Type-Theory of Acyclic Algorithms with Generalised Immediate Terms

Roussanka Loukanova
Stockholm University, Stockholm, Sweden

Keywords: Mathematics of Algorithms, Acyclic Recursion, Types, Algorithmic Semantics, Denotation, Immediate Terms, Immediate Denotation, Canonical Form.

Abstract: The paper extends the higher-order, type-theory L^{λ}_{ar} of acyclic algorithms by classifying some explicit terms as generalised immediate terms. We introduce a restricted, specialised iβ-rule and reduction of generalised immediate terms to their iβ-canonical forms. Then, we incorporate the iβ-rule in the reduction calculus of L^{λ}_{ar}. The new reduction calculus provides more efficient algoritms for computation of values of terms, by using iterative calculations.

1 INTRODUCTION

This paper is on development of a new approach to the mathematical notion of algorithm, by providing recursive computations as iteration using basic components in structured memory units. Originally, the approach was initiated by (Moschovakis, 1994). In recent years, related work has been reinitiated in new directions of higher-order type-theoriesof algorithms and information. A potential prospect for applications of the new approach is to data science and computational semantics of artificial and natural languages, from the perspective of AI. In particular, the theory of acyclic recursion L^{λ}_{ar}, see (Moschovakis, 2006), models the concepts of meaning and synonymy in typed models. The formal system L^{λ}_{ar} is a higher-order type theory, which is a proper extension of Gallin's TY_2, see (Gallin, 1975), and thus, of Montague's Intensional Logic (IL), see (Montague, 1973). The type theory L^{λ}_{ar} and its calculi extend Gallin's TY_2, at the level of the formal language and its semantics, by using several means: (1) two kinds of variables (*recursion variables*, called alternatively *locations*, and *pure variables*); (2) by formation of an additional kind of *recursion terms*; (3) systems of rules that form various calculi, i.e., reduction calculi and the calculus of algorithmic synonymy.

In the first part of the paper, we give the formal definitions of the syntax and denotational semantics of the language of L^{λ}_{ar}, by providing intuitive descriptions. Then, we give an informal description of the algorithmic semantics of L^{λ}_{ar}, and based on that, we present a formal introduction to the major notions of algorithmic equivalence between terms of L^{λ}_{ar}. In this introduction of L^{λ}_{ar}, we have reformulated some of its major theoretical characteristics, from the perspective of potential theoretical and practical developments, for applications to AI.

The second part of the paper is devoted to extending the reduction calculus of L^{λ}_{ar}. The purpose is to reduce the complexity of the algorithmic computations of the interpretations of L^{λ}_{ar} terms, by simplifying them. The new reduction system preserves the major characteristics of the algorithms, while reducing computational steps that are inessential from computational perspective.

Major characteristics and results of the theory of L^{λ}_{ar} and its original reduction system, are essential for the mathematical notion of algorithm, and related applications to algorithmic semantics, especially in AI areas. The first part of this paper introduces some of the major concepts of L^{λ}_{ar}, which are necessary for the new contribution in the second part.

2 SYNTAX AND SEMANTICS

Syntax of L^{λ}_{ar}. The formal language and calculus $L^{\lambda}_{ar}(K)$ properly extends a corresponding version of the classic typed λ-calculus, e.g., Ty_2 (Gallin, 1975; Gallin, 2011), and thus, of Montague Intensional Logic (IL) (Montague, 1973).

Types. The set Types is defined by the following rules of a Context-free Grammar (CFG), in Backus-Naur Form (BNF):

Loukanova, R.
Type-Theory of Acyclic Algorithms with Generalised Immediate Terms.
DOI: 10.5220/0007410207460754
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 746-754
ISBN: 978-989-758-350-6

$$T ::= \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (T \to T) \tag{1}$$

The type e is the type of the semantic entities and the expressions denoting entities; t of the truth values and corresponding expressions, s of the states. The types $(\tau \to \sigma)$ are the types of functions from objects of type τ to objects of type σ, and of expressions denoting such functions.

