Construcción Formal de Programas en Teoría de Tipos

Gustavo Betarte

Carlos Luna

Instituto de Computación (InCo)

Objetivos

- Generales:

  Iniciación al uso de métodos formales para la producción de software correcto por construcción

- Particulares:
  - **presentación de laTeoría de Tipos como lógica de programación**
  - **familiarización con ambientes de desarrollo de programas basados en ese formalismo**

Costrucción Formal de Programas en Teoría de Tipos

Métodos formales:

- **Lógicas de Programación**
  - **Teorías de Tipos**
    - **Asistentes de Pruebas**
      - **Coq**

- **Lenguaje formal para la especificación de programas (Teoría de Tipos)**
- **Razonamiento sobre la especificación y demostración propiedades**
- **Asistentes mecánicos**

Programa

- **Asistentes de pruebas para lógicos y matemáticos**
  - **Una presentación formal de la lógica proposicional y de primer orden**
- **Asistentes de pruebas para programadores**
  - **Cálculo lambda con definiciones inductivas como lenguaje de programación funcional**
- **Pruebas y programas. Especificaciones y Tipos**
  - **Isomorfismo de Curry-Howard**
  - **Extracción de programas a partir de pruebas**
  - **Caso de estudio: Modelado y verificación de políticas de seguridad en VirtualCert**

Capítulo 1: Asistentes de Pruebas para Lógicos y Matemáticos

1. Lógica Proposicional

0. Motivación

Ejemplo: División

Sean *a* y *b* $\in$ N, b≠0.

Calculamos *a* div *b* y a mod *b*
***simultáneamente* haciendo recursión en *a*:**

- **0** divmod ***b*** **= <0,0>**
- **(n+1)** divmod ***b*** **= let <q,r> = n** divmod ***b***

in **if r < b-1**

then <q,r+1>

else <q+1,0>

División: especificación y prueba

- **Especificación: Para todos $a,b \in \mathbb{N}$ tq. $b \neq 0$ existen q y r tq. $a=b.q+r$ y $r < b$.**
- **Prueba de que el programa es correcto: por inducción en *a:***

- **0** divmod ***b*** **= <0,0>** ➡ **0 = *b*.0 + 0 y 0 < *b***
- **(n+1)** divmod ***b*** **=**

```
let <q,r> = n divmod b
in  if r < b-1
        then <q,r+1>
        else <q+1,0>
```

➡

supongamos *n=b.q+r* y *r < b.*

luego, si *r < b - 1*

entonces *n+1 = b.q+(r+1)* y

r+1<b

sino *n+1 = b.(q+1)+0* y *0<b*

1. Cálculo proposicional

Lógica como herramienta para especificar y probar corrección de programas.

Cálculo proposicional

Sintaxis

- **variables de proposición: A,B,C,....**
- **conectivos: $\bot$, $\wedge$, $\vee$, $\rightarrow$, ~, $\leftrightarrow$**

Semántica

- **noción de verdad: existencia de prueba (lógica constructivista)**

Cálculo proposicional en Coq

- **Prop es la familia de proposiciones**
- **Declaración de variables de proposición:**

  Variable A B C: Prop
- **Conectivos:**
  - $\wedge, \vee, \rightarrow, \bot$ **son *primitivos*** (/\, \/, ->, False)
  - **~ y $\leftrightarrow$ son *definidos*:**

    $\sim\alpha$ **:=** $(\alpha \rightarrow \bot)$

    $\forall\alpha \leftrightarrow \beta$ **:=** $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

Construcción de pruebas en Coq

Para comenzar:

- **Queremos probar cierta *fórmula objetivo* (*goal*)**
- **Para ello, utilizaremos diferentes *tácticas* que transforman al objetivo original e introducen hipótesis adicionales**

Desarrollo de pruebas lógicas

Situación general:

- **existen varios objetivos a probar, cada uno a partir de ciertas hipótesis:**

$$\frac{\Gamma_1}{\alpha_1} \quad \frac{\Gamma_2}{\alpha_2} \quad \ldots \quad \frac{\Gamma_n}{\alpha_n}$$

secuentes

donde:

- **cada Γ_i es un conjunto de fórmulas (hipótesis) de la forma:**
  H0: γ_0, H1: γ_1,... Hk: γ_k.
- **α_i es una fórmula**
- **cada α_i debe ser probada a partir de Γ_i**

Tácticas = Reglas de inferencia

Una táctica transforma un secuente de la forma:

