Relatons between let-Terms of Lambda-Calculus and where-Terms of Type-Theory of Recursion

Roussanka Loukanova

Institute of Mathematics and Informatics (IMI)
Bulgarian Academy of Sciences (BAS), Bulgaria

WG6 meeting in Leuven in April 2024

4-5 April 2024

https://europroofnet.github.io/wg6-leuven/
https://europroofnet.github.io/wg6-leuven/programme/
https://europroofnet.github.io/wg6-leuven/programme/#loukanova

Algorithmic CompSynSem Interfaces within L^{λ}_{ar} / L^{λ}_{r} Moschovakis [8]

$$\underbrace{\text{Syn of } L^{\lambda}_{ar} / L^{\lambda}_{r} \iff \underbrace{\text{Canonical Terms} \Longrightarrow \underbrace{\text{Denotations}}_{\text{Denotational Semantics}}}_{\text{Algorithmic Semantics}}}_{\text{Algorithmic CompSynSem Interfaces}}$$

- Denotational Semantics of L^{λ}_{ar} / L^{λ}_{r}: by induction on terms
- Reduction Calculus of L^{λ}_{ar} / L^{λ}_{r}: defined by (10+3+n) red. rules

$$A \Rightarrow B \qquad \text{(10 by Moschovakis; 3+n by Loukanova)} \tag{1}$$

- The reduction calculus of L^{λ}_{ar} / L^{λ}_{r} is effective (by a theorem):
  For every $A \in$ Terms, there is unique, up to congruence, canonical form cf(A), s.th.:

$$A \Rightarrow_{\text{cf}} \text{cf}(A) \tag{2}$$

- Algorithmic Semantics of L^{λ}_{ar} / L^{λ}_{r}
  For every algorithmically meaningful $A \in$ Terms:
  - cf(A) determines the algorithm alg(A) for computing den(A)

Syntax of Type Theory of Algorithms (TTA): Types, Vocabulary

- Gallin Types (1975)

$$\tau ::= \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (\tau \to \tau) \qquad \text{(Types)}$$

- Abbreviations

$$\widetilde{\sigma} \equiv (\mathsf{s} \to \sigma), \quad \text{for state-dependent objects of type } \widetilde{\sigma} \qquad \text{(3a)}$$

$$\widetilde{\mathsf{e}} \equiv (\mathsf{s} \to \mathsf{e}), \quad \text{for state-dependent entities} \qquad \text{(3b)}$$

$$\widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t}), \quad \text{for state-dependent truth values} \qquad \text{(3c)}$$

- Typed Vocabulary, for all $\sigma \in$ Types

$$K_\sigma = \mathsf{Consts}_\sigma = \{\mathsf{c}_0^\sigma, \mathsf{c}_1^\sigma, \dots\} \qquad \text{(4a)}$$

$$\wedge, \vee, \to \ \in \mathsf{Consts}_{(\tau \to (\tau \to \tau))}, \ \tau \in \{\mathsf{t}, \widetilde{\mathsf{t}}\} \quad \text{(logical constants)} \quad \text{(4b)}$$

$$\neg \in \mathsf{Consts}_{(\tau \to \tau)}, \ \tau \in \{\mathsf{t}, \widetilde{\mathsf{t}}\} \quad \text{(logical constant for negation)} \quad \text{(4c)}$$

$$\mathsf{PureV}_\sigma = \{v_0^\sigma, v_1^\sigma, \dots\} \qquad \text{(4d)}$$

$$\mathsf{RecV}_\sigma = \mathsf{MemoryV}_\sigma = \{p_0^\sigma, p_1^\sigma, \dots\} \qquad \text{(4e)}$$

$$\mathsf{PureV}_\sigma \cap \mathsf{RecV}_\sigma = \varnothing, \qquad \mathsf{Vars}_\sigma = \mathsf{PureV}_\sigma \cup \mathsf{RecV}_\sigma \qquad \text{(4f)}$$

Terms of Type Theory of Algorithms (TTA): L_{ar}^{λ} acyclic recursion (L_{r}^{λ} full recursion)

$$\mathsf{A} :\equiv \mathsf{c}^{\sigma} : \sigma \mid X^{\sigma} : \sigma \mid \mathsf{B}^{(\rho\to\sigma)}(\mathsf{C}^{\rho}) : \sigma \mid \lambda(v^{\rho})\,(\mathsf{B}^{\sigma}) : (\rho\to\sigma) \quad (5a)$$

$$\mid \mathsf{A}_0^{\sigma_0} \text{ where } \{\, p_1^{\sigma_1} := \mathsf{A}_1^{\sigma_1}, \ldots, \ldots, p_n^{\sigma_n} := \mathsf{A}_n^{\sigma_n} \,\} : \sigma_0 \quad (5b)$$

$$\mid \wedge\,(A_2^{\tau})(A_1^{\tau}) : \tau \mid \vee\,(A_2^{\tau})(A_1^{\tau}) : \tau \mid \to (A_2^{\tau})(A_1^{\tau}) : \tau \quad (5c)$$

$$\mid \neg(B^{\tau}) : \tau \quad (5d)$$

$$\mid \forall(v^{\sigma})(B^{\tau}) : \tau \mid \exists(v^{\sigma})(B^{\tau}) : \tau \qquad \text{(pure quantifiers)} \quad (5e)$$

$$\mid \mathsf{A}_0^{\sigma_0} \text{ such that } \{\, \mathsf{C}_1^{\tau_1}, \ldots, \mathsf{C}_m^{\tau_m} \,\} : \sigma_0' \quad (5f)$$

