Partiality and Recursion in Higher-Order Logic

Łukasz Czajka

Institute of Informatics
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

19 Mar 2013

This presentation is about

- a certain approach to allowing **arbitrary recursive definitions in higher-order logic**,

This presentation is about

- a certain approach to allowing **arbitrary recursive definitions in higher-order logic**,
- and about partiality, but only as far as it arises from non-terminating recursion,

This presentation is about

- a certain approach to allowing **arbitrary recursive definitions in higher-order logic**,
- and about partiality, but only as far as it arises from non-terminating recursion,
    - we are not interested in things like $x/0$, $\mathrm{head}(\mathrm{nil})$, etc.

Higher-order logic

Syntax

- Types: $\mathcal{T} \quad ::= \quad \mathrm{Prop} \mid \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$

Higher-order logic

Syntax

- Types: $\mathcal{T} \quad ::= \quad \mathrm{Prop} \mid \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$
- Constants:
  - $\forall_\tau \in \Sigma_{(\tau \to \mathrm{Prop}) \to \mathrm{Prop}}$
  - $\supset \in \Sigma_{\mathrm{Prop} \to \mathrm{Prop} \to \mathrm{Prop}}$

Higher-order logic

Syntax

- Types: $\mathcal{T} \quad ::= \quad \mathrm{Prop} \mid \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$
- Constants:
  - $\forall_\tau \in \Sigma_{(\tau \to \mathrm{Prop}) \to \mathrm{Prop}}$
  - $\supset \in \Sigma_{\mathrm{Prop} \to \mathrm{Prop} \to \mathrm{Prop}}$
- Set of terms T_τ of type τ of the simply-typed λ-calculus:
  - $\Sigma_\tau, V_\tau \subseteq T_\tau$,
  - if $t \in T_{\tau_1 \to \tau_2}$ and $s \in T_{\tau_1}$ then $ts \in T_{\tau_2}$,
  - if $x \in V_{\tau_1}$ and $t \in T_{\tau_2}$ then $\lambda x : \tau \,.\, t \in T_{\tau_1 \to \tau_2}$.

Higher-order logic

Syntax

- Types: $\mathcal{T} \quad ::= \quad \mathrm{Prop} \mid \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$
- Constants:
  - $\forall_\tau \in \Sigma_{(\tau\to\mathrm{Prop})\to\mathrm{Prop}}$
  - $\supset \in \Sigma_{\mathrm{Prop}\to\mathrm{Prop}\to\mathrm{Prop}}$
- Set of terms T_τ of type τ of the simply-typed λ-calculus:
  - $\Sigma_\tau, V_\tau \subseteq T_\tau$,
  - if $t \in T_{\tau_1\to\tau_2}$ and $s \in T_{\tau_1}$ then $ts \in T_{\tau_2}$,
  - if $x \in V_{\tau_1}$ and $t \in T_{\tau_2}$ then $\lambda x : \tau \,.\, t \in T_{\tau_1\to\tau_2}$.
- Usual conventions:
  - φ, ψ, etc. – terms of type Prop,
  - $\varphi \supset \psi \equiv \supset\varphi\psi$,
  - $\forall x : \tau \,.\, \varphi \equiv \forall_\tau(\lambda x : \tau \,.\, \varphi)$.

Higher-order logic

Ratural deduction rules (simple intuitionistic intensional variant)

$$\frac{}{\Delta, \varphi \vdash \varphi}$$

$$\supset_i: \frac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e: \frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i : \frac{\Delta \vdash \varphi}{\Delta \vdash \forall x : \tau \,.\, \varphi} \; x \in V_\tau, x \notin FV(\Delta)$$

$$\forall_e : \frac{\Delta \vdash \forall x : \tau \,.\, \varphi}{\Delta \vdash \varphi[x/t]} \; t \in T_\tau$$

$$\mathrm{conv} : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

Higher-order logic with recursion (inconsistent)

The naive approach

Forget the type information.

Higher-order logic with recursion (inconsistent)

Syntax

- No types.

Higher-order logic with recursion (inconsistent)

Syntax

- No types.
  - Domains may be represented by constants (predicates).

