

PPA

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

Manuel Panichelli

Departamento de Computación, FCEyN, UBA

Diciembre 2024

Introducción

- Los **asistentes de demostración** son herramientas que facilitan la escritura y el chequeo de demostraciones por computadora.
- Usos usuales: formalización de teoremas matemáticos y verificación de programas.
- Ventajas:¹
 - Facilitan la colaboración a gran escala (mediante la confianza en el asistente).
 - Habilitan generación automática de demostraciones con IA. Por ej. un *LLM* (como *ChatGPT*) suele devolver alucinaciones, que pueden ser filtradas automáticamente con un asistente.

¹Terrence Tao - Machine Assisted Proof

Implementan distintas *teorías*. (TODO: No me gusta teoria, se usa para teorías de primer orden) Ejemplos:

- Mizar (lógica de primer orden)
- Coq (teoría de tipos)
- Agda (teoría de tipos)
- Isabelle (lógica de orden superior / teoría de conjuntos ZF)

- Diseñamos e implementamos **PPA** (*Pani's Proof Assistant*), un asistente de demostración para lógica **clásica** de primer orden.
- Permite hacer extracción de testigos: dada una demostración de $\exists x.p(x)$, encuentra t tal que $p(t)$.
- Aporte principal: implementación de extracción de testigos para lógica clásica de forma directa (vemos los detalles después).

Representación de demostraciones

Queremos escribir demostraciones en la computadora. ¿Cómo las representamos?. Veamos un ejemplo.

- Tenemos dos premisas
 - 1 Los alumnos que faltan a los exámenes, los reprueban.
 - 2 Si se reprueba un final, se recursa la materia.
- A partir de ellas, podríamos demostrar que si un alumno falta a un final, entonces recursa la materia.

Teorema

Si ((falta entonces reprueba) y (reprueba entonces recursa)) y falta, entonces recursa

Demostración

- Asumo que falta. Quiero ver que recursa.
- Sabemos que si falta, entonces reprueba. Por lo tanto reprobó.
- Sabemos que si reprueba, entonces recursa. Por lo tanto recursó. ☐

- La demostración anterior es poco precisa. No se puede representar rigurosamente.
- Necesitamos **sistemas deductivos**: sistemas lógicos formales usados para demostrar setencias. Pueden ser representados como un tipo abstracto de datos.
- Usamos **deducción natural**. Compuesto por,
 - **Lenguaje formal**: lógica de primer orden.
 - **Reglas de inferencia**: lista de reglas que se usan para probar teoremas a partir de axiomas y otros teoremas. Por ejemplo, *modus ponens* (si es cierto $A \rightarrow B$ y A , se puede concluir B) o *modus tollens* (si es cierto $A \rightarrow B$ y $\neg B$, se puede concluir $\neg A$)
 - **Axiomas**: fórmulas de L que se asumen válidas. Todos los teoremas se derivan de axiomas. Se usan para modelar *teorías* de primer orden (por ej. teoría de estudiantes en la facultad).

Definición (Términos)

Los términos están dados por la gramática:

$$\begin{array}{ll} t ::= x & \text{(variables)} \\ \quad | f(t_1, \dots, t_n) & \text{(funciones)} \end{array}$$

Definición (Fórmulas)

Las fórmulas están dadas por la gramática:

$$\begin{array}{ll} A, B ::= p(t_1, \dots, t_n) & \text{(predicados)} \\ \quad | \perp \mid \top & \text{(falso o } bottom \text{ y verdadero o } top) \\ \quad | A \wedge B \mid A \vee B & \text{(conjunción y disyunción)} \\ \quad | A \rightarrow B \mid \neg A & \text{(implicación y negación)} \\ \quad | \forall x.A \mid \exists x.A & \text{(cuantificador universal y existencial)} \end{array}$$

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

Notación

Usamos

- x, y, z, \dots como **variables**.
- f, g, h, \dots como **símbolos de función**.
- p, q, r, \dots como **símbolos de predicado**.
- t, u, \dots para referirnos a **términos**.
- $a, b, c, \dots, A, B, C, \dots$ y φ, ψ, \dots para referirnos a **fórmulas**.