- $\mathsf{c}^{\tau} \in \mathsf{Consts}_{\tau}$, $X^{\tau} \in \mathsf{PureV}_{\tau} \cup \mathsf{RecV}_{\tau}$, $v^{\sigma} \in \mathsf{PureV}_{\sigma}$
- $\mathsf{B}, \mathsf{C} \in \mathsf{Terms}$, $p_i^{\sigma_i} \in \mathsf{RecV}_{\sigma_i}$, $A_i^{\sigma_i} \in \mathsf{Terms}_{\sigma_i}$, $\mathsf{C}_j^{\tau_j} \in \mathsf{Terms}_{\tau_j}$
- In (5c)–(5e), (5f): $\tau, \tau_j \in \{\, \mathsf{t}, \widetilde{\mathsf{t}} \,\}$, $\widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t})$ (for propositions)
- Acyclicity Constraint (AC), for L_{ar}^{λ}; without it, L_{r}^{λ} with full recursion

$$\{\, p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n} \,\} \quad (n \geq 0) \text{ is acyclic iff} \quad (6a)$$

$$\text{for some rank} : \{p_1, \ldots, p_n\} \to \mathbb{N} \quad (6b)$$

$$\text{if } p_j \text{ occurs freely in } A_i, \text{ then } \mathrm{rank}(p_i) > \mathrm{rank}(p_j) \quad (6c)$$

Types of Restrictor Terms

In the restrictor term (5f) / (7),

$$A_0^{\sigma_0} \text{ such that } \{ C_1^{\tau_1}, \ldots, C_n^{\tau_n} \} : \sigma_0' \tag{7}$$

for each $i = 1, \ldots, n$:

- $\tau_i \equiv \mathsf{t}$ (state independent truth values), or
- $\tau_i \equiv \widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t})$ (state dependent truth values)

$$\sigma_0' \equiv \begin{cases} \sigma_0, & \text{if } \tau_i \equiv \mathsf{t}, \text{ for all } i \in \{1, \ldots, n\} & \text{(8a)} \\ \sigma_0 \equiv (\mathsf{s} \to \sigma), & \text{if } \tau_i \equiv \widetilde{\mathsf{t}}, \text{ for some } i \in \{1, \ldots, n\}, \text{ and} & \text{(8b)} \\ & \quad \text{for some } \sigma \in \text{Types}, \ \sigma_0 \equiv (\mathsf{s} \to \sigma) & \\ \widetilde{\sigma_0} \equiv (\mathsf{s} \to \sigma_0), & \text{if } \tau_i \equiv \widetilde{\mathsf{t}}, \text{ for some } i \in \{1, \ldots, n\}, \text{ and} & \text{(8c)} \\ & \quad \text{there is no } \sigma, \text{ s.th. } \sigma_0 \equiv (\mathsf{s} \to \sigma) & \end{cases}$$

Definition (Explicit and λ-*Calculus* Terms)

- $A \in$ Terms is explicit iff the constant where designating the recursion operator does not occur in A (cf(A) can be where-term)
- $A \in$ Terms is a *λ-calculus term* iff
  it is explicit and no recursion variable occurs in it

Definition (Immediate and Proper Terms)

- The set ImT of immediate terms is defined by recursion (9)

$$T :\equiv V \mid p(v_1)\dots(v_m) \mid \lambda(u_1)\dots\lambda(u_n)p(v_1)\dots(v_m) \tag{9}$$

  for $V \in$ Vars, $p \in$ RecV, $u_i, v_j, \in$ PureV,
  $i = 1, \dots, n,\ j = 1, \dots, m,\ (m, n \geq 0)$
- Every $A \in$ Terms that is not immediate is proper:

$$\mathrm{PrT} = (\mathrm{Terms} - \mathrm{ImT}) \tag{10}$$

Immediate terms do not carry algorithmic sense.

Development of Scott let-Expressions by where-Recursion Terms: Key Factors

- Dana S. Scott [12] introduced the let-expressions by the
- Gordon Plotkin [9] further formalized LCF

Algorithmic Generalization of Scott let-Expressions by Moschovakis where-Recursion Terms

Algorithmic Syntax-Semantics Interfaces of L^{λ}_{ar} / L^{λ}_{r} provide algorithmic generalization of the Scott let-expressions to where-recursion terms.