Higher-order logic with recursion (inconsistent)

Syntax

- No types.
  - Domains may be represented by constants (predicates).
- Constants: $\forall, \supset \in \Sigma$.

Higher-order logic with recursion (inconsistent)

Syntax

- No types.
  - Domains may be represented by constants (predicates).
- Constants: $\forall, \supset \in \Sigma$.
- Set of terms T of the *untyped* λ-calculus:
  - $\Sigma, V \subseteq T$,
  - if $t \in T$ and $s \in T$ then $ts \in T$,
  - if $x \in V$ and $t \in T$ then $\lambda x \,.\, t \in T$.

Higher-order logic with recursion (inconsistent)

Syntax

- No types.
  - Domains may be represented by constants (predicates).
- Constants: $\forall, \supset \in \Sigma$.
- Set of terms T of the *untyped* λ-calculus:
  - $\Sigma, V \subseteq T$,
  - if $t \in T$ and $s \in T$ then $ts \in T$,
  - if $x \in V$ and $t \in T$ then $\lambda x \,.\, t \in T$.
- Usual conventions:
  - $\varphi \supset \psi \equiv \supset \varphi\psi$,
  - $\forall x \,.\, \varphi \equiv \forall(\lambda x \,.\, \varphi)$.

Higher-order logic with recursion (inconsistent)

Recursion

In the untyped λ-calculus any set of equations of the form

$$\begin{aligned} z_1 x_1 \ldots x_m &=_\beta \Phi_1(z_1, \ldots, z_n, x_1, \ldots, x_m) \\ &\vdots \\ z_n x_1 \ldots x_m &=_\beta \Phi_n(z_1, \ldots, z_n, x_1, \ldots, x_m) \end{aligned}$$

has a solution for $z_1, \ldots, z_n$, where the expressions $\Phi_i(z_1, \ldots, z_n, x_1, \ldots, x_m)$ are arbitrary terms with the free variables listed.

Higher-order logic with recursion (inconsistent)

In the untyped λ-calculus any set of equations of the form

$$\begin{aligned} z_1 x_1 \dots x_m &=_\beta \Phi_1(z_1, \dots, z_n, x_1, \dots, x_m) \\ &\vdots \\ z_n x_1 \dots x_m &=_\beta \Phi_n(z_1, \dots, z_n, x_1, \dots, x_m) \end{aligned}$$

has a solution for $z_1, \dots, z_n$, where the expressions $\Phi_i(z_1, \dots, z_n, x_1, \dots, x_m)$ are arbitrary terms with the free variables listed.

In other words, for any such set of equations, there exist terms $t_1, \dots, t_n$ such that for any terms $s_1, \dots, s_m$ we have $t_i s_1 \dots s_m =_\beta \Phi_i(t_1, \dots, t_n, s_1, \dots, s_m)$ for each $i = 1, \dots, n$.

Higher-order logic with recursion (inconsistent)

Rules (minimal logic)

$$\frac{}{\Delta, \varphi \vdash \varphi}$$

$$\supset_i : \frac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e : \frac{\Delta \vdash \varphi \supset \psi \qquad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i : \frac{\Delta \vdash \varphi}{\Delta \vdash \forall x \,.\, \varphi}\ x \notin FV(\Delta) \qquad \forall_e : \frac{\Delta \vdash \forall x \,.\, \varphi}{\Delta \vdash \varphi[x/t]}\ t \in T$$

$$\mathrm{conv} : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

Higher-order logic with recursion (inconsistent)

Rules (minimal logic)

$$\frac{}{\Delta, \varphi \vdash \varphi}$$

$$\supset_i: \frac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e: \frac{\Delta \vdash \varphi \supset \psi \qquad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i : \frac{\Delta \vdash \varphi}{\Delta \vdash \forall x \,.\, \varphi}\ x \notin FV(\Delta) \qquad \forall_e : \frac{\Delta \vdash \forall x \,.\, \varphi}{\Delta \vdash \varphi[x/t]}\ t \in T$$

$$\text{conv} : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

This is actually (more or less) what people (Church, Curry, ...) initially tried in the 1930s ...

