Algorithmic Eta-reduction in Type-theory of Acyclic Recursion

Roussanka Loukanova[1,2]
[1]*Stockholm University, Stockholm, Sweden*
[2]*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev Str., Block 8, 1113 Sofia, Bulgaria*

Keywords: Algorithms, Acyclic Recursion, Type Theory, Reduction Calculi, Canonical Forms, Eta-reduction.

Abstract: The paper extends the standard reduction calculus of the higher-order Type-theory of Acyclic Recursion to η-reduction. This is achieved by adding a restricted η-rule, which is applicable only to terms in canonical forms satisfying certain conditions. Canonical forms of terms determine the iterative algorithms for computing the semantic denotations of the terms. Unnecessary λ-abstractions and corresponding functional applications in canonical forms contribute to algorithmic complexity. The η-rule provides a simple way to reduce complexity by maintenance of essential algorithmic structure of computations.

1 INTRODUCTION

This paper is part of theoretical development and applications of a new approach to the mathematical notion of algorithm, originally introduced by the formal languages of recursion (Moschovakis, 1989). The simply-typed theory of recursion $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ introduced in (Moschovakis, 2006) is a higher-order type theory, which is a proper extension of Gallin type theory TY_2 (Gallin, 1975). The book (Moschovakis, 2019) investigates complexity in its untyped version.

The formal languages FLR introduced in a sequence of papers (Moschovakis, 1989; Moschovakis, 1993; Moschovakis, 1997), while untyped systems formalising untyped domains of recursive functions, allow full recursion with cyclicity, and are equivalent to any of the classic theories of the mathematical notion of algorithm. The formal syntax of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, presented originally in (Moschovakis, 2006), is typed and allows only recursion terms with acyclic systems of assignments, thus, formalising algorithms that end after finite iterations of computations. Typed languages of full recursion $\mathrm{L}^{\lambda}_{\mathrm{r}}$ have the typed syntax of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, without the acyclicity requirement. The classes of languages of recursion (FLR, $\mathrm{L}^{\lambda}_{\mathrm{r}}$, and $\mathrm{L}^{\lambda}_{\mathrm{ar}}$) have two semantic layers: denotational semantics and algorithmic semantics. The recursion terms of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ are essential for representing algorithmic computations of semantic information.

This paper concerns the typed theory of algoritms $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, which is already well developed, and provides new approaches to intelligent foundations, with versatile applications, and especially to the areas of Artificial Intelligence (AI). In this paper, we introduce a technique, which, by adding a simple additional reduction rule to the reduction calculi of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, is important for applications of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ to the areas of AI.

The purpose of the reduction calculi of the formal language of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is to reduce every $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ term A to a term in a canonical form that represents the computational steps for the computation of the denotation of A. The reduction calculi of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ reflect fundamental algorithmic patterns in computations. Among the potential applications of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ are intelligent software systems, e.g., in robotics and in AI, that perform algorithmic procedures, which determine reliable performance. The prominent applications of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ are to computational semantics of formal and natural languages, and, in particular, semantics of programming and other specification languages in Computer Science and AI, as well as Natural Language Processings (NLP), and computational grammars that cover semantics.

The reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ reduces every $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-term to its canonical form. The canonical forms not only preserve the denotations of the $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms, they reveal the algorithmic steps encoded by the terms for computing their denotations. The canonical form $\mathsf{cf}(A)$ of every $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-term A determines the algorithm that computes the denotation of A in the semantics domain of every given semantic structure. The λ-rule of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is one of the most important reduction rules

Loukanova, R.
Algorithmic Eta-reduction in Type-theory of Acyclic Recursion.
DOI: 10.5220/0009182410031010
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 1003-1010
ISBN: 978-989-758-395-7; ISSN: 2184-433X

for the recursion and abstraction operators. Nevertheless, in many cases, the λ-rule, creates redundant λ-abstractions over variables that do not occur freely in the recursion terms (e.g., see Example 3.1). For example, it is important to simplify terms having such extra abstractions, when translating, i.e., rendering, natura language (NL) expressions to semantic representations by $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms.

This paper presents a restricted η-reduction that simplifies terms in canonical forms. The η-rule does not preserve the strictest referential synonymy of its input and output terms, which are in canonical forms. Importantly, the η-rule takes care to maintain closely the algorithmic meaning of the terms in the entire reduction sequence, while reducing computational complexity caused by excessive, superfluous *lambda*-abstractions and corresponding functional applications. It preserves the denotations of the input and reduced terms. (Loukanova, 2019c; Loukanova, 2019b) introduce and investigate the properties of more intricate rules and reduction calculi for removing redundant *lambda*-abstractions. The η-reduction is also interesting. It is easier to use, by extending the ordinal reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, since it is applied directly, only to terms in canonical forms.

(Loukanova, 2011d) introduces very briefly the η-rule, for the purpose of applying it to semantic representations of human language, without looking into its properties. We reformulate the rule here, for presenting its properties with respect to its existing and potential applications to the areas of Artificial Intelligence (AI). In particular, we point to applications that need context dependent information and algorithmic computations that depend on states of information, including agents.

We should stress that the η-rule introduced here is about recursion terms and the algorithms designated by them. It is different from, while related to, the denotational η-rule in standard λ-calculi.