The algorithmic semantics by L^{λ}_{ar} / L^{λ}_{r} is provided by:

1. Reduction calculus of L^{λ}_{ar} / L^{λ}_{r} of (10+) reduction rules, based on:
2. Division of the variables into two kinds:

$$\mathsf{PureV}_{\sigma} \quad \text{(pure vars for } \lambda\text{-abstraction and quantifiers)} \tag{11a}$$

$$\mathsf{RecV}_{\sigma} \quad \text{(recursion vars for assignments in recursion terms)} \tag{11b}$$

3. Division of the terms into immediate ImT and proper PrT terms: $\mathsf{PrT} = (\mathsf{Terms} - \mathsf{ImT})$
4. Reductions to canonical forms $A \Rightarrow_{\mathsf{cf}} \mathsf{cf}(A)$: $\mathsf{cf}(A)$ determines $\mathsf{alg}(A)$, for the algorithmically meaningful $A \in \mathsf{PrT}$

Scott let-Expressions and where-Recursion Terms

- Assume $A \in$ Terms is of the form (12a)–(12b)

$$A \equiv \mathsf{cf}_{\gamma^*}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\} \tag{12a}$$

$$\mathsf{rank}(p_i) = i, \text{ for } i \in \{1, \dots, n\} \tag{12b}$$

- The λ-abstraction (13b) is characteristic for the let-expression (13a)
- λ-abstraction is not possible directly over $p_i \in$ RecV, in (12a)–(12b)

- In let-expressions (13a), $x_i \in \mathsf{PureV}_{\tau_i}$, for the λ-abstraction (13b)
- The replacements (13c) handle the mismatch pure vars for λ-abstraction vs. recursion vars for assignments.

- Assume the abbreviations (13a)–(13b) in L^{λ}_{ar} / L^{λ}_{r}:

$$A' \equiv \text{let } x_1 = D_1, \dots, x_n = D_n \text{ in } D_0 \tag{13a}$$

$$\equiv \lambda(x_1)\big(\dots[\lambda(x_n)(D_0)](D_n)\dots\big)(D_1) \tag{13b}$$

$$x_i \in \mathsf{PureV}_{\tau_i},\ x_i \notin \mathsf{Vars}(A),\ n \geq 1, \text{ for } i \in \{1, \dots, n\}$$

$$D_j \equiv A_j\{p_1 :\equiv x_1, \dots, p_n :\equiv x_n\}, \text{ for } j \in \{0, \dots, n\} \tag{13c}$$

We shall consider a special case of $n = 1$. It suffices for a demonstration.

Reduction of Scott let-Expressions to Canonical where-Recursion Terms

Lemma

Assume that $A, C, A_1 \in$ Terms *that are as in* (14a)–(14b), *Given that:*

- C, A_1 *are explicit, irreducible;* A_1 *is proper,*
- $p_1 \notin \mathsf{FreeV}(C)$, $x_1 \notin \mathsf{Vars}(A)$,
- $z \notin \mathsf{FreeV}(\lambda(\overrightarrow{u})x_1(\overrightarrow{v}))$:

$$A \equiv \mathsf{cf}_{\gamma^*}(A) \equiv \underbrace{\lambda(z)\big[C\big(\lambda(\overrightarrow{u})p_1(\overrightarrow{v})\big)\big]}_{A_0} \text{ where } \{\, p_1 := A_1 \,\} \tag{14a}$$

$$A_0 \equiv \lambda(z)\big[C\big(\lambda(\overrightarrow{u})p_1(\overrightarrow{v})\big)\big] \tag{14b}$$

Then, the let-expression A' *is not algorithmically equivalent to* A

$$A \not\approx_{\gamma^*} A' \equiv \text{let } x_1 = A_1 \text{ in } A_0 \tag{15a}$$

$$\equiv \big[\lambda(x_1)\big(A_0\{p_1 :\equiv x_1\}\big)\big](A_1) \tag{15b}$$

$$\approx_{\gamma^*} \mathsf{cf}_{\gamma^*}(A') \tag{15c}$$

Reduction of Scott let-Expressions to Canonical where-Recursion Terms: Proof

Proof: The full proof is given in Loukanova [6]. Part of the proof:

$$A' \equiv \big[\lambda(x_1)\big(A_0\{p_1 :\equiv x_1\}\big)\big](A_1) \tag{16a}$$

$$\equiv \lambda(x_1)\Big[\underbrace{\big[\lambda(z)\big[C\big(\lambda(\overrightarrow{u})p_1(\overrightarrow{v})\big)\big]\big]}_{A_0}\{p_1 :\equiv x_1\}\Big](A_1) \tag{16b}$$

$$\Rightarrow \lambda(x_1)\Big[\lambda(z)\big[C(r_1)\big] \text{ where } \{\, r_1 := \lambda(\overrightarrow{u})x_1(\overrightarrow{v})\,\}\Big](A_1) \tag{16c}$$

by Lemma 3 [6], (lq-comp), (ap-comp)

$$\Rightarrow \Big[\lambda(x_1)\big[\lambda(z)\big[C(r_1^1(x_1))\big]\big]\text{where } \{\, r_1^1 := \lambda(x_1)\lambda(\overrightarrow{u})x_1(\overrightarrow{v})\,\}\Big](A_1) \tag{16d}$$

by (ξ) for $\lambda(x_1)$, (ap-comp)

$$\Rightarrow \lambda(x_1)\big[\lambda(z)\big[C(r_1^1(x_1))\big]\big](A_1) \text{ where } \{\, r_1^1 := \lambda(x_1)\lambda(\overrightarrow{u})x_1(\overrightarrow{v})\,\} \tag{16e}$$

by (recap)

