Type-theory of acyclic recursion and its calculi

Roussanka Loukanova

Stockholm University, Sweden

Second Workshop on Mathematical Logic and its Applications
5-9 March 2018, Kanazawa, Japan

Outline

Moschovakis Theory of Algorithms

- Moschovakis [1], 1989, initiated:
  untyped Theory of Algorithms: algorithms with full recursion
- Moschovakis [2], 2006, introduced a
  Type Theory of Acyclic Algorithms, by demonstrating it for Computational Semantics of Human Language
- Ongoing development:
  - Type Theory of Acyclic Algorithms, L^{λ}_{ar}:
    algorithms with acyclic recursion
  - Type Theory of Algorithms, L^{λ}_{r}:
    algorithms with full recursion
  - Dependent-Type Theory of Situated Information and Algorithms
- Applications to:
  - Computational Syntax-Semantics of Human Language
  - Computational Neuroscience (a little bit)

Algorithms, i.e., computations, as semantics of recursion terms

$$\underbrace{\text{Syntax of } \mathrm{L}_{\mathrm{ar}}^{\lambda}\ (\mathrm{L}_{\mathrm{r}}^{\lambda}) \Longrightarrow \text{Algorithms (for Computations)} \Longrightarrow \text{Denotations}}_{\text{Semantics of } \mathrm{L}_{\mathrm{ar}}^{\lambda}(\mathrm{L}_{\mathrm{r}}^{\lambda})}$$

- denotational semantics of $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ is by induction on term structure
- the reduction calculus of $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ $(\mathrm{L}_{\mathrm{r}}^{\lambda})$ defines a reduction relation:

$$A \Rightarrow B$$

- the reduction calculus of $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ effectively reduces every term to its canonical form:

$$A \Rightarrow_{\mathsf{cf}} \mathsf{cf}(A)$$

- the algorithm for computing $\mathsf{den}(A)$, for a meaningful term A, is determined by $\mathsf{cf}(A)$:

$$\mathsf{den}(A) = \mathsf{den}(\mathsf{cf}(A))$$

Gallin Types: $\sigma :\equiv \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (\tau_1 \to \tau_2)$
Constants: $\mathsf{Const}_\tau = \{\mathsf{c}_0^\tau, \mathsf{c}_1^\tau, \dots\}$
Variables: $\mathsf{PureVars}_\tau = \{v_0^\tau, v_1^\tau, \dots\}$, $\mathsf{RecVars}_\tau = \{p_0^\tau, p_1^\tau, \dots\}$

Terms of $\mathrm{L}_{ar}^{\lambda}$ (L_{r}^{λ}):

$$A :\equiv \mathsf{c}^\tau : \tau \mid x^\tau : \tau \tag{1a}$$
$$\mid B^{(\sigma\to\tau)}(C^\sigma) : \tau \tag{1b}$$
$$\mid \lambda(v^\sigma)(B^\tau) : (\sigma \to \tau) \tag{1c}$$
$$\mid A_0^\sigma \text{ where } \{p_1^{\sigma_1} := A_1^{\sigma_1}, \dots, p_n^{\sigma_n} := A_n^{\sigma_n}\} : \sigma \tag{1d}$$

given that:

- $\mathsf{c}^\tau \in \mathsf{Const}^\tau$, $x^\tau \in \mathsf{PureVars}^\tau \cup \mathsf{RecVars}^\tau$
- $p_i \in \mathsf{RecVars}^{\sigma_i}$, $A_i \in \mathsf{Terms}^{\sigma_i}$
- $\{p_1^{\sigma_1} := A_1^{\sigma_1}, \dots, p_n^{\sigma_n} := A_n^{\sigma_n}\}$ satisfies the Acyclicity Constraint iff:
  - there exists a function

$$\mathsf{rank}\colon \{p_1, \dots, p_n\} \to \mathbb{N}$$

s.th. if p_j occurs freely in A_i, then $\mathsf{rank}(p_i) > \mathsf{rank}(p_j)$

Abbreviations

- $er \equiv error$
- $\widetilde{\tau} \equiv (\mathsf{s} \to \tau)$, where $\tau \in$ Types (the type of state dependent objects of type σ)
- Sequences

$$\begin{aligned} \overrightarrow{X} &\equiv X_1, \ldots, X_n, \\ &\text{for } X_i \in \text{Terms for all } i \in \{1, \ldots, n\} \\ &\text{or } X_i \in \text{Types for all } i \in \{1, \ldots, n\} \end{aligned} \tag{2}$$

