UNIVERSIDAD NACIONAL DE RIO CUARTO

Trabajo de tesis para la obtención del grado de Licenciatura en Ciencias de la Computación

Título : Verificación de modelos con Cálculo-μ

Autor
Luciano Putruele

Director de Tesis: *Dr. Pablo F. Castro*
Co-Director de Tesis: *Dr. Germán Regis*

Rio Cuarto, Argentina
Abril 2016

Resumen

En esta tesis desarrollaremos una herramienta de verificación de modelos (Model Checking), llamada MC2, sobre programas formalizados en un lenguaje lógico simple que también será desarrollado en este trabajo. La herramienta de verificación se encargará de, valga la redundancia, verificar propiedades, caracterizadas en la forma de la lógica temporal Calculo Mu, sobre dichos programa. Cabe destacar que el lenguaje de programación y el verificador funcionan como una sola unidad ya que en un mismo programa MC2 se define tanto el modelo como las propiedades a verificar (si las hay). A todo esto, el metalenguaje utilizado para el desarrollo de estas herramientas es Haskell.

Una motivación para el desarrollo de esta tesis fue la utilización de un lenguaje funcional (Haskell) para el desarrollo de la herramienta en su totalidad, en lugar de utilizar lenguajes imperativos u orientados a objetos.

Lo que se propone con este proyecto es explorar otras alternativas para especificar propiedades que un modelo deba satisfacer, siendo la alternativa en este trabajo el Cálculo Mu, en vez de las lógicas temporales más utilizadas en este tipo de herramientas, como ser LTL, CTL y CTL*, ya que esto trae la ventaja de que el Cálculo Mu es más expresivo que los anteriormente nombrados. Adicionalmente los modelos de los programas MC2 se basan en la definición de reglas de transición usando proposiciones lógicas atómicas con lo cuál es transparente ver la estructura del modelo como una máquina de transición de estados más allá de que internamente se los trata simbólicamente como fórmulas para mejorar el rendimiento.

Palabras clave: .

Agradecimientos

Agradezco a ...

Una dedicatoria muy especial es para ...

List of Figures

Contents

Chapter 1

Introducción

La verificación de modelos o comunmente *model checking* es una técnica automática para verificar sistemas reactivos con una cantidad finita de estados, por ejemplo protocolos de comunicación y diseños de circuitos. Las especificaciones de las propiedades a verificar son expresadas en una lógica temporal proposicional, y el sistema esta modelado como un grafo. Se utiliza una busqueda eficiente para determinar automáticamente si las especificaciones son satisfechas por el grafo. Esta técnica fue desarrollada originalmente en 1981 por Clarke y Emerson. Quielle y Sifakis descubrieron independientemente una técnica similar de verificación poco después. Esta técnica tiene varias ventajas importantes sobre probadores de teoremas para verificación de circuitos y protocolos. La mas importante es que es automática. Normalmente, el usuario provee una representación de alto nivel del modelo y una especificación de la propiedad que se desea verificar. El model checker terminará devolviendo la respuesta True indicando que el modelo satisface la especificación o dará una traza de ejecución a modo de contraejemplo si el modelo no satisface la propiedad. Esta es una propiedad muy importante a la hora de encontrar bugs sutiles.

1.1 Objetivos

El objetivo principal de este proyecto es explorar otras alternativas para especificar propiedades que un modelo deba satisfacer, siendo la alternativa en este trabajo el Cálculo-μ, en lugar de las lógicas temporales más utilizadas en este tipo de herramientas, como ser LTL, CTL y CTL*, además de que esto trae la ventaja de que el Cálculo-μ es más expresivo que los anteriormente nombrados, por lo

tanto, las propiedades descritas en LTL,CTL y CTL* pueden ser descritan también usando Cálculo-μ. Como objetivo secundario cabe destacar la utilización del paradigma funcional de programación para el desarrollo de la herramienta en su totalidad, en lugar de utilizar paradigmas imperativos u orientados a objetos que son normalmente mas utilizados en el área.

