INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA
ESCUELA DE INGENIERÍA Y GESTIÓN

VERIFICACIÓN UTILIZANDO DYNAMITE DE LA CORRECCIÓN DEL MODELO CHORD

AUTOR/ES: Dantur, Juan Pablo (Leg. N° 54623)
Ocamica, Santiago (Leg. N° 53346)

DOCENTE/S TITULAR/ES O TUTOR/ES: Frías, Marcelo; Moscato, Mariano

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN INFORMÁTICA

Lugar: ITBA, Sede Distrito Tecnológico, Lavardén 315, CABA
Fecha: 12/08/2019

Abstract

La herramienta Dynamite permite traducir modelos formales realizados en el lenguaje de modelado Alloy en teorías del demostrador semiautomático PVS, pero no fue probado con modelos tan complejos como un modelo completo del protocolo Chord. El objetivo inicial de este trabajo fue demostrar la correctitud del protocolo Chord utilizando Dynamite. Durante el desarrollo del Proyecto Final nos encontramos con limitaciones relacionadas a la traducción de Alloy a PVS por parte de Dynamite. Por este motivo, el objetivo del trabajo pasó a ser el de probar Dynamite con el modelo de Chord mencionado anteriormente y de esta forma evaluar la aplicabilidad de Dynamite para este objetivo. Se pudo contribuir al avance de Dynamite a través del reporte de problemas relacionados a la traducción del modelo de Alloy de Chord. También quedaron demostrados con Dynamite todos los lemas sobre los espacios de identificadores y dos de los cinco teoremas principales sobre la correctitud de Chord dado que su invariante se cumple.

Objetivos

Inicialmente el objetivo de este trabajo fue demostrar de manera semiautomática utilizando Dynamite la correctitud del modelo de Chord planteado en [1], con las correcciones y demostraciones planteadas en [2], utilizando como input el modelo de Alloy definido en [3] y lemas auxiliares definidos sobre ese modelo.

En el intento de demostrar los teoremas se descubrió que Dynamite no es completamente compatible con el modelo de Alloy utilizado, ya que Dynamite fue implementado sobre una versión de Alloy anterior a la utilizada en [3], por lo que el objetivo del trabajo se modificó a testear el software Dynamite y a reportar las limitaciones encontradas en el mismo al equipo de mantenimiento de Dynamite, utilizando un modelo cuya complejidad supera largamente a los utilizados anteriormente.

El Protocolo Chord

Chord es un protocolo y algoritmo para la implementación de una tabla de hash distribuida (DHT) peer-to-peer con una distribución de claves basada en *consistent hashing*. En Chord los nodos se referencian formando un *anillo* de tamaño $\mathbf{2^n}$ en el cual los nodos mantienen referencias a nodos a distancias $[2^0,2^1, \ldots, 2^{n-1}]$ en la llamada *finger table*. En el anillo a cada nodo se le asigna un identificador pseudoaleatorio perteneciente al espacio **mod 2^n-1** basado el hash SHA1 de su dirección IP. Esta distribución de nodos en el anillo tiene alta probabilidad de tener buenas características de *balanceo de carga y almacenamiento*. La *finger table* se utiliza en los comandos de búsqueda para optimizar las mismas, reduciendo la cantidad de consultas de O(n) a $O(\log_2 n)$.

Chord fue definido por *Stoica et al.* por primera vez en [1], que establece condiciones de estado inicial y operaciones de mantenimiento del anillo ante unión, abandono y falla de nodos del anillo. El objetivo de estas es mantener las propiedades de *tolerancia a fallas de nodos*, *ruteo eficiente*, y *consistencia*. Luego *Zave* en [2] probó la existencia de fallas en el mantenimiento de las propiedades anteriores y estableció nuevas condiciones de estado inicial y operaciones de mantenimiento, además de demostrar respecto a la correctitud de las mismas la inexistencia de contraejemplos para redes de hasta 10 nodos utilizando el Analizador de Alloy. En el mismo trabajo, *Zave* define una *invariante* que garantiza la correctitud del protocolo.

El lenguaje de modelado Alloy

Alloy se compone de un lenguaje que sirve para describir estructuras y una herramienta para explorarlas.

