Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Anton Galitch

Director:

Codirector:

Buenos Aires, 2014

Resumen

tes etst set set set set s

Palabras claves: test

Índice general

1. INTRODUCCIÓN

1.1. Motivación

Si bien el testing es la técnica más frecuentemente utilizada como validación de corrección del software, no es posible alcanzar resultados contundentes mediante su aplicación. En la amplia mayoría de los casos, esta validación parcial es suficiente. Sin embargo, en el desarrollo de sistemas críticos es necesario contar con técnicas que permitan alcanzar mayores grados de certeza sobre la satisfacción de cierto conjunto de propiedades.

En tales contextos se suelen utilizar herramientas basadas en métodos formales tales como demostradores de teoremas, con los que se puede lograr certeza absoluta sobre las satisfacción de las propiedades deseadas. No obstante, una contra importante de estos métodos es su falta de automaticidad. Esta falencia está usualmente ligada al poder expresivo del lenguaje utilizado: a mayor expresividad del lenguaje, las herramientas de análisis brindan menos posibilidades de automaticidad.

Un ejemplo clásico de esta correlación está dado por los lenguajes de primer orden. Su expresividad impide que se pueda construir un programa que sea capaz de decidir sobre la verdad o falsedad de una sentencia cualquiera. Sin embargo, se han desarrollado diversas herramientas que son capaces de realizar análisis automáticos para una gran cantidad de casos (como Mace4 [5], E [6], SPASS [7], etc.).

Por otro lado, la complejidad actual de los sistemas informáticos fuerza a los equipos de desarrollo a distribuir la tarea de relevar los requerimientos referidos a distintos aspecto de estos sistemas. Es sabido que diferentes aspectos de un sistema de software son mejor capturados por lenguajes de diferentes características. Esta apreciación vale tanto para lenguajes semiformales como la enorme diversidad de lenguajes diagramáticos de UML [8], como para lenguajes formales en donde diferentes lógicas resultan apropiadas para poner de relieve diferentes comportamientos de un sistema informático (algunos ejemplos son, linear temporal logics, tanto su versión proposicional, de primer orden o con operadores de pasado [9], [11], para caracterizar propiedades de las ejecuciones de un sistema, lógicas dinámicas, también en sus versiones proposicional y de primer orden [12], etc.). Si bien esta práctica de construir especificaciones heterogéneas facilita la comprensión y documentación de los sistemas, impone una nueva dificultad a los métodos de análisis, que usualmente se aplican sobre un lenguaje único.

Para salvar esta dificultad, desde hace ya algunos años, se ha estado trabajando en el desarrollo de fundamentos formales y herramientas de análisis de sistemas descriptos en forma heterogénea. Del lado de los fundamentos formales encontramos los trabajos seminales en el campo de la especificación algebraica de software como son [13], [15], [16], [18], mientras que del lado de las herramientas se observan trabajos como [20], [21], [22], [24].

Otra dimensión de la heterogeneidad es la enorme diversidad de herramientas de análisis que existen hoy en día para cada lenguaje. Así, dado un lenguaje de especificación o diseño de software particular, encontramos un verdadero ecosistema de herramientas que nos brindan la posibilidad de aplicar diferentes técnicas de validación y verificación de propiedades descriptas en ese lenguaje y por lo tanto, es necesario brindar fundamentos formales a un concepto de análisis que permita la interrelación, estructurada formalmente,

de estas técnicas. Con Heterogenius [4] exploramos la construcción de esta herramienta cuya racionalización formal tiene origen en Argentum.

Heterogenious es hoy una herramienta que integra gran variedad de lenguajes a través de traducciones que preservan la semántica de las especificaciones, y a través de diversas herramientas que permiten el análisis de propiedades en dichos lenguajes. Actualmente, una enorme variedad de herramientas, tanto demostradores semiautomáticos como bounded model-checkers automáticos, han elegido TPTP [1] como lenguaje nativo para la realización del análisis lo que habilita el uso de una enorme variedad de métodos automáticos y semiautomáticos de análisis para especificaciones que pueden ser escritas en este lenguaje u otro para el cual existe un co-morfismo a este.

En esta tesis investigaremos nuevas metodologías tendientes a sacar provecho de la madurez de los sistemas actuales de demostración automática de teoremas de primer orden descritos en lenguaje TPTP, con el objetivo de construir herramientas que sirvan de ayuda en los procesos de análisis formal de software crítico. Estas metodologías se plasmarán en mejoras concretas al sistema Heterogenius, aumentando su automaticidad en los casos que sea posible.

1.2. Objetivos Especificos

En ésta tesis busca mejorar el análisis heterogeneo, mediante la extensión del árbol de análisis y la incorporación de nuevas operaciones; experimentar con nuevas tecnologías, especialmente las herramientas automáticas que trabajan con lógica de primer orden y ampliar la funcionalidad de Heterogenius.

1.2.1. Extensión del árbol de análisis

Con el objetivo de permitir documentar todo el proceso de demostración (caminos alternativos tomados, decisiones que no produjeron ningún resultado exitoso, etc), se decidió ampliar el concepto del árbol de analisis. Para ésto se incluyó un nuevo tipo de ramificación, que permite soportar ramas de demostraciones alternativas y ramas de decisiones no exitosas.