The language $\mathrm{L}^{\lambda}_{\mathrm{ar}}(K)$ has typed constants: $K = \bigcup_{\tau \in \mathsf{Types}} K_\tau$, where $K_\tau = \{\mathsf{c_0}^\tau, \dots, \mathsf{c}^\tau_k, \dots\}$. Distinctively, $\mathrm{L}^{\lambda}_{\mathrm{ar}}(K)$ is that it has two kinds of typed variables: $\mathsf{Vars} = \mathsf{PureVars} \cup \mathsf{RecVars}$, called *pure variables*, $\mathsf{PureVars}$, and *recursion variables*, $\mathsf{RecVars}$, so that for each $\tau \in \mathsf{Types}$, $\mathsf{PureVars}_\tau = \{\mathsf{v}_0, \mathsf{v}_1, \dots\}$ and $\mathsf{RecVars}_\tau = \{\mathsf{r}_0, \mathsf{r}_1, \dots\}$.

Terms. The set Terms of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is defined in a recursive style, with the type assignments either as superscripts or with colon sign:

$$A :\equiv \mathsf{c}^\tau \mid x^\tau \mid \tag{2a}$$
$$\big[B^{(\sigma\to\tau)}(C^\sigma)\big]^\tau \mid \tag{2b}$$
$$\big[\lambda v^\sigma (B^\tau)\big]^{(\sigma\to\tau)} \mid \tag{2c}$$
$$\big[A_0^{\sigma_0} \text{ where } \{\, p_1^{\sigma_1} := A_1^{\sigma_1}, \dots, p_n^{\sigma_n} := A_n^{\sigma_n} \,\}\big]^{\sigma_0} \tag{2d}$$

$\mathsf{c}^\tau \in K_\tau$; $x^\tau \in \mathsf{PureVars}_\tau \cup \mathsf{RecVars}_\tau$; $v^\sigma \in \mathsf{PureVars}_\sigma$; $A, B, A_i^{\sigma_i} \in \mathsf{Terms}$ $(i = 0, \dots, n)$; $p_i \in \mathsf{RecVars}_{\sigma_i}$ $(i = 1, \dots, n)$; $\{p_1^{\sigma_1} := A_1^{\sigma_1}, \dots, p_n^{\sigma_n} := A_n^{\sigma_n}\}$ is an acyclic sequence of assignments satisfying AC:

Acyclicity Constraint AC 1. For any given terms $A_1 : \sigma_1, \dots, A_n : \sigma_n$, and recursion variables $p_1 : \sigma_1, \dots, p_n : \sigma_n$, the set $\{p_1 := A_1, \dots, p_n := A_n\}$ is an *acyclic system of assignments* iff there is a ranking function $\mathsf{rank} : \{p_1, \dots, p_n\} \to \mathbb{N}$ such that, if p_j occurs freely in A_i then $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$.

Denotational Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$. The language $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ has denotational semantics that is given by a definition of a denotational function for semantic structures with typed domain frames. The denotation function of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is defined by the structure of the $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ terms.

A *semantic structure* (*model*) of L^{λ}_{ar} is a tuple $\mathfrak{A} = \langle \mathbb{T}, I \rangle$, where:

(S1) $\mathbb{T}$, called the *frame* of $\mathfrak{A}$, is a set of typed sets of objects:
$\mathbb{T} = \{\mathbb{T}_\sigma \mid \sigma \in \mathsf{Types}\}$ where $\mathbb{T}_e \neq \varnothing$ is a set of *entities*, $\mathbb{T}_t = \{0, 1, er\} \subseteq \mathbb{T}_e$ of *truth values*, $\mathbb{T}_s \neq \varnothing$ of *states*

(S2) $\mathbb{T}$ is a standard frame:
$\mathbb{T}_{(\tau_1 \to \tau_2)} = \{\, p \mid p \colon \mathbb{T}_{\tau_1} \to \mathbb{T}_{\tau_2} \,\}$

(S3) $I \colon K \to \cup\mathbb{T}$, is the *interpretation function* of $\mathfrak{A}$, and for every $\mathsf{c} \in K_\tau$, $I(\mathsf{c}) = c$, for some $c \in \mathbb{T}_\tau$