1.2 Estructura

Primero analizaremos conceptos básicos para la comprensión de esta tesis, conceptos como la verificación de modelos, representación de estos modelos, el concepto de lógicas temporales, y en particular el Cálculo-μ. Más tarde introduciremos la noción de verificación simbólica de modelos para así luego entrar en detalle sobre la implementación del model checker MC2. Luego estableceremos la idea detrás del lenguaje MC2, para entrar luego en detalle con la sintaxis y la semántica del lenguaje. Por último se analizará la implementación concreta del Por último veremos algunos ejemplos para afianzar el entendimiento de la aplicación práctica de esta herramienta.

Chapter 2

Conceptos preliminares

En el diseño de software y hardware para sistemas complejos, cada vez es más el tiempo y esfuerzo dedicado a la verificación en vez de la construcción. Se buscan técnicas para reducir y facilitar el trabajo de la verificación y a la vez incrementar su cobertura. Los métodos formales ofrecen un gran potencial para obtener una integración temprana de la verificación en el proceso de diseño, para proveer técnicas de verificación mas efectivas, y para reducir el tiempo de verificación en general.

La verificación de modelos o model checking es una técnica automática de verificación de propiedades sobre sistemas con una cantidad finita de estados. Es una alternativa interesante con respecto al testing o las simulaciones ya que a diferencia de estas técnicas, el model checking hace una prueba exhaustiva del sistema, es decir, analiza todas las trazas posibles de la ejecución del sistema en cuestión. Sin embargo, esto trae un problema, esto es el problema de la explosión de estados. Esto ocurre en sistemas con muchas interacciones internas, y que pueden hacer crecer exponencialmente el espacio de estados posibles del sistema, ya que la prueba es exhaustiva no se puede ignorar ningún estado posible. En los últimos años se ha logrado un gran progreso en cómo lidiar con este problema mediante formas más compactas de representar al sistema, como por ejemplo, una representación simbólica del modelo del sistema.

El modelo del sistema generalmente es generado automaticamente desde una descripción del modelo en un un lenguaje similar a alguno de programación como C, Java, etc. Hay que notar que la especificación de la propiedad prescribe lo que el systema debe y no debe hacer, en cambio la descripción del modelo señala como se comporta el sistema. El verificador de modelos examina todos los estados relevantes del sistema para verificar si satisface o no la propiedad deseada.

El proceso del verificación de modelos consta de varias fases diferenciables:

Modelado: Hay que modelar el sistema en cuestión usando el lenguaje de descripción de modelos del verificador, y formalizar la propiedad que se desea verificar usando el lenguaje de especificación de propiedades.

Ejecución: Ejecutar el verificador para corroborar la validez de la propiedad en el modelo del sistema.

Análisis: Si la propiedad fue satisfecha, verificar la próxima propiedad (si la hay), si en cambio, no fue satisfecha, hay que refinar el modelo y/o la propiedad y finalmente, repetir el proceso.

2.1 Modelado de sistemas

En esta sección veremos cómo representar un modelo explícitamente mediante una estructura de Kripke, más tarde veremos otra forma de representación llamada simbólica que representa el modelo mediante una fórmula lógica de primer orden.

Sea AP un conjunto de proposiciones atómicas, una estructura de Kripke M sobre AP es una cuatro-upla $M = (S, S_0, R, L)$ donde:

1. S es un conjunto finito de estados.
2. $S_0 \in S$ es el conjunto de estados iniciales.
3. $R \in S \times S$ es una relación de transición total, es decir para cada estado $s \in S$ existe un estado $s' \in S$ tal que $R(s, s')$ vale.
4. $L\colon S \to 2^{AP}$ es una función que etiqueta a cada estado con el conjunto de proposiciones atómicas que son verdaderas en ese estado. Un camino en la estructura M desde un estado s es una secuencia infinita de estados $p = s_0, s_1, s_2, s_3, ...$, tal que $s = s_0$ y $R(s_i, s_{i+1})$ vale para todo $i > 0$.