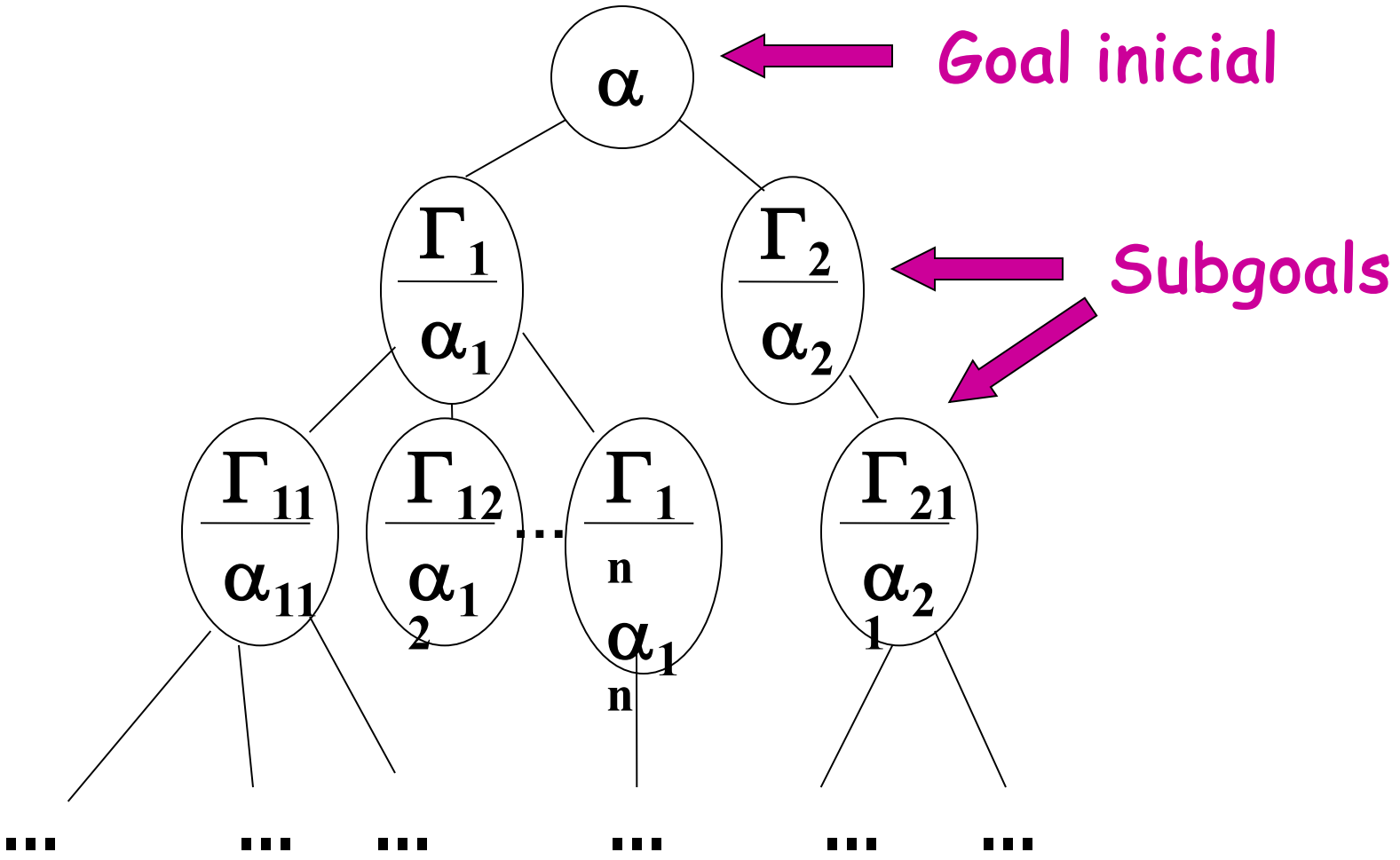
$$\frac{\Gamma_i}{\alpha_i}$$

en *cero o más* secuentes:

$$\frac{\Gamma_{i1}}{\alpha_{i1}} \quad \frac{\Gamma_{i2}}{\alpha_{i2}} \quad \ldots \quad \frac{\Gamma_{in}}{\alpha_{in}}$$

tales que probar los últimos es *suficiente* para construir una prueba del secuente original

Prueba completa = Árbol



Tácticas

caso trivial

$$\frac{\Gamma,\ H: \alpha}{\alpha}$$

assumption

Probado!

Tácticas (II)
Reglas de introducción

introducción $\rightarrow$

$$\frac{\Gamma}{\alpha \rightarrow \beta}$$

intro H

$$\frac{\Gamma \quad \mathbf{H}: \alpha}{\beta}$$

- **el identificador H es opcional**
- **variantes:** intros, intros $H_1 \ldots H_n$

Tácticas (III)

Reglas de introducción (cont.)

introducción $\vee$ (Izq.)

$$\frac{\Gamma}{\alpha \vee \beta}$$ left $$\frac{\Gamma}{\alpha}$$

introducción $\vee$ (Der.)

$$\frac{\Gamma}{\alpha \vee \beta}$$ right $$\frac{\Gamma}{\beta}$$

Tácticas (IV)

Reglas de introducción (cont.)

introducción $\wedge$

$$\frac{\Gamma}{\alpha \wedge \beta}$$ **split** $$\frac{\Gamma}{\alpha} \qquad \frac{\Gamma}{\beta}$$

"introducción" $\bot$

$$\frac{\Gamma}{\textbf{False}}$$ **absurd** α $$\frac{\Gamma}{\alpha} \qquad \frac{\Gamma}{\sim\alpha}$$