(S4) the set of the *variable assignments* or *valuations*, in $\mathfrak{A}$ is:

$$G = \{\, g \mid g \colon \mathsf{Vars} \to \cup\mathbb{T} \text{ and } g(x) \in \mathbb{T}_\sigma, \text{ for every } x \colon \sigma \} \tag{3}$$

Definition 1 (Denotation Function)**.** There is a function $\mathsf{den}^{\mathfrak{A}} \colon \mathsf{Terms} \to (G \to \bigcup\mathbb{T})$ defined by structural induction on $A \in \mathsf{Terms}_\sigma$.

(D1) *Variables and constants:*

$$\mathsf{den}^{\mathfrak{A}}(x)(g) = g(x), \text{ for all } x \in \mathsf{Vars} \tag{4a}$$
$$\mathsf{den}^{\mathfrak{A}}(\mathsf{c})(g) = I(\mathsf{c}), \text{ for all } c \in K \tag{4b}$$

(D2) *Application terms:*

$$\mathsf{den}^{\mathfrak{A}}(A(B))(g) = \mathsf{den}^{\mathfrak{A}}(A)(g)(\mathsf{den}^{\mathfrak{A}}(B)(g)) \tag{5}$$

(D3) *λ-abstraction terms:* For all $x : \tau$ and $B : \sigma$, $\mathsf{den}^{\mathfrak{A}}(\lambda(x)(B))(g) : \mathbb{T}_\tau \to \mathbb{T}_\sigma$ is the function such that, for every $t \in \mathbb{T}_\tau$,

$$[\mathsf{den}^{\mathfrak{A}}(\lambda(x)(B))(g)](t) = \mathsf{den}^{\mathfrak{A}}(B)(g\{x := t\}) \tag{6}$$

(D4) *Recursion terms:*

$$\mathsf{den}^{\mathfrak{A}}(A_0 \text{ where } \{\, p_1 := A_1, \dots, p_n := A_n \})(g) \tag{7a}$$
$$= \mathsf{den}^{\mathfrak{A}}(A_0)(g\{\, p_1 := \overline{p}_1, \dots, p_n := \overline{p}_n \}) \tag{7b}$$

where $\overline{p}_i \in \mathbb{T}_{\tau_i}$ are computed by recursion on $\mathsf{rank}(p_i)$, so that:

$$\overline{p}_i = \mathsf{den}^{\mathfrak{A}}(A_i)(g\{\, p_{i,1} := \overline{p}_{i,1}, \dots, p_{i,k_i} := \overline{p}_{i,k_i} \}) \tag{8}$$

where $p_{i,1}, \dots, p_{i,k_i}$ are the recursion variables $p_j \in \{p_1, \dots, p_n\}$ with $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$.

For a semantic structure $\mathfrak{A}$ with standard frame $\mathbb{T}$, the denotation function $\mathsf{den}^{\mathfrak{A}}$, defined by the above induction, exists and is unique.

We say that two terms $A, B \in \mathsf{Terms}$ are *denotationally equivalent*, and write $A = B$, when A and B have the same denotations, for all $\mathfrak{A}$ and variable valuations in $\mathfrak{A}$, i.e.:

for a given $\mathfrak{A}$:

$$\mathfrak{A} \models A = B \iff A \overset{\mathfrak{A}}{=} B \tag{9a}$$
$$\iff \mathsf{den}^{\mathfrak{A}}(A)(g) = \mathsf{den}^{\mathfrak{A}}(B)(g) \text{ for all } g \in G \text{ in } \mathfrak{A} \tag{9b}$$

$$A = B \iff \text{for all } \mathfrak{A},\ \mathfrak{A} \models A = B \tag{10}$$

In this paper, we assume that $\mathfrak{A}$ is a given, fixed structure. Often, when it is understood, we skip writing $\mathfrak{A}$, e.g., in the subscripts, and in $\mathsf{den}^{\mathfrak{A}}$, by den.