- Abbreviated sequences of mutually recursive assignments:

$$\overrightarrow{p} := \overrightarrow{A} \equiv \left[p_1 := A_1, \ \ldots, \ p_n := A_n\right] \quad (n \geq 0) \tag{3}$$

Denotational Semantics of L^{λ}_{ar}

For any *semantic structure* $\mathfrak{A}(\mathsf{Const}) = \langle \mathbb{T}, \mathcal{I} \rangle$, where

- $\mathbb{T} = \{\mathbb{T}_\sigma \mid \sigma \in \mathit{Types}\}$ is a frame of typed objects,
- $\mathcal{I}\colon\ \mathsf{Const} \longrightarrow \mathbb{T}$ is the *interpretation function*, respecting the types: for every $\sigma \in \mathit{Types}\}$, $\mathcal{I}(\mathsf{Const}_\sigma) = \mathbb{T}_\sigma$

with $G = \{g \mid g\colon\ \mathsf{PureVars} \cup \mathsf{RecVars} \longrightarrow \mathbb{T}\}$
(the set of all variable valuations),

the *denotation function*: $\mathsf{den}^{\mathfrak{A}} \equiv \mathsf{den}\colon\ \mathsf{Terms} \longrightarrow \{\, f \mid f\colon G \longrightarrow \mathbb{T}\,\}$
is defined by recursion on the structure of the terms:

(D1) $\mathsf{den}(x)(g) = g(x)$; $\mathsf{den}(c)(g) = \mathcal{I}(c)$

(D2) $\mathsf{den}(A(B))(g) = \mathsf{den}(A)(g)(\mathsf{den}(B)(g))$

(D3) $\mathsf{den}(\lambda x(B))(g)\big(t\big) = \mathsf{den}(B)(g\{x := t\})$, for every $t \in \mathbb{T}_\tau$

(to be continued)

The denotation function for the recursion terms (continuation)

(D4) $\text{den}(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})(g) =$
$\text{den}(A_0)(g\{p_1 := \overline{p}_1, \ldots, p_n := \overline{p}_n\})$,

where $\overline{p}_i \in \mathbb{T}_{\tau_i}$ are defined by recursion on $\text{rank}(p_i)$:

$$\overline{p_i} = \text{den}(A_i)(g\{p_{k_1} := \overline{p}_{k_1}, \ldots, p_{k_m} := \overline{p}_{k_m}\})$$

given that $p_{k_1}, \ldots, p_{k_m}$ are all of the recursion variables $p_j \in \{p_1, \ldots, p_n\}$, s.t. $\text{rank}(p_j) < \text{rank}(p_i)$.

Intuitively:

- $\text{den}(A_1)(g), \ldots, \text{den}(A_n)(g)$ are computed recursively and stored in $p_1, \ldots, p_n$, respectively
- the denotation $\text{den}(A_0)(g)$ may depend on the values stored in $p_1, \ldots, p_n$

[Congruence] If $A \equiv_{\mathsf{c}} B$, then $A \Rightarrow B$ (cong)

[Transitivity] If $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$ (trans)

[Compositionality]

- If $A \Rightarrow A'$ and $B \Rightarrow B'$, then $A(B) \Rightarrow A'(B')$ (ap-comp / rep1)
- If $A \Rightarrow B$, then $\lambda(u)(A) \Rightarrow \lambda(u)(B)$ (λ-comp / rep2)
- If $A_i \Rightarrow B_i$ $(i = 0, \ldots, n)$, then

$$A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \} \Rightarrow B_0 \text{ where } \{ p_1 := B_1, \ldots, p_n := B_n \}$$ (wh-comp / rep3)

Reduction Rules (to be continued)

[Head Rule] given that no p_i occurs freely in any B_j,

$$\big(A_0 \text{ where } \{ \overrightarrow{p} := \overrightarrow{A} \}\big) \text{ where } \{ \overrightarrow{q} := \overrightarrow{B} \} \Rightarrow A_0 \text{ where } \{ \overrightarrow{p} := \overrightarrow{A},\ \overrightarrow{q} := \overrightarrow{B} \} \tag{head}$$