Además se introdujeron reglas de cálculo de secuentes nuevas para permitir un mejor control de la aplicación de herramientas automáticas externas.

1.2.2. Extensión de Heterogenius

El lenguaje de lógica de primer orden *TPTP-FOF*, al ser muy difundido en la comunidad de investigadores de demostradores automáticos de teoremas es soportado por numerosas herramientas. Entre ellas EProver y SPASS, demostradores automáticos de teoremas; EProver y Mace4, buscadores de modelos.

Para permitir el uso de todas éstas herramientas y otras, se decidió integrar *TPTP-FOF* con Heterogenius mediante la implementación de una $\rho-$translation desde el lenguaje *PDOCFA*.

1.2.3. Heterogeneidad Verdadera

Heterogenius en su primera versión permitió realizar demostraciones heterogeneas mediante traducciones de secuentes. Cada secuente tenía fórmulas de un mismo lenguaje

haciendo que los mismos sean en realidad homogeneos.

Se decidió ampliar el concepto de heterogeneidad expandiendolo también a los secuentes. De esta forma se permitió tener mayor flexibilidad en las demostraciones al poder soportar fórmulas de distintos lenguajes en el mismo secuente.

2. PRELIMINARES

2.1. Cálculo de Secuentes

2.2. Demostraciones Heterogeneas

2.3. Heterogenius

Heterogenius [4] es un software que, como su principal objetivo, permite realizar demostraciones interactivas mediante el cálculo de secuentes, representando lo que se quiere demostrar en distintos lenguajes, aplicando reglas de cálculo de secuentes y usando las herramientas automáticas externas.

Su principal caracteristica es que soporta demostraciones heterogeneas, o sea las demostraciones pueden aceptar secuentes escritos en distintos lenguajes. Los lenguajes soportados actualmente son dos: *Alloy* y *PDOCFA*. Entre otras cosas Heterogenius presenta las siguientes caracteristicas:

- Pone en práctica la idea de *árbol de análisis* el cuál es el elemento principal donde se realiza el proceso de demostración.
- Separa conceptualmente y arquitectónicamente las herramientas utilizadas durante el análisis.
- Ofrece interacción entre varias herramientas externas.
- Posee una arquitectura modular y extensible.
- Posee una interfaz de usuario moderna e intuitiva.

Además de ser un demostrador interactivo, Heterogenius también es una plataforma en la cuál se puede experimentar con lenguajes y herramientas automáticas nuevas.

2.4. TPTP-FOF

TPTP (mil problemas para demostradores automáticos, por sus siglas en ingles) [1] es una biblioteca de problemas para demostradores automáticos de teoremas. La principal motivación para *TPTP* es permitir el testeo y evaluación de diferentes sistemas de demostradores automáticos de teoremas. Los problemas están en cuatro lenguajes: *THF*, *TFF*, *FOF* y *CNF*.

Anualmente se realiza la competencia *CASC* (CADE ATP System Competition [2]), una competencia de demostradores automáticos de teoremas donde los sistemas participanetes compiten para probar la mayor cantidad de problemas de *TPTP*. Ésta competencia resulta una muy buena prueba para evaluar el funcionamiento de las herramientas automáticas disponibles.

FOF [3] es un lenguaje de lógica de primer órden con igualdad. Su elección de *FOF* está motivada por su difundido uso y soporte que tiene de las herramientas automáticas y su expresividad a nivel de sintaxis. Una especificación de *TPTP-FOF* es una lista de fórmulas que además tienen un nombre y un tipo. Por ejemplo:

```
fof(john,axiom,(
    human(john) )).

fof(all_created_equal,axiom,(
    ! [H1,H2] :
      ( ( human(H1)
         & human(H2) )
     => created_equal(H1,H2) ) )).

fof(john_failed,axiom,(
    grade(john) = f )).

fof(someone_got_an_a,axiom,(
    ? [H] :
      ( human(H)
      & grade(H) = a ) )).

fof(distinct_grades,axiom,(
    a != f )).

fof(grades_not_human,axiom,(
    ! [G] : ~ human(grade(G)) )).

fof(someone_not_john,conjecture,(
    ? [H] :
      ( human(H)
      & H != john ) )).
```

Especificación típica en el lenguaje *TPTP-FOF*. La sintaxis contiene todos los elementos de la lógica de primer órden. **![X]** es el cuantificador universal sobre la variable X; **?[X]** es el cuantificador existencial sobre la variable X. $human()$, $grade()$, $created\_equal()$ son predicados; las variables empiezan con una letra mayuscula y las constantes con minuscula. Los operadores disponibles son $\sim$, |, &, =>, <=>, = y ! = y corresponden con la negación, la disyunción, la conjunción, la implicación, la doble implicación, la igualdad y la desigualdad.

En el ejemplo anterior la última fórmula es de tipo *conjecture*. Ésto indica al demostrador que es la fórmula que se quiere probar. Las otras fórmulas, de tipo *axiom* se interpretan como axiomas.