Proper vs. Immediate Terms. Obtaining the denotational value $\mathsf{den}(A)$ of a canonically immediate term $A \in \mathsf{CImT}$ does not involve any algorithmic steps for its calculations. It is computed immediately from the values of its free variables by a relevant valuation function $g \in G$, and by functional application, or λ-abstractions, according to Definition 1 of the denotation function. This is done without resorting to algorithmic steps and without extraction of denotation values from memory locations, where they have been saved after other computation steps.

On the other hand, the denotation $\mathsf{den}(A)$ of a proper term $A \in \mathsf{PrT}$ requires calculations by an algorithm depending on its syntactic structure, and using also Definition 1.

The denotation values of generalised immediate terms are obtained immediately, by employing their β-canonical forms, which are canonically immediate terms. That is, these denotations are obtained by using the valuation function for free variables and function applications to values, and also by forming new functions by λ-abstractions, according to Definition 1,

3 REDUCTION CALCULUS

Definition 2 (Congruence)**.** The *congruence* relation is the smallest equivalence relation $\equiv_{\mathsf{c}}$ between $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms:

1. reflexivity, symmetricity, transitivity
2. the term formation rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, for constants, variables, application, λ-abstraction, and acyclic recursion
3. renaming bound, pure and recursion variables, without causing variable collisions
4. re-ordering of assignments

Reduction Rules.

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:

$$\text{If } A \Rightarrow A' \text{ and } B \Rightarrow B', \text{ then } A(B) \Rightarrow A'(B') \tag{c-ap}$$

$$\text{If } A \Rightarrow B, \text{ then } \lambda(u)(A) \Rightarrow \lambda(u)(B) \tag{c-λ}$$

$$\begin{aligned}&\text{If } A_i \Rightarrow B_i, \text{ for } i = 0, \ldots, n, \text{ then}\\ &A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \}\\ \Rightarrow\ &B_0 \text{ where } \{ p_1 := B_1, \ldots, p_n := B_n \}\end{aligned} \tag{c-rec}$$

Head Rule: (head)

$$\begin{aligned}&\big(A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\}\big) \text{ where } \{\overrightarrow{q} := \overrightarrow{B}\}\\ \Rightarrow\ &A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A},\ \overrightarrow{q} := \overrightarrow{B}\}\end{aligned}$$

given that no p_i occurs freely in any B_j, for $i = 1, \ldots, n$, $j = 1, \ldots, m$

Bekič-Scott rule: (B-S)

$$\begin{aligned}&A_0 \text{ where } \{ p := \big(B_0 \text{ where } \{\overrightarrow{q} := \overrightarrow{B}\}\big),\\ &\qquad \overrightarrow{p} := \overrightarrow{A} \}\\ \Rightarrow\ &A_0 \text{ where } \{ p := B_0, \overrightarrow{q} := \overrightarrow{B},\\ &\qquad \overrightarrow{p} := \overrightarrow{A}\}\end{aligned}$$

given that no q_i occurs free in any A_j, for $i = 1, \ldots, n$, $j = 1, \ldots, m$

Recursion-application rule: (recap)

$$\begin{aligned}&(A_0 \text{ where } \{ \overrightarrow{p} := \overrightarrow{A} \})(B)\\ \Rightarrow\ &A_0(B) \text{ where } \{ \overrightarrow{p} := \overrightarrow{A} \}\end{aligned}$$

given that no p_i occurs free in B for $i = 1, \ldots, n$

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

λ-rule: (λ)

$$\begin{aligned}&\lambda(u)(A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \})\\ \Rightarrow\ &\lambda(u)A'_0 \text{ where } \{ p'_1 := \lambda(u)A'_1, \ldots,\\ &\qquad p'_n := \lambda(u)A'_n \}\end{aligned}$$

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.:

$$\begin{aligned}&A'_i \equiv A_i\{p_1 :\equiv p'_1(u), \ldots, p_n :\equiv p'_n(u)\}\\ &\text{for all } i \in \{1, \ldots, n\}\end{aligned} \tag{11}$$