[Bekič-Scott Rule] given that no q_i occurs freely in any A_j,

$$A_0 \text{ where } \{ p := \big(B_0 \text{ where } \{ \overrightarrow{q} := \overrightarrow{B} \}\big),\ \overrightarrow{p} := \overrightarrow{A} \} \Rightarrow A_0 \text{ where } \{ p := B_0, \overrightarrow{q} := \overrightarrow{B},\ \overrightarrow{p} := \overrightarrow{A} \} \tag{B-S}$$

[Recursion-Application Rule] given that no p_i occurs freely in B,

$$\big(A_0 \text{ where } \{ \overrightarrow{p} := \overrightarrow{A} \}\big)(B) \tag{7}$$

$$\Rightarrow A_0(B) \text{ where } \{ \overrightarrow{p} := \overrightarrow{A} \} \tag{recap}$$

Reduction Rules (to be continued)

[Application Rule] given that $B \in$ PrT is a proper term, and fresh $p \in \big[\,\text{RecVars} - \big(\,\text{FV}\,(A(B)) \cup \text{BV}\,(A(B))\big)\big]$,

$$A(B) \Rightarrow \Big[A(p) \text{ where } \{\, p := B \,\}\Big] \qquad \text{(ap)}$$

[λ-rule] given fresh $p_i' \in \big[\,\text{RecVars} - \big(\,\text{FV}(A) \cup \text{BV}(A)\big)\big]$, $i = 1, \dots, n$, for $A \equiv A_0$ where $\{\, p_1 := A_1, \dots, p_n := A_n \,\}$

$$\lambda(u)\Big(A_0 \text{ where } \big\{\, p_1 := A_1, \dots, p_n := A_n \,\big\}\Big) \qquad (\lambda)$$
$$\Rightarrow \Big[\,\lambda(u)A_0' \text{ where } \big\{\, p_1' := \lambda(u)A_1', \dots, p_n' := \lambda(u)A_n' \,\big\}\Big]$$

where, for all $i = 0, \dots, n$,

$$A_i' \equiv \Big[A_i\big\{\, p_1 :\equiv p_1'(u), \dots, p_n :\equiv p_n'(u) \,\big\}\Big] \qquad (9)$$

Theorem (Canonical Form Theorem)

For each $A \in$ Terms, *there is a unique up to congruence, irreducible* $\mathsf{cf}(A) \in$ Terms *s.th.:*

1. $\mathsf{cf}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$
   for some explicit, irreducible $A_0, \ldots, A_n \in$ Terms $(n \geq 0)$
2. $A \Rightarrow \mathsf{cf}(A)$

Algorithmic Equivalence

Intuitively: L_r^{λ} is a formalization of the mathematical notion of algorithm, for computing values of recursive functions, designated by recursion terms and expressed by terms in canonical forms.
I.e., the concept of algorithm is defined formally, at the object level of its syntax.

Theorem (of Algorithmic Equivalence / Synonymy)

Two terms $A, B \in$ Terms *are* *algorithmically equivalent*, $A \approx B$, *iff there are explicit, irreducible terms* $A_0, A_1, \ldots, A_n$, $B_0, B_1, \ldots, B_n$ $(n \geq 0)$ *s.th.:*

- $A \Rightarrow_{\mathsf{cf}} A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$
- $B \Rightarrow_{\mathsf{cf}} B_0$ where $\{p_1 := B_1, \ldots, p_n := B_n\}$
- $\models A_i = B_i \quad (i = 0, \ldots, n)$, *i.e.,*

$$\mathsf{den}(A_i)(g) = \mathsf{den}(B_i)(g), \quad \textit{for all } g \in G \tag{10}$$

A simple math example: pattern occurring in quantifiers of Human Language

$x \in \mathsf{PureVars}_{\sigma}$, $b \in \mathsf{Const}_{\tau_1}$, $p \in \mathsf{RecVars}_{\tau_1}$, $p' \in \mathsf{RecVars}_{(\sigma \to \tau_1)}$,
$f : (\tau_1 \to \tau_2)$, an explicit and irreducible term:

$$A^{(\sigma\to\tau_2)} \equiv \lambda(x)\big[f(b)\big] \tag{11a}$$

$$\Rightarrow \lambda(x)\Big[f(p) \text{ where } \{ p := b \}\Big] \quad \text{by (ap)} \tag{11b}$$