In Section 2, we give the formal definitions of the syntax of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$. We introduce the denotational and algorithmic semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ with some intuitions and examples. Then, we introduce the rules of the reduction calculus of the type theory $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, which is central to the referential intensions, i.e., to the algorithmic meaning in a selected specific semantic domain of applications, and some key theoretical results. The central part of the paper is on introducing the new, additional η-rule for reduction of the $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ terms, which are in canonical forms, to simpler η-canonical forms that formalise more efficient algorithmic computations.

2 TYPE-THEORY OF ACYCLIC RECURSION

2.1 Types

The set Types is the smallest set defined recursively as follows, by :

$$\tau :\equiv \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (\tau_1 \to \tau_2) \qquad \text{(Types)}$$

The type e is for *entities*, also called *individuals*; s is for states consisting of various information, e.g., such as possible worlds, context, time and/or space locations, some agents, e.g., a speaker; t is for truth values. For an elaboration of possible choices of context information, which include speaker agents, see (Loukanova, 2011b). The type $(\mathsf{s} \to \tau)$ is for context dependent objects.

2.2 Syntax of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

Vocabulary of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

- *Constants:* $K = \bigcup_{\tau \in \mathsf{Types}} K_\tau$,
  where, for each $\tau \in \mathsf{Types}$, K_τ is a finite set:
  $K_\tau = \{ \mathsf{c}^\tau_0, \mathsf{c}^\tau_1, \ldots, \mathsf{c}^\tau_{k_\tau} \}$
- *Pure Variables:* $\mathsf{PureV} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{PureV}_\tau$,
  where, for each $\tau \in \mathsf{Types}$,
  $\mathsf{PureV}_\tau = \{ \mathsf{v}^\tau_0, \mathsf{v}^\tau_1, \ldots \}$ (a denumerable set)
- *Recursion Variables (Memory Locations):*
  $\mathsf{RecV} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{RecV}_\tau$,
  where, for each $\tau \in \mathsf{Types}$,
  $\mathsf{RecV}_\tau = \{ \mathsf{p}^\tau_0, \mathsf{p}^\tau_1, \ldots \}$ (a denumerable set)

Definition 1 (The set Terms of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$).

$$A :\equiv \mathsf{c}^\tau : \tau \mid x^\tau : \tau \mid B^{(\sigma \to \tau)}(C^\sigma) : \tau \qquad (1)$$

$$\mid \lambda(v^\sigma)\,(B^\tau) : (\sigma \to \tau) \qquad (2)$$

$$\mid A^\sigma_0 \text{ where } \{ p^{\sigma_1}_1 := A^{\sigma_1}_1, \ldots, p^{\sigma_n}_n := A^{\sigma_n}_n \} : \sigma \qquad (3)$$

for $A_1 : \sigma_1, \ldots, A_n : \sigma_n$ are terms ($n \geq 0$), and $p_1 : \sigma_1$, $\ldots$, $p_n : \sigma_n$ are pairwise different recursion variables (memory locations), and the sequence of *assignments* $\{ p_1 := A_1, \ldots, p_n := A_n \}$ satisfies the Acyclicity Constraint:

Acyclicity Constraint. A sequence of assignments of the form:

$$\{ p_1 := A_1, \ldots, p_n := A_n \}$$

is *acyclic* iff there is a ranking function

$$\mathsf{rank} : \{ p_1, \ldots, p_n \} \longrightarrow \mathbb{N}$$

such that, for all $p_i, p_j \in \{ p_1, \ldots, p_n \}$, if p_j occurs freely in A_i, then $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$.

Intuitively, an acyclic sequence of assignments $\{p_1 := A_1, \ldots, p_n := A_n\}$ defines recursive computations of the values $\mathsf{den}(A_1)$, $\ldots$, $\mathsf{den}(A_n)$ to be assigned to the locations $p_1, \ldots, p_n$, which close-off after a finite number of steps; ranking $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$ means that the value A_i assigned to p_i, may depend on the values of the free occurrences of the location p_j in A_i, and of all other free occurrences of locations with lower rank than p_j.

Some Notations and Abbreviations

- The symbol "=" is a predicate constant of the language $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ for identity, and also used for the identity relation, which it denotes
- The symbol "$\equiv$" is a meta-symbol for literal identity between expressions
- The symbol "$:\equiv$" is a meta-symbol that we use in definitions, e.g., of types and terms, and for the replacement operation
- Often, we skip some "understood" parentheses
- The type σ of a term A may be depicted either as a superscript, A^{σ}, or by a colon, $A : \sigma$

We use the following abbreviations for "folding" and "unfolding" sequences of assignments. For any terms $A_1 : \sigma_1, \ldots, A_n : \sigma_n, A_{n+1} : \sigma_{n+1}, C, D : \tau$, and recursion variables $p_1 : \sigma_1, \ldots, p_n : \sigma_n$, (where $n \geq 0$):

$$\overrightarrow{p} := \overrightarrow{A} \equiv p_1 := A_1, \ \ldots, \ p_n := A_n \tag{4a}$$

$$\overrightarrow{p} := \overrightarrow{A}\{C :\equiv D\} \equiv \tag{4b}$$

$$p_1 := A_1\{C :\equiv D\}, \ \ldots, \ p_n := A_n\{C :\equiv D\} \tag{4c}$$

where, for all $i = 1, \ldots, n$, $A_i\{C :\equiv D\}$ are the result of the replacement of all occurrences of C, respectively in A_i without causing variable clashes.

