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

PPA - Un asistente de demostración para lógica de primer orden con extracción de testigos usando la traducción de Friedman

Tesis de Licenciatura en Ciencias de la Computación

Manuel Panichelli

Director: Pablo Barenbaum

Buenos Aires, 2024

PPA - UN ASISTENTE DE DEMOSTRACIÓN PARA LÓGICA DE PRIMER ORDEN CON EXTRACCIÓN DE TESTIGOS USANDO LA TRADUCCIÓN DE FRIEDMAN

La princesa Leia, líder del movimiento rebelde que desea reinstaurar la República en la galaxia en los tiempos ominosos del Imperio, es capturada por las malévolas Fuerzas Imperiales, capitaneadas por el implacable Darth Vader. El intrépido Luke Skywalker, ayudado por Han Solo, capitán de la nave espacial "El Halcón Milenario", y los androides, R2D2 y C3PO, serán los encargados de luchar contra el enemigo y rescatar a la princesa para volver a instaurar la justicia en el seno de la Galaxia (aprox. 200 palabras).

Palabras claves: Guerra, Rebelión, Wookie, Jedi, Fuerza, Imperio (no menos de 5).

PPA - A PROOF-ASSISTANT FOR FIRST-ORDER LOGIC WITH WITNESS EXTRACTION USING FRIEDMAN'S TRANSLATION

In a galaxy far, far away, a psychopathic emperor and his most trusted servant – a former Jedi Knight known as Darth Vader – are ruling a universe with fear. They have built a horrifying weapon known as the Death Star, a giant battle station capable of annihilating a world in less than a second. When the Death Star's master plans are captured by the fledgling Rebel Alliance, Vader starts a pursuit of the ship carrying them. A young dissident Senator, Leia Organa, is aboard the ship & puts the plans into a maintenance robot named R2-D2. Although she is captured, the Death Star plans cannot be found, as R2 & his companion, a tall robot named C-3PO, have escaped to the desert world of Tatooine below. Through a series of mishaps, the robots end up in the hands of a farm boy named Luke Skywalker, who lives with his Uncle Owen & Aunt Beru. Owen & Beru are viciously murdered by the Empire's stormtroopers who are trying to recover the plans, and Luke & the robots meet with former Jedi Knight Obi-Wan Kenobi to try to return the plans to Leia Organa's home, Alderaan. After contracting a pilot named Han Solo & his Wookiee companion Chewbacca, they escape an Imperial blockade. But when they reach Alderaan's coordinates, they find it destroyed - by the Death Star. They soon find themselves caught in a tractor beam & pulled into the Death Star. Although they rescue Leia Organa from the Death Star after a series of narrow escapes, Kenobi becomes one with the Force after being killed by his former pupil - Darth Vader. They reach the Alliance's base on Yavin's fourth moon, but the Imperials are in hot pursuit with the Death Star, and plan to annihilate the Rebel base. The Rebels must quickly find a way to eliminate the Death Star before it destroys them as it did Alderaan (aprox. 200 palabras).

Keywords: War, Rebellion, Wookie, Jedi, The Force, Empire (no menos de 5).

AGRADECIMIENTOS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce sapien ipsum, aliquet eget convallis at, adipiscing non odio. Donec porttitor tincidunt cursus. In tellus dui, varius sed scelerisque faucibus, sagittis non magna. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Mauris et luctus justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris sit amet purus massa, sed sodales justo. Mauris id mi sed orci porttitor dictum. Donec vitae mi non leo consectetur tempus vel et sapien. Curabitur enim quam, sollicitudin id iaculis id, congue euismod diam. Sed in eros nec urna lacinia porttitor ut vitae nulla. Ut mattis, erat et laoreet feugiat, lacus urna hendrerit nisi, at tincidunt dui justo at felis. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Ut iaculis euismod magna et consequat. Mauris eu augue in ipsum elementum dictum. Sed accumsan, velit vel vehicula dignissim, nibh tellus consequat metus, vel fringilla neque dolor in dolor. Aliquam ac justo ut lectus iaculis pharetra vitae sed turpis. Aliquam pulvinar lorem vel ipsum auctor et hendrerit nisl molestie. Donec id felis nec ante placerat vehicula. Sed lacus risus, aliquet vel facilisis eu, placerat vitae augue.