3. APORTES

3.1. Heterogenius y el Árbol de Análisis

El árbol de análisis de Heterogenius es el elemento principal de un proceso de demostración. Es donde se realizan todas las acciones y es donde se refleja el camino tomado para lograr una demostración exitosa. Cada nodo del árbol representa un secuente en algún lenguaje soportado por Heterogenius. Las aristas corresponden con las acciones ejecutadas. Dependiendo del lenguaje en el que esté el secuente se habilitan diferentes acciones, pero en general se las puede dividir en tres categorías:

- las de cálculo de secuentes, son acciones que transforman un secuente en otro. Algunas pueden producir multiples secuentes (por ejemplo la acción *Case*) creando varias ramas que tienen que ser demostradas para lograr un resultado en el nodo raiz.
- las acciones de traducción, traducen un secuente de un lenguaje a otro. Dependiendo de la expresividad del lenguaje esta traducción puede ser con perdida o no.
- las acciones de búsqueda de contraejemplos, implican el uso de herramientas externas para buscar contraejemplos en el secuente donde se aplique la acción. El resultado puede ser positivo, se encontró un contraejemplo o negativo, no se encontró nada pero ésto no nos dice nada de la existencia del contraejemplo.

3.1.1. Limitaciones

En su versión actual, Heterogenius presenta varios problemas y limitaciones. A continuación se detallarán algunas de éstas limitaciones que fueron tratadas en ésta tesis:

Lo primero que se puede notar es la falta de *heterogeneidad verdadera*. Si bien Heterogenius permite realizar una demostración en diferentes lenguajes, cada secuente se limita a tener un solo lenguaje. Debido a ésto se puede decir que Heterogenius, en realidad permite tener demostraciones heterogeneas con secuentes homogeneos.

Otro gran problema con el que nos encontramos es la incapacidad del árbol de análisis de documentar el proceso de análisis completo. O sea, el árbol solamente muestra las acciones y pasos que llevan al éxito de una demostración. En el momento del análisis cuando una rama no lleva al resultado deseado y se quiere probar otro tipo de acciones, es necesario borrar la rama que no dió resultado. Ésto lleva a que se pierdan partes del análisis que pueden ser útiles ya que documentan las acciones probadas (y que no dieron un resultado deseado) y no nos permite tener un historial de todo lo que se hizo en el análisis.

El árbol de análisis tiene solamente un tipo de ramas, que representan acciones obligatorias que se tienen que llevar a cabo para que el resultado se propague al nodo raiz. Ésto limita la posibilidad de realizar demostraciones alternativas y experimentar en una demostración con diferentes tipos de herramientas, acciones, etc.

Debido a que las herramientas automáticas no siempre producen un resultado, decidimos que es necesario también documentar la aplicación de las acciones aún si no tienen un resultado definido. Todo ésto aporta al objetivo general de documentar todas las acciones realizadas durante el análisis.

3.2. Rediseño del Árbol de Análisis

Para solucionar las limitaciones presentadas nos propusimos a rediseñar el árbol de análisis.

El primer cambio que realizamos es la implementación de *heterogeneidad verdadera*. Para lograr ésto flexibilizamos los secuentes permitiendo que también sean heterogeneos. Ésto nos permite manejar las demostraciones con mayor flexibilidad y abstraernos aún más del lenguaje usado para describir el problema. La implementación y fundamentos teóricos se explican con mayor detalle en la sección 3.3 .

En el árbol de análisis éste cambio se va a reflejar en la visualización de los nodos. Los nodos con secuentes heterogeneos van a estar marcados por una *H* y los nodos homogeneos quedarán sin ninguna marca.

Fig. 3.1: El nodo con H contiene un secuente heterogéneo. El otro nodo es homogéneo.

Para permitir la documentación y el historial de todas las acciones aplicadas asi como la creación de caminos de análisis alternativos introdujimos el concepto de *ramificación alternativa*:

3.2.1. Ramas alternativas

Las ramas alternativas representan la existencia de multiples caminos en los que se subdivide el análisis para lograr un resultado.. En la interface de Heterogenius éste tipo de ramas, se representa con lineas punteadas y su significado semántico es el de un operador lógico **"o"**. Se corresponde con un camino alternativo en una demostración.

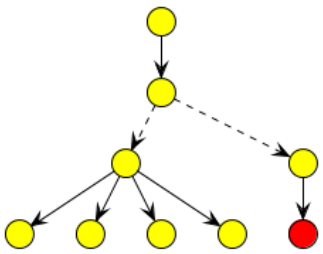


Fig. 3.2: La segunda rama alternativa presenta un contraejemplo. Ésto indica que existe un contraejemplo para el nodo del cual salen las ramas alternativas.

Un nodo con hijos conectados por las ramas alternativas, se entiende que vale si ***alguna*** de las ramas valen. Ésto es diferente de la ramificación normal (lineas continuas) que indica que el nodo padre vale si todos sus hijos valen.

Al aplicar una ramificación alternativa a un nodo del árbol de análisis, el nodo es copiado y agregado como sus propios hijos. Ésto nos permite trabajar sobre las copias del nodo original. Cuando es necesario también se puede agregar ramas alternativas durante un análisis.

La principal ventaja de usar caminos alternativos es la de poder documentar todo el análisis que se hizo y las decisiones tomadas, incluso las decisiones que no llevaron al cumplimiento del objetivo. Por otro lado también nos permite experimentar con diferentes formas de probar lo mismo.