Reduction of Scott let-Expressions to Canonical where-Recursion Terms: Proof Cont.

$$\Rightarrow \Big[\lambda(x_1)\big[\lambda(z)\big[C(r^1_1(x_1))\big]\big](p_1) \text{ where } \{p_1 := A_1\}\Big] \text{ where } \{\, r^1_1 := \lambda(x_1)\lambda(\overrightarrow{u})x_1(\overrightarrow{v})\,\} \tag{17a}$$

by (ap), (rec-comp)

$$\Rightarrow \lambda(x_1)\big[\lambda(z)\big[C(r^1_1(x_1))\big]\big](p_1) \text{ where } \{p_1 := A_1,\ r^1_1 := \lambda(x_1)\lambda(\overrightarrow{u})x_1(\overrightarrow{v})\,\} \quad \text{by (head)} \tag{17b}$$

$$\equiv \mathsf{cf}_{\gamma^*}(A') \approx_{\gamma^*} A' \tag{17c}$$

$$\not\approx_{\gamma^*} A \tag{17d}$$

Thus, (15a) holds: $A \not\approx_{\gamma^*} A'$, by Theorem 6 from (14a) and (17b).

Proposition

In general, the algorithmic equivalence does not hold between the $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ *recursion terms of the form* (12a) *and the* λ*-calculus terms* (13a)–(13b), *which are characteristic for the corresponding let-expressions in* λ*-calculus.*

Proof: By Lemma 3

Scott Question

Question rised by Dana S. Scott, on Loukanova [6]:

> *In Section 2.3 "Denotational Semantics" it looks to me that you are using the category of sets. Have you thought of other categories?*

Lines of initiated and future work on Type-Theory of Recursion, incorporating states, situations, situated objects, situated and types:

- L_{ar}^{λ} type theory of acyclic algorithms that close-off
- L_{r}^{λ} type theory of full recursion, incl., partial functions
- For L_{ar}^{λ} and L_{r}^{λ}, semantic domains of denotational semantics can be:
  - of the category sets: Zermelo-Fraenkel Set Theory ZFC: up to now
  - proper classes of non-well founded sets: to be added
- Dependent-Type Theory of Full Recursion & Situated Information (DTTSitInfo /DTTSI), Loukanova since 1989, recent [1, 5]

For proper classes of non-well founded sets, see Rathjen [10, 11]

Development of Type-Theory of (Acyclic) Algorithms L_r^λ (L_{ar}^λ) and Dependent-Type Theory of Situated Info (DTTSitInfo)

Classes of type theories modeling states & situated info & algorithms

$$\text{Montague IL} \subsetneq \text{Gallin } TY_2 \subsetneq \text{Moschovakis } L_{ar}^\lambda \subsetneq \text{Moschovakis } L_r^\lambda \quad (18a)$$

$$\subsetneq \text{DTTSitInfo} \quad (18b)$$

- Type-Theory of (Acyclic) Recursion / Algorithms, L_r^λ (L_{ar}^λ): provides:
  - a math notion of algorithm
  - Computational Semantics of formal (FL) and natural (NL) languages
- L_{ar}^λ / L_r^λ is type theory of algorithms with acyclic / full recursion:
  - Introduced by Moschovakis [8]
  - Math development by Loukanova [2, 3, 4, 7, 6]
- logic operators, by logic constants of suitable types
- underspecification, generalized quantifiers, pure logic quantifiers
- extended reduction calculus of L_{ar}^λ / L_r^λ
- proof that L_{ar}^λ & L_r^λ extend classic λ-calculus, algorithmically, [6]
- Dependent-Type Theory of Situated Info (DTTSitInfo / DTTSI)

Motivation for Type Theory L_{ar}^{λ} and Outlook: Theory & Applications

- L_{ar}^{λ} provides Computational Semantics:
  - for Natural Language (NL), Formal Languages (FL), Programming Languages:
  - for greater semantic distinctions than type-theoretic semantics by λ-calculi, including any Montagovian grammars for NL
- L_{ar}^{λ} provides Parametric Algorithms
  - Parameters can be instantiated depending on context info, specific areas and and specific domains of applications
  - Domains and applications using natural language
  - Syntax-Semantics Interfaces with semantic ambiguities and underspecification
- L_{ar}^{λ} with logical operators and pure quantifiers can be used for:
  - proof-theoretic computational semantics and reasoning
  - inferences of semantic information
  - Canonical forms can be used by automatic provers and proof assistants

Looking Forward with Thanks!