Definition 3 (Reduction Relation, ⇒)**.**

- The *reduction relation* between terms is the smallest relation, denoted by ⇒, between terms that is closed under the reduction rules
- For any two terms A and B, A reduces to B, denoted by $A \Rightarrow B$, iff B can be obtained from A by a finite number of applications of reduction rules

Definition 4 (Term Irreducibility)**.** We say that a term $A \in \mathsf{Terms}$ is *irreducible* if and only if

$$\text{for all } B \in \mathsf{Terms}, \text{ if } A \Rightarrow B, \text{ then } A \equiv_{\mathsf{c}} B \tag{12}$$

The following theorems are major results that are essential for algorithmic semantics.

For every term $A \in \mathrm{L}^{\lambda}_{\mathrm{ar}}$, there is an irreducible term C, denoted by $\mathsf{cf}(A)$ and called the *canonical form* of A, which satisfies Theorem 1.

Theorem 1 (Extended Canonical Form Theorem: existence and uniqueness of the canonical forms)**.** *See (Moschovakis, 2006), § 3.1 and § 3.14. For every term $A \in \mathrm{L}^{\lambda}_{\mathrm{ar}}$, the following properties hold:*

1. *(Existence of a canonical form of A) There exist explicit, irreducible terms $A_0, \ldots, A_n$ ($n \geq 0$) such that*
$$\mathsf{cf}(A) \equiv A_0 \text{ where } \{ p_1 := A_1, \ldots, \\ p_n := A_n \}, \tag{13}$$
Thus, $\mathsf{cf}(A)$ *is irreducible*
2. *A constant* $\mathsf{c} \in K$ *or a variable* $x \in \mathsf{Vars}$ *occurs freely in* $\mathsf{cf}(A)$ *if and only if it occurs freely in A*
3. $A \Rightarrow \mathsf{cf}(A)$
4. *If A is irreducible, then* $\mathsf{cf}(A) \equiv_{\mathsf{c}} A$
5. *If* $A \Rightarrow B$*, then* $\mathsf{cf}(A) \equiv_{\mathsf{c}} \mathsf{cf}(B)$
6. *(Uniqueness of the canonical forms up to congruence) If* $A \Rightarrow B$ *and B is irreducible, then* $B \equiv_{\mathsf{c}} \mathsf{cf}(A)$*, i.e.,* $\mathsf{cf}(A)$ *is an unique, up to congruence, irreducible term*

Proof. By induction on term structure and using the reduction rules □

We write

$$A \Rightarrow_{\mathsf{cf}} B \iff B \equiv_{\mathsf{c}} \mathsf{cf}(A) \tag{14}$$
$$A \Rightarrow_{\mathsf{cf}} \mathsf{cf}(A) \tag{15}$$

Definition 5 (Syntactic Equivalence $\approx_s$)**.** For any $A, B \in \mathsf{Terms}$,

$$A \approx_s B \iff \mathsf{cf}(A) \equiv_{\mathsf{c}} \mathsf{cf}(B) \tag{16}$$

For more about syntactic synonymy (i.e., syntactic equivalence), see (Moschovakis, 2006). The difference between the syntactic synonymy $\approx_s$ and algorithmic synonymy $\approx$ in the selected $\mathfrak{A}(K)$, is that syntactic synonymy does not apply to denotationally equivalent constants and syntactic constructs such as λ-abstracts. For instance, assuming that *dog* and *canine* are constants, such that $\mathsf{den}^{\mathfrak{A}}(dog) = \mathsf{den}^{\mathfrak{A}}(canine)$, then $dog \approx canine$ (by the Referential Synonymy Theorem 1), because both terms are in canonical forms. On the other hand, $dog \not\approx_s canine$, since $dog \not\equiv_{\mathsf{c}} canine$. Also, $\mathsf{den}^{\mathfrak{A}}(dog) = \mathsf{den}^{\mathfrak{A}}(\lambda(x)dog(x))$ (by the clauses (D1), (D3) of the Definition 1 of the denotation function). Therefore, $dog \approx \lambda(x)dog(x)$ (by the Referential Synonymy Theorem 3), because both terms are in canonical forms. On the other hand, $dog \not\approx_s \lambda(x)dog(x)$, because $dog \not\equiv_{\mathsf{c}} \lambda(x)dog(x)$.