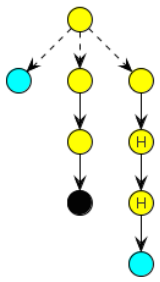


Fig. 3.3: Tres ramas alternativas: la primera y la última indican que no se encontró ningún contraejemplo. La segunda rama muestra que se pudo demostrar que el secuente vale, por lo cuál el secuente del nodo raiz también vale.

3.2.2. Nueva Clasificación de Acciones

Debido a que la clasificación original de las acciones (3.1) es poco general, propusimos cambiar las categorias para permitir una mayor flexibilidad a la hora de agregar herramientas automáticas y acciones nuevas.

La nueva clasificación propuesta es:

- Acciones de cálculo de secuentes: son todas las acciones que toman un secuente como su entrada y devuelven uno o mas secuentes. Algunas de éstas acciones son: *Use*, *Lemma*, *Case*, *Skolem*, etc.
- Acciones de herramientas estructurales: son acciones que trabajan directamente sobre la estructura del árbol de análisis. Los traductores-ρ forman parte de éste grupo, asi como las nuevas acciones introducidas: *carga de antecedentes externos*, *traducción-ρ de fórmulas* y *proyección de fórmulas*.
- Acciones de herramientas automaticas: éste grupo representa a las acciones para las que se usan herramientas automáticas como lo son los demostradores automáticos de teoremas y los buscadores de contraejemplos.

Ésta clasificación se refleja en la arquitectura de Heterogenius y permite guiar las futuras extensiones y funcionalidades adicionales.

3.3. Heterogeneidad Verdadera

Para solucionar otra de las limitaciones presentes en Heterogenius extendimos el concepto de heterogeneidad para lograr tener demostraciones verdaderamente heterogeneas en lugar de demostraciones homogeneas en un árbol de análisis heterogeneo.

La diferencia principal radica en que con la nueva implementación, los secuentes pueden soportar fórmulas de diferentes lenguajes. Así un secuente puede ser de tipo homogeneo o heterogeneo. En el primer caso todas las fórmulas del secuente usan el mismo lenguaje; en el segundo las fórmulas son de lenguajes distintos.

La ventaja de los secuentes heterogeneos es que se puede combinar fórmulas (lemmas, propiedades, teoremas) provenientes de distintas especificaciones escritas en lenguajes diferentes. De éste fórma nos podemos abstraer del lenguaje en el que están escritas las fórmulas y concentrarnos en el análisis.

La principal limitación de los secuentes heterogeneos es que las herramientas (calculadores de secuentes, buscadores de contraejemplos, demostradores automáticos) trabajan con secuentes escritos en un solo lenguaje, o sea secuentes homogeneos. Debido a ésto se proveen nuevas operaciones para el manejo de fórmulas dentro de un secuente:

3.3.1. Operaciones para el manejo de fórmulas

Cada una de las siguientes operaciones puede cambiar o no la heterogeneidad de un secuente. Dependiendo de los lenguajes de las fórmulas del resultado, el secuente puede pasar a ser heterogeneo, homogeneo o mantener su tipo.

Proyección

Dado un secuente, se selecciona un subconjunto de las fórmulas que se quiere proyectar y el nuevo secuente se forma a partir de las fórmulas seleccionadas.

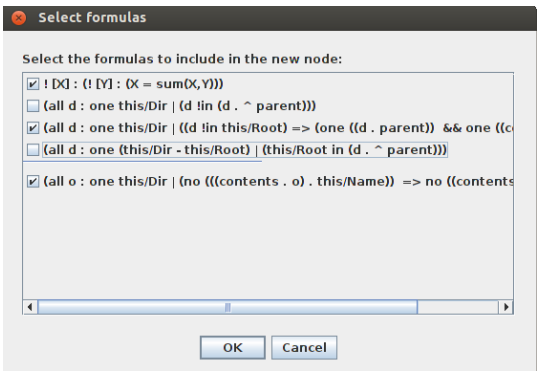


Fig. 3.4: Se marcan las fórmulas que se quiere proyectar al nuevo secuente.

Dado un secuente:

$$\frac{\alpha_1,\ldots,\alpha_n}{\alpha_{n+1},\ldots,\alpha_m}$$

y un subconjunto $\mathcal{C} \subseteq \{1 \ldots m\}$, el secuente resultante:

$$\frac{\alpha_i \text{ con } i = 1 \dots n \text{ y } i \in \mathcal{C}}{\alpha_j \text{ con } j = n+1 \dots m \text{ y } j \in \mathcal{C}}$$

Introducción de antecedentes desde una fuente externa

Ésta operación permite cargar desde un archivo de especificación, ya sea *Alloy* o *FOF* axiomas e introducirlas como antecedentes del secuente analizado.

Dado un secuente:

$$\frac{\alpha_1,\dots,\alpha_n}{\alpha_{n+1},\dots,\alpha_m}$$

y un conjunto de fórmulas nuevas $\{\beta_1 \dots \beta_k\}$. El nuevo secuente es:

$$\frac{\alpha_1,\dots,\alpha_n,\ \beta_1,\dots,\beta_k}{\alpha_{n+1},\dots,\alpha_m}$$

Traducción

Se extendió el concepto de traducciones ρ para que se puedan traducir fórmulas por separado. El secuente resultante contendrá las fórmulas del secuente analizado en el lenguaje seleccionado.