Denotational Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ The language $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ has denotational semantics provided by a denotational function $\mathsf{den}^{\mathfrak{A}}$, for any given typed semantic structure $\mathfrak{A}$ with typed domain frames and variable assignments g in $\mathfrak{A}$. The definition of $\mathsf{den}^{\mathfrak{A}}$ is by structural induction on the terms, for details, see (Moschovakis, 2006) and (Loukanova, 2019c; Loukanova, 2019b). Often, we shall designate the denotation function by den without the superscript.

2.3 Reduction Calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

The reduction rules define a *reduction* relation between terms.

The *congruence* relation, $\equiv_{\mathsf{c}}$, is the smallest relation between $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ terms ($A \equiv_{\mathsf{c}} B$) that is reflexive, symmetric, transitive, and closed under: term formation rules (application, λ-abstraction, and acyclic recursion); renaming bound variables (pure and recursion), without causing variable collisions; and re-ordering of the assignments within the acyclic sequences of assignments of the recursion terms , i.e., for any permutation π: $\{1, \ldots, n\} \to \{1, \ldots, n\}$:

Reduction Rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

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 $A(B) \Rightarrow A'(B')$ (rep1)
- If $A \Rightarrow B$, then $\lambda u\,(A) \Rightarrow \lambda u\,(B)$ (rep2)
- If $A_i \Rightarrow B_i$, for $i = 0, \ldots, n$, then (rep3)
  A_0 where $\{p_1 := A_1, \ldots, p_n := A_n\}$
  $\Rightarrow B_0$ where $\{p_1 := B_1, \ldots, p_n := B_n\}$

The Head Rule (head)

$\big(A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\}\big) \text{ where } \{\overrightarrow{q} := \overrightarrow{B}\}$
$\Rightarrow A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}, \ \overrightarrow{q} := \overrightarrow{B}\}$
given that no p_i occurs freely in any B_j, for $i = 1, \ldots, n$, $j = 1, \ldots, m$

The Bekič-Scott Rule (B-S)

$A_0 \text{ where } \{p := \big(B_0 \text{ where } \{\overrightarrow{q} := \overrightarrow{B}\}\big), \overrightarrow{p} := \overrightarrow{A}\}$
$\Rightarrow A_0 \text{ where } \{p := B_0, \overrightarrow{q} := \overrightarrow{B}, \ \overrightarrow{p} := \overrightarrow{A}\}$
given that no q_j occurs freely in any A_i, for $i = 1, \ldots, n$, $j = 1, \ldots, m$

The Recursion-application Rule (recap)

$(A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\})(B)$
$\Rightarrow A_0(B) \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\}$
given that no p_i occurs freely in B, for $i = 1, \ldots, n$

The Application Rule (ap)

$A(B) \ \Rightarrow \ A(p) \text{ where } \{p := B\}$
given that B is a proper term and p is a fresh location

The λ-Rule (λ)

$\lambda u\,(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})$
$\Rightarrow \lambda u A'_0 \text{ where } \{p'_1 := \lambda u A'_1, \ldots, p'_n := \lambda u A'_n\}$,
where for all $i = 1, \ldots, n$, p'_i is a fresh location and A'_i is the result of the replacement of the free occurrences of $p_1, \ldots, p_n$ in A_i with $p'_1(u), \ldots, p'_n(u)$, respectively, i.e.:

$$A'_i :\equiv A_i\{p_1 :\equiv p'_1(u), \ldots, p_n :\equiv p'_n(u)\}$$

2.4 Some Major Properties of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

The following theorems are important for the algorithmic meanings of the terms.

Theorem 1 (Canonical Form Theorem)**.** *(For details, see (Moschovakis, 2006), § 3.1.) For each term A, there is a unique, up to congruence, irreducible term C, denoted by* $\mathsf{cf}(A)$ *and called the canonical form of A, such that:*

1. $\mathsf{cf}(A) \equiv A_0$ where $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$ *for some explicit, irreducible terms* $A_1, \ldots, A_n$ $(n \geq 0)$
2. $A \Rightarrow \mathsf{cf}(A)$
3. *if* $A \Rightarrow B$ *and B is irreducible, then* $B \equiv_c \mathsf{cf}(A)$*, i.e.,* $\mathsf{cf}(A)$ *is the unique, up to congruence, irreducible term to which A can be reduced*

Theorem 2. *(See (Moschovakis, 2006), § 3.11.) For any* $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ *terms A and B, if* $A \Rightarrow B$*, then*

$$\mathsf{den}(A) = \mathsf{den}(B) \tag{5a}$$

$$\mathsf{den}(A) = \mathsf{den}(\mathsf{cf}(A)) \tag{5b}$$

The canonical forms have a distinguished feature that is part of their computational (algorithmic) role: they provide algorithmic patterns of semantic computations. The more general terms provide algorithmic patterns that consist of sub-terms with components that are recursion variables; the most basic assignments of recursion variables (of lowest ranks) provide the specific basic data that feeds-up the general computational patterns. The more general terms and sub-terms classify language expressions with respect to their semantics and determine the algorithms for computing the denotations of the expressions.