$$\Rightarrow_{\mathsf{cf}} \lambda(x) f(p'(x)) \text{ where } \{ p' := \lambda(x) b \} \quad \text{by } (\lambda) \tag{11c}$$

$$\not\approx \lambda(x) f(p) \text{ where } \{ p := b \} \tag{11d}$$

$$\equiv B^{(\sigma\to\tau_2)} \tag{11e}$$

Since $\lambda(x)b : (\sigma \to \tau_1)$ and $b : \tau_1$ $\quad \therefore$ $\mathsf{den}(\lambda(x)b) \neq \mathsf{den}(b)$
$\therefore \; A \not\approx B$
However: $A \approx_{\gamma} B \approx_{\gamma^*} B$

$$A \Rightarrow_{\mathsf{cf}} \lambda(x) f(p'(x)) \text{ where } \{ p' := \lambda(x) b \} \tag{12a}$$

$$\Rightarrow_{(\gamma)} \lambda(x) f(p) \text{ where } \{ p := b \} \tag{12b}$$

$$\equiv B \equiv \mathsf{cf}_{\gamma}(A) \approx_{\gamma} A \tag{12c}$$

Definition (γ-condition)

A term $A \in$ Terms satisfies the γ-condition for an assignment $p := \lambda(v)P$ iff A is of the form:

$$A \equiv A_0 \text{ where } \{ \overrightarrow{a} := \overrightarrow{A},\ p := \lambda(v)P,\ \overrightarrow{b} := \overrightarrow{B} \} \tag{13}$$

such that

1. $v \notin \mathrm{FreeV}(P)$
2. All occurrences of p in A_0, $\overrightarrow{A}$ and $\overrightarrow{B}$ are occurrences in a sub-term $p(v)$ that are in the scope of $\lambda(v)$ (modulo congruence with respect to renaming the scope variable v).

(γ)-rule

$$A \equiv \quad A_0 \text{ where } \{ \overrightarrow{a} := \overrightarrow{A},\ p := \lambda(v)P,\ \overrightarrow{b} := \overrightarrow{B} \} \tag{14a}$$

$$\Rightarrow_{(\gamma)} A_0' \text{ where } \{ \overrightarrow{a} := \overrightarrow{A'},\ p' := P,\ \overrightarrow{b} := \overrightarrow{B'} \} \tag{14b}$$

given that:

- the term $A \in$ Terms satisfies the γ-condition (in Definition 3) for the assignment $p := \lambda(v)P$
- $p' \in \mathrm{RV}_\tau$ is a fresh recursion variable
- $\overrightarrow{X'} \equiv \overrightarrow{X}\{p(v) :\equiv p'\}$ is the result of the replacements

  $X_i\{p(v) :\equiv p'\}$
  i.e., replacing all occurrences of $p(v)$ by p', in all parts X_i (for $X_i \equiv A_i$, $X_i \equiv B_i$) in (14a)–(14b), modulo congruence with respect to renaming the scope variable v ($i \in \{0, \ldots, n_X\}$)

A Pattern Term for Binary, Generalised Quantifiers (e.g., in human language)

- $D, K \in$ Const, $\;H \in$ Terms, $\;D, K : \widetilde{\mathsf{e}}$, $\;H : (\widetilde{\mathsf{e}} \rightarrow (\widetilde{\mathsf{e}} \rightarrow \widetilde{\mathsf{t}}))$

$$A \equiv \Big[\lambda y \big[\big[some(D)\big]\big(\lambda x_d\, H(x_d)(y)\big)\big]\Big](K) \tag{15a}$$

$$\Rightarrow \ldots \tag{15b}$$

$$\begin{aligned}\Rightarrow \Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big) \text{ where}\\ \{\, d' := \lambda(y_k)D,\\ h := \lambda(y_k)\,\lambda(x_d)H(x_d)(y_k)\,\}\Big](K)\end{aligned} \tag{15c}$$

$$\Rightarrow_{\mathsf{cf}}\ \mathsf{cf}(A) \equiv \tag{15d}$$

$$\begin{aligned}\Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where}\\ \{\, h := \lambda(y_k)\,\lambda(x_d)H(x_d)(y_k),\\ d' := \lambda(y_k)D,\ k := K\,\}\end{aligned} \tag{15e}$$

$$\begin{aligned}\Rightarrow_{(\gamma)} \big[\lambda(y_k) some(d)\big(h(y_k)\big)\big](k) \text{ where}\\ \{\, h := \lambda(y_k)\,\lambda(x_d)H(x_d)(y_k),\\ d := D,\ k := K\,\}\end{aligned} \tag{15f}$$