Theorem 2. *For any* $A, B \in \mathsf{Terms}$,

$$A \Rightarrow B \implies A \approx_s B \implies A \approx B \implies A = B \tag{17}$$

4 ALGORITHMIC SEMANTIC DATA

We shall consider a given, fixed $\mathfrak{A}(K)$, which takes a collection of data as its typed domains of objects and functions. The constants from K of $\mathrm{L}^{\lambda}_{\mathrm{ar}}(K)$ are interpreted in the typed domains determined by the data. For each term A, the variable valuations $g \in G$ provide values $g(\chi)$ of the free variables $\chi \in \mathsf{FreeV}(A)$.

4.1 On the Algorithmic Semantics

The notion of intension in the class of formal languages of recursion covers the most essential, computational aspect of the concept of meaning. The notion of algorithm in $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ is provided by formal syntax-semantics interface of the theory and calculi of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$.

Informally, for a given meaningful, i.e., proper, term $A \in \mathsf{Terms}$ and a semantic structure $\mathfrak{A}$, the *referential intension* of A is the *algorithm* for computing $\mathsf{den}(A)$ with interpretational reference to $\mathfrak{A}$. That is, $A \in \mathsf{Terms}$ is a mathematical 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 subterm A_0 and of the terms $A_1, \ldots, A_n$ in the system of assignments of its canonical form:

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

Two meaningful expressions are synonymous iff their referential intensions are naturally isomorphic, i.e., they are the same algorithms. Thus, the algorithmic meaning of a proper term (i.e., its algorithmic sense) is the information about how to "compute" its denotation step-by-step: a proper term A determines the algorithm for computing $\mathsf{den})A)$ by carrying instructions within its term structure, which are revealed by its canonical form $\mathsf{cf}(A)$, for computing what the parts denote in $\mathfrak{A}$ of a data system. Thus, the canonical form $\mathsf{cf}(A)$ of a proper $A \in \mathsf{Terms}$ determines the algorithmic steps for computing semantic denotations by using all necessary basic components of $\mathsf{cf}(A)$:

1. the basic instructions (facts), which consist of $\{p_1 := A_1, \ldots, p_n := A_n\}$, i.e., each $\mathsf{den}(A_i)(g)$ is computed and "saved" in p_i accordingly, by recursion with respect to $\mathsf{rank}(p_i)$
2. $\mathsf{den}(A_i)(g)$ of the head term A_0 is computed by using the values in p_i
3. finally $\mathsf{den}(A)(g) = \mathsf{den}(A_i)(g)$
4. the acyclicity constraint over each term A guarantees a terminating algorithm by the 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 $\mathsf{den}(A) = \mathsf{den}(A_0)$

Table 1: Algorithmic Semantics in L^{λ}_{ar}.

$$\underbrace{L^{\lambda}_{ar} \to \text{Computations: Algorithms} \to \text{Denotations}}_{\text{Algorithmic Semantics}}$$

$$A \Rightarrow_{\mathsf{cf}} \mathsf{cf}(A) \tag{18a}$$

$\mathsf{cf}(A)$ determines the *algorithm* for computing $\mathsf{den}(A)$

$$\mathsf{den}(A) = \mathsf{den}\big(\mathsf{cf}(A)\big) \tag{18b}$$

The class of formal languages of Moschovakis recursion offers formalisation of central computational aspects, by (at least) two semantic "levels":

1. algorithms, as mathematical objects, recursors, which are determined by the terms in canonical form
2. denotations, computed by recursive algorithms

The canonically immediate terms have canonical forms and denotations, but they do not determine any algorithms. For every canonically immediate term $A \equiv \lambda(\vec{x})\big(X(\vec{y})\big)$, where $X \in \mathsf{Vars}$, its denotation $\mathsf{den}(A)(g)$ is obtained immediately, from the variable values $g(y)$, for all $y \in \mathsf{FreeV}(A)$, without performing any designated algorithmic calculations.