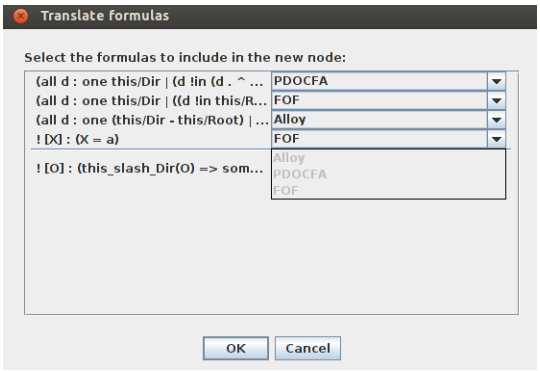


Fig. 3.5: Para cada fórmula se puede seleccionar el lenguaje al que se quiere traducir.

Dado un secuente S

$$\frac{\alpha_1,\dots,\alpha_n}{\alpha_{n+1},\dots,\alpha_m}$$

y una relacion $\mathcal{T} : Formula \times Lenguaje$ que indica el lenguaje seleccionado para cada fórmula del secuente S, el secuente resultante S' es:

$$\frac{\beta_1,\dots,\beta_n}{\beta_{n+1},\dots,\beta_m}$$

donde $\beta_i = \rho(\alpha_i, \mathcal{T}(\alpha_i))$.

con $\rho(\alpha, l) : Formula \times Lenguaje \to Formula$: función de traducción para la fórmula α al lenguaje l.

3.4. Extensión de Heterogenius con Herramientas de Lógica de Primer Orden

Existen numerosas herramientas que funcionan con el lenguaje de lógica de primer orden *TPTP-FOF*. Para permitir la integración de estas herramientas con Heterogenius y abrir el camino para la interacción con las futuras tecnologias basadas en éste lenguaje, se agregó *TPTP-FOF* al motor de Heterogenius como un lenguaje de análisis.

Junto a la integración de *TPTP-FOF*, se incorporaron los siguientes mecanismos para poder usar las herramientas correspondientes:

- Se permitió la carga de especificaciones escritas puramente en *TPTP-FOF*.
- Se agregó una $\rho-$translation desde las formulas *PDOCFA* a *TPTP-FOF*. [TODO: referenciar la seccion que explica esto en detalle].

Teniendo el soporte de *TPTP-FOF* por parte del motor de cálculo de secuentes de Heterogenius, integramos algunas de las herramientas mas difundidas en el ámbito de demostradores automáticos de teoremas para el lenguaje *TPTP-FOF*: *E-Prover* y *SPASS* como calculadores de secuentes; *E-Prover* y *Mace4* como buscadores de contraejemplos.

3.4.1. Herramientas Usadas

E-Prover

Es un demostrador automático de teoremas de lógica de primer orden basado en el calculo por superposición. Además realiza búsquedas de modelos por lo cual también se usa como un buscador de contraejemplos.

SPASS

Es un demostrador automático de teoremas de lógica de primer orden con igualdad desarrollado por el Instituto Max Planck.

Desde el 2000, tanto *SPASS* como *E-Prover* ocupan los primeros lugares en la competencia anual de demostradores de teoremas *CASC* (CADE ATP System Competition).

Mace4

Es un buscador de modelos finitos y contraejemplos para lógica de primer orden. Se seleccionó debido a su notoriedad en el ámbito de herramientas automáticas de lógica de primer orden.

3.4.2. Cálculo de secuentes

Al introducir el soporte para lenguajes de primer órden y herramientas automáticas fue necesario extender el cálculo de secuentes agregando algunas reglas nuevas.

Sea $\Gamma \vdash \alpha$ el secuente que se quiere analizar, se introducen las siguientes reglas:

Regla 1: $\dfrac{\Gamma \vdash \alpha}{\top}$ (si vale $\Gamma \vdash^{fof} \alpha$)

Ésta regla indica que si se logra encontrar una demostración del secuente $\Gamma \vdash \alpha$ con un demostrador automático *TPTP-FOF*, el resultado es $\top$ y se termina la demostración.

$$\textbf{Regla 2: } \frac{\Gamma \vdash \alpha}{\bot} \text{ (si existe } \mathcal{M} \in Mod^{fof}(\Gamma) \text{ y } \mathcal{M} \not\models^{fof} \alpha \text{)}$$

Si de lo contrario, se logra encontrar un modelo de Γ que no satisface α, éste modelo es un contraejemplo y el resultado del análisis es $\bot$.

$$\textbf{Regla 3: } \frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha} \text{ (si no)}$$

Por último, debido a que la lógica de primer orden no es completa y la ejecución de las herramientas automáticas está limitada por un timeout, un caso posible es el de no encontrar ni una demostración ni un contraejemplo. En éste caso no se sabe nada y el resultado es el mismo secuente.

Al usar algún demostrador automático de teoremas se aplican las reglas 1 y 3. En cambio cuando se realiza una búsqueda de contraejemplo, las reglas usadas son 2 y 3.