Definition (Congruence Relation, informally)

The *congruence* relation is the smallest equivalence relation (i.e., reflexive, symmetric, transitive) between $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms, $A \equiv_{\mathsf{c}} B$, that is closed under:

1. operators of term-formation:
   - application
   - λ-abstraction
   - logic operators
   - pure, logic quantifiers
   - acyclic recursion
   - restriction
2. renaming bound variables (pure and recursion), without causing variable collisions
3. re-ordering of the assignments within the acyclic sequences of assignments in the recursion terms
4. re-ordering of the restriction sub-terms in the restriction terms

[Congruence] If $A \equiv_c B$, 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)

- If $A \Rightarrow B$, and $\xi \in \{ \lambda, \exists, \forall \}$, then $\xi(u)(A) \Rightarrow \xi(u)(B)$ (lq-comp)

- If $A_i \Rightarrow B_i$ $(i = 0, \ldots, n)$, then

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

- If $A_0 \Rightarrow B_0$ and $C_i \Rightarrow R_i$ $(i = 0, \ldots, n)$, then

$$A_0 \text{ such that } \{ C_1, \ldots, C_n \}$$
$$\Rightarrow B_0 \text{ such that } \{ R_1, \ldots, R_n \}$$
(st-comp)

Reduction Rules (to be continued)

[Head Rule] Given that $p_i \neq q_j$ and no p_i occurs freely in any B_j,

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

[Bekič-Scott Rule] Given that $p_i \neq q_j$ and no q_i occurs freely in any A_j

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

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

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

Reduction Rules (to be continued)

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

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

[λ and Quantifiers Rules] Let $\xi \in \{\, \lambda, \exists, \forall \,\}$.
Given fresh $p_i' \in \big[\,\mathsf{RecV} - \big(\,\mathsf{FV}(A) \cup \mathsf{BV}(A)\big)\big]$, $i = 1, \ldots, n$, for $A \equiv A_0$ where $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$ and replacements A_i' in (22):

$$A_i' \equiv \Big[A_i\{\, p_1 :\equiv p_1'(u), \ldots, p_n :\equiv p_n'(u) \,\}\Big] \qquad (22)$$

$$\begin{aligned} &\xi(u)\,\Big(A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}\Big) \\ &\Rightarrow \xi(u)\, A_0' \text{ where } \{\, p_1' := \lambda(u)\, A_1', \ldots, p_n' := \lambda(u)\, A_n' \,\} \end{aligned} \qquad (\xi)$$

- each $R_i^{\tau_i} \in$ Terms in $\overrightarrow{R}$ is immediate and $\tau_i \in \{\, \mathsf{t}, \widetilde{\mathsf{t}} \,\}$
- each $C_j^{\tau_j} \in$ Terms is proper and $\tau_j \in \{\, \mathsf{t}, \widetilde{\mathsf{t}} \,\}$ $(j = 1, \ldots, m,\ m \geq 0)$
- $a_0, c_j \in$ RecV $(j = 1, \ldots, m)$ fresh

(st1) Rule A_0 is an immediate term, $m \geq 1$

$$
\begin{aligned}
&(A_0 \text{ such that } \{\, C_1, \ldots, C_m, \overrightarrow{R} \,\}) \\
\Rightarrow\ &(A_0 \text{ such that } \{\, c_1, \ldots, c_m, \overrightarrow{R} \,\}) \\
&\qquad \text{where } \{\, c_1 := C_1,\ \ldots, c_m := C_m \,\}
\end{aligned}
\tag{st1}
$$

(st2) Rule A_0 is a proper term

$$
\begin{aligned}
&(A_0 \text{ such that } \{\, C_1, \ldots, C_m, \overrightarrow{R} \,\}) \\
\Rightarrow\ &(a_0 \text{ such that } \{\, c_1, \ldots, c_m, \overrightarrow{R} \,\}) \\
&\qquad \text{where } \{\, a_0 := A_0, \\
&\qquad\qquad c_1 := C_1,\ \ldots, c_m := C_m \,\}
\end{aligned}
\tag{st2}
$$

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: (25a)–(25c):

$$A \equiv A_0 \text{ where } \{\, \overrightarrow{a} := \overrightarrow{A}, \tag{25a}$$

$$p := \lambda(\overrightarrow{u})\lambda(v)P, \tag{25b}$$

$$\overrightarrow{b} := \overrightarrow{B}\,\} \tag{25c}$$

such that the following holds:

1. $v \notin$ FreeVars(P)
2. All occurrences of p in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are occurrences:
   - in $p(\overrightarrow{u})(v)$
   - which are in the scope of $\lambda(v)$
     modulo renaming the bound variables $\overrightarrow{u}, v$

(γ^*)-rule

$$
\begin{aligned}
A \equiv A_0 \text{ where } \{ \overrightarrow{a} &:= \overrightarrow{A}, && \text{(26a)}\\
p &:= \lambda(\overrightarrow{u})\lambda(v)P, && \text{(26b)}\\
\overrightarrow{b} &:= \overrightarrow{B} \; \} && \text{(26c)}\\
\Rightarrow_{(\gamma^*)} A'_0 \text{ where } \{ \overrightarrow{a} &:= \overrightarrow{A}', && \text{(26d)}\\
p' &:= \lambda(\overrightarrow{u})P, && \text{(26e)}\\
\overrightarrow{b} &:= \overrightarrow{B'} \; \} && \text{(26f)}
\end{aligned}
$$