Deducción natural

Definiciones

- Γ es un **contexto de demostración**, conjunto de fórmulas que se asumen válidas
- Notación: $\Gamma, \varphi = \Gamma \cup \{\varphi\}$
- \vdash es la **relación de derivabilidad** definida a partir de las *reglas de inferencia*. Permite escribir juicios $\Gamma \vdash \varphi$.
- Intuición: “ φ es una consecuencia de las suposiciones de Γ ”
- El juicio es cierto si en una cantidad finita de pasos podemos concluir φ a partir de las fórmulas de Γ , los axiomas y las reglas de inferencia.
- Decimos que φ es *derivable* a partir de Γ .

Definición (Reglas de inferencia)

$$\frac{}{\Gamma, A \vdash A} \text{Ax}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \text{I} \rightarrow$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \text{E} \rightarrow$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \text{I} \wedge$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \text{E} \wedge_1$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \text{E} \wedge_2$$

Dos tipos para cada conector y cuantificador, dada una fórmula formada con un conector:

- **Introducción:** ¿Cómo la demuestro?
- **Eliminación:** ¿Cómo la uso para demostrar otra?

Ejemplo

Vamos a demostrar el ejemplo informal en deducción natural. Lo modelamos para un alumno y materia particulares. Notamos:

- $X \equiv \text{reprueba}(\text{juan}, \text{final}(\text{logica}))$
- $R \equiv \text{recurso}(\text{juan}, \text{logica})$
- $F \equiv \text{falta}(\text{juan}, \text{final}(\text{logica}))$

Queremos probar entonces

$$\left((F \rightarrow X) \wedge (X \rightarrow R) \right) \rightarrow (F \rightarrow R)$$

Ejemplo

$$\begin{array}{c}
 \frac{}{\Gamma \vdash (F \rightarrow X) \wedge (X \rightarrow R)} \text{Ax} \\
 \hline
 \frac{}{\Gamma \vdash X \rightarrow R} \text{E}\wedge_1 \quad \frac{}{\Gamma \vdash X} \Pi \\
 \hline
 \frac{}{\Gamma = (F \rightarrow X) \wedge (X \rightarrow R), F \vdash R} \text{E}\rightarrow \\
 \hline
 \frac{}{(F \rightarrow X) \wedge (X \rightarrow R) \vdash F \rightarrow R} \text{I}\rightarrow \\
 \hline
 \vdash \left((F \rightarrow X) \wedge (X \rightarrow R) \right) \rightarrow (F \rightarrow R) \text{I}\rightarrow
 \end{array}$$

donde

$$\begin{array}{c}
 \frac{}{\Gamma \vdash (F \rightarrow X) \wedge (X \rightarrow R)} \text{Ax} \\
 \hline
 \frac{}{\Gamma \vdash F \rightarrow X} \Pi = \frac{}{\Gamma \vdash X} \text{E}\wedge_2 \quad \frac{}{\Gamma \vdash F} \text{Ax} \\
 \hline
 \text{E}\rightarrow
 \end{array}$$

Definición (Reglas de inferencia)

$$\frac{}{\Gamma \vdash A \vee \neg A} \text{LEM}$$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} E_{\perp}$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} I_{\neg}$$

$$\frac{}{\Gamma \vdash \top} I_{\top}$$

$$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} E_{\neg}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} I_{\vee_1}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} I_{\vee_2}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} E_{\vee}$$

Definición (Sustitución)

Notamos como $A\{x := t\}$ a la sustitución de todas las ocurrencias libres de la variable x por el término t en la fórmula A .