Un modelo definido con el lenguaje Alloy describe un conjunto de estructuras implícitamente mediante varias restricciones que tienen las mismas. En el caso de Chord, se describen propiedades que debe cumplir en todo instante de tiempo una red (modelada mediante un conjunto de Nodos, donde cada Nodo conoce a su sucesor entre otras cosas). También se describen “funciones” mediante las cuales se cambia el estado de una red, que reciben un estado y retornan otro estado de esa misma red.

La herramienta “Analizador de Alloy” sirve para examinar las restricciones definidas en el modelo de Alloy y verificar parcialmente si se cumplen en la estructura definida. La verificación es parcial dado que Alloy no está diseñado para verificar formalmente que las propiedades se cumplan. Solo puede verificar que no existan contraejemplos para una propiedad en un espacio dado, o que se cumpla bajo ciertas condiciones. También se puede usar la herramienta para generar instancias del modelo definido y verificar a simple vista las propiedades de las mismas.

La Herramienta Dynamite

Dadas las limitaciones previamente mencionadas de Alloy, es decir, la incapacidad para verificar una propiedad, se desarrolló un demostrador semiautomático de teoremas compatible con el lenguaje de modelado Alloy, llamado Dynamite [4]. Para realizar el trabajo, se utilizó Dynamite [4]. Dynamite es una extensión de PVS que incluye un cálculo completo para Alloy. También incluye extensiones para reducir el esfuerzo de las pruebas mediante el análisis automático de nuevas hipótesis con la ayuda del analizador de Alloy. Dynamite complementa el análisis parcial automático ofrecido por Alloy con verificación semiautomática a través de la prueba de teoremas.

Para explicar el funcionamiento de esta herramienta, se debe conocer el funcionamiento de las herramientas Alloy y PVS. La herramienta Alloy consiste de un lenguaje open-source en el cual se pueden modelar diversas estructuras de manera declarativa (como son, por ejemplo, un sistema de archivos, o una red de nodos conectados de manera circular). Para lograr esto permite definir distintas "clases" con propiedades, así como herencia entre las mismas, similar a un lenguaje orientado a objetos. Además de esto permite establecer propiedades, restricciones y "hechos" respecto de las estructuras que se pueden definir. Se pueden definir *asserts,* que son afirmaciones sobre la estructura en cuestión que luego deberán ser demostradas para todas las instancias de la estructura. Si bien Alloy cuenta con una interfaz donde pueden analizarse los *asserts*, el problema es que Alloy sólo puede demostrarlos con el auxilio de un SAT solver. Esto último requiere fijar una cota máxima para el tamaño de los dominios de datos que se utilizarán en el modelo. Esto puede servir para descartar *asserts* falsos, pero no sirve para demostrar efectivamente que un *assert* es verdadero ya que una aserción puede no tener contraejemplos dentro de las dimensiones seleccionadas, pero tenerlos al considerar dominios con más datos.

Por otra parte, PVS es un entorno que sirve para demostrar propiedades formalmente. Cuenta con distintas estrategias que pueden ser usadas para demostraciones. Por ejemplo, permite instanciar un cuantificador universal ("Para todo x, se cumple propiedad y") y re-escribirlo como una sentencia particular ("Para x_1, se cumple propiedad y").

Lo que permite hacer Dynamite, es importar modelos de Alloy para ser utilizados con PVS y así hacer demostraciones semiautomáticas sobre los modelos definidos, lo cual no se podría hacer solo con Alloy ya que está diseñado para hacer búsquedas sobre un espacio acotado.

Dynamite debe ejecutarse sobre una interfaz Emacs, y consiste en una herramienta que procesa todas las propiedades definidas en la biblioteca estándar de Alloy, así como las que fueron definidas para la estructura en particular (en este caso, los modelos de Chord), y permite utilizarlas junto con estrategias de PVS vía comandos para demostrar formalmente diversos *asserts* correspondientes a la estructura definida.

Instalación y primeros pasos en Dynamite

Lo primero que se tuvo que hacer fue la instalación de Dynamite. La complejidad de este paso se debe a que Dynamite requiere versiones específicas de Alloy, PVS y Emacs, y requirió modificar algunos archivos de bash que contienen las rutas a los binarios de dichos programas.