Adding the γ-rule to the set of reduction rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ yields:

- non-deterministic reductions sequences: $A \Rightarrow_{(\gamma)} B$
- some terms can be reduced to different γ-irreducible terms that are not algorithmically equivalent

$$A \Rightarrow_{(\gamma)} B_1,\ A \Rightarrow_{(\gamma)} B_2,\ B_1 \not\approx B_2,\ \ B_1,\ B_2 \text{ are } \gamma\text{-irreducible} \quad (16)$$

A solution:
The rules are applied to the innermost, reducible sub-terms, i.e., from inside-out: $A \stackrel{\text{in}}{\Rightarrow}_{(\gamma)} B$

Theorem (γ-Canonical Form Theorem: for innermost γ-reductions)

For each $A \in$ Terms, *there is a unique up to congruence,* γ*-irreducible* $\mathrm{cf}_{\gamma}(A) \in$ Terms *s.th.:*

1. $\mathrm{cf}_{\gamma}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$ *for some explicit,* γ*-irreducible terms* $A_0, \ldots, A_n \in$ Terms $(n \geq 0)$
2. $A \stackrel{\text{in}}{\Rightarrow}_{(\gamma)} \mathrm{cf}_{\gamma}(A)$

γ^*-Reduction — stronger reduction

Definition ($\gamma*$-condition)

A term $A \in$ Terms satisfies the γ^*-condition for an assignment $p := \lambda(\overrightarrow{u}^{\overrightarrow{\sigma}})\,\lambda(v^{\sigma})P^{\tau} : (\overrightarrow{\sigma} \to (\sigma \to \tau))$, with respect to $\lambda(v^{\sigma})$, iff A is of the form: (17a)–(17c):

$$A \equiv A_0 \text{ where } \{\, \overrightarrow{a} := \overrightarrow{A}, \tag{17a}$$

$$p := \lambda(\overrightarrow{u})\,\lambda(v)P, \tag{17b}$$

$$\overrightarrow{b} := \overrightarrow{B}\,\} \tag{17c}$$

such that the following holds:

1. $v \notin \mathsf{FreeV}(P)$
2. All occurrences of p in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are occurrences in $p(\overrightarrow{u})(v)$, modulo renaming the variables $\overrightarrow{u}, v$

(γ^*)-rule

$$
\begin{aligned}
A \equiv A_0 \text{ where } \{\, \overrightarrow{a} &:= \overrightarrow{A}, && \text{(18a)}\\
p &:= \lambda(\overrightarrow{u})\, \lambda(v) P, && \text{(18b)}\\
\overrightarrow{b} &:= \overrightarrow{B} \,\} && \text{(18c)}\\
\Rightarrow_{(\gamma^*)} A_0' \text{ where } \{\, \overrightarrow{a} &:= \overrightarrow{A}', && \text{(18d)}\\
p' &:= \lambda(\overrightarrow{u}) P', && \text{(18e)}\\
\overrightarrow{b} &:= \overrightarrow{B'} \,\} && \text{(18f)}
\end{aligned}
$$

given that:

- $A \in$ Terms satisfies the γ^*-condition (in Definition 5) for $p := \lambda(\overrightarrow{u})\, \lambda(v) P : (\overrightarrow{\sigma} \to (\sigma \to \tau))$, with respect to $\lambda(v)$
- $p' \in \mathsf{RecVars}_{(\overrightarrow{\sigma} \to \tau)}$ is a fresh recursion variable
- $\overrightarrow{X'} \equiv \overrightarrow{X}\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$ is the result of the replacements

  $X_i\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$,
  i.e., replacing all occurrences of $p(\overrightarrow{u})(v)$ by $p'(\overrightarrow{u})$, in all corresponding parts $X_i \equiv A_i$, $X_i \equiv B_i$, in (18a)–(18f), modulo renaming the variables $\overrightarrow{u}, v$

Theorem (γ^*-Canonical Form Theorem)

For each $A \in$ Terms, *there is a unique up to congruence,* γ^**-irreducible* $\mathsf{cf}_{\gamma^*}(A) \in$ Terms, *s.th.:*