Definición (Reglas de cuantificadores)

$$\frac{\Gamma \vdash A \quad x \notin fv(\Gamma)}{\Gamma \vdash \forall x.A} \text{I}\forall$$

$$\frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A\{x := t\}} \text{E}\forall$$

$$\frac{\Gamma \vdash A\{x := t\}}{\Gamma \vdash \exists x.A} \text{I}\exists$$

$$\frac{\Gamma \vdash \exists x.A \quad \Gamma, A \vdash B \quad x \notin fv(\Gamma, B)}{\Gamma \vdash B} \text{E}\exists$$

Reglas admisibles

- Mencionamos *modus tollens* pero no aparece en las reglas de inferencia.
- Queremos un sistema lógico **minimal**: no agregamos como regla de inferencia lo que podemos derivar a partir de las existentes, las reglas **admisibles**.
- Se implementan como *macros*: cada uso de la regla admisible se reemplaza por su demostración.

Lema (Modus tollens)

$$\frac{\frac{\frac{\Gamma \vdash (A \rightarrow B) \wedge \neg B}{\Gamma \vdash \neg B} Ax \quad E\wedge_2 \quad \frac{\frac{\frac{\Gamma \vdash (A \rightarrow B) \wedge \neg B}{\Gamma \vdash A \rightarrow B} Ax \quad E\wedge_1 \quad \frac{\Gamma \vdash A}{\Gamma \vdash B} E\rightarrow}{\Gamma \vdash \perp} E\neg}{\Gamma = (A \rightarrow B) \wedge \neg B, A \vdash \perp} I\neg}{(A \rightarrow B) \wedge \neg B \vdash \neg A} I\rightarrow}{\vdash (A \rightarrow B \wedge \neg B) \rightarrow \neg A} I\rightarrow$$

Sustitución sin capturas

Para la sustitución $A\{x := t\}$ queremos evitar la **captura de variables**, por ejemplo

$$(\forall y.p(x))\{x := y\} \stackrel{?}{=} \forall y.p(\textcolor{red}{y})$$

sustituyendo sin más, capturamos a la variable x que ahora está ligada. Lo evitamos **automáticamente**: cuando se encuentra con una captura, se renombra la variable ligada de forma que no ocurra

$$(\forall y.p(x))\{x := y\} = \forall \textcolor{red}{z}.p(y)$$

Alfa equivalencia

- Si tenemos una hipótesis $\exists x.p(x)$ queremos poder usarla para demostrar $\exists y.p(y)$.
- No son iguales, pero son **α -equivalentes**: si renombramos variables ligadas de forma apropiada, son iguales.
- Algoritmo naíf: cuadrático en la estructura de la fórmula, renombrando recursivamente.
- Algoritmo cuasilineal: manteniendo dos sustituciones, una por fórmula.

Ejemplo

$$\begin{array}{ll} (\exists x.f(x)) \stackrel{\alpha}{=} (\exists y.f(y)) & \{\}, \{\} \\ \iff f(x) \stackrel{\alpha}{=} f(y) & \{x \mapsto z\}, \{y \mapsto z\} \\ \iff x \stackrel{\alpha}{=} y & \{x \mapsto z\}, \{y \mapsto z\} \\ \iff z = z. & \end{array}$$

PPA

Aparentemente hay una forma canónica de presentar demostraciones matemáticas². Descubierta e implementada independientemente en Mizar, Isar (Isabelle), etc. Combinación de ideas:

- **Deducción natural en estilo de *Fitch***. Notación equivalente en la cual las demostraciones son representadas como listas de fórmulas en lugar de árboles. Las que aparecen antes justifican las que aparecen después.
- **Reglas de inferencia “*big step*”**: una forma de afirmar que $A_1, \dots, A_n \vdash A$ es válida, sin tener que demostrarlo a mano.
- **Sintaxis similar a un lenguaje de programación** en lugar del lenguaje natural usado para demostraciones.