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 $\lambda(v)$
- $p' \in \mathsf{RecV}_{(\overrightarrow{\sigma} \to \tau)}$ 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 (26a)–(26f), modulo renaming the variables $\overrightarrow{u}, v$

Theorem (γ^*-Canonical Form: Existence and Uniqueness)

See Loukanova [2, 3, 4], Moschovakis [8].
For every $A \in$ Terms, *there exists a unique up to congruence, irreducible term* $\mathsf{cf}_{\gamma^*}(A) \in$ Terms, *such that:*

1. *for some explicit, irreducible terms* $A_0, \ldots, A_n \in$ Terms $(n \geq 0)$

$$\mathsf{cf}_{\gamma^*}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \tag{27}$$

$$A \Rightarrow \mathsf{cf}_{\gamma^*}(A) \tag{28}$$

2. *for every* B, *such that* $A \Rightarrow B$ *and* B *is irreducible, it holds that* $B \equiv_c \mathsf{cf}_{\gamma^*}(A)$,
i.e., $\mathsf{cf}_{\gamma^*}(A)$ *is unique, up to congruence*

3. $\mathsf{Consts}(\mathsf{cf}_{\gamma^*}(A)) = \mathsf{Consts}(A)$

4. $\mathsf{FreeV}(\mathsf{cf}_{\gamma^*}(A)) = \mathsf{FreeV}(A)$

The proof is by induction on term structure of A, (5a)–(5e), (5f).

Algorithmic Semantic of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ / L^{λ}_{r}

In the original reduction calculus by Moschovakis [8], the Canonical Form Theorem 6 is about $\mathsf{cf}(A)$. Often, we shall write:

$$\mathsf{cf}(A) \equiv \mathsf{cf}_{\gamma^*}(A) \tag{29}$$

- For every term $A \in$ Terms, by the Canonical Form Theorem 6:

$$A \Rightarrow \mathsf{cf}_{\gamma^*}(A)$$

- For every proper (i.e., non-immediate) $A \in$ Terms, $\mathsf{cf}_{\gamma^*}(A)$ determines the algorithm $\mathsf{alg}(A)$ for computing $\mathsf{den}(A)$

Theorem (Effective Reduction Calculi)

For every term $A \in$ Terms, its canonical form $\mathsf{cf}_{\gamma^*}(\mathrm{A})(A)$ is effectively computed, by the extended reduction calculus.

Definition (of Algorithmic Equivalence / Synonymy)

Two terms $A, B \in$ Terms are algorithmically equivalent, $A \approx B$, in a given semantic structure $\mathfrak{A}$, i.e., referentially synonymous in $\mathfrak{A}$, iff

- A and B are both immediate, or
- A and B are both proper

and there are explicit, irreducible terms (of appropriate types), $A_0, \ldots, A_n, B_0, \ldots, B_n$, $n \geq 0$, such that:

1. $A \Rightarrow_{\mathsf{cf}} A_0$ where $\{ p_1 := A_1, \ldots, p_n := A_n \} \equiv \mathsf{cf}_{\gamma^*}(A)$
2. $B \Rightarrow_{\mathsf{cf}} B_0$ where $\{ p_1 := B_1, \ldots, p_n := B_n \} \equiv \mathsf{cf}(B)$
3. for all $i \in \{ 0, \ldots, n \}$
   a. for every $x \in \mathsf{PureV} \cup \mathsf{RecV}$,
   $$x \in \mathsf{FreeV}(A_i) \quad \text{iff} \quad x \in \mathsf{FreeV}(B_i) \tag{30}$$
   b. $\mathsf{den}(A_i) = \mathsf{den}(B_i)$

Type Theory L^{λ}_{ar} / L^{λ}_{r} is more expressive than Gallin TY2

Theorem (Conditions for Explicit and Non-Explicit Terms)

Extending Theorem §3.24, Moschovakis [8].

1. *Necessary Condition for Explicit Terms:*
   For every explicit $A \in$ Terms, *there is no* $p \in$ RecV *such that*
   a. p *is bound via the recursion operator* where *in* $\mathrm{cf}_{\gamma^*}(A)$
   b. p *occurs in more than one of the parts* A_i $(0 \leq i \leq n)$ *of* $\mathrm{cf}_{\gamma^*}(A)$
2. *Sufficient Condition for Non-Explicit Terms:*
   Assume that $A \in$ Terms *and* $p \in$ RecV *are such that*
   a. p *is bound via the recursion operator* where *in* $\mathrm{cf}_{\gamma^*}(A)$
   b. p *occurs in (at least) two parts* A_i $(0 \leq i \leq n)$ *of* $\mathrm{cf}_{\gamma^*}(A)$, *which have denotations essentially depending on* p, *e.i.:*

   Then, there is no explicit term $B \in$ Terms, *such that* B *is algorithmically equivalent to* A, $B \approx A$,
   Therefore, there is no λ*-calculus term* B, *such that* $B \approx A$.