Una vez lograda la instalación de Dynamite, el próximo paso a realizar fue aprender sintaxis de Alloy junto con el procedimiento para demostrar propiedades en Dynamite. Para aprender la sintaxis de Alloy, se construyó una modelo que representa un File System. Este modelo contiene *Objects*, los cuales pueden ser de dos tipos: *File* o *Dir*. También define el tipo *Name* y el tipo *Root*, un subtipo de *Dir* del cual solo puede haber una instancia (escrito en Alloy como *one sig Root extends Dir*). Cada *Dir* posee dos propiedades: *parent* que representa 0 o 1 *Dir* padre (escrito en Alloy como *lone Dir*) y *contents*, que representa una relación entre un *Name* y 0 o 1 *Object*. Dicho de otra forma, cada *Dir* se relaciona con 0 o 1 *Dir* mediante la relación binaria *parent*, y la relación ternaria *contents* relaciona un par conformado por un *Dir* y un *Name* con 0 o 1 *Object*.

Además de los objetos mencionados, el File System cuenta con una función *ancestors*, la cual dado un *Dir d* representa la unión de *d.parent, d.parent.parent, …*, un predicado *parentIsWellDefined*, que incluye todos los *Dir d* para los cuales el padre está bien definido, es decir, cualquier *Dir* contenido en *d.contents* va a tener como padre a *d*, y por último una serie de *facts* (Propiedades que se dan por válidas) y *asserts* (Propiedades que se desean demostrar).

Contribuciones

Problemas de Dynamite encontrados

Incompatibilidad con el keyword de Alloy “seq”

La versión con la que comenzamos a realizar el trabajo era incompatible con el keyword de alloy `seq` utilizado para declarar un campo como una secuencia ordenada de datos.
Una versión publicada durante la realización del proyecto corrige este problema.

Problema con expand “lte”

`(expand "lte”)` debía expandir `lte[n1,n2]` a `lt[n1,n2] || n1=n2` pero no hacía nada.
Inicialmente se resolvió con un parche provisorio en Dynamite, y luego fue publicada una nueva versión corrigiendo este problema de forma correcta.

Problema con la traducción del predicado elems

```
Rule? (inst -1 "s_1.principals" "elems[(m_1.(s_1.succ)).list]")
```

Al intentar instanciar un cuantificador con una expresión que incluye la función `elems` Dynamite lanzaba un error de chequeo de tipos debido a la forma en que estaba traduciendo esta función. Esto fue resuelto en una versión publicada de Dynamite.

Traducción a Dynamite de conjuntos por comprensión

Al intentar traducir el predicado `PrincipalsAreRingMembers` se obtiene un resultado incorrecto por un error de Dynamite al traducir los conjuntos definidos por comprensión que impide continuar con las demostraciones que incluyan este predicado. La solución a este problema queda pendiente al día de presentación de este informe.

Chord

Las demostraciones específicas de Chord de más alto nivel se pueden dividir en tres grupos. Las que prueban que la invariante implica que el modelo es correcto:

- InvariantEnsuresNoDuplicates
- InvariantEnsuresOrderedSuccessorLists
- InvariantEnsuresPrincipalsInRing
- InvariantEnsuresOneOrderedRing
- InvariantEnsuresConnectedAppendages

Las que prueban que la invariante se mantiene entre los cambios de estado permitidos en la red:

- FailPreservesInvariant
- JoinPreservesInvariant
- StabilizeFromSuccessorPreservesInvariant
- StabilizeFromPredecessorPreservesInvariant
- RectifyPreservesInvariant

Y las que prueban que las operaciones generan progreso en el estado de la red hacia un ideal:

- NonIdealImpliesChangeEnabled
- IdealImpliesNoChangeEnabled
- EffectiveStabilizeFromSuccChangesState
- EffectiveStabilizeFromPrdcChangesState
- EffectiveRectifyFromSuccChangesState
- EffectiveRectifyFromPrdcChangesState