² *Mathematical Vernacular* de Freek Wiedijk

Diseñamos e implementamos el *lenguaje* PPA, inspirado en el *mathematical vernacular*. Ejemplo:

```
1  axiom falta_reprueba: forall A . forall E .  
2      falta(A, E) -> reprueba(A, E)  
3  axiom reprueba_recura: forall A . forall M .  
4      reprueba(A, final(M)) -> recursa(A, M)  
5  
6  theorem falta_entonces_recura: forall A . forall M .  
7      falta(A, final(M)) -> recursa(A, M)  
8  proof  
9      let A  
10     let M  
11     suppose falta: falta(A, final(M))  
12     have reprueba: reprueba(A, final(M)) by falta, falta_reprueba  
13     thus recursa(A, M) by reprueba, reprueba_recura  
14 end
```

Un **programa** de PPA consiste en una lista de **declaraciones**, que pueden ser

- **Axiomas**: fórmulas que se asumen válidas

```
axiom <name> : <form>
```

- **Teoremas**: fórmulas junto con sus demostraciones.

```
theorem <name> : <form>
```

```
proof
```

```
  <steps>
```

```
end
```

- **Variables** (<var>)

$(_ | [A-Z])[a-zA-Z0-9_ -]*(\ ')^*$

- **Identificadores** (<id>)

$[a-zA-Z0-9_ - \! ? ! \# \$ \% * \backslash \< \> \backslash = \backslash ? \backslash @ \backslash ^] + (\ ')^*$

- **Nombres** (<name>)

$\backslash "[^ \backslash "]^* \backslash "$

Fórmulas y términos

Términos:

- Variables: `<var>`
- Funciones: `<id>(<term>, ..., <term>)`

Funciones:

- Predicados: `<id>(<term>, ..., <term>)`
- `<form>` & `<form>`
- `<form>` | `<form>`
- `<form>` `->` `<form>`
- `<form>` `<->` `<form>`
- `~ <form>`
- **exists** `<var>` . `<form>`
- **forall** `<var>` . `<form>`
- **true, false**
- `(<form>)`

- Lista de comandos que reducen sucesivamente la *tesis* (fórmula a demostrar) hasta agotarla por completo.
- Corresponden aproximadamente a reglas de inferencia de deducción natural (vistas como una demostración en el estilo de Fitch).
- Tienen disponible un **contexto** con todas las hipótesis asumidas (como axiomas) o demostradas (teoremas y comandos que demuestran hipótesis auxiliares).

by - El mecanismo principal de demostración

- `<form> by <name>, ..., <name>` afirma que la fórmula es una consecuencia lógica de las fórmulas que corresponden a las hipótesis provistas.
- Por debajo usa un *solver* completo para lógica proposicional pero heurístico para primer orden.
- Se usa para eliminar implicaciones y universales.
- Usado por dos comandos principales: **thus** y **have**.

Thus

thus <form> **by** <name>, ..., <name>

Si <form> es *parte* de la tesis, y el *solver* puede demostrar la implicación, lo demuestra automáticamente y lo descarga de la tesis.

```
1  axiom ax1: a -> b
2  axiom ax2: b -> c
3
4  theorem t1: a -> c
5  proof
6    suppose a: a
7
8    // La tesis ahora es c
9    thus c by a, ax1, ax2
10 end
```

```
1  axiom ax: forall X . f(X)
2
3  theorem t: f(n)
4  proof
5    thus f(n) by ax
6  end
```

Certificador

- Las demostraciones de PPAse *certifican* generando una demostración de deducción natural.
- Cumple con el criterio de de bruijn

Extracción de testigos

- PPA (*Pani's proof assistant*) es un **asistente de demostraciones** inspirado en Mizar.
- Es un **lenguaje de programación** implementado en Haskell que permite escribir y chequear demostraciones en lógica *clásica* de primer orden.
- A diferencia de Prolog, **no demuestra todo automáticamente***. Deben ser escritas *rigurosamente* por el usuario.
- (WIP) permite la extracción de testigos mediante la **traducción de Friedman**.