Higher-order logic with recursion (inconsistent)

Curry's paradox

. . . and obviously it didn't work.

Higher-order logic with recursion (inconsistent)

Curry's paradox

... and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

Higher-order logic with recursion (inconsistent)

Curry's paradox

. . . and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

Higher-order logic with recursion (inconsistent)

Curry's paradox

. . . and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

(2) $\varphi \vdash \varphi \supset \psi$ by conv from (1),

Higher-order logic with recursion (inconsistent)

Curry's paradox

...and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

(2) $\varphi \vdash \varphi \supset \psi$ by conv from (1),

(3) $\varphi \vdash \psi$ by $\supset_e$ from (1) and (2),

Higher-order logic with recursion (inconsistent)

Curry's paradox

. . . and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

(2) $\varphi \vdash \varphi \supset \psi$ by conv from (1),

(3) $\varphi \vdash \psi$ by $\supset_e$ from (1) and (2),

(4) $\vdash \varphi \supset \psi$ by $\supset_i$ from (3),

Higher-order logic with recursion (inconsistent)

Curry's paradox

...and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

(2) $\varphi \vdash \varphi \supset \psi$ by conv from (1),

(3) $\varphi \vdash \psi$ by $\supset_e$ from (1) and (2),

(4) $\vdash \varphi \supset \psi$ by $\supset_i$ from (3),

(5) $\vdash \varphi$ by conv from (4),

Higher-order logic with recursion (inconsistent)

Curry's paradox

...and obviously it didn't work.

For an arbitrary given term ψ, define φ by $\varphi =_\beta \varphi \supset \psi$.

(1) $\varphi \vdash \varphi$ by axiom,

(2) $\varphi \vdash \varphi \supset \psi$ by conv from (1),

(3) $\varphi \vdash \psi$ by $\supset_e$ from (1) and (2),

(4) $\vdash \varphi \supset \psi$ by $\supset_i$ from (3),

(5) $\vdash \varphi$ by conv from (4),

(6) $\vdash \psi$ by $\supset_e$ from (4) and (5).

Higher-order logic

A digression

In ordinary higher-order logic constants of type $\mathrm{Prop} \to \tau$, $(\alpha \to \mathrm{Prop}) \to \tau$, etc. for $\tau \neq \mathrm{Prop}$ are allowed.

Higher-order logic

A digression

In ordinary higher-order logic constants of type $\mathrm{Prop} \to \tau$, $(\alpha \to \mathrm{Prop}) \to \tau$, etc. for $\tau \neq \mathrm{Prop}$ are allowed.

It is rather obvious that one could invent a higher-order logic allowing arbitrary recursive *programs*, where the programs and the logic would not be mixed.

Higher-order logic

A digression

In ordinary higher-order logic constants of type $\mathrm{Prop} \to \tau$, $(\alpha \to \mathrm{Prop}) \to \tau$, etc. for $\tau \neq \mathrm{Prop}$ are allowed.

It is rather obvious that one could invent a higher-order logic allowing arbitrary recursive *programs*, where the programs and the logic would not be mixed.

- E.g. have a domain for untyped programs in ordinary higher-order logic.

Higher-order logic

A digression

In ordinary higher-order logic constants of type $\mathrm{Prop} \to \tau$, $(\alpha \to \mathrm{Prop}) \to \tau$, etc. for $\tau \neq \mathrm{Prop}$ are allowed.

It is rather obvious that one could invent a higher-order logic allowing arbitrary recursive *programs*, where the programs and the logic would not be mixed.

- E.g. have a domain for untyped programs in ordinary higher-order logic.
- But we study recursion *in* **higher-order logic**.

Higher-order logic with recursion

The correct version

- We need types after all.

Higher-order logic with recursion

The correct version

- We need types after all.
- But they will be *internal* to the system.

Higher-order logic with recursion

The correct version

- We need types after all.
- But they will be *internal* to the system.
- In this context it is perhaps better to think of types as *sets*.