The proof is by Moschovakis [8] I provide it for the extended L^{λ}_{ar} / L^{λ}_{r}

Reductions with Pure Quantifier Rules: Algorithmic Patterns and Instantiations

- Assume $cube, large_0 \in \mathsf{Consts}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})}$, in the typical Aristotelian form:

$$\text{Some cube is large} \xrightarrow{\text{render}} B \equiv \exists x(cube(x) \land large_0(x)) \tag{31a}$$

$$B \Rightarrow \exists x((c \land l) \text{ where } \{\, c := cube(x),\ l := large_0(x)\, \}) \tag{31b}$$

by 2 x (ap) (ap-comp), (recap), (rec-comp), (head), (lq-comp)

$$\Rightarrow \underbrace{\exists x(c'(x) \land l'(x))}_{B_0 \text{ algorithmic pattern}} \text{ where } \{ \tag{31c}$$

$$\underbrace{c' := \lambda(x)(cube(x)),\ l' := \lambda(x)(large_0(x))}_{\text{instantiations of memory slots } c',\ l'} \} \equiv \mathsf{cf}(B) \tag{31d}$$

from (31c), by (ξ) to $\exists$

$$\approx \underbrace{\exists x(c'(x) \land l'(x))}_{B_0 \text{ algorithmic pattern}} \text{ where } \{ \underbrace{c' := cube,\ l' := large_0}_{\text{instantiations of memory slots } c',\ l'} \} \equiv B' \tag{31e}$$

$$\text{by Def. 8 from (31c)–(31d), } \mathsf{den}(\lambda(x)(cube(x))) = \mathsf{den}(cube), \quad \mathsf{den}(\lambda(x)(large_0(x))) = \mathsf{den}(large_0) \tag{31f}$$

$$\text{Some cube is large} \xrightarrow{\text{render}} T, \quad \mathit{large} \in \mathsf{Consts}_{((\tilde{\mathsf{e}}\to\tilde{\mathsf{t}})\to(\tilde{\mathsf{e}}\to\tilde{\mathsf{t}}))} \tag{32a}$$

$$T \equiv \exists x\big[\mathit{cube}(x) \land \underbrace{\mathit{large}(\mathit{cube})(x)}_{\text{by predicate modification}}\big] \Rightarrow \ldots \tag{32b}$$

$$\Rightarrow \exists x\big[(c_1 \land l) \text{ where } \{\, c_1 := \mathit{cube}(x), \tag{32c}$$

$$l := \mathit{large}(c_2)(x),\ c_2 := \mathit{cube}\,\}\big] \tag{32d}$$

$$\Rightarrow \exists x(c_1'(x) \land l'(x)) \text{ where } \{\, c_1' := \lambda(x)(\mathit{cube}(x)), \tag{32e}$$

$$l' := \lambda(x)(\mathit{large}(c_2'(x))(x)),\ c_2' := \lambda(x)\,\mathit{cube}\,\} \tag{32f}$$

$$\equiv \mathsf{cf}(T) \qquad \text{(32e)–(32f) is by } (\xi) \text{ on (32c)–(32d)}$$

$$\Rightarrow_{\gamma^*} \exists x(c_1'(x) \land l'(x)) \text{ where } \{\, c_1' := \lambda(x)(\mathit{cube}(x)), \tag{32g}$$

$$l' := \lambda(x)(\mathit{large}(c_2)(x)),\ c_2 := \mathit{cube}\,\} \tag{32h}$$

$$\equiv \mathsf{cf}_{\gamma^*}(T)$$

$$\approx \exists x(c_1'(x) \land l'(x)) \text{ where } \{\, c_1' := \mathit{cube}, \tag{32i}$$

$$l' := \lambda(x)(\mathit{large}(c_2)(x)),\ c_2 := \mathit{cube}\,\} \tag{32j}$$

$$\text{Some cube is large} \xrightarrow{\text{render}} C, \quad large \in \mathsf{Consts}_{((\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}})\to(\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}}))}$$

$$C \equiv \underbrace{\exists x\big[c'(x) \land large(c')(x)\big]}_{E_0} \text{ where } \{\, c' := cube \,\} \tag{33a}$$

$$\Rightarrow \underbrace{\exists x\Big[\big(c'(x) \land l\big) \text{ where } \{\, l := large(c')(x) \,\}\Big]}_{E_1} \text{ where } \{\, c' := cube \,\} \tag{33b}$$

from (33a), by (ap) to $\land$ of E_0; (lq-comp); (rec-comp)

$$\Rightarrow \Big[\underbrace{\exists x\big(c'(x) \land l'(x)\big) \text{ where } \{\, l' := \lambda(x)\big(large(c')(x)\big) \,\}}_{E_2}\Big] \text{ where } \{\, c' := cube \,\} \tag{33c}$$

from (33b), by (ξ) to $\exists$

$$\Rightarrow \underbrace{\exists x\big(c'(x) \land l'(x)\big)}_{C_0 \text{ an algorithmic pattern}} \text{ where } \{\, \underbrace{c' := cube,\ l' := \lambda(x)\big(large(c')(x)\big)}_{\text{instantiations of memory } c',\, l'} \,\} \equiv \mathsf{cf}(C) \tag{33d}$$

from (33c), by (head); (cong)