Algorithmic Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$**.** The notion of algorithmic semantics, i.e., algorithmic intension, in the languages of recursion, (Moschovakis, 2006), covers the most essential, computational aspect of the concept of meaning. The *referential intension*, $\mathsf{int}(A)$, of a meaningful term A is the tuple of functions (a recursor) that is defined by the denotations $\mathsf{den}(A_i)$ $(i \in \{0, \ldots n\})$ of the parts (i.e., the head sub-term A_0 and of the terms A_1, …, A_n in the system of assignments) of its canonical form:

$$\mathsf{cf}(A) \equiv A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}$$

Intuitively, for each meaningful term A, the intension of A, $\mathsf{int}(A)$, is the *algorithm* for computing its denotation $\mathsf{den}(A)$. Two meaningful expressions are synonymous iff their referential intensions are naturally isomorphic, i.e., they are the same algorithm. Thus, the algorithmic meaning of a meaningful term (i.e., its sense) is the information about how to compute its denotation step-by-step: a meaningful term has sense by carrying instructions within its structure, which are revealed by its canonical form, for acquiring what they denote in a model. The canonical form $\mathsf{cf}(A)$ of a meaningful term A encodes its intension, i.e., the algorithm for computing its denotation, via: (1) the basic instructions (facts), which consist of $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$ and the head term A_0, which are needed for computing the denotation $\mathsf{den}(A)$, and (2) a terminating rank order of the recursive steps that compute each $\mathsf{den}(A_i)$, for $i \in \{0, \ldots, n\}$, for incremental computation of the denotation $\mathsf{den}(A) = \mathsf{den}(A_0)$.

The reduction calculus of the type theory $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is effective. The calculus of the intensional synonymy, i.e., algorithmic equivalence, in the type-theory $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ has a restricted β-reduction rule, which contributes to the high expressiveness of the language of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, see (Moschovakis, 2006) and (Loukanova, 2011a; Loukanova, 2011c).

Theorem 3 (Referential Synonymy Theorem)**.** *(See (Moschovakis, 2006), § 3.4.) Two terms* A, B *are referentially synonymous,* $A \approx B$*, i.e., algorithmically equivalent, with respect to a given semantic structure* $\mathfrak{A}$ *iff there are explicit, irreducible terms (of appropriate types),* $A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_n$*, $n \geq 0$, such that:*

1. $A \Rightarrow_{\mathsf{cf}} A_0$ where $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$,
2. $B \Rightarrow_{\mathsf{cf}} B_0$ where $\{\, p_1 := B_1, \ldots, p_n := B_n \,\}$,
3. *(a) for every* $x \in \mathsf{PureV} \cup \mathsf{RecV}$,

$$x \in \mathsf{FreeVars}(A_i) \text{ iff } x \in \mathsf{FreeVars}(B_i), \quad \text{for } i \in \{0, \ldots, n\} \tag{6}$$

(b) for all $g \in G$*:*

$$\mathsf{den}(A_i)(g) = \mathsf{den}(B_i)(g),\ i \in \{0, \ldots, n\} \tag{7}$$

3 ETA-REDUCTION

In this section, at first, we present an example from natural language to motivate the usefulness of the additional η-rule, and then we introduce the η-rule and the extended reduction calculus.

Example 3.1. The detailed steps of rendering the sentence (8a) into a term, and its reduction to the canonical form (8b)–(8e), are given in (Loukanova, 2019b), as motivation for γ-reduction, which is more general and complex.

$$\text{Kim hugs some dog} \xrightarrow{\text{render}} A\ldots \tag{8a}$$

$$\mathsf{cf}(A) \equiv_{\mathsf{c}} [\lambda y_k(some\big(d(y_k)\big)(h(y_k)))](k) \text{ where} \tag{8b}$$

$$\{k := kim, \tag{8c}$$

$$h := \lambda y_k \lambda x_d\, hugs(x_d)(y_k), \tag{8d}$$

$$d := \lambda y_k\, dog\} \tag{8e}$$

Then, by Theorem 3:

$$\mathsf{cf}(A) \not\approx B \equiv_{\mathsf{c}} \big[\lambda y_k\, some(d')\big(h(y_k)\big)\big](k) \text{ where} \tag{9a}$$

$$\{k := kim, \tag{9b}$$

$$h := \lambda y_k \lambda x_d\, hugs(x_d)(y_k), \tag{9c}$$

$$d' := dog\} \tag{9d}$$

The term B in (9a)-(9b) is in a canonical form, but it is not algorithmically equivalent (referentially synonymous) to the term (8b)-(8e), by the reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$. That is, these two terms are not algorithmically equivalent with respect to the strictest notion of algorithm introduced in (Moschovakis, 2006).

Definition 2 (η-rule)**.** Let A be a term in a canonical form:

$$A \equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A},$$

$$p_{n+1} := \lambda v A_{n+1}, \tag{10a}$$

$$\overrightarrow{q} := \overrightarrow{B}\}$$

$$\equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n,$$

$$p_{n+1} := \lambda v A_{n+1}, \tag{10b}$$

$$q_1 := B_1, \ldots, q_k := B_k\}$$

with $n \geq 0$, $k \geq 0$, such that

1. $v : \sigma$ is a pure variable and $p_{n+1} : (\sigma \to \tau)$ is a recursion variable (location).
2. The explicit, irreducible term $A_{n+1} : \tau$ does not have any (free) occurrences of v (and p_{n+1}).
3. All the occurrences of p_{n+1} in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are occurrences of the term $p_{n+1}(v)$, which are in the scope of λv (modulo appropriate renaming of v).

