Demostraciones ejecutables

Ciclo de charlas CCCC

12 de agosto de 2022

Pablo Barenbaum

Instituto de Ciencias de la Computación
FCEyN, UBA, Argentina

Universidad Nacional de Quilmes / CONICET
Argentina

Lógica	Computación

Lógica

$$\frac{A \text{ form} \qquad B \text{ form}}{(A \to B) \text{ form}}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

Deducción natural

Gentzen ($\sim$1934)

Computación

Lógica	Computación
$\dfrac{A \text{ form} \qquad B \text{ form}}{(A \to B) \text{ form}}$	$\dfrac{A \text{ type} \qquad B \text{ type}}{(A \to B) \text{ type}}$
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.\, e : A \to B}$
$\dfrac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash e' : A}{\Gamma \vdash e\, e' : B}$
Deducción natural	**Cálculo-λ simplemente tipado**
Gentzen ($\sim$1934)	Church ($\sim$1940)

Lógica	Computación

Lógica	Computación
Proposición, fórmula	Tipo, especificación

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
???	Ejecutar un programa

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
???	Polimorfismo paramétrico (*generics*)

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
Lógica de segundo orden	Polimorfismo paramétrico (*generics*)

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
Lógica de segundo orden	Polimorfismo paramétrico (*generics*)
Lógica de primer orden	???

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
Lógica de segundo orden	Polimorfismo paramétrico (*generics*)
Lógica de primer orden	Tipos dependientes

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
Lógica de segundo orden	Polimorfismo paramétrico (*generics*)
Lógica de primer orden	Tipos dependientes
⋮	⋮

Lógica	Computación
Proposición, fórmula	Tipo, especificación
Demostración	Programa
Normalizar una demostración	Ejecutar un programa
Lógica de segundo orden	Polimorfismo paramétrico (*generics*)
Lógica de primer orden	Tipos dependientes
⋮	⋮

"Isomorfismo de Curry–Howard"
"Correspondencia entre proposiciones y tipos"
"Correspondencia entre pruebas y programas"

¿Para qué estudiar esta correspondencia?

¿Para qué estudiar esta correspondencia?

Diseñar lenguajes de programación en los cuales los tipos permitan expresar propiedades sobre el comportamiento de los programas.

```
inversa :  Matriz → Matriz
inversa =  ...

inversa-correcta : ∀ (m :  Matriz) → det m ≠ 0
                   → m * inversa m = id
inversa-correcta = ...
```

¿Para qué estudiar esta correspondencia?

Diseñar lenguajes de programación en los cuales los tipos permitan expresar propiedades sobre el comportamiento de los programas.

```
inversa :  Matriz → Matriz
inversa =  ...

inversa-correcta : ∀ (m :  Matriz) → det m ≠ 0
                   → m * inversa m = id
inversa-correcta = ...
```

Más aún: **dar programas correctos por construcción**.

```
inversa' : ∀ (m :  Matriz) → det m ≠ 0
           → ∃ (m' :  Matriz) × (m * m' = id)
inversa' = ...
```

¿Para qué estudiar esta correspondencia?

Diseñar lenguajes de programación en los cuales los tipos permitan expresar propiedades sobre el comportamiento de los programas.

```
inversa :  Matriz → Matriz
inversa =  ...

inversa-correcta : ∀ (m :  Matriz) → det m ≠ 0
                   → m * inversa m = id
inversa-correcta = ...
```

Más aún: **dar programas correctos por construcción**.

```
inversa' : ∀ (m :  Matriz) → det m ≠ 0
           → ∃ (m' :  Matriz) × (m * m' = id)
inversa' = ...
```

El sistema lógico es consistente. **Todos los programas terminan.**

Ejemplos: Coq, Agda, Lean, Isabelle, F*, ...