Asistentes de demostraciones (*proof assistants*)

- Son programas que *asisten* al usuario a la hora de escribir demostraciones, permiten representarlas en un programa
- Aplicaciones: Formalización de teoremas, verificación formal de programas, etc.
- Ejemplos: Coq, Isabelle (Isar), **Mizar**, ...
- Ventajas³:
 - facilitan colaboración a gran escala (via confianza en el checker)
 - habilitan generación automática de demostraciones con ML. Un LLM suele devolver alucinaciones, pero pueden ser chequeadas

³Terrence Tao - Machine Assisted Proof

¿Por qué certificados?

- Si formalizamos una demostración en PPA y queremos chequear que sea correcta, hay que confiar en la implementación del *proof assistant*.
- **Criterio de De Bruijn:** si guardamos una demostración de bajo nivel de forma completa, puede ser chequeada por un programa independiente (que es sencillo de implementar).
- Cumplida por Coq, pero no Mizar⁴.

⁴Adam Naumowicz - A brief overview of Mizar

- Es un lenguaje. Frontend implementado con un *parser generator* (happy + alex)
- Permite definir axiomas y teoremas con sus demostraciones, que al **certificarse** generan una demostración en deducción natural.
- Basado en *Mathematical Vernacular*⁵: un lenguaje formal para escribir demostraciones similar al natural.

⁵The Mathematical Vernacular - Freek Wiedijk

Ejemplo

Una demostración es una secuencia de *comandos*, que pueden ir sucesivamente reduciendo la *tesis* (objetivo a probar) y agregando hipótesis a un contexto. Se mapean a reglas de deducción natural.

Teorema

theorem "implication transitivity":

$(a \rightarrow b) \ \& \ (b \rightarrow c) \rightarrow (a \rightarrow c)$ // Tesis

proof

suppose h1: $(a \rightarrow b) \ \& \ (b \rightarrow c)$

// Tesis: $a \rightarrow c$

suppose h2: a

// Tesis: c

thus c **by** h1, h2

end

- El mecanismo principal para demostrar es el **by**, que *automáticamente* demuestra que un hecho es consecuencia de una lista de hipótesis.
- Se usa en lugar de $E \rightarrow$ y $E\forall$
- Es completo para lógica proposicional pero heurístico para LPO

Ejemplo

```
axiom ax1: a -> b
axiom ax2: a
theorem thm: b
proof
  thus b by ax1, ax2
end
```

Para demostrar $((a \rightarrow b) \wedge a) \rightarrow b$ lo hacemos por el absurdo: negamos y encontramos una contradicción.

Primero convertimos la fórmula a forma normal disyuntiva (DNF)

$$\neg[((a \rightarrow b) \wedge a) \rightarrow b]$$

$$\equiv \neg[\neg((a \rightarrow b) \wedge a) \vee b]$$

$$\equiv \neg\neg((a \rightarrow b) \wedge a) \wedge \neg b$$

$$\equiv ((a \rightarrow b) \wedge a) \wedge \neg b$$

$$\equiv (\neg a \vee b) \wedge a \wedge \neg b$$

$$\equiv (\neg a \vee b) \wedge a \wedge \neg b$$

$$\equiv (\neg a \wedge a \wedge \neg b) \vee (b \wedge a \wedge \neg b)$$

$$(x \rightarrow y \equiv \neg x \vee y)$$

$$(\neg(x \vee y) \equiv \neg x \wedge \neg y)$$

$$(\neg\neg x \equiv x)$$

$$(x \rightarrow y \equiv \neg x \vee y)$$

$$((x \vee y) \wedge z \equiv (x \wedge z) \vee (y \wedge z))$$

Contradicción

Ya tenemos la fórmula en DNF, ahora tenemos que demostrar la contradicción. Lo hacemos refutando cada cláusula

$$(\neg a \wedge a \wedge \neg b) \vee (b \wedge a \wedge \neg b) \vdash \perp$$

Reglas

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} E\vee$$

$$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} E\neg$$

Teniendo en el contexto $\Gamma = \{h_1 : b_1, \dots, h_n : b_n\}$ para certificar

thus a by $h_1 \dots h_n$

- Debe demostrar $b_1 \wedge \dots \wedge b_n \rightarrow a$
- Lo hace por absurdo: la niega y encuentra una contradicción
- Primero la convierte a forma normal disyuntiva (DNF)
- Luego refuta cada cláusula (conjunción de literales)
 - False ($\perp \wedge p \wedge q$)
 - Literales opuestos ($p(a) \wedge \neg p(a) \wedge q$)
 - Eliminación de existencial ($\forall x. p(x) \wedge \neg p(a)$)