1. $\mathsf{cf}_{\gamma^*}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$
   for some explicit, γ^**-irreducible* $A_0, \ldots, A_n \in$ Terms *(*$n \geq 0$*)*
2. $A \Rightarrow_{(\gamma^*)} \mathsf{cf}_{\gamma^*}(A)$

γ^*-Algorithmic Equivalence

Theorem (γ^*-Algorithmic Equivalence)

Two terms $A, B \in$ Terms *are* *γ^*-algorithmically equivalent*, $A \approx_{\gamma^*} B$, *iff there are explicit, γ^*-irreducible terms* $A_0, A_1, \ldots, A_n$, $B_0, B_1, \ldots, B_n$ *($n \geq 0$), s.th.:*

- $A \Rightarrow_{\mathsf{cf}_{\gamma^*}} A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\} \equiv \mathsf{cf}_{\gamma^*}(A)$
- $B \Rightarrow_{\mathsf{cf}_{\gamma^*}} B_0$ where $\{p_1 := B_1, \ldots, p_n := B_n\} \equiv \mathsf{cf}_{\gamma^*}(B)$
- $\models A_i = B_i \ \ (i = 0, \ldots, n)$, *i.e.,*

$$\mathrm{den}(A_i)(g) = \mathrm{den}(B_i)(g), \quad \textit{for all } g \in G \tag{19}$$

Theorem

For any $A, B \in$ Terms,

$$A \Rightarrow B \implies A \approx_s B \tag{20a}$$

$$\implies A \approx B \tag{20b}$$

$$\implies A \approx_{\gamma^*} B \implies A \models\!\dashv B \tag{20c}$$

Theorem

1. *For all* $A, B \in$ Terms

$$A \approx_{\gamma} B \Longrightarrow A \approx_{\gamma^*} B \tag{21}$$

The proofs are (longish) by using induction on term structure, definitions, and theorems of algorithmic equivalence.

Theorem

1. *There exist* $A, B \in$ Terms, *such that* $A \approx_{\gamma} B$, $A \not\approx B$. *I.e., in general:*

$$A \approx_{\gamma} B \not\Rightarrow A \approx B \tag{22}$$

2. *There exist* $A, B \in$ Terms, *such that* $A \approx_{\gamma^*} B$, $A \not\approx B$. *I.e., in general:*

$$A \approx_{\gamma^*} B \not\Rightarrow A \approx B \tag{23}$$

3. *There exist* $A, B \in$ Terms, *such that* $A \approx_{\gamma^*} B$, $A \not\approx_{\gamma} B$. *I.e., in general:*

$$A \approx_{\gamma^*} B \not\Rightarrow A \approx_{\gamma} B \tag{24}$$

$$A \equiv \lambda(x_1)\,\lambda(x_2)\Big[f(p) \text{ where } \{\, p := [P(q)](x_2),\quad q := Q(x_1)\}\Big] \tag{25a}$$

$$\overset{\text{in}}{\Rightarrow}_{(\lambda)} \lambda(x_1)\Big[\lambda(x_2)f(p^1(x_2)) \text{ where } \{\; p^1 := \lambda(x_2)\Big[\big[P(q^1(x_2))\big](x_2)\Big],\quad q^1 := \lambda(x_2)Q(x_1)\}\Big] \tag{25b}$$

$$\overset{\text{in}}{\Rightarrow}_{(\gamma)} \lambda(x_1)\Big[\lambda(x_2)f(p^1(x_2)) \text{ where } \{\; p^1 := \lambda(x_2)\Big[\big[P(q^1(x_2))\big](x_2)\Big],\quad q^1_1 := Q(x_1)\}\Big] \tag{25c}$$

$$\overset{\text{in}}{\Rightarrow}_{(\lambda)} \lambda(x_1)\,\lambda(x_2)f(p^1_1(x_1)(x_2)) \text{ where } \{\; p^2_1 := \lambda(x_1)\,\lambda(x_2)\Big[\big[P(q^2_1(x_1))\big](x_2)\Big],\quad q^2_1 := \lambda(x_1)Q(x_1)\} \equiv \mathsf{cf}_\gamma(A) \equiv B \tag{25d}$$

$$\therefore\;\; A \not\approx B,\;\; A \approx_\gamma B \equiv \mathsf{cf}_\gamma(A)$$