Líneas de trabajo

Colaboradores

Proposiciones clásicas como tipos ..... T. Freund

Tipos cuantitativos ..... B. Accattoli, D. Kesner, M. Milicich

Notación para reescrituras de orden superior ..... E. Bonelli

Cálculo-λ funcional–lógico ..... F. Giordano, F. Lochbaum, M. Milicich

Líneas de trabajo

Colaboradores

Proposiciones clásicas como tipos . . . . . T. Freund

Tipos cuantitativos . . . . . B. Accattoli, D. Kesner, M. Milicich

Notación para reescrituras de orden superior . . . . . E. Bonelli

Cálculo-λ funcional–lógico . . . . . F. Giordano, F. Lochbaum, M. Milicich

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.

Pero tienen que ser constructivas.

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.
Pero tienen que ser constructivas.

Ejemplo de demostración no constructiva

$P(x)$	:	"padezco x"
$C(x)$	:	"creo que padezco x"
H	:	"hipocondría"

Postulado. $P(H) \longleftrightarrow \exists x.(C(x) \wedge \neg P(x))$
"Padezco hipocondría si y sólo si creo que padezco algo que no padezco."

Teorema. $C(H) \to P(H)$
"Si creo que padezco hipocondría, padezco hipocondría."

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.
Pero tienen que ser constructivas.

Ejemplo de demostración no constructiva

$P(x)$	:	"padezco x"
$C(x)$	:	"creo que padezco x"
H	:	"hipocondría"

Postulado. $P(H) \longleftrightarrow \exists x.(C(x) \wedge \neg P(x))$
"Padezco hipocondría si y sólo si creo que padezco algo que no padezco."

Teorema. $C(H) \to P(H)$
"Si creo que padezco hipocondría, padezco hipocondría."

Demostración. Supongamos $C(H)$. Se da $P(H) \vee \neg P(H)$.

Si $P(H)$, listo.

Si $\neg P(H)$, tenemos $C(H) \wedge \neg P(H)$. Luego $P(H)$. Absurdo.

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.
Pero tienen que ser constructivas.

Ejemplo de demostración no constructiva

$P(x)$	:	"padezco x"
$C(x)$	:	"creo que padezco x"
H	:	"hipocondría"

Postulado. $P(H) \longleftrightarrow \exists x.(C(x) \wedge \neg P(x))$
"Padezco hipocondría si y sólo si creo que padezco algo que no padezco."

Teorema. $C(H) \to P(H)$
"Si creo que padezco hipocondría, padezco hipocondría."

Demostración. Supongamos $C(H)$. Se da $P(H) \vee \neg P(H)$.

Si $P(H)$, listo.

Si $\neg P(H)$, tenemos $C(H) \wedge \neg P(H)$. Luego $P(H)$. Absurdo.

Asumiendo $C(H)$, la demostración no exhibe un x tal que $C(x) \wedge \neg P(x)$.

Proposiciones clásicas como tipos

Las demostraciones se pueden ejecutar.
Pero tienen que ser constructivas.

Ejemplo de demostración no constructiva

$P(x)$	:	"padezco x"
$C(x)$	:	"creo que padezco x"
H	:	"hipocondría"

Postulado. $P(H) \longleftrightarrow \exists x.(C(x) \wedge \neg P(x))$
"Padezco hipocondría si y sólo si creo que padezco algo que no padezco."

Teorema. $C(H) \to P(H)$
"Si creo que padezco hipocondría, padezco hipocondría."

Demostración. Supongamos $C(H)$. Se da $\underbrace{P(H) \vee \neg P(H)}_{\text{tercero excluido}}$.

Si $P(H)$, listo.

Si $\neg P(H)$, tenemos $C(H) \wedge \neg P(H)$. Luego $P(H)$. Absurdo.

Asumiendo $C(H)$, la demostración no exhibe un x tal que $C(x) \wedge \neg P(x)$.

Proposiciones clásicas como tipos

Lógica clásica

A diferencia de la lógica intuicionista

- Admite el principio del tercero excluido.
- *A priori* no constructiva.