Higher-order logic with recursion

Syntax

- Constants: $\forall, \supset, \rightarrow, \mathrm{Prop}, \mathrm{Type} \in \Sigma$ plus one constant for each base type ($\mathcal{B} \subseteq \Sigma$).

Higher-order logic with recursion

Syntax

- Constants: $\forall, \supset, \to, \mathrm{Prop}, \mathrm{Type} \in \Sigma$ plus one constant for each base type ($\mathcal{B} \subseteq \Sigma$).
- Set of terms T of the untyped λ-calculus with constants from Σ.

Higher-order logic with recursion

Syntax

- Constants: $\forall, \supset, \rightarrow, \mathrm{Prop}, \mathrm{Type} \in \Sigma$ plus one constant for each base type ($\mathcal{B} \subseteq \Sigma$).
- Set of terms T of the untyped λ-calculus with constants from Σ.
- Conventions:
  - $\varphi \supset \psi \equiv \supset \varphi \psi$,
  - $\forall x : \tau \,.\, \varphi \equiv \forall \tau (\lambda x \,.\, \varphi)$,
  - $\alpha \rightarrow \beta \equiv \rightarrow \alpha \beta$,
  - $\alpha : \beta \equiv \beta \alpha$.

Higher-order logic with recursion

Informal intuitive semantics

Higher-order logic with recursion

Informal intuitive semantics

- The universe contains everything: programs, formulas, truth values, types, kinds, numbers, data values, “undefined” values, programs mixed with formulas, . . .

Higher-order logic with recursion

Informal intuitive semantics

- The universe contains everything: programs, formulas, truth values, types, kinds, numbers, data values, “undefined” values, programs mixed with formulas, ...
- Terms denote elements of this mega-universe.

Higher-order logic with recursion

Informal intuitive semantics

- The universe contains everything: programs, formulas, truth values, types, kinds, numbers, data values, "undefined" values, programs mixed with formulas, ...
- Terms denote elements of this mega-universe.
- We do not know *a priori* which category a given term belongs to.

Higher-order logic with recursion

Informal intuitive semantics

- The universe contains everything: programs, formulas, truth values, types, kinds, numbers, data values, "undefined" values, programs mixed with formulas, . . .
- Terms denote elements of this mega-universe.
- We do not know *a priori* which category a given term belongs to.
- The inference rules allow us to find out.

Higher-order logic with recursion

Informal intuitive semantics

- $t : \mathrm{Prop}$ is true iff t is true or false,
- $\alpha : \mathrm{Type}$ is true iff α is a type,
- $t : \alpha$ is true iff t has type α, assuming α is a type,
- $\forall x : \alpha.\varphi$ is true iff α is a type and for all t of type α, $\varphi[x/t]$ is true,
- $\forall x : \alpha.\varphi$ is false iff α is a type and there exists t of type α such that $\varphi[x/t]$ is false,
- $t_1 \vee t_2$ is true iff t_1 is true or t_2 is true,
- $t_1 \vee t_2$ is false iff t_1 is false and t_2 is false,
- $t_1 \supset t_2$ is true iff t_1 is false or both t_1 and t_2 are true,
- $t_1 \supset t_2$ is false iff t_1 is true and t_2 is false,
- $\neg t$ is true iff t is false,
- $\neg t$ is false iff t is true.

Higher-order logic with recursion

Typing rules

$$\frac{}{\Delta \vdash \bot : \mathrm{Prop}} \qquad \frac{}{\Delta \vdash \mathrm{Prop} : \mathrm{Type}}$$

$$\frac{\alpha \in \mathcal{B}}{\Delta \vdash \alpha : \mathrm{Type}} \qquad \frac{\Delta \vdash \alpha : \mathrm{Type}}{\Delta \vdash (\forall \alpha) : (\alpha \to \mathrm{Prop}) \to \mathrm{Prop}}$$

$$\frac{\Delta \vdash \varphi : \mathrm{Prop} \qquad \Delta, \varphi \vdash \psi : \mathrm{Prop}}{\Delta \vdash (\varphi \supset \psi) : \mathrm{Prop}}$$

$$\frac{\Delta \vdash (\varphi \supset \psi) : \mathrm{Prop}}{\Delta \vdash \varphi : \mathrm{Prop}}$$