Then, for any fresh recursion variable $p'_{n+1} : \tau$

$$A_0 \text{ where } \{$$

$$\overrightarrow{p} := \overrightarrow{A}, \tag{11a}$$

$$p_{n+1} := \lambda v A_{n+1},$$

$$\overrightarrow{q} := \overrightarrow{B}\}$$

$$\Rightarrow_{\eta} A_0\{p_{n+1}(v) :\equiv p'_{n+1}\} \text{ where } \{$$

$$\overrightarrow{p} := \overrightarrow{A}\{p_{n+1}(v) :\equiv p'_{n+1}\}, \tag{11b}$$

$$p'_{n+1} := A_{n+1},$$

$$\overrightarrow{q} := \overrightarrow{B}\{p_{n+1}(v) :\equiv p'_{n+1}\}\}$$

where, $\overrightarrow{A}\{p_{n+1}(v) :\equiv p'_{n+1}\}$ is the result of the replacement $A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}$ of all occurrences of $p_{n+1}(v)$ with p'_{n+1}, in all terms A_i of $\overrightarrow{A}$, and $\overrightarrow{B}\{p_{n+1}(v) :\equiv p'_{n+1}\}$ is the is the result of the replacement $B_j\{p_{n+1}(v) :\equiv p'_{n+1}\}$ of all occurrences of $p_{n+1}(v)$ with p'_{n+1}, in all terms B_j of $\overrightarrow{B}$.

Note: In the η-rule and corresponding applications, for all $i \in \{0,\ldots,n\}$ and $j \in \{0,\ldots,k\}$, the replacements $A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}$ and $B_j\{p_{n+1}(v) :\equiv p'_{n+1}\}$ are such that the occurrences of the term $p_{n+1}(v)$ in A_i and B_j are in the scope of λv.

Theorem 4 (Denotational Equivalence by η-rule)**.** *Let A be a term in a canonical form:*

$$A \equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A},$$

$$p_{n+1} := \lambda v A_{n+1},\ \overrightarrow{q} := \overrightarrow{B}\} \tag{12a}$$

($n \geq 0$) such that:

1. *$v : \sigma$ is a pure variable and $p_{n+1} : (\sigma \to \tau)$ is a recursion variable.*
2. *The explicit, irreducible term $A_{n+1} : \tau$ does not have any (free) occurrences of v (and p_{n+1}).*
3. *All the occurrences of p_{n+1}, in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$, are occurrences of the term $p_{n+1}(v)$, which are in the scope of λv (modulo appropriate renaming of v).*

Let $p'_{n+1} : \tau$ be a fresh recursion variable, and A' be a term as in (13b) *(by the η-rule):*

$$A \equiv [A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A},$$

$$p_{n+1} := \lambda v A_{n+1}, \tag{13a}$$

$$\overrightarrow{q} := \overrightarrow{B}\}]$$

$$\Rightarrow_{\eta}\ A' \equiv [A_0\{p_{n+1}(v) :\equiv p'_{n+1}\} \text{ where}$$

$$\{\overrightarrow{p} := \overrightarrow{A}\{p_{n+1}(v) :\equiv p'_{n+1}\}, \tag{13b}$$

$$p'_{n+1} := A_{n+1},$$

$$\overrightarrow{q} := \overrightarrow{B}\{p_{n+1}(v) :\equiv p'_{n+1}\}\}]$$

Then,

$$A \not\approx A' \tag{14a}$$

$$\mathsf{cf}(A') \equiv_{\mathsf{c}} A' \tag{14b}$$

and, for all $g \in G$, $i \in \{0,\ldots,n\}$, and $j \in \{1,\ldots,k\}$, the following denotational equalities hold:

$$\mathsf{den}(A)(g) = \mathsf{den}(A')(g) \tag{15}$$

and:

$$\mathsf{den}(A_i)(g\{\overrightarrow{p} := \overrightarrow{\overline{p}},\ p_{n+1} := \overline{p}_{n+1},$$

$$\overrightarrow{q} := \overrightarrow{\overline{q}}\}) \tag{16a}$$

$$= \mathsf{den}\Big(A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}\Big)(g\{\overrightarrow{p} := \overrightarrow{\overline{p'}},$$

$$p'_{n+1} := \overline{p'}_{n+1},\ \overrightarrow{q} := \overrightarrow{\overline{q'}}\}) \tag{16b}$$

and:

$$\mathsf{den}(B_j)(g\{\overrightarrow{p} := \overrightarrow{\overline{p}},\ p_{n+1} := \overline{p}_{n+1},\ \overrightarrow{q} := \overrightarrow{\overline{q}}\}) \tag{17a}$$

$$= \mathsf{den}\left(B_j\{p_{n+1}(v) :\equiv p'_{n+1}\}\right)(g\{\overrightarrow{p} := \overrightarrow{\overline{p'}},\ p'_{n+1} := \overline{p'}_{n+1},\ \overrightarrow{q} := \overrightarrow{\overline{q'}}\}) \tag{17b}$$

where, for all $i \in \{1,\ldots,n+1\}$ and $j \in \{1,\ldots,k\}$, the values $\overline{p}_i \in \mathbb{T}_{\sigma_i}$, $\overline{p'}_i \in \mathbb{T}_{\sigma_i}$, $\overline{q}_j \in \mathbb{T}_{\tau_j}$, and $\overline{q'}_j \in \mathbb{T}_{\tau_j}$ are calculated by recursion on the rank *of $\overrightarrow{p}$, p_{n+1}, $\overrightarrow{p'}$, p'_{n+1}, $\overrightarrow{q}$, $\overrightarrow{q'}$.*