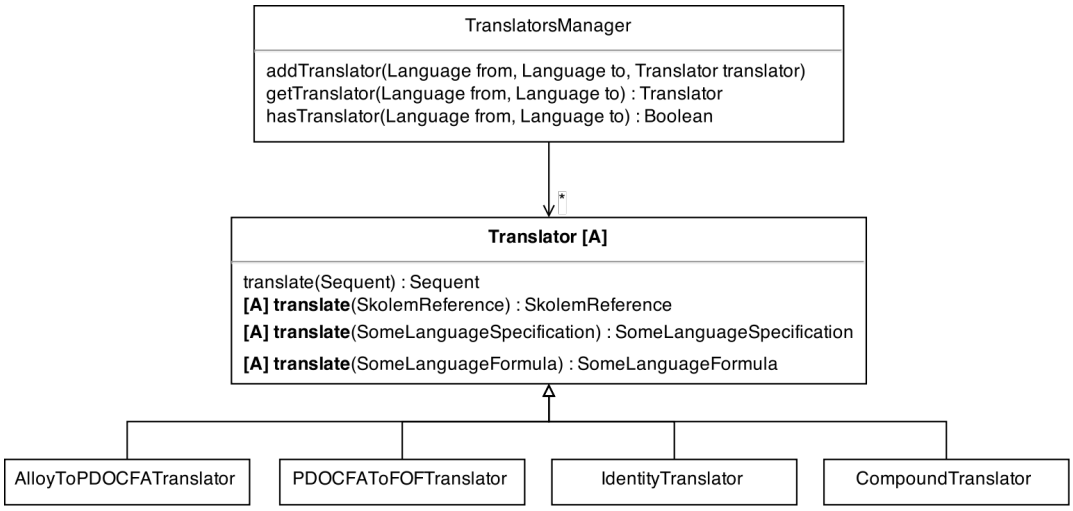
TODO: explicar la traduccion PDOCFA -¿ FOF

3.5. Detalles de Implementación

3.5.1. Extensión de las traducciones Rho

Debido a los cambios introducidos fue necesario refactorizar el diseño de la infraestructura de las traducciones ρ. Lo primero que se hizo fue agregar un *TranslationsManager*, un objeto encargado de manejar todas las traducciones soportadas por el sistema.

Por otro lado los traductores (subclases de *Translator*) deben implementar los tres métodos abstractos definidos en la clase padre. Cada uno de éstos métodos permite un control más fino de las traducciones al separar el secuente en sus partes, que son: una referencia de skolemizacíon, una especificación y la fórmula analizada.



Se proveen los traductores de *Alloy* a *PDOCFA*, de *PDOCFA* a *TPTP-FOF* asi como el *CompoundTranslator* que permite componer los traductores para lograr traducciones transitivas, por ejemplo de *Alloy* a *TPTP-FOF*.

3.5.2. Demostradores de teoremas

Preparación del secuente

El formato *TPTP-FOF* que usan las herramientas automáticas agregadas presenta un archivo de texto con una lista de fórmulas escritas en el lenguaje *TPTP-FOF*. Cada una de éstas fórmulas debe tener un tipo que puede ser: *axiom* o *conjecture* (existen mas tipos pero son irrelevantes en éste caso).

Una fórmula de tipo *axiom* se considera verdadera y se la toma como parte de la especificación. Las fórmulas de tipo *conjecture* son las que el demostrador automático va a tratar de probar.

Para convertir el secuente analizado al formato *TPTP-FOF* se agregan todas las fórmulas del antecedente con el tipo *axiom* y las fórmulas del consecuente con el tipo *conjecture*.

Al ejecutar un demostrador automático con la entrada preparada de éste modo, se va a tratar de probar las fórmulas del consecuente usando que las fórmulas del antecedente son verdaderas.

Integración con Heterogenius

Para integrar las herramientas automáticas, tanto los demostradores de teoremas como los buscadores de contraejemplos fue necesario refactorizar y extender el diseño de algunas partes de la arquitectura de Heterogenius. Algunos de los objetivos y criterios que se usaron durante el rediseño fueron lograr abstraer las herramientas usadas y hacer un diseño lo suficientemente extensible y abierto para la integración con nuevas herramientas en el futuro.