In (Moschovakis, 2006), the recursor of a given $A \in \mathsf{Terms}$ is called its referential intension. We prefer to call these mathematical objects algorithms, e.g., recursive procedures, for computing denotations. This is more appropriate since it covers the mathematical concept of algorithm, from foundational view. We would try to avoid reloading the classic terminology of the concept of intension, which has been established since Montague's Intensional Logic (IL), see (Montague, 1973). In addition, the denotation function den in L^{λ}_{ar} covers the classic à la Montague notion of intension. For every state dependent type $(\mathsf{s} \to \tau)$:

$$\mathsf{den}(A)(g) \colon \mathbb{T}_s \to \mathbb{T}_\tau, \text{every term } A : (\mathsf{s} \to \tau) \tag{19}$$

The general scheme in Table 1 depicts the algorithmic semantics of L^{λ}_{ar} and its role for its denotational semantics.

4.2 Algorithmic Equivalence

Here, we give criteria for algorithmic equivalence between terms that are interpreted in a given $\mathfrak{A}(K)$. The selected semantic structure $\mathfrak{A}$ is based on a collection of data structured in typed domains of objects, with interpretations of constants and variable valuations in the data domains.

Theorem 3 (Algorithmic Equivalence in $\mathfrak{A}$)**.** *(This is a slightly extended version of the corresponding Referential Synonymy Theorem in (Moschovakis, 2006)) Two terms A, B are algorithmically (i.e., referentially) synonymous, $A \approx B$, if and only if one of the following cases holds:*

1. *both terms A and B are immediate and for all valuations $g \in G$, $\mathsf{den}(A)(g) = \mathsf{den}(B)(g)$*
2. *or, both A and B are proper terms, and there are explicit, irreducible terms of corresponding types, $A_i : \sigma_i$ and $B_i : \sigma_i$, $i = 1, \dots, n$ ($n \geq 0$) such that:*

$$A^{\sigma_0} \Rightarrow_{\mathsf{cf}} A_0^{\sigma_0} \text{ where } \{\, p_1 := A_1^{\sigma_1}, \dots, p_n := A_n^{\sigma_n} \,\} \tag{19a}$$

$$B^{\sigma_0} \Rightarrow_{\mathsf{cf}} B_0^{\sigma_0} \text{ where } \{\, p_1 := B_1^{\sigma_1}, \dots, p_n := B_n^{\sigma_n} \,\} \tag{19b}$$

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

$$\mathsf{den}(A_i)(g) = \mathsf{den}(B_i)(g), \text{ for all } g \in G \tag{20a}$$

Informally, A and B are algorithmically equivalent, $A \approx B$, in a given semantic structure $\mathfrak{A}$, if and only if one of the following cases holds

1. Both A and B are immediate, i.e., do not have any algorithmic meanings, and A and B have the same denotations in $\mathfrak{A}$
2. A and B are proper terms with algorithmic meanings, and the denotations of A and B, in $\mathfrak{A}$, are equal and computed by the same algorithm. It is determined by their canonical forms, $\mathsf{cf}(A)$ and $\mathsf{cf}(B)$, with corresponding basic algorithmic steps, represented by the denotationally equal, irreducible parts A_i and B_i. The denotations of A_i and B_i are computed inductively, i.e., by mutual recursion from the lowest up to the highest induction rank

In Table 2, the restrictions CRestr1(b) and CRestr2(c) are necessary.