Proposiciones clásicas como tipos

Lógica clásica

A diferencia de la lógica intuicionista

- Admite el principio del tercero excluido.
- *A priori* no constructiva.

¿Se le puede dar una interpretación computacional?

Varios intentos:

Cálculo-λ simétrico . . . . . F. Barbanera, S. Berardi

Cálculo $\lambda\mu$ . . . . . M. Parigot

Cálculo $\bar{\lambda}\mu\tilde{\mu}$ . . . . . P.-L. Curien, H. Herbelin

. . .

Proposiciones clásicas como tipos

Lógica clásica

A diferencia de la lógica intuicionista

- Admite el principio del tercero excluido.
- *A priori* no constructiva.

¿Se le puede dar una interpretación computacional?

Varios intentos:

Cálculo-λ simétrico . . . . . F. Barbanera, S. Berardi

Cálculo $\lambda\mu$ . . . . . M. Parigot

Cálculo $\bar{\lambda}\mu\tilde{\mu}$ . . . . . P.-L. Curien, H. Herbelin

. . .

Cálculo-λ^{PRK}

con T. Freund

$$A^{\oplus} \simeq (A^{\ominus} \to A^{+}) \qquad A^{\ominus} \simeq (A^{\oplus} \to A^{-})$$

Simétrico con respecto a la dualidad de de Morgan.

Confluencia, terminación, semántica de Kripke, extensión a segundo orden.

Tipos cuantitativos

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.

```
f :  Int → Int → Int
f n m = n + n
```

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ *e* termina	

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ *e* termina	*e* tiene tipo $\Longleftrightarrow$ *e* termina

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ *e* termina	*e* tiene tipo $\Longleftrightarrow$ *e* termina
Inferencia decidible.	

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ *e* termina	*e* tiene tipo $\Longleftrightarrow$ *e* termina
Inferencia decidible.	Inferencia indecidible.

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ *e* termina	*e* tiene tipo $\Longleftrightarrow$ *e* termina
Inferencia decidible.	Inferencia indecidible.
Captura propiedades estáticas. Asegurar terminación. Asegurar invariantes.	

Tipos cuantitativos

Tipos simples	**Tipos cuantitativos** (intersección no idempotente)
Cada subexpresión tiene un tipo.	Cada subexpresión tiene tantos tipos como veces se usa.
`f : Int → Int → Int` `f n m = n + n`	`f : [Int, Int] → [] → Int` `f n m = n + n`
e tiene tipo $\Longrightarrow$ e termina	e tiene tipo $\Longleftrightarrow$ e termina
Inferencia decidible.	Inferencia indecidible.
Captura propiedades estáticas.	Captura propiedades dinámicas.
Asegurar terminación. Asegurar invariantes.	Medir el tiempo de ejecución. Medir el tamaño del resultado.

Tipos cuantitativos

Algunos problemas:

- Dar un sistema de tipos cuantitativos para el *useful strong call-by-need* de Accattoli y Dal Lago.

con B. Accattoli y D. Kesner

Tipos cuantitativos

Algunos problemas:

- Dar un sistema de tipos cuantitativos para el *useful strong call-by-need* de Accattoli y Dal Lago.

  con B. Accattoli y D. Kesner

- Establecer una correspondencia entre derivaciones en sistemas de tipos cuantitativos y secuencias de reducción estratégicas.

  con E. Bonelli y M. Milicich

Tipos cuantitativos

Algunos problemas:

- Dar un sistema de tipos cuantitativos para el *useful strong call-by-need* de Accattoli y Dal Lago.

  con B. Accattoli y D. Kesner

- Establecer una correspondencia entre derivaciones en sistemas de tipos cuantitativos y secuencias de reducción estratégicas.

  con E. Bonelli y M. Milicich

- Estudiar la acción de las traducciones a *continuation-passing style* sobre los tipos cuantitativos.

  con D. Kesner y M. Milicich