Índice general

1. INTRODUCCIÓN

Los asistentes de demostración son herramientas que facilitan la escritura y chequeo de demostraciones en una computadora. Pueden ser usados para formalizar teoremas, realizar verificación formal de programas, entre otros. A diferencia de escribir demostraciones y chequearlas manualmente, facilitan la colaboración a gran escala: no es necesario confiar ni revisar a detalle las demostraciones que hace el resto del equipo, alcanza con que el asistente las considere válidas. También permiten generar al menos partes de demostraciones mediante LLMs, apoyándose en el asistente para que las chequee por nosotros.

Trabajan con distintas *teorías*. Por ejemplo, el asistente Mizar con lógica de primer orden, Coq con teoría de tipos (cálculo de construcciones o CoC) y Agda también (teoría unificada de tipos dependientes, basada en teoría de tipos de Martin-Löf).

Una propiedad deseable cumplida por muchos asistentes es el **criterio de De Bruijn**. En principio, para estar seguros de que una demostración es correcta sería necesario confiar en la implementación del asistente, que puede ser muy compleja. Un asistente se dice que cumple con el criterio si construye una demostración en un formato elemental, sencillo, que pueda ser chequeada por un programa independiente, escrito por cualquiera que desconfíe de la implementación original del asistente. De esa forma no es necesario confiar en ella.

En este trabajo implementamos una asistente de demostración "PPA" (*Pani's proof assistant*) inspirado en Mizar. Trabaja sobre teorías de lógica clásica de primer orden. Cumple con el criterio de De Bruijn porque *certifica* las demostraciones en el lenguaje de alto nivel PPA, generando demostraciones de bajo nivel de deducción natural: un sistema lógico que permite escribir demostraciones mediante reglas de inferencia. Además, permite extraer testigos de existenciales. Su sintaxis está inspirada en el *mathematical vernacular* introducido por Freek Wiedijk en [Wie], que tiene por objetivo ser lo más parecido posible al lenguaje natural. No es un demostrador automático de teoremas, pero incluye un pequeño demostrador heurístico para lógica de primer orden y completo para proposicional, que simplifica la escritura de demostraciones.

Para la extracción de testigos, las demostraciones deben ser **constructivas**: una demostración de $\exists x.p(x)$ nos debe decir cómo encontrar a un objeto t que cumpla con $p(t)$. PPA usa lógica clásica, que no siempre es constructiva por los principios de razonamiento clásicos equivalentes como LEM. Siempre vale $A \vee \neg A$, y podemos demostrar usando eso sin determinar exactamente cual de las dos vale. Los asistentes como Coq lo aseguran mediante el uso de **lógica intuicionista**, que siempre es constructiva.

En este trabajo, la extracción se hace en dos pasos. Primero hacemos uso de la **traducción de Friedman** [Miq11], que permite traducir una demostración clásica a una intuicionista para cierta clase de fórmulas: las Π^0_2, de la pinta $\forall x_1 \dots \forall x_n.\exists y.\varphi$. Luego, *normalizamos* la demostración intuicionista, y de su forma normal extraemos el testigo del existencial. No conocemos un asistente de demostración de lógica clásica que implemente la traducción de Friedman original para extraer testigos. Este es el aporte principal de este trabajo.

1.1. Lógica de primer orden

A continuación se presentan definiciones preliminares de lógica de primer orden. Suponemos dados,

- Un conjunto infinito numerable de **variables** $\{x, y, z, \dots\}$
- Un conjunto infinito numerable de **símbolos de función** $\{f, g, h, \dots\}$

- Un conjunto infinito numerable de **símbolos de predicado** $\{p, q, r, \dots\}$

Def. 1 (Términos)**.** Los términos están dados por la gramática

$$\begin{aligned} t ::= x & \qquad \text{(variables)} \\ \mid f(t_1, \dots, t_n) & \qquad \text{(funciones)} \end{aligned}$$