Sea $V = v_1, v_2, ..., v_n$ el conjunto de variables del sistema y sea D el dominio, llamaremos una valuación de V a una función que asocia a cada variable de V un valor de D.

Un estado del sistema se puede representar como una valuación de las variables del sistema. Una proposición atómica de la forma $v = d$ donde $v \in V$ y $d \in D$ será verdadera en un estado s si y solo si $s(v) = d$. Dada una valuación, podemos escribir una fórmula que sea verdadera precisamente para esa valuación, por ejemplo si tenemos $V = \{x, y, z\}$ y la valuación $(x \leftarrow True, y \leftarrow True, z \leftarrow False)$ entonces derivamos la fórmula $(x \wedge y \wedge !z)$. En general, una fórmula puede ser verdadera para varias valuaciones. Si adoptamos la convención de que una fórmula representa el conjunto de todas las valuaciones que la hacen verdadera, entonces podremos describir ciertos conjuntos de estados como fórmulas de primer orden.

En particular, el conjunto de los estados iniciales del sistema puede describirse como una fórmula de primer orden S_0 sobre las variables en V. Una transición del sistema se puede representar como un par ordenado de valuaciones, de forma similar podemos describir conjuntos de transiciones mediante una formula para ese par, pero para poder expresar la fórmula se necesita una copia V' de V para hablar de siguiente estado, en V' todas las variables estan primadas. Por ejemplo si tenemos una transición $(x \leftarrow True, y \leftarrow True, z \leftarrow False, (x \leftarrow True, y \leftarrow True, z \leftarrow True))$, podemos derivar la fórmula $(x \wedge y \wedge !z \wedge x' \wedge y' \wedge z')$.

Consideremos el siguiente ejemplo, tenemos $V = \{x, y, z\}$ y $D = \{True, False\}$, $S_0(x, y, z) = (x = True \wedge y = True \wedge z = False)$, y tenemos solo una transición: $z := x \wedge y$,consideremos False = 0 y True = 1 por una cuestión de facilidad de lectura. Definimos asi la estructura de kripke (3.1) de la siguiente manera:

$$S = D \times D \times D$$

$$S_0 = \{(1, 1, 0)\}$$

$$R = \{((1, 1, 0), (1, 1, 1)), ((1, 1, 1), (1, 1, 1))\}$$

$$L(1, 1, 0) = \{x = 1, y = 1, z = 0\},$$

$$L(1, 1, 1) = \{x = 1, y = 1, z = 1\}$$

El único camino posible en esta estructura partiendo del estado inicial es: (1, 1, 0), (1, 1, 1), (1, 1, 1), (1, 1, 1) ...

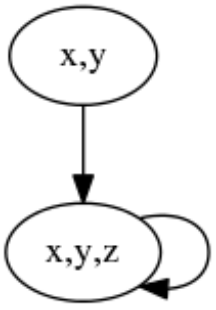

Figure 2.1: Estructura de Kripke para este ejemplo.

2.2 Especificación de propiedades

Ahora describiremos una lógica para especificar propiedades deseadas en una estructura de Kripke u otra máquina de transición de estados. La lógica utiliza proposiciones atómicas y operadores como la disyunción y la negación para construir expresiones más complicadas que describan propiedades sobre estados. La lógica temporal es un formalismo que permite describir secuencias de transiciones entre estados en un sistema reactivo, nos interesa saber si en algún momento se llega a un estado determinado o que nunca se llegue a un deadlock. Para esto introduce nuevos operadores especiales que permiten hablar sobre tiempo. Estos operadores pueden combinarse con los operadores lógicos conocidos. Analizaremos a continuación una lógica temporal muy potente llamada Cálculo-μ.