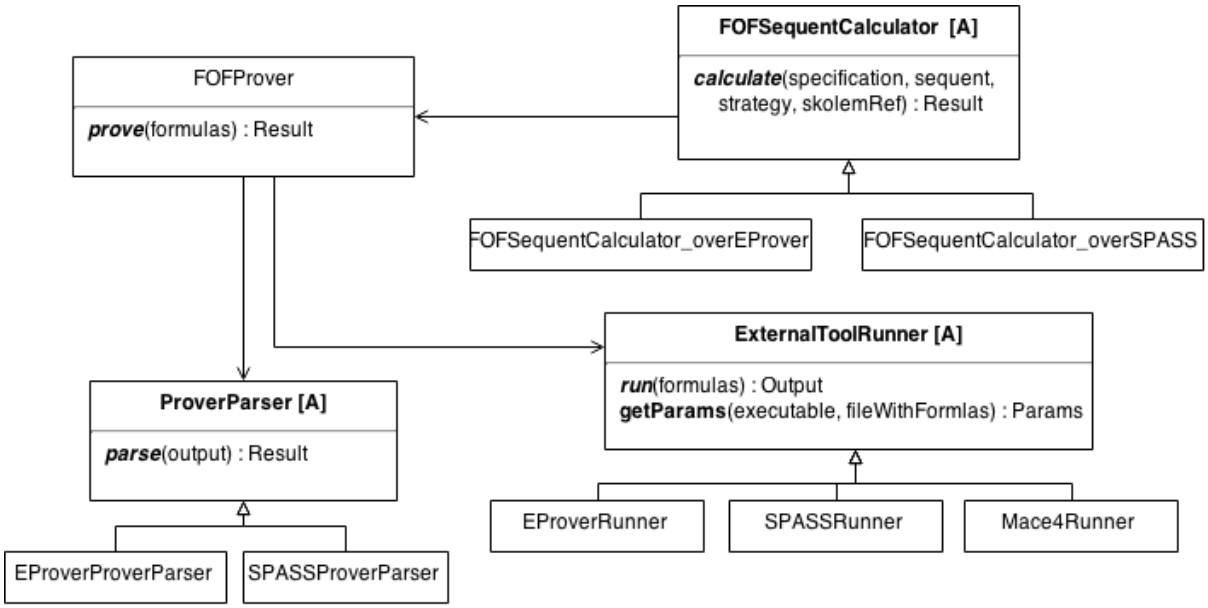


Fig. 3.6: Demostradores automáticos usados como calculadores de secuentes.

Un calculador de secuentes basado en un demostrador automático del lenguaje *TPTP-FOF* se crea subclasificando la clase abstracta **FOFSequentCalculator**. La nueva clase va a contener un objeto *FOFProver* compuesto con los objetos de las subclases de **ProverParser** y de **ExternalToolRunner** correspondientes.

Para agregar una herramienta nueva primero se debe subclasificar la clase abstracta **ExternalToolRunner**. Ésta clase abstrae las particularidades de ejecución de la herramienta especificando los parametros necesarios. Luego se extiende la clase **ProverParser** que se encarga de procesar el texto correspondiente a la salida de la herramienta.

3.5.3. Búsqueda de contraejemplos

Preparación del secuente

Tanto *Mace4* como *E-Prover* se usan para buscar contraejemplos de los secuentes *TPTP-FOF*. Como las dos herramientas son buscadores de modelos, para preparar el secuente lo que se hace es armar una lista de fórmulas de tipo *axiom* a partir de las fórmulas del antecedente y del consecuente del secuente analizado.

Cuando todas las fórmulas que se envian como entrada a los buscadores de modelos son de tipo *axiom*. Las herramientas tratan de buscar un modelo para el conjunto de los axiomas recibidos.

Lo que se quiere lograr es encontrar un contraejemplo, entonces se hace necesario negar el secuente y buscar un modelo para la negación:

Sea $\{\alpha_1 \dots \alpha_n\} \Rightarrow \{\beta_1 \dots \beta_m\}$ el secuente bajo análisis. La negación se puede escribir como:

$$\bigwedge_{i=1}^{n} \alpha_i \wedge \bigwedge_{j=1}^{m} \neg\beta_j \tag{3.1}$$

Con lo cual para armar la lista de fórmulas de entrada, para cada α_i del antecedente se agrega α_i como axioma y para cada β_i del consecuente se agrega $\neg\beta_i$ también como axioma.

Encontrar un modelo para una especificación armada de éste modo implica la existencia de un contraejemplo para el secuente procesado.

Integración con Heterogenius

También como en el caso de los demostradores automáticos fue necesario refactorizar el diseño de los buscadores de contraejemplos para permitir una mayor flexibilidad a la hora de agregar nuevas herramientas con soporte de *TPTP-FOF*.

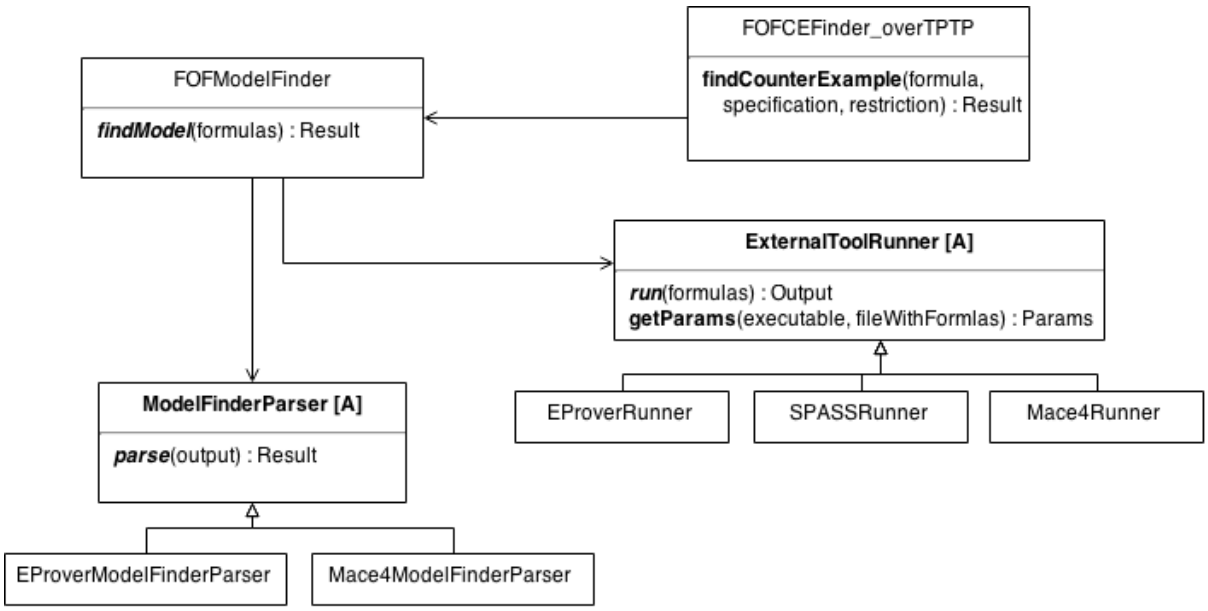


Fig. 3.7: Buscadores de contraejemplos.