Proof. The proof is long, by induction on the rank values, and is not in the subject of this paper. □

Definition 3 (η-reduction)**.** The η-*reduction* relation in $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is the smallest relation $\Rightarrow^*_{\eta}$ between $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms, such that:

(1) For any $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms A and B,
if $\mathsf{cf}(A) \Rightarrow_{\eta} B$, then $A \Rightarrow^*_{\eta} B$ (η-red)

(2) $\Rightarrow^*_{\eta}$ is closed under transitivity, congruence, and is compositional with respect to term formation rules, i.e.:

Transitivity If $A \Rightarrow^*_{\eta} B$ and $B \Rightarrow^*_{\eta} C$, then $A \Rightarrow^*_{\eta} C$ (η-tr)

Congruence If $A \equiv_c B$, then $A \Rightarrow^*_{\eta} B$ (η-cong)

Compositionality

- If $A \Rightarrow^*_{\eta} A'$ and $B \Rightarrow^*_{\eta} B'$, then $A(B) \Rightarrow^*_{\eta} A'(B')$ (η-rep1)
- If $A \Rightarrow^*_{\eta} B$, then $\lambda u\,(A) \Rightarrow^*_{\eta} \lambda u\,(B)$ (η-rep2)
- If $A_i \Rightarrow^*_{\eta} B_i$, for $i = 0, \ldots, n$, then (η-rep3)
  A_0 where $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$
  $\Rightarrow^*_{\eta} B_0$ where $\{\, p_1 := B_1, \ldots, p_n := B_n \,\}$

Definition 4 (η-equivalence relation $\approx_{\eta}$)**.** For any $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms A and B

$$A \approx_{\eta} B \iff \text{for some } C,\ \mathsf{cf}(A) \Rightarrow^*_{\eta} C \ \text{ and } \ C \approx B \tag{18}$$

Corollary 1. *For any $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms A and B,*

(1) if $A \Rightarrow^*_{\eta} B$, then $\mathsf{den}(A) = \mathsf{den}(B)$

(2) if $A \approx_{\eta} B$, then $\mathsf{den}(A) = \mathsf{den}(B)$

Proof. ((1)) is proved by induction on the definition of the $\Rightarrow^*_{\eta}$. Then, ((2)) follows by the Definition 4 of $\approx_{\eta}$. □

Corollary 2. *There exist (many) $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms A, B, and C such that*

$$A \Rightarrow B \Rightarrow^*_{\eta} C \Longrightarrow A \approx B \Longrightarrow A \approx_{\eta} B \approx_{\eta} C \tag{19a}$$

$$\text{while } C \not\approx B \text{ and } C \not\approx A \tag{19b}$$

4 USEFULNESS OF THE ETA-RULE

In this section, we present some arguments for the use of η-rule and η-reduction, for existing and potential applications.

Maintenance of Essential Algorithmic Computations. While the equalities (15), (16a), (17a) are about denotations, they do not use the denotational traditional β-conversion or any other syntactical manipulations over the terms A and A', except the η-rule for $A \Rightarrow^*_{\eta} A'$.

Preservance of Algorithmic Structure. The η-rule preserves very closely the computational structure of the term A in a canonical form. The term A', which is such that $A \Rightarrow^*_{\eta} A'$, is also in a canonical form, with parts that are almostly the same as the corresponding parts of A, with the exception of the replacements $\{p_{n+1}(v) :\equiv p'_{n+1}\}$ and skipped λv from the term part of $p_{n+1} := \lambda v A_{n+1}$ to the term part of $p'_{n+1} := A_{n+1}$.

The where assignments formalise some essentials of object declarations in object oriented programming languages, and in general, of function declarations in programming languages.

The equalities of the denotations of the corresponding parts, given in (16a)–(16b) and (17a)–(17b), are proved by recursion on rank. The denotations of the parts may, in general, strictly depend on the values of the recursion variables that have lesser rank. The denotational equality of the corresponding parts, e.g., of A_i and $A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}$, in (20a)–(20b), holds for the variable assignment g due to its update for the *local variables within the scope of* where, which are constrained by the recursion assignments.

$$\mathsf{den}(A_i)(g\{\overrightarrow{p} := \overrightarrow{\overline{p}},\ p_{n+1} := \overline{p}_{n+1},\ \overrightarrow{q} := \overrightarrow{\overline{q}}\}) \tag{20a}$$

$$= \mathsf{den}\left(A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}\right)(g\{\overrightarrow{p} := \overrightarrow{\overline{p'}},\ p'_{n+1} := \overline{p'}_{n+1},\ \overrightarrow{q} := \overrightarrow{\overline{q'}}\}) \tag{20b}$$