$$
\begin{aligned}
A \equiv\ & \lambda(x_1)\,\lambda(x_2)\Big[f(p) \text{ where } \{\, p := [P(q)](x_2), \\
& \qquad\qquad\qquad\qquad\qquad q := Q(x_1)\}\Big]
\end{aligned} \tag{26a}
$$

$$
\begin{aligned}
\Rightarrow_{(\lambda)}\ & \lambda(x_1)\Big[\, \lambda(x_2) f(p^1(x_2)) \text{ where } \{ \\
& \qquad\qquad p^1 := \lambda(x_2)\Big[\big[P(q^1(x_2))\big](x_2)\Big], \\
& \qquad\qquad q^1 := \lambda(x_2) Q(x_1)\}\Big]
\end{aligned} \tag{26b}
$$

$$
\begin{aligned}
\Rightarrow_{(\lambda)}\ & \lambda(x_1)\,\lambda(x_2) f(p^1(x_1)(x_2)) \text{ where } \{ \\
& \qquad\qquad p^2 := \lambda(x_1)\,\lambda(x_2)\Big[\big[P(q^2(x_1)(x_2))\big](x_2)\Big], \\
& \qquad\qquad q^2 := \lambda(x_1)\,\lambda(x_2) Q(x_1)\} \equiv \mathsf{cf}(A) \not\equiv \mathsf{cf}_{\gamma}(A)
\end{aligned} \tag{26c}
$$

$$
\begin{aligned}
\Rightarrow_{(\gamma^*)}\ & \lambda(x_1)\,\lambda(x_2) f(p^1_1(x_1)(x_2)) \text{ where } \{ \\
& \qquad\qquad p^2_1 := \lambda(x_1)\,\lambda(x_2)\Big[\big[P(q^2_1(x_1))\big](x_2)\Big], \\
& \qquad\qquad q^2_1 := \lambda(x_1) Q(x_1)\}
\end{aligned} \tag{26d}
$$

$$
\equiv \mathsf{cf}_{\gamma^*}(A) \equiv \mathsf{cf}_{\gamma}(A) \equiv B \tag{26e}
$$

$$
\therefore\ \ A \not\approx B,\ \ A \approx_{\gamma^*} B
$$

$$\text{Kim hugs some dog} \tag{27a}$$

$$\xrightarrow{\text{render}} A \equiv \Big[\lambda y \big[\big[some(dog)\big]\big(\lambda x_d\, hugs(x_d)(y)\big)\big]\Big](kim) \tag{27b}$$

$$\Rightarrow \ldots \tag{27c}$$

$$\begin{aligned}\Rightarrow \Big[\,\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big) \text{ where}\\ \{\, d' := \lambda(y_k)\,dog,\\ h := \lambda(y_k)\,\lambda(x_d) hugs(x_d)(y_k)\,\}\Big](kim)\end{aligned} \tag{27d}$$

$$\Rightarrow_{\mathsf{cf}}\ \mathsf{cf}(A) \equiv \tag{27e}$$

$$\begin{aligned}\Big[\,\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where}\\ \{\, h := \lambda(y_k)\,\lambda(x_d) hugs(x_d)(y_k),\\ d' := \lambda(y_k) dog,\ k := kim\,\}\end{aligned} \tag{27f}$$

$$\begin{aligned}\Rightarrow_{(\gamma)}\ \big[\,\lambda(y_k) some(d)\big(h(y_k)\big)\big](k) \text{ where}\\ \{\, h := \lambda(y_k)\,\lambda(x_d) hugs(x_d)(y_k),\\ d := dog,\ k := kim\,\}\end{aligned} \tag{27g}$$

Some Current Tasks (among many others) and Future Work

- My current focus is on
  - development of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$
  - applications to computational semantics and computational syntax-semantics interface of natural language
  - applications to computational neuroscience
- Longer-term work
  - Type Theory of Algorithms, $\mathrm{L}^{\lambda}_{\mathrm{r}}$:
    algorithms with full recursion
  - Dependent-Type Theory of Situated Information and Algorithms:
    - dependent types
    - situated, partial, and parametric components

Some References I

Yiannis N Moschovakis.
The formal language of recursion.
The Journal of Symbolic Logic, 54(04):1216–1252, 1989.

Yiannis N. Moschovakis.
A logical calculus of meaning and synonymy.
Linguistics and Philosophy, 29(1):27–89, Feb 2006.
URL: http://dx.doi.org/10.1007/s10988-005-6920-7,
doi:10.1007/s10988-005-6920-7.