$$\frac{\Delta \vdash \varphi}{\Delta \vdash \varphi : \mathrm{Prop}}$$

Higher-order logic with recursion

Typing rules continued

$$\to_i\colon \frac{\Delta \vdash \alpha : \mathrm{Type} \qquad \Delta, x : \alpha \vdash t : \beta \qquad x \notin FV(\Delta, \alpha, \beta)}{\Delta \vdash (\lambda x\,.\,t) : \alpha \to \beta}$$

$$\to_e\colon \frac{\Delta \vdash t_1 : \alpha \to \beta \qquad \Delta \vdash t_2 : \alpha}{\Delta \vdash t_1 t_2 : \beta}$$

$$\to_t\colon \frac{\Delta \vdash \alpha : \mathrm{Type} \qquad \Delta \vdash \beta : \mathrm{Type}}{\Delta \vdash (\alpha \to \beta) : \mathrm{Type}}$$

Higher-order logic with recursion

Rules for logical connectives

$$\frac{}{\Delta, \varphi \vdash \varphi}$$

$$\supset_i : \frac{\Delta, \varphi \vdash \psi \qquad \Delta \vdash \varphi : \mathrm{Prop}}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e : \frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i : \frac{\Delta, x : \tau \vdash \varphi \qquad \Delta \vdash \tau : \mathrm{Type}}{\Delta \vdash \forall x : \tau \,.\, \varphi} \; x \notin FV(\Delta)$$

$$\forall_e : \frac{\Delta \vdash \forall x : \tau \,.\, \varphi \qquad \Delta \vdash t : \tau}{\Delta \vdash \varphi[x/t]}$$

$$\mathrm{conv} : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

Higher-order logic with recursion

Rules for logical connectives

$$\frac{}{\Delta, \varphi \vdash \varphi}$$

$$\supset_i: \frac{\Delta, \varphi \vdash \psi \qquad \Delta \vdash \varphi : \mathrm{Prop}}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e: \frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i : \frac{\Delta, x : \tau \vdash \varphi \qquad \Delta \vdash \tau : \mathrm{Type}}{\Delta \vdash \forall x : \tau \,.\, \varphi} \; x \notin FV(\Delta)$$

$$\forall_e : \frac{\Delta \vdash \forall x : \tau \,.\, \varphi \qquad \Delta \vdash t : \tau}{\Delta \vdash \varphi[x/t]}$$

$$\mathrm{conv} : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

This system is consistent.

Higher-order logic with recursion

This is (more or less) what Haskell Curry's school developed to save the program of (illative) combinatory logic after the discovery of paradoxes

Higher-order logic with recursion

This is (more or less) what Haskell Curry's school developed to save the program of (illative) combinatory logic after the discovery of paradoxes, but:

- until relatively recently there have been few consistency proofs,

Higher-order logic with recursion

This is (more or less) what Haskell Curry's school developed to save the program of (illative) combinatory logic after the discovery of paradoxes, but:

- until relatively recently there have been few consistency proofs,
  - till the 1990s no consistency proofs for systems strong enough to interpret traditional *first-order* logic,

Higher-order logic with recursion

This is (more or less) what Haskell Curry's school developed to save the program of (illative) combinatory logic after the discovery of paradoxes, but:

- until relatively recently there have been few consistency proofs,
  - till the 1990s no consistency proofs for systems strong enough to interpret traditional *first-order* logic,
- the known consistency proofs for strong systems (including the proof in the paper appendix) are quite complicated.