The recursion variables $\overrightarrow{p}$, p_{n+1}, $\overrightarrow{p'}$, p'_{n+1}, $\overrightarrow{q}$, $\overrightarrow{q'}$, are bound, i.e., *constrained* to the values determined by the assignments in the scope of where. In case that $\mathsf{rank}(p_i) > \mathsf{rank}(p_{n+1})$, the value $\mathsf{den}(A_i)(g)$ of the part A_i can depend on the value of p_{n+1}, and respectively, $\mathsf{den}\left(A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}\right)(g)$ on the value of p'_{n+1}. Without the constraints $p_{n+1} := \lambda v A_{n+1}$ and $p'_{n+1} := A_{n+1}$, a variable assignment g can be such that for some $a \in \mathbb{T}_{\sigma}$, $g(p_{n+1})(a) \neq g(p'_{n+1})$. Then, without the specified update of g, outside the scope of

where, i.e. outside the local system of assignments, it is possible that $\mathsf{den}(A_i)(g) \neq \mathsf{den}\big(A_i\{p_{n+1}(v) :\equiv p'_{n+1}\}\big)(g)$.

Local Scope of Recursion Operator. We explain the role of the local scope of the recursion operator designated by the constant where.

Let us consider again the terms in Example 3.1. The corresponding parts, which are subject of the η-rule, $\lambda y_k(some\big(d(y_k)\big)(h(y_k)))$, of the term (8b)–(8e), and $\lambda y_k(some\big(d'\big)(h(y_k)))$, of (9a)–(9b), depend on the values of d and d', respectively. The denotations of these terms are equal only within the local scopes of where, with the variable assignment updated, i.e., constrained by the corresponding assignments $d := \lambda y_k dog$ and $d' := dog$. Without these constraints, it is possible that, for some $g \in G$, we have the denotational inequality in (21a)–(21b):

$$\mathsf{den}\Big(\lambda y_k(some\big(d(y_k)\big)(h(y_k)))\Big)(g) \tag{21a}$$

$$\neq \mathsf{den}\Big(\lambda y_k(some\big(d'\big)(h(y_k)))\Big)(g) \tag{21b}$$

because $g(d) \in \mathbb{T}_{(\widetilde{\mathsf{e}}\to(\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}}))}$ and $g(d') \in \mathbb{T}_{(\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}})}$ can be any objects in these domains. For example, $\big(g(d)\big)(k)$ can depend on a, and it is possible that there is some $k \in \mathbb{T}_{\widetilde{\mathsf{e}}}$, such that $\big(g(d)\big)(k) \neq g(d')$ so that, the following holds:

$$\mathsf{den}\Big(\lambda y_k(some\big(d(y_k)\big)(h(y_k)))\Big)(k) \tag{22a}$$

$$\neq \mathsf{den}\Big(\lambda y_k(some\big(d'\big)(h(y_k)))\Big)(k) \tag{22b}$$

Then, from (22a) by the denotation function, den, we have (23a).

$$I(some)\Big(\big(g(d)\big)(k)\Big)\Big(\big(g(h)\big)(k)\Big) \tag{23a}$$

$$\neq I(some)\Big(g(d')\Big)\Big(\big(g(h)\big)(k)\Big) \tag{23b}$$

For all variable assignments g in a given semantic structure $\mathfrak{A}$, we get $\mathsf{den}(A)(g) = \mathsf{den}(A')(g)$, because the recursion variables $\overrightarrow{p}$, p_{n+1}, $\overrightarrow{p'}$, p'_{n+1}, $\overrightarrow{q}$, $\overrightarrow{q'}$, are bound, i.e., *constrained* to the values determined by the assignments in the scope of where. The term A carries the binding constraints together with the scope of where.

Theorem 4 is not just about denotational semantics of terms related by the η-rule. Many different terms (as in many formal languages) have the same denotations. Some manipulations and operators over terms, do not preserve denotations, i.e., they may produce new terms that have different denotations from the original terms over which they operate. While the η-rule does not preserve the original, strict algorithmic steps, i.e., the referential intension, Theorem 4 proves that the η-rule, applied on a canonical form $\mathsf{cf}(A)$, preserves its denotational semantics, by minimal divergence from the algorithmic steps determined by the original canonical form $\mathsf{cf}(A)$ on which it is applied. This additional reduction is useful for various tasks, e.g., in translations between NL expressions and generation of NL from semantic representations. Theorem 4 extended to the $\approx_\eta$ equivalence by Corollary1 shows that the $\approx_\eta$ equivalence is one of the many equivalence relations between terms, which is stronger than denotational equality and weaker than referential synonymy.

5 EXISTING AND POTENTIAL APPLICATIONS

In this section, we point to existing and potential applications of the Type-Theory of Acyclic Recursion $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, in areas that are within the subareas of Artificial Intelligence (AI). The reduction calculs of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ extended to η-reduction is useful in such applications, for reducing algorithmic complexity.

- programming languages: for algorithmic (procedural) semantics of programming languages
- compiler programming languages: for automatic conversion of recursive programs into iterative programs, where the reduction calculus is build into parts of compilation processing
- algorithm specifications with higher-order type theory of algorithms
- data science / database
- Computational Semantics (Loukanova, 2016)
- Syntax-Semantics Interface in NLP, computational grammar, and lexicon of human language (Loukanova, 2019d; Loukanova, 2019a)
- Language Processing / Technology
- Computational Neuroscience (Loukanova, 2017)

6 OUTLOOK AND FUTURE WORK

We have formulated the η-rule to simplify the canonical forms in some cases with occurrences of vacuous λ-abstractions and corresponding functional applications, by preserving all other structural components

of the canonical terms. The proof of Theorem 4 involves details of general canonical forms and induction steps. It demonstrates what part of a canonical form is reduced, by inessential divergence from the strictest referential synonymy of the entire term. It demonstrates that the replacements preserve the denotation of the entire term.