Tácticas (V)
Reglas de eliminación

eliminación $\rightarrow$

$$\frac{\Gamma \quad H: \alpha \rightarrow \beta}{\beta}$$

apply H

$$\frac{\Gamma \quad H: \alpha \rightarrow \beta}{\alpha}$$

en general:

$$\frac{\Gamma \quad H: \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \beta}{\beta}$$

apply H

$$\frac{\Gamma \quad H: \ldots}{\alpha_1} \qquad \frac{\Gamma \quad H: \ldots}{\alpha_2} \qquad \ldots$$

Tácticas (VI)

Reglas de eliminación (cont.)

eliminación $\vee$

$$\frac{\Gamma \quad H: \alpha \vee \beta}{\sigma}$$

elim H

$$\frac{\Gamma \quad H: \alpha \vee \beta}{\alpha \to \sigma} \qquad \frac{\Gamma \quad H: \alpha \vee \beta}{\beta \to \sigma}$$

eliminación $\wedge$

$$\frac{\Gamma \quad H: \alpha \wedge \beta}{\sigma}$$

elim H

$$\frac{\Gamma \quad H: \alpha \wedge \beta}{\alpha \to \beta \to \sigma}$$

Tácticas (VII)

Reglas de eliminación (cont.)

eliminación ⊥

Γ
H: False

σ

elim H

Probado!

Tácticas (VIII)
Otras tácticas

Utilización de lema intermediario

$$\frac{\Gamma}{\alpha} \quad \text{cut } \beta \quad \frac{\Gamma}{\beta \rightarrow \alpha} \quad \frac{\Gamma}{\beta}$$

Explicitación de la prueba:

$$\frac{\Gamma}{\alpha} \quad \text{exact } t \quad \textbf{Probado!}$$

donde: t debe ser una prueba de α (figura en las hipótesis de Γ o es el nombre de un lema ya probado)

Tácticas (IX)
Eliminación de definiciones

Eliminación de definición

$$\frac{\Gamma}{\sim\alpha}$$ unfold not $$\frac{\Gamma}{\alpha \rightarrow \text{False}}$$

Eliminación de definición

$$\frac{\Gamma}{\alpha \leftrightarrow \beta}$$ unfold iff $$\frac{\Gamma}{(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)}$$

unfold las hipótesis: unfold *nom* in *h*

Observaciones

- **En general:**
  - **si *term* es una prueba de una proposición α (que figura en las hipótesis o es el nombre de un lema ya probado), entonces elim *term*, aplica la regla de eliminación del conectivo principal de α.**
  - **apply y elim no se aplican solamente a hipótesis sino también a lemas previamente demostrados.**

Esquemas de proposición

Si demostramos

```
Variable A:Prop.
Lemma L1 : A→ A.
Proof. … Qed.
```

La prueba es análoga para *cualquier* proposición *A*.

Un *esquema de proposición* se expresa como:

Para toda proposición *A*, se cumple *A*→ *A*.

En Coq este esquema se expresa:

```
forall A:Prop, A→ A
```

Esquemas de proposición
Instanciación

Ejemplo:

Si ya demostramos en Coq

```
Lemma Lid : forall A:Prop, A→ A
Proof.
…
Qed.
```

Para demostrar ~False → ~False

podemos utilizar exact (Lid ~False).

2. Cálculo proposicional clásico

Principio del tercero excluído:

Para toda proposición P, se cumple $P \vee \sim P$

En lógica constructiva, una proposición es <u>verdadera</u> *si y sólo si* podemos <u>exhibir una prueba.</u>

- **Sabemos probar instancias de este axioma, pero no sabemos probar el esquema pues para cada proposición deberíamos exhibir una prueba de ella o de su negación ¿como haríamos con las conjeturas y resultados indecidibles?**

<u>Ejemplo</u>: sabemos probar $1=0 \vee \sim 1=0$

no sabemos (hoy) probar $P=NP \vee \sim P=NP$

Cálculo proposicional clásico

El principio del tercero excluído no se puede demostrar en la lógica constructiva

→ puede agregarse como un AXIOMA

En Coq este axioma está declarado en el módulo *Classical* y se llama *classic* .

```
Require Import Classical.
        ...
check (classic A).
        (classic A) : (A \/ ~A)
```

3. Razonamiento automatizado

Tautologías constructivas

$$\frac{\Gamma}{\alpha}$$ tauto **Probado!**

tauto **implementa una estrategia de construcción de pruebas de tautologías constructivas.**

Esta estrategia tiene éxito cuando α es una *tautología* cuya prueba *no usa el tercero excluído*.

Construye una prueba.

Táctica (Semi)Automática

Aplicación (hacia atrás) de lemas e hipótesis

$$\frac{\Gamma}{\alpha}\ \texttt{auto}$$ **Probado!**

O bien

$$\frac{\Gamma}{\alpha}\ \texttt{auto}\ \frac{\Gamma}{\alpha}$$

auto implementa una estrategia de construcción de pruebas similar a la de Prolog (razonamiento hacia atrás) aplicando lemas e hipótesis. Considera únicamente aquellos lemas declarados como Hint. Construye una prueba.