Def. 2 (Fórmulas)**.** Las fórmulas están dadas por la gramática

$$\begin{aligned} A, B ::= p(t_1, \dots, t_n) & \qquad \text{(predicados)} \\ \mid \bot \mid \top & \qquad \text{(verdadero y falso)} \\ \mid A \wedge B & \qquad \text{(conjunción)} \\ \mid A \vee B & \qquad \text{(disyunción)} \\ \mid A \to B & \qquad \text{(implicación)} \\ \mid \neg A & \qquad \text{(negación)} \\ \mid \forall x.A & \qquad \text{(cuantificador universal)} \\ \mid \exists x.A & \qquad \text{(cuantificador existencial)} \end{aligned}$$

Los predicados son **fórmulas atómicas**. Los de aridad 0 además son llamados *variables proposicionales.*

Notación. Usamos

- $a, b, c, \dots$, $A, B, C, \dots$ y $\varphi, \psi, \dots$ para referirnos a fórmulas.
- $t, u, \dots$ para referirnos a términos

Def. 3 (Variables libres y ligadas)**.** Las variables pueden ocurrir libres o ligadas. Los cuantificadores ligan a las variables, y usamos $fv(()A)$ para referirnos a las variables libres de una fórmula. Se define por inducción estructural de la siguiente forma.

- Términos

$$\begin{aligned} fv(x) &= \{x\} \\ fv(f(t_1, \dots, t_n)) &= \bigcup_{i \in 1 \dots n} fv(t_i) \end{aligned}$$

- Fórmulas

$$\begin{aligned} fv(\bot) &= \emptyset \\ fv(\top) &= \emptyset \\ fv(p(t_1, \dots, t_n)) &= \bigcup_{i \in 1 \dots n} fv(t_i) \\ fv(A \wedge B) &= fv(A) \cup fv(B) \\ fv(A \vee B) &= fv(A) \cup fv(B) \\ fv(A \to B) &= fv(A) \cup fv(B) \\ fv(\neg A) &= fv(A) \\ fv(\forall x.A) &= fv(A) \setminus x \\ fv(\exists x.A) &= fv(A) \setminus x \end{aligned}$$

1.2. Arquitectura de ppa

A lo largo del trabajo, usaremos "PPA" para referirnos a dos cosas separadas: PPA el lenguaje para escribir demostraciones, y `ppa` la herramienta que implementa el lenguaje y con su extracción de testigos. Esta última está implementada en **Haskell**. Funcionalmente tiene la siguiente arquitectura, representada en la Figura 1.1 (Arquitectura funcional de ppa)

- El usuario escribe una demostración en alto nivel en el lenguaje PPA
- Puede *chequear* la demostración, que primero la **certifica** generando una demostración de deducción natural (el "certificado") que es chequeada por su correctitud.
- Luego, si es una demostración de un existencial, el usuario puede optar por *extraer un testigo*: primero se **traduce** la demostración de clásica a intuicionista, y luego se reduce hacia su formal normal de la cual se puede tomar el testigo.

Las funciones principales son las siguientes

- `certify :: Program -> Result Context`. Implementada por el módulo `PPA.Certifier`. Certifica un *programa* (axiomas y teoremas) generando un *contexto* (una demostración en deducción natural para cada teorema).
- `check :: Env -> Proof -> Form -> CheckResult`. Implementada por el módulo `ND.Checker`. Chequea una demostración de una fórmula asumiendo un entorno o contexto de demostración.
- `translateFriedman :: Proof -> FormPi02 -> (Proof, R)`. Implementada por el módulo `Extractor.Translator.Proof`. Traduce una demostración clásica de una fórmula Π^0_2 generando una demostración intuicionista, usando la traducción de Friedman relativizando la negación con `R`.
- `reduce :: Proof -> Proof`. Reduce una demostración a su forma normal. Implementada en el módulo `Extractor.Reducer`.