The presented η-rule has applications to computational semantics of human languages and to semantics of programs and compilers. Replacements based on the η-rule are not always possible because there are intervening (algorithmic) structures. The effect of a general β-replacement would have been like reversing iteration to recursive program interpreter. Something like a compiler that translates a program with recursion (i.e. a L^{λ}_{ar} term) to a program with induction, and then finds functional components that compute constant functions (not depending on inputs), and in attempts to provide the constant value, without using the "vacuous" function applications, is reversing the tail recursion into recursion. The presented η-rule avoids this.

In the analyses of certain classes of human language expressions, at least those that we have covered in recent work, the η-rule provides simplification of canonical terms that are otherwise irreducible.

REFERENCES

Gallin, D. (1975). *Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics*. North-Holland Publishing Company, Amsterdam and Oxford, and American Elsevier Publishing Company.

Loukanova, R. (2011a). From Montague's Rules of Quantification to Minimal Recursion Semantics and the Language of Acyclic Recursion. In Bel-Enguix, G., Dahl, V., and Jiménez-López, M. D., editors, *Biology, Computation and Linguistics*, volume 228 of *Frontiers in Artificial Intelligence and Applications*, pages 200–214. IOS Press, Amsterdam; Berlin; Tokyo; Washington, DC.

Loukanova, R. (2011b). Modeling Context Information for Computational Semantics with the Language of Acyclic Recursion. In Pérez, J. B., Corchado, J. M., Moreno, M., Julián, V., Mathieu, P., Canada-Bago, J., Ortega, A., and Fernández-Caballero, A., editors, *Highlights in Practical Applications of Agents and Multiagent Systems*, volume 89 of *Advances in Intelligent and Soft Computing*, pages 265–274. Springer Berlin Heidelberg.

Loukanova, R. (2011c). Reference, Co-reference and Antecedent-anaphora in the Type Theory of Acyclic Recursion. In Bel-Enguix, G. and Jiménez-López, M. D., editors, *Bio-Inspired Models for Natural and Formal Languages*, pages 81–102. Cambridge Scholars Publishing.

Loukanova, R. (2011d). Semantics with the Language of Acyclic Recursion in Constraint-Based Grammar. In Bel-Enguix, G. and Jiménez-López, M. D., editors, *Bio-Inspired Models for Natural and Formal Languages*, pages 103–134. Cambridge Scholars Publishing.

Loukanova, R. (2016). Relationships between Specified and Underspecified Quantification by the Theory of Acyclic Recursion. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 5(4):19–42.

Loukanova, R. (2017). Binding Operators in Type-Theory of Algorithms for Algorithmic Binding of Functional Neuro-Receptors. In Ganzha, M., Maciaszek, L., and Paprzycki, M., editors, *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*, volume 11 of *Annals of Computer Science and Information Systems*, pages 57–66. Polish Information Processing Society.

Loukanova, R. (2019a). Computational Syntax-Semantics Interface with Type-Theory of Acyclic Recursion for Underspecified Semantics. In Osswald, R., Retoré, C., and Sutton, P., editors, *IWCS 2019 Workshop on Computing Semantics with Types, Frames and Related Structures. Proceedings of the Workshop*, pages 37–48. The Association for Computational Linguistics (ACL).

Loukanova, R. (2019b). Gamma-Reduction in Type Theory of Acyclic Recursion. *Fundamenta Informaticae*, 170(4):367–411.

Loukanova, R. (2019c). Gamma-star canonical forms in the type-theory of acyclic algorithms. In van den Herik, J. and Rocha, A. P., editors, *Agents and Artificial Intelligence*, pages 383–407, Cham. Springer International Publishing.

Loukanova, R. (2019d). Syntax-semantics interfaces of modifiers. In Rodríguez, S., Prieto, J., Faria, P., Kłos, S., Fernández, A., Mazuelas, S., Jiménez-López, M. D., Moreno, M. N., and Navarro, E. M., editors, *Distributed Computing and Artificial Intelligence, Special Sessions, 15th International Conference*, pages 231–239, Cham. Springer International Publishing.

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

Moschovakis, Y. N. (1993). Sense and denotation as algorithm and value. In Oikkonen, J. and Väänänen, J., editors, *Logic Colloquium '90: ASL Summer Meeting in Helsinki*, volume Volume 2 of *Lecture Notes in Logic*, pages 210–249. Springer-Verlag, Berlin.

Moschovakis, Y. N. (1997). The logic of functional recursion. In Dalla Chiara, M. L., Doets, K., Mundici, D., and van Benthem, J., editors, *Logic and Scientific Methods*, volume 259, pages 179–207. Springer, Dordrecht.

Moschovakis, Y. N. (2006). A Logical Calculus of Meaning and Synonymy. *Linguistics and Philosophy*, 29(1):27–89.

Moschovakis, Y. N. (2019). *Abstract Recursion and Intrinsic Complexity*, volume 45 of *Lecture Notes in Logic*. Cambridge University Press.