2.3 Cálculo-μ

El Cálculo-μ es un poderoso lenguaje para expresar propiedades de sistemas de transición de estados al usar operadores de punto fijo. El Cálculo-μ ha generado mucho interés entre investigadores en verificación asistida por computadoras. Este interés surge del hecho de que muchas lógicas temporales pueden ser codificadas por el Cálculo-μ. Otra fuente de interés en el Cálculo-μ viene de la existencia de algoritmos eficientes de verificación de modelos para este formalismo. Como consecuencia, los procedimientos de verificación para muchas lógicas temporales y modales pueden ser descriptas al traducirse al Cálculo-μ. Hay varias versiones del Cálculo-μ, concretamente usaremos la versión proposicional de Kozen.

2.3.1 Sintaxis

Sea $M = (S, T, L)$ una estructura de Kripke y sea $VAR = Q, Q1, Q2, \ldots$ un conjunto de variables relacionales, donde a cada variable relacional se le puede asignar un subconjunto de S, construimos una μ-fórmula como sigue:

Si $p \in AP$, entonces p es una fórmula.
Si $Q \in VAR$, entonces Q es una fórmula.
Si f y g son fórmulas, entonces $\neg f$, $f \vee g$, y $f \wedge g$ son fórmulas.
Si f es una fórmula, entonces $\Box f$ y $\Diamond f$ son fórmulas.
Si $Q \in VAR$ y f es una fórmula entonces $\mu Q.f$ y $\nu Q.f$ son fórmulas.

Las variables pueden estar libres o ligadas en una fórmula a través de un operador de punto fijo. Una fórmula cerrada es una fórmula sin variables libres.

2.3.2 Semántica

El significado intuitivo de $\Diamond f$ es "Es posible realizar una transición a un estado donde f vale", similarmente $\Box f$ significa "f vale en todos los estados alcanzables por medio de una transición" Los operadores μ y ν expresan puntos fijos menores y mayores respectivamente. El conjunto vacío de estados se denota con $False$ y el conjunto de todos los estados S se denota con $True$.

Ejemplos:

$\nu Z \cdot f \wedge \Box Z$ se interpreta como "f es verdadera siempre en todo camino".
$\mu Z \cdot f \vee \Diamond Z$ se interpreta como "existe un camino hacia un estado donde f vale".
$\nu Z \cdot \Diamond True \wedge \Box Z$ se interpreta como "no hay estados que no tengan transiciones hacia otros estados".

Formalmente, una fórmula f se interpreta como un conjunto de estados donde f es verdadera, escribimos este conjunto como $[[f]]$ sobre un sistema de transición de estados M y un ambiente $e : VAR \to 2^S$, denotaremos $e[Q \leftarrow W]$ como un ambiente que es igual a e solo que Q ahora tiene el valor W. el conjunto $[[f]]$ sobre M y e se define recursivamente de la siguiente manera:

$$
\begin{aligned}
[[p]]\ M\ e &= \{s \mid p \in L(s)\} \\
[[Q]]\ M\ e &= e(Q) \\
[[\neg f]]\ M\ e &= S \setminus [[f]]\ M\ e \\
[[f \wedge g]]\ M\ e &= [[f]]Me \cap [[g]]\ M\ e \\
[[f \vee g]]\ M\ e &= [[f]]Me \cup [[g]]\ M\ e \\
[[\Diamond f]]\ M\ e &= \{s \mid \exists t : s \to t \wedge t \in [[f]]\ M\ e\} \\
[[\Box f]]\ M\ e &= \{s \mid \forall t : s \to t \to t \in [[f]]\ M\ e\}
\end{aligned}
$$

$[[\mu Q.f]]\ M\ e$ es el menor punto fijo del predicado transformador $t : 2^S \to 2^S$ definido como $t(W) = [[f]]\ M\ e[Q \leftarrow W]$

$[[\nu Q.f]]\ M\ e$ es el mayor punto fijo del predicado transformador $t : 2^S \to 2^S$ definido como $t(W) = [[f]]\ M\ e[Q \leftarrow W]$

2.3.3 Algoritmo de verificación de modelos explícitos

La semántica anterior es a su vez el algoritmo de verificación explicita de modelos para Cálculo-μ.