Higher-order logic with recursion

Rules for other connectives (actually not all derivable)

$$\exists_i : \frac{\Delta \vdash \alpha : \mathrm{Type} \qquad \Delta \vdash t : \alpha \qquad \Delta \vdash \varphi[x/t]}{\Delta \vdash \exists x : \alpha \,.\, \varphi}$$

$$\exists_e : \frac{\Delta \vdash \exists x : \alpha \,.\, \varphi \qquad \Delta, x : \alpha, \varphi \vdash \psi \qquad x \notin FV(\Delta, \psi, \alpha)}{\Delta \vdash \psi}$$

$$\vee_{i1} : \frac{\Delta \vdash \varphi}{\Delta \vdash \varphi \vee \psi} \qquad \vee_{i2} : \frac{\Delta \vdash \psi}{\Delta \vdash \varphi \vee \psi}$$

$$\vee_e : \frac{\Delta \vdash \varphi_1 \vee \varphi_2 \qquad \Delta, \varphi_1 \vdash \psi \qquad \Delta, \varphi_2 \vdash \psi}{\Delta \vdash \psi}$$

$$\vee_t : \frac{\Delta \vdash \varphi : \mathrm{Prop} \qquad \Delta \vdash \psi : \mathrm{Prop}}{\Delta \vdash (\varphi \vee \psi) : \mathrm{Prop}}$$

Higher-order logic with recursion

Rules for other connectives (actually not all derivable)

$$\wedge_{e1} : \frac{\Delta \vdash \varphi \wedge \psi}{\Delta \vdash \varphi} \qquad \wedge_{e2} : \frac{\Delta \vdash \varphi \wedge \psi}{\Delta \vdash \psi}$$

$$\wedge_i : \frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}$$

$$\perp_e : \frac{\Delta \vdash \perp}{\Delta \vdash \varphi}$$

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,
- $\lceil x \rceil = x$ if x is a variable,

Relation to ordinary logic
Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,
- $\lceil x \rceil = x$ if x is a variable,
- $\lceil t_1 t_2 \rceil = \lceil t_1 \rceil \lceil t_2 \rceil$,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,
- $\lceil x \rceil = x$ if x is a variable,
- $\lceil t_1 t_2 \rceil = \lceil t_1 \rceil \lceil t_2 \rceil$,
- $\lceil \lambda x : \tau \,.\, t \rceil = \lambda x \,.\, \lceil t \rceil$,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,
- $\lceil x \rceil = x$ if x is a variable,
- $\lceil t_1 t_2 \rceil = \lceil t_1 \rceil \lceil t_2 \rceil$,
- $\lceil \lambda x : \tau \,.\, t \rceil = \lambda x \,.\, \lceil t \rceil$,
- $\lceil \varphi \supset \psi \rceil = \lceil \varphi \rceil \supset \lceil \psi \rceil$,

Relation to ordinary logic

Translation

We define a translation $\lceil - \rceil$ from the language of ordinary higher-order logic to the language of the system above.

- $\lceil \tau \rceil = \tau$ if τ is a base type ($\tau \in \mathcal{B}$),
- $\lceil \mathrm{Prop} \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if c is a constant,
- $\lceil x \rceil = x$ if x is a variable,
- $\lceil t_1 t_2 \rceil = \lceil t_1 \rceil \lceil t_2 \rceil$,
- $\lceil \lambda x : \tau \,.\, t \rceil = \lambda x \,.\, \lceil t \rceil$,
- $\lceil \varphi \supset \psi \rceil = \lceil \varphi \rceil \supset \lceil \psi \rceil$,
- $\lceil \forall x : \tau \,.\, \varphi \rceil = \forall x : \lceil \tau \rceil \,.\, \lceil \varphi \rceil$.

Relation to ordinary logic

Context-providing mapping

A mapping Γ from sets of terms in ordinary higher-order logic to sets of terms in our system.

Relation to ordinary logic

Context-providing mapping

A mapping Γ from sets of terms in ordinary higher-order logic to sets of terms in our system.
For a set of terms Δ, the set $\Gamma(\Delta)$ consists of:

- $x : \lceil \tau \rceil$ for all types τ and all variables $x \in FV(\Delta)$ of type τ,

Relation to ordinary logic

Context-providing mapping

A mapping Γ from sets of terms in ordinary higher-order logic to sets of terms in our system.

For a set of terms Δ, the set $\Gamma(\Delta)$ consists of:

- $x : \lceil \tau \rceil$ for all types τ and all variables $x \in FV(\Delta)$ of type τ,
- $c : \lceil \tau \rceil$ for all types τ and all constants c of type τ,

Relation to ordinary logic

Context-providing mapping

A mapping Γ from sets of terms in ordinary higher-order logic to sets of terms in our system.