Desafío: ¡Hay que generar una demostración de deducción natural!

¿Cómo demostramos el pasaje de uno al otro?

$$\neg[((a \rightarrow b) \wedge a) \rightarrow b] \vdash \perp$$

$$\vdots$$

$$(\neg a \wedge a \wedge \neg b) \vee (b \wedge a \wedge \neg b) \vdash \perp$$

Generando demostraciones para todas las equivalencias, y convirtiendo la fórmula paso por paso (*“small step”*)

$$\neg\neg x \equiv x$$

$$\neg\perp \equiv \top$$

$$\neg\top \equiv \perp$$

$$x \rightarrow y \equiv \neg x \vee y$$

$$\neg(x \vee y) \equiv \neg x \wedge \neg y$$

$$\neg(x \wedge y) \equiv \neg x \vee \neg y$$

$$(x \vee y) \wedge z \equiv (x \wedge z) \vee (y \wedge z)$$

$$z \wedge (x \vee y) \equiv (z \wedge x) \vee (z \wedge y)$$

$$x \vee (y \vee z) \equiv (x \vee y) \vee z$$

$$x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$$

Para poder hacerlo paso por paso también hace falta demostrar la *congruencia* de los operadores

$$a \vee \neg(b \vee c) \equiv a \vee (\neg b \wedge \neg c)$$

En general,

$$\alpha \equiv \alpha' \Rightarrow \alpha \wedge \beta \equiv \alpha' \wedge \beta$$

$$\beta \equiv \beta' \Rightarrow \alpha \wedge \beta \equiv \alpha \wedge \beta'$$

Análogo para \vee, \neg

¿Por qué este mecanismo?

- Es un procedimiento completo para LP pero **heurístico** para LPO, puede fallar (i.e no demuestra cualquier cosa)
- Satisfacibilidad de LPO es indecidible
- Mecanismos como *resolución general* se pueden colgar
- Podríamos haber hecho otro, queríamos hacer *alguno*

- La lógica **clásica** no siempre es constructiva, por el *principio del tercero excluido* (LEM):

para toda proposición A , es verdadera ella o su negación

$$A \vee \neg A$$

- La lógica **intuicionista** se puede describir de forma sucinta como la lógica clásica sin LEM. Equivalentemente, tampoco vale la *eliminación de la doble negación* ($\neg\neg A \rightarrow A$)

Demostración no constructiva

Teorema

Existen dos números irracionales a, b tq a^b es racional.

Sabemos que $\sqrt{2}$ es irracional, y por LEM que $\sqrt{2}^{\sqrt{2}}$ es o racional o irracional.

- Si $\sqrt{2}^{\sqrt{2}}$ es racional, tomamos $a = b = \sqrt{2}$
- Sino, tomamos $a = \sqrt{2}^{\sqrt{2}}$, $b = \sqrt{2}$ y luego

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\sqrt{2}} = \sqrt{2}^2 = 2$$

que es racional.

¡No nos dice cuales son a y b !

- Queremos “reducir” o “ejecutar” los programas para obtener testigos de existenciales.
- La lógica clásica no es ejecutable (no constructiva). La intuicionista sí
- Friedman traduce de clásica a intuicionista. Caveat: solo fórmulas $\in \Pi_2^0$ (i.e de la forma $\forall x_1 \dots \forall x_n \exists y. \varphi$)

¿En dónde estamos parados?