Chapter 3

Verificación simbólica de modelos

El algoritmo de verificación de modelos con estados explícitos para Cálculo-μ presentado anteriormente tiene un problema, es muy susceptible a que ocurra una explosión en el tamaño del modelo, especialmente si el grafo de transición de estados se extrae de un sistema concurrente con muchos componentes. En esta sección se describe un algoritmo de verificación de modelos simbólicos para Cálculo-μ que opera sobre estructuras de Kripke, esta vez representadas no de manera explícita, sino de manera simbólica a través de fórmulas lógicas.

3.1 Representación de fórmulas lógicas

Los árboles binarios de decisión ordenados (OBDDs) son formas canónicas de representación de fórmulas lógicas. Son considerablemente mas compactos que las formas normales tradicionales como la forma normal conjuntiva y la forma normal disyuntiva, y pueden ser manipulados eficientemente. Por esto, los OBDDs han sido utilizados ampliamente para una variedad de aplicaciones en el diseño asistido por computadoras, incluyendo simulacion simbólica, verificación de lógica combinatoria y, mas recientemente, verificación de sistemas concurrentes con estados finitos.

Para entender la necesidad de usar OBDDs, consideremos primero los árboles binarios de decisión. Un arbol binario de decisión es un árbol dirigido con raiz que consiste en vertices terminales y no terminales. Cada vertice no terminal v esta etiquetado por una variable $var(v)$ y tiene dos hijos: $izq(v)$ corresponde al caso en que v tenga el valor 0 y $der(v)$ en caso contrario. Cada vértice terminal esta etiquetado por una constante $valor(v)$ la cual es 0 o 1. Un árbol binario de

decisión para la fórmula $f(a,b,c) = (a \wedge b) \vee (a \wedge c)$ es mostrado en la figura . Uno puede decidir si una asignación particular a las variables hace verdadera la fórmula o no al atravesar el árbol desde la raíz hasta un vertice terminal. Por ejemplo, la asignación $\{a \leftarrow 1, b \leftarrow 0, c \leftarrow 0\}$ lleva al vértice terminal 0, por lo tanto la fórmula es falsa para esta asignación.