For a set of terms Δ, the set $\Gamma(\Delta)$ consists of:

- $x : \lceil \tau \rceil$ for all types τ and all variables $x \in FV(\Delta)$ of type τ,
- $c : \lceil \tau \rceil$ for all types τ and all constants c of type τ,
- $\tau : \mathrm{Type}$ for all $\tau \in \mathcal{B}$,

Relation to ordinary logic

Context-providing mapping

A mapping Γ from sets of terms in ordinary higher-order logic to sets of terms in our system.
For a set of terms Δ, the set $\Gamma(\Delta)$ consists of:

- $x : \lceil\tau\rceil$ for all types τ and all variables $x \in FV(\Delta)$ of type τ,
- $c : \lceil\tau\rceil$ for all types τ and all constants c of type τ,
- $\tau : \mathrm{Type}$ for all $\tau \in \mathcal{B}$,
- $y : \tau$ for all $\tau \in \mathcal{B}$ and some variable y of type τ such that $y \notin FV(\Delta)$.

Relation to ordinary logic

Soundness and completeness of the translation

Theorem (essentially known since about the 1970s)

If $\Delta \vdash_{\mathrm{PRED}\omega} \varphi$ *then* $\Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$.

Relation to ordinary logic

Theorem (essentially known since about the 1970s)

If $\Delta \vdash_{\mathrm{PRED}\omega} \varphi$ *then* $\Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$.

Conjecture

If $\Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$ *then* $\Delta \vdash_{\mathrm{PRED}\omega} \varphi$.

Relation to ordinary logic

Soundness and completeness of the translation

Theorem (essentially known since about the 1970s)

If $\Delta \vdash_{\mathrm{PRED}\omega} \varphi$ *then* $\Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$.

Conjecture

If $\Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$ *then* $\Delta \vdash_{\mathrm{PRED}\omega} \varphi$.

Theorem

$$\Delta \vdash_{\mathrm{FOL}} \varphi \quad \text{iff} \quad \Gamma(\Delta \cup \{\varphi\}), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$$

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop}\,.\,((p \supset \bot) \supset \bot) \supset p}$$

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop} \,.\, ((p \supset \bot) \supset \bot) \supset p}$$

- Choice.

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop}\,.\,((p \supset \bot) \supset \bot) \supset p}$$

- Choice.
- Extensionality wrt. Leibniz equality.

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop}\,.\,((p \supset \bot) \supset \bot) \supset p}$$

- Choice.
- Extensionality wrt. Leibniz equality.
- A conditional.

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop} \,.\, ((p \supset \bot) \supset \bot) \supset p}$$

- Choice.
- Extensionality wrt. Leibniz equality.
- A conditional.
- Predicate subtypes.

Other features

See the paper for details.

- Classical logic.

$$\frac{}{\Delta \vdash \forall p : \mathrm{Prop}\,.\,((p \supset \bot) \supset \bot) \supset p}$$

- Choice.
- Extensionality wrt. Leibniz equality.
- A conditional.
- Predicate subtypes.
- A class of **inductive types**.

Induction

We now have higher-order logic with recursion

Induction

We now have higher-order logic with recursion, but the crucial question is:

How much can we infer about terms whose type is not yet known?

Induction

We now have higher-order logic with recursion, but the crucial question is:

How much can we infer about terms whose type is not yet known?

- With the basic system the answer is: not much.

Induction

We now have higher-order logic with recursion, but the crucial question is:

How much can we infer about terms whose type is not yet known?

- With the basic system the answer is: not much.
- We need induction principles applicable to *arbitrary* terms.

Induction

We now have higher-order logic with recursion, but the crucial question is:

How much can we infer about terms whose type is not yet known?

- With the basic system the answer is: not much.
- We need induction principles applicable to *arbitrary* terms.
- Our system allows all inductive types without functional arguments, but here we concentrate solely on natural numbers.

Induction

For natural numbers

We need an induction principle **applicable to untyped terms**.

Induction

For natural numbers

We need an induction principle **applicable to untyped terms**.

