Simply typed λ-calculus

Sabine Broda
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

MAP-i, Porto 2008

Simple Types

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
    - each type-variable a is a type (atomic);

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Convention: $\to$ associates to the right

Simple Types

- infinite set of *type-variables* $a, b, c, \ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Convention: $\to$ associates to the right

$$a \to b \to c \to d$$

stands for

$$(a \to (b \to (c \to d)))$$

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Convention: $\to$ associates to the right

$$a \to b \to c \to d$$
stands for
$$(a \to (b \to (c \to d)))$$

Examples: a, $a \to a$, $((a \to b) \to a) \to a$

Type-assignment

Type-assignment

- an expression $M : \alpha$ is a *type-assignment* (M is called its subject);

Type-assignment

- an expression $M : \alpha$ is a *type-assignment* (M is called its subject);
- a *type-context* is a finite, perhaps empty, set of type-assignments

$$\Gamma = \{x_1 : \alpha_1, \ldots, x_n : \alpha_n\},$$

such that $x_1, \ldots, x_n$ are distinct term-variables.

The system $\lambda\!\rightarrow$-Curry

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

The system $\lambda\!\to$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

(axiom) $\dfrac{}{x : \alpha \ \vdash \ x : \alpha}$

The system $\lambda\!\to$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{x : \alpha \quad \vdash \quad x : \alpha}$$

$$\text{(app)} \quad \frac{\Gamma_1 \vdash M : \alpha \to \beta \qquad \Gamma_2 \vdash N : \alpha}{\Gamma_1 \cup \Gamma_2 \vdash MN : \beta} \qquad (\Gamma_1 \cup \Gamma_2 \text{ consistent})$$

The system $\lambda{\rightarrow}$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{x : \alpha \vdash x : \alpha}$$

$$\text{(app)} \quad \frac{\Gamma_1 \vdash M : \alpha \to \beta \qquad \Gamma_2 \vdash N : \alpha}{\Gamma_1 \cup \Gamma_2 \vdash MN : \beta} \qquad (\Gamma_1 \cup \Gamma_2 \text{ consistent})$$

$$\text{(abs)} \quad \frac{\Gamma \vdash M : \beta}{\Gamma \setminus \{x : \alpha\} \vdash \lambda x.M : \alpha \to \beta}$$

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{x : \alpha \;\vdash\; x : \alpha}$$

$$\text{(app)} \quad \frac{\Gamma_1 \vdash M : \alpha \to \beta \qquad \Gamma_2 \vdash N : \alpha}{\Gamma_1 \cup \Gamma_2 \vdash MN : \beta} \qquad (\Gamma_1 \cup \Gamma_2 \text{ consistent})$$

$$\text{(abs)} \quad \frac{\Gamma \;\vdash\; M : \beta}{\Gamma \setminus \{x : \alpha\} \;\vdash\; \lambda x.M : \alpha \to \beta}$$

A deduction Δ of $\Gamma \vdash M : \tau$ is a tree of formulae, those at the tops of branches being axioms and those below being deduced from those immediately above them by a rule ((app) or (abs)) and with bottom formula $\Gamma \vdash M : \tau$.

Related problems:

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$?
  (M is said to be typable)

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$? (M is said to be typable)
- *Inhabitation:* Given Γ and τ, is there M such that $\Gamma \vdash M : \tau$? (If $\Gamma = \emptyset$, we say that τ is inhabited; also M is called an inhabitant of τ)

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$? (M is said to be typable)
- *Inhabitation:* Given Γ and τ, is there M such that $\Gamma \vdash M : \tau$? (If $\Gamma = \emptyset$, we say that τ is inhabited; also M is called an inhabitant of τ)

All these problems are decidable!

Exercises

1. Show that $\vdash \lambda x.x : a \to a$.
2. Show that $\vdash \lambda x.x : (a \to b) \to a \to b$.
3. Find Γ and α such that $\Gamma \vdash (\lambda xy.xy)z : \alpha$.
4. Find α such that $\vdash \lambda x.xx : \alpha$.
5. Find M such that $\vdash M : a \to b \to a$.
6. Find M such that $\vdash M : ((a \to b) \to a) \to a$.