EffectiveRectifyFromPrdcChangesState
EffectiveRectifyFromSuccChangesState
EffectiveStabilizeFromPrdcChangesState
EffectiveStabilizeFromSuccChangesState
IdealImpliesNoChangeEnabled
NonIdealImpliesChangeEnabled
LossOfPrincipal
RectifyPreservesInvariant
StabilizeFromPredecessorPreservesInvariant
StabilizeFromSuccessorPreservesInvariant
JoinPreservesInvariant
FailPreservesInvariant
InvariantEnsuresConnectedAppendages
InvariantEnsuresOneOrderedRing
InvariantEnsuresPrincipalsInRing
UnnamedAssertion
PrincipalsInSuccLists
NodeCardInclusion
SetCard
InTransitivity
DuplicatesInExtSuccImpliesAllPrincipals
InvariantEnsuresNoDuplicates
subSeqCard
InvariantEnsuresOrderedSuccessorLists
NoDuplicatesAltDef
InsertInStartContainsObjectInStart
Untitled
lteTotal
AnyIncludedInAny
ltIrreflexive
AnyBetweenAny
ltTotal
IncludedReversesBetween
ltTransitive
LocationOfUnskippedNode
ltAntisymmetric
distinctAtoms
ChainedIntervals
lteTransitive
lteSymmetry

Chord: Demostraciones exitosas

Pudimos demostrar con éxito todos los lemas acerca de los espacios de identificadores en un anillo, entre ellos: AnyBetweenAny, LocationOfUnskippedNode y ChainedIntervals.
Las demostraciones de Chord que pudimos probar son ambas del grupo que prueba que la invariante implica que el modelo es correcto: InvariantEnsuresNoDuplicates y InvariantEnsuresOrderedSuccessorLists

Chord: Demostraciones fallidas

Las demostraciones que intentamos realizar sin éxito son InvariantEnsuresPrincipalsInRing, InvariantEnsuresOneOrderedRing ya que Dynamite tiene un error para traducir los conjuntos definidos por comprensión utilizados en esa parte del modelo y en gran parte del resto que quedó sin demostrar.

Conclusiones

Dynamite es una herramienta muy poderosa que facilita la demostración formal de propiedades de modelos previamente modelados analizados con Alloy, pero todavía tiene mucho espacio de mejora en cuanto a la traducción de modelos de Alloy que utilizan características avanzadas de este como por ejemplo conjuntos definidos por comprensión.

El reporte de los problemas encontrados con Dynamite durante la realización de este trabajo llevó a que se lancen varias versiones nuevas corrigiendo limitaciones de compatibilidad y traducción de modelos de Alloy, pero se requiere un análisis más exhaustivo para poder corregir Dynamite en su totalidad y poder demostrar correctamente todas las propiedades planteadas en el modelo de Alloy analizado.

Trabajo Futuro

En este trabajo sólo se analizó un subconjunto de la totalidad de los teoremas de Chord, quedan dos grandes ramas del modelo por probar y analizar la aplicabilidad de Dynamite para probar cada teorema. Para poder probar lo restante se deberá corregir la traducción de los conjuntos definidos por comprensión e intentar.

Un enfoque interesante puede ser intentar continuar probando Chord de una forma más “top-down” es decir arrancando por las pruebas de más alto nivel introduciendo lemas asumidos en principio ciertos y demostrados luego.

Otra opción es intentar probar otro modelo con la herramienta Dynamite, ya que otros dominios pueden utilizar otras características de Alloy cuya traducción no haya sido probada correcta anteriormente.

Referencias Bibliográficas

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan,“Chord: A scalable peer-to-peer lookup service for Internet applications” in *Proceedings of ACM SIGCOMM.* ACM, August 2001.

[2] Pamela Zave “Reasoning about Identifier Spaces: How to Make Chord Correct” *IEEE Transactions on Software Engineering,* 43(12):1144-1156, December 2017

[3] http://www.pamelazave.com/correctChord.als

[4] Mariano M. Moscato, Carlos G. Lopez Pombo and Marcelo F. Frias, 2013. “Dynamite: A Tool for the Verification of Alloy Models Based on PVS” *ACM Trans. Softw. Eng. Methodol.* V, N, Article A (January YYYY), 37 pages.