1.3. Estructura del escrito

El trabajo se divide en 5 capítulos principales además de la introducción y la conclusión. Comenzamos por el Capítulo 2 (Deducción natural) en el que se presenta de forma completa el sistema de deducción natural usado para los certificados. Se introducen las reglas de inferencia junto con sus intuiciones, el concepto de *reglas admisibles*, demostraciones de ejemplo y algunos algoritmos implementados (chequeo, alfa equivalencia de fórmulas y sustitución sin capturas).

En el Capítulo 3 (El lenguaje PPA) introducimos el lenguaje desde el punto de vista de un usuario: cómo están compuestos los programas, cómo usar el pequeño demostrador (**by**) para facilitar la escritura de demostraciones, y una descripción exhaustiva de todos los comandos soportados. En Capítulo 4 (El certificador de PPA) se muestra la implementación interna del *certificador* de PPA, cómo se generan demostraciones en deducción natural a

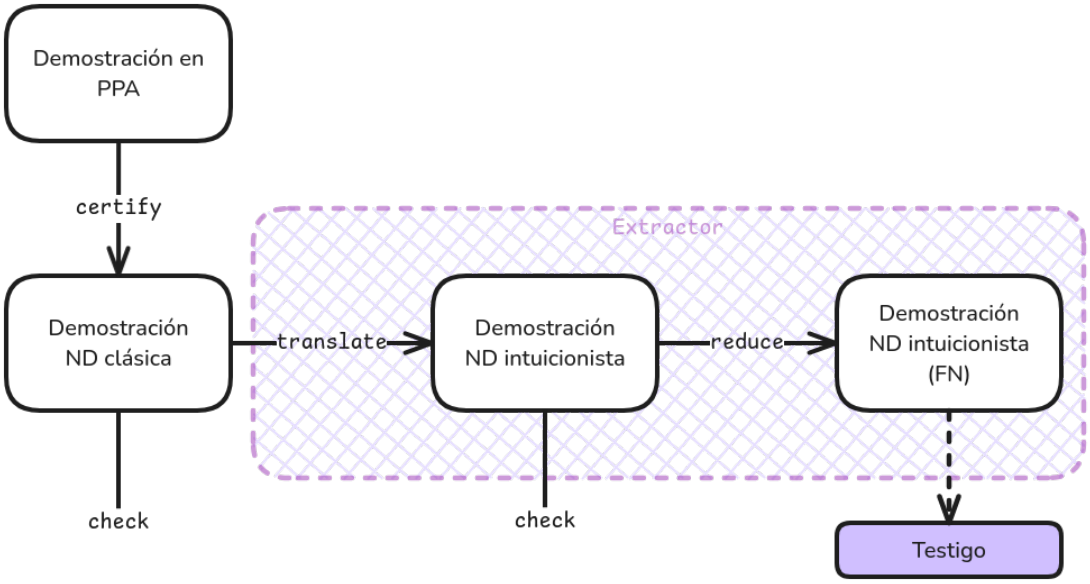

Fig. 1.1: Arquitectura funcional de `ppa`

partir de programas. De forma central se explica el funcionamiento del **`by`**, que es el corazón del certificador.

En el Capítulo 5 (Extracción de testigos de existenciales) se muestra el proceso completo de extracción de testigos, comenzando por la traducción de Friedman y luego la normalización de demostraciones intuicionistas.

Finalmente, en el Capítulo 6 (La herramienta `ppa`) se detallan los pasos para instalar y usar la herramienta `ppa`, y se mencionan algunos detalles de implementación como el compilador y el modelado de deducción natural.

2. DEDUCCIÓN NATURAL

3. EL LENGUAJE PPA

4. EL CERTIFICADOR DE PPA

5. EXTRACCIÓN DE TESTIGOS DE EXISTENCIALES

6. LA HERRAMIENTA `ppa`

7. CONCLUSIONES

BIBLIOGRAFíA

[Miq11] Alexandre Miquel. «Existential witness extraction in classical realizability and via a negative translation». En: *Log. Methods Comput. Sci.* 7.2 (2011). DOI: 10.2168/LMCS-7(2:2)2011. URL: https://doi.org/10.2168/LMCS-7(2:2)2011.

[Wie] Freek Wiedijk. *Mathematical Vernacular*. https://www.cs.ru.nl/~freek/notes/mv.pdf.