Subject-reduction/expansion

Subject-reduction/expansion

Subject-reduction: If $M \rightarrow_\beta N$ and $\Gamma \vdash M : \sigma$, then $\Gamma_N \vdash M : \sigma$.

Subject-reduction/expansion

Subject-reduction: If $M \to_\beta N$ and $\Gamma \vdash M : \sigma$, then $\Gamma_N \vdash M : \sigma$.

Subject-expansion? Take $M = (\lambda xyz.xz(yz))(\lambda xy.x)$, $N = \lambda xy.y$ and $\sigma = a \to b \to b$. Show that $M \to_\beta N$, $\vdash N : \sigma$ and $\nvdash M : \sigma$.

Subject-reduction/expansion

Subject-reduction: If $M \to_\beta N$ and $\Gamma \vdash M : \sigma$, then $\Gamma_N \vdash M : \sigma$.

Subject-expansion? Take $M = (\lambda xyz.xz(yz))(\lambda xy.x)$, $N = \lambda xy.y$ and $\sigma = a \to b \to b$. Show that $M \to_\beta N$, $\vdash N : \sigma$ and $\nvdash M : \sigma$.

Conclusion?

Subject-reduction/expansion

Subject-reduction: If $M \to_\beta N$ and $\Gamma \vdash M : \sigma$, then $\Gamma_N \vdash M : \sigma$.

Subject-expansion? Take $M = (\lambda xyz.xz(yz))(\lambda xy.x)$, $N = \lambda xy.y$ and $\sigma = a \to b \to b$. Show that $M \to_\beta N$, $\vdash N : \sigma$ and $\nvdash M : \sigma$.

Conclusion?

And for $M = (\lambda xy.y)(\lambda x.xx)$?

Principal pairs/types

Principal pairs/types

Principal pairs: (Γ, σ) is a principal pair for M iff
- $\Gamma \vdash M : \sigma$;
- if $\Gamma' \vdash M : \sigma'$, then there is a substitution S such that $S(\Gamma) = \Gamma'$ and $S(\sigma) = \sigma'$.

Principal pairs/types

Principal pairs: (Γ, σ) is a principal pair for M iff
- $\Gamma \vdash M : \sigma$;
- if $\Gamma' \vdash M : \sigma'$, then there is a substitution S such that $S(\Gamma) = \Gamma'$ and $S(\sigma) = \sigma'$.

Principal type theorem: There is an algorithm *pp*, such that for every term M, *pp* computes a principal pair for M if some exists, and fails otherwise.

Properties

Properties

- Confluence (Church-Rosser);

Properties

- Confluence (Church-Rosser);
- strong normalization;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;
- subject-reduction;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;
- subject-reduction;
- principal types;

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^α, x^β, $\ldots y^\alpha$, $\ldots$;

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^α, x^β, $\ldots$ y^α, $\ldots$;
  - each annotated variable x^α is a λ-term of type α;

The system $\lambda{\rightarrow}$-Church

- term-variables annotated with types: x^α, x^β, $\ldots y^\alpha$, $\ldots$;
  - each annotated variable x^α is a λ-term of type α;
  - if M and N are λ-terms, respectively of type $\alpha \rightarrow \beta$ and α, then (MN) is a λ-term of type β, (*application*);

The system $\lambda \rightarrow$-Church

- term-variables annotated with types: x^α, x^β, $\ldots y^\alpha$, $\ldots$;
    - each annotated variable x^α is a λ-term of type α;
    - if M and N are λ-terms, respectively of type $\alpha \rightarrow \beta$ and α, then (MN) is a λ-term of type β, (*application*);
    - if M is a λ-term of type β and x^α an annotated variable, then $(\lambda x^\alpha.M)$ is a λ-term of type $\alpha \rightarrow \beta$, (*abstraction*).

The system $\lambda{\rightarrow}$-Church

- term-variables annotated with types: $x^\alpha, x^\beta, \ldots y^\alpha, \ldots;$
  - each annotated variable x^α is a λ-term of type α;
  - if M and N are λ-terms, respectively of type $\alpha \to \beta$ and α, then (MN) is a λ-term of type β, (*application*);
  - if M is a λ-term of type β and x^α an annotated variable, then $(\lambda x^\alpha.M)$ is a λ-term of type $\alpha \to \beta$, (*abstraction*).

Church vs. Curry Differences and similarities...