- this won't do:

$$\forall f : \mathrm{Nat} \to \mathrm{Prop} \, . \, ((f 0 \wedge (\forall x : \mathrm{Nat} \, . \, fx \supset f(sx))) \supset \forall x : \mathrm{Nat} \, . \, fx)$$

Induction

For natural numbers

We need an induction principle **applicable to untyped terms**.

- this won't do:

$$\forall f : \mathrm{Nat} \to \mathrm{Prop}\,.\; ((f0 \wedge (\forall x : \mathrm{Nat}\,.\, fx \supset f(sx))) \supset \forall x : \mathrm{Nat}\,.\, fx)$$

- this will:

$$n_i : \frac{\Delta \vdash t0 \qquad \Delta, x : \mathrm{Nat}, tx \vdash t(sx) \qquad x \notin FV(\Delta, t)}{\Delta \vdash \forall x : \mathrm{Nat}\,.\, tx}$$

Inferring types by induction

Theorem (informal formulation)

Let f be a function for which there exists a measure μ (in natural numbers) on its arguments such that it can be proven that in a finite number of exhaustive cases μ decreases with each recursive call to f. If under the assumption that f has type β it can be proven that the body of f has type β, then f has type β.

Inferring types by induction

Theorem (informal formulation)

Let f be a function for which there exists a measure μ (in natural numbers) on its arguments such that it can be proven that in a finite number of exhaustive cases μ decreases with each recursive call to f. If under the assumption that f has type β it can be proven that the body of f has type β, then f has type β.

A precise statement of this theorem is somewhat lengthy and we omit it (see the paper for details).

Inferring types by induction

Theorem (informal formulation)

Let f be a function for which there exists a measure μ (in natural numbers) on its arguments such that it can be proven that in a finite number of exhaustive cases μ decreases with each recursive call to f. If under the assumption that f has type β it can be proven that the body of f has type β, then f has type β.

A precise statement of this theorem is somewhat lengthy and we omit it (see the paper for details).

Proof.

An easy application of the induction principle. $\square$

Conclusion

- The need to occasionally provide explicit typing derivations as additional premises of some inference rules is the only essential property of the system different from ordinary logic.

Conclusion

- The need to occasionally provide explicit typing derivations as additional premises of some inference rules is the only essential property of the system different from ordinary logic.
- In an implementation of our logic manual typing might be needed only when the function considered
  - is not simply typable,

Conclusion

- The need to occasionally provide explicit typing derivations as additional premises of some inference rules is the only essential property of the system different from ordinary logic.
- In an implementation of our logic manual typing might be needed only when the function considered
  - is not simply typable,
  - is not structurally recursive,

Conclusion

- The need to occasionally provide explicit typing derivations as additional premises of some inference rules is the only essential property of the system different from ordinary logic.
- In an implementation of our logic manual typing might be needed only when the function considered
  - is not simply typable,
  - is not structurally recursive,
  - we can't easily find a decreasing measure,

Conclusion

- The need to occasionally provide explicit typing derivations as additional premises of some inference rules is the only essential property of the system different from ordinary logic.
- In an implementation of our logic manual typing might be needed only when the function considered
  - is not simply typable,
  - is not structurally recursive,
  - we can't easily find a decreasing measure,
  - thus it cannot be (straightforwardly) defined in simple extensions of higher-order logic with terminating recursion.

Clarification

The theorems outlined above have constructive proofs (except for completeness of the translation of FOL). Therefore, there exists an algorithm which transforms appropriate annotations on types and/or measures into derivations in our logic.

Bibliography

Czajka, Ł.:
Partiality and recursion in higher-order logic.
W: Foundations of Software Science and Computation Structures, FOSSACS 2013, Proceedings. Volume 7794 of LNCS, Springer (2013) 177-192

Czajka, Ł.:
Higher-order illative combinatory logic.
Journal of Symbolic Logic (2013) Accepted.

Czajka, Ł.:
A semantic approach to illative combinatory logic.
W: Computer Science Logic, CSL 2011, Proceedings. Volume 12 of LIPIcs, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2011) 174–188