El diseño de los buscadores de contraejemplos basados en *TPTP-FOF* es análogo a los calculadores de secuentes. Se usa la misma clase **ExternalToolRunner** que en el caso anterior para la abstracción de la ejecución de la herramienta en sí. Pero en lugar de subclasificar **ProverParser** se subclasifica la clase abstracta **ModelFinderParser** que tiene un comportamiento análogo.

4. CASO DE ESTUDIO

5. CONCLUSIONES Y TRABAJOS FUTUROS

....

Bibliografía

[1] Sutcliffe, G. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning. Vol. 43. Num. 4. Pp. 337-362. 2009.

[2] The CADE ATP System Competition `http://www.cs.miami.edu/~tptp/CASC/`

[3] FOF: First Orden Formula `http://www.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html`

[4] Manuel Gimenez, Mariano Miguel Moscato, Carlos G. Lopez Pombo, and Marcelo F. Frias. Heterogenius: a framework for hybrid analysis of heterogeneous software specifications. In Aguirre and Ribeiro [25], pages 1045–1058. Workshop affiliated to [26]. 2013.

[5] McCune, W., "Prover9 and Mace4", http://www.cs.unm.edu/ mccune/Prover9, 2005-2010.

[6] Schulz, S. System Description: E 1.8, Proceedings of the 19th LPAR, Stellenbosch, 2013, pp. 477-483, LNCS 8312 © Springer Verlag.

[7] Weidenbach C., Dimova D., Fietzke A., Kumar R., Suda M. and Wischnewski P., 2009, SPASS Version 3.5. in 22nd International Conference on Automated Deduction, CADE 2009, LNCS 5663, pp. 140-145.

[8] Grady Booch, Jim Rumbaugh, and Ivar Jacobson. The unified modeling language user guide. Addison–Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[9] Amir Pnueli. The temporal logic of programs. In Proceedings of 18th. Annual IEEE Symposium on Foundations of Computer Science [10], pages 46–57.

[10] IEEE Computer Society. 18th. Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, CA, USA, 1977. IEEE Computer Society.

[11] Zohar Manna and Amir Pnueli. Temporal Verification of Reactive Systems. Springer-Verlag, New York, 1995.

[12] David Harel, Dexter Kozen, and J. Tiuryn. Dynamic logic. Foundations of Computing. MIT Press, Cambridge, MA, USA, 2000.

[13] Joseph A. Goguen and Rod M. Burstall. Introducing institutions. In Clarke and Kozen [14], pages 221–256.

[14] Edmund M. Clarke and Dexter Kozen, editors. Carnegie Mellon Workshop on Logic of Programs, volume 184 of Lecture Notes in Computer Science. Springer-Verlag, 1984.

[15] Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. Journal of the ACM, 39(1):95–146, 1992.

[16] Andrzej Tarlecki. Moving between logical systems. In Haveraaen et al. [17], pages 478–502.

[17] Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors. Selected papers from the 11th Workshop on Specification of Abstract Data Types, joint with the 8th COMPASS Workshop on Recent

[18] Razvan Diaconescu. Grothendieck institutions. Applied Categorical Structures, 10(4):383–402, 2002.

[19] Mogens Nielsen and Uffe Engberg, editors. International Con- ference on Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, London, UK, 2002. Springer-Verlag.

[20] Till Mossakowski. Heterogeneous development graphs and heterogeneous borrowing. In Nielsen and Engberg [19], pages 326–341.

[21] Razvan Diaconescu and Kokichi Futatsugi. Logical semantics for CafeOBJ. Technical Report JAIST-IS-RR-96-0024S, Japan Advanced Institute of Science and Technology, 1996.

[22] Razvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. Theoretical Computer Science, 285(2):289–318, 2002.

[23] Michael Johnson and Varmo Vene, editors. 11th. International Conference on Algebraic Methodology and Software Technology, AMAST 2006, volume 4019 of Lecture Notes in Computer Science, Kuressaare, Estonia, July, 5–8 2006. Springer-Verlag.

[24] Carlos G. Lopez Pombo and Marcelo F. Frias. Fork algebras as a sufficiently rich universal institution. In Johnson and Vene [23], pages 235–247.

[25] Nazareno M. Aguirre and Leila Ribeiro, editors. Latin American Workshop on Formal Methods 2013, August 2013. Workshop affiliated to [26].

[26] Pedro R. D'Argenio and Hernán Melgratti, editors. 24th International Conference on Concurrency Theory – CONCUR 2013, volume 8052 of Lecture Notes in Computer Science. Springer- Verlag, 2013.