Proposition

- *The* $\mathrm{L}_{\mathrm{ar}}^{\lambda}$*-terms* $C \approx \mathsf{cf}(C)$ *in* (33a)–(33d), *and many other* $\mathrm{L}_{\mathrm{ar}}^{\lambda}$*-terms, are not algorithmically equivalent to any explicit terms*
- *Therefore,* $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ *is a strict, proper extension of Gallin TY2 and Montagovian IL.*

Therefore:

Placement of $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ in a class of type theories

$$\text{Montague IL} \subsetneq \text{Gallin TY}_2 \subsetneq \text{Moschovakis } \mathrm{L}_{\mathrm{ar}}^{\lambda} \subsetneq \text{Moschovakis } \mathrm{L}_{r}^{\lambda} \tag{34}$$

Some References I

Loukanova, R.: Situation Theory, Situated Information, and Situated Agents.
In: N.T. Nguyen, R. Kowalczyk, A. Fred, F. Joaquim (eds.) Transactions on Computational Collective Intelligence XVII, *Lecture Notes in Computer Science*, vol. 8790, pp. 145–170. Springer, Berlin, Heidelberg (2014).
URL https://doi.org/10.1007/978-3-662-44994-3_8

Loukanova, R.: Gamma-Reduction in Type Theory of Acyclic Recursion.
Fundamenta Informaticae **170**(4), 367–411 (2019).
URL https://doi.org/10.3233/FI-2019-1867

Some References II

- Loukanova, R.: Gamma-Star Canonical Forms in the Type-Theory of Acyclic algorithms.
  In: J. van den Herik, A.P. Rocha (eds.) Agents and Artificial Intelligence. ICAART 2018, *Lecture Notes in Computer Science, book series LNAI*, vol. 11352, pp. 383–407. Springer International Publishing, Cham (2019).
  DOI https://doi.org/10.1007/978-3-030-05453-3_18

- Loukanova, R.: Type-Theory of Acyclic Algorithms for Models of Consecutive Binding of Functional Neuro-Receptors.
  In: A. Grabowski, R. Loukanova, C. Schwarzweller (eds.) AI Aspects in Reasoning, Languages, and Computation, vol. 889, pp. 1–48. Springer International Publishing, Cham (2020).
  URL https://doi.org/10.1007/978-3-030-41425-2_1

Some References III

Loukanova, R.: Algorithmic Dependent-Type Theory of Situated Information and Context Assessments.
In: S. Omatu, R. Mehmood, P. Sitek, S. Cicerone, S. Rodríguez (eds.) Distributed Computing and Artificial Intelligence, 19th International Conference, vol. 583, pp. 31–41. Springer International Publishing, Cham (2023).
DOI 10.1007/978-3-031-20859-1_4.
URL https://doi.org/10.1007/978-3-031-20859-1_4

Loukanova, R.: Logic Operators and Quantifiers in Type-Theory of Algorithms.
In: D. Bekki, K. Mineshima, E. McCready (eds.) Logic and Engineering of Natural Language Semantics, pp. 173–198. Springer Nature Switzerland, Cham (2023).
DOI https://doi.org/10.1007/978-3-031-43977-3_11

Some References IV

- Loukanova, R.: Restricted Computations and Parameters in Type-Theory of Acyclic Recursion.
  ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal **12**(1), 1–40, e29081 (2023).
  URL https://doi.org/10.14201/adcaij.29081
- Moschovakis, Y.N.: A Logical Calculus of Meaning and Synonymy.
  Linguistics and Philosophy **29**(1), 27–89 (2006).
  URL https://doi.org/10.1007/s10988-005-6920-7
- Plotkin, G.D.: LCF considered as a programming language.
  Theoretical Computer Science **5**(3), 223–255 (1977).
  URL https://doi.org/10.1016/0304-3975(77)90044-5
- Rathjen, M.: The anti-foundation axiom in constructive set theories.
  In: G. Mints, R. Muskens (eds.) Games, Logic, and Constructive Sets, pp. 87–108. CSLI Publications, Stanford, California (2003)

Some References V

Rathjen, M.: Predicativity, circularity, and anti-foundation.
In: G. Link (ed.) One hundred years of Russell's paradox (De Gruyter Series in Logic and Its Applications), vol. 6, pp. 191–219. Walter de Gruyter, Berlin, New York (2004)

Scott, D.S.: A type-theoretical alternative to ISWIM, CUCH, OWHY.
Theoretical Computer Science **121**(1), 411–440 (1993).
URL https://doi.org/10.1016/0304-3975(93)90095-B