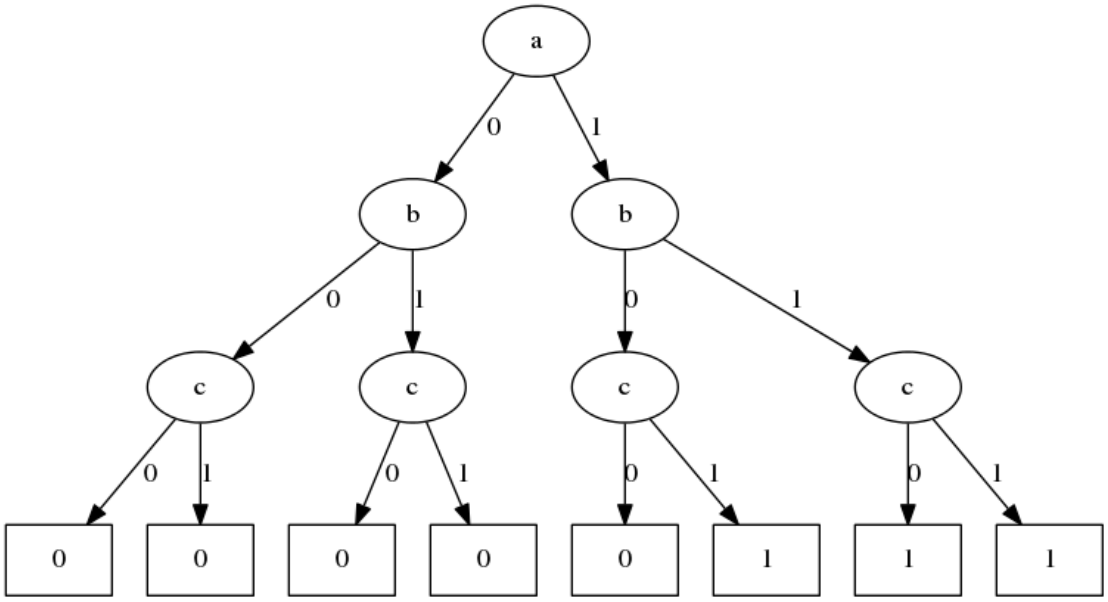

Figure 3.1: Árbol binario de decisión para este ejemplo.

Los árboles binarios de decisión no proveen una representación muy concisa para las funciones lógicas. De hecho, tienen el mismo tamaño que las tablas de verdad. Afortunadamente, es común que haya mucha redundancia en tales árboles. Por ejemplo, en la figura todos los caminos donde a tiene el valor 0 llevan al nodo terminal 0, por lo tanto no seria necesario analizar los valores de b y c en esta rama. Esto lleva a pensar que hay formas de reducir el tamaño del árbol unificando subárboles isomorfos. Esto da como resultado un grafo acíclico dirigido (DAG) llamado diagrama binario de decisión (BDD). Mas precisamente, un BDD es un grafo con raiz, dirigido y acíclico con dos tipos de vertices, vertices terminales y no terminales. Estos tienen el mismo significado que en el caso de los árboles. Cada BDD B con raiz v determina una función lógica $f_v(x_1, ..., x_n)$ de la siguiente manera:

1. Si v es un vértice terminal: (a) Si $valor(v) = 1$ entonces $f_v(x_1, ..., x_n) = 1$. (b) Si $valor(v) = 0$ entonces $f_v(x_1, ..., x_n) = 0$.

2. Si v es un vértice no terminal con $var(v) = x_i$ entonces f_v es la función

$$f_v(x_1, ..., x_n) = (\neg x_i \wedge f_{izq(v)}(x_1, ..., x_n)) \vee (x_i \wedge f_{der(v)}(x_1, ..., x_n))$$

3.2 Diagramas binarios de decisión

Chapter 4

Lenguaje MC2

MC2 es un lenguaje de modelado de sistemas basado en la noción de las estructuras de Kripke, es decir que con este lenguaje se puede establecer cómo se comportará una máquina de transición de estados en cuyos estados valen ciertas proposiciones atómicas. En esta sección describiremos la sintaxis de MC2, y luego la semántica.

4.1 Tipos

En MC2 tenemos proposiciones atómicas (AP) representadas por cadenas, cada una tiene asociada un valor lógico (True o False), para lo cual existe un tipo Env (environment) que consta de una lista de pares de proposiciones atómicas y sus valores lógicos asociados. Un valor de tipo Env representa el estado del sistema en un momento dado.

$typeName = StringtypeAP = StringtypeEnv = [(AP, Bool)]typeAssoc = Name -> OBDDAP$

Assoc es un tipo que se utiliza en la semántica de las fórmulas de cálculo mu, este representa una función que toma el nombre de una variable y devuelve el valor asociado (representado por un OBDD).

4.2 Sintaxis

Una especificación MC2 se divide en cuatro partes bien diferenciadas: Una sección donde se declaran las variables (proposiciones atómicas), otra donde se describen las reglas de transición, luego una sección donde se establece el estado inicial, y por último una parte donde se detalla la/s propiedades a verificar en el modelo.

Bibliography

[1] J. Doe, *The Book without Title*. Dummy Publisher, 2100.

Apéndices

Appendix A

Código

Código.