Theorem 4 (Compositionality Theorem for algorithmic synonymy in a selected, fixed semantic structure $\mathfrak{A}$)**.** *For all $A \in \mathsf{Terms}_\sigma$, $B, C \in \mathsf{Terms}_\tau$, $x \in \mathsf{PureVars}_\tau$, such that the substitutions $A\{x :\equiv B\}$ and $A\{x :\equiv C\}$ are free, i.e., do not cause variable collisions:*

$$B \approx C \quad \longrightarrow \quad A\{x :\equiv B\} \approx A\{x :\equiv C\} \tag{21}$$

Proof. The proof is by induction on the term structure of A, by using the rules of referential synonymy in Table 2. □

Table 2: The calculus of algorithmic synonymy $\approx$, with respect to referential intensions in a fixed $\mathfrak{A}$.

$$\frac{A \approx_s B}{A \approx B} \qquad \text{(SynA)}$$

$$A \approx A \qquad \frac{B \approx A}{A \approx B} \qquad \frac{A \approx B \qquad B \approx C}{A \approx C} \qquad \text{(EqRel)}$$

$$\frac{A_1 \approx B_1 \qquad A_2 \approx B_2}{A_1(A_2) \approx B_1(B_2)} \qquad \text{(ApC)}$$

$$\frac{A \approx B}{\lambda(u)A \approx \lambda(u)B} \qquad (\lambda\text{C})$$

$$\frac{A_0 \approx B_0 \qquad A_1 \approx B_1 \qquad \dots \qquad A_n \approx B_n}{A_0 \text{ where } \{\vec{a} := \vec{A}\} \approx B_0 \text{ where } \{\vec{b} := \vec{B}\}} \qquad \text{(whC)}$$

$$\frac{\models C = D}{C \approx D} \;(C, D \text{ e.i., CRestr1}) \qquad \text{(eiEq)}$$

$$\frac{}{\big(\lambda(u)C\big)(v) \approx C\{u :\equiv v\}} \;(C \text{ e.i., CRestr2}) \qquad (\beta\text{Eq})$$

where "e.i." stands for "explicit, irreducible term(s)", and the restrictions CRestr1–CRestr2 hold:

CRestr1 in (eiEq):

(a) $\models C = D$ iff
for all $g \in G$, $\mathsf{den}^{\mathfrak{A}}(C)(g) = \mathsf{den}^{\mathfrak{A}}(B)(g)$

(b) C, D are e.i., both immediate or both proper

CRestr2 in (βEq):

(a) C is e.i. term

(b) $u, v \in \mathsf{PureVars}$, the substitution $C\{u :\equiv v\}$ is free

(c) $\big(\lambda(u)C\big)(v)$, $C\{u :\equiv v\}$ are both immediate, or both proper terms

5 GENERALISED IMMEDIATE TERMS

5.1 Canonically Immediate Terms

In this section, we generalise the concept of immediately obtained denotations.

A set of special terms, called immediate terms, has a significant role in the reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ and in the notion of algorithmic semantics. Informally, the immediate terms are formed only from variables of both kinds, by a succession of applications of a recursion variable to pure variables, which can be followed, on the top level by a succession of λ-abstractions.

At first, we shall define the set of the *canonically immediate terms* by Definition 22a using the Backus-Naur notational style, TBNF. In (Moschovakis, 2006), these terms, without allowing pure variables as applicant, are just all the immediate terms. The canonically immediate terms can have pure and recursion variables as the applicant of an applicative immediate term, while the arguments can only be pure variables.

Definition 6 (Canonically Immediate Terms, CImT, explicating types: TBNF Notational Style)**.** The set CImT of the canonically immediate terms consists of the terms defined as follows:

$$T^{\tau} :\equiv X^{\tau} \mid V^{(\tau_1 \to \dots \to (\tau_m \to \tau))}(v_1^{\tau_1}) \dots (v_m^{\tau_m}) \qquad (22a)$$

$$T^{(\sigma_1 \to \dots \to (\sigma_n \to \tau))} :\equiv \qquad (22b)$$
$$\lambda(u_1^{\sigma_1}) \dots \lambda(u_n^{\sigma_n}) V^{(\tau_1 \to \dots \to (\tau_m \to \tau))}(v_1^{\tau_1}) \dots (v_m^{\tau_m})$$

where $n \geq 0$, $m \geq 0$; $u_i \in \mathsf{PureVars}_{\sigma_i}$, for $i = 1, \dots, n$; $v_j \in \mathsf{PureVars}_{\tau_j}$, for $j = 1, \dots, m$; $V \in \mathsf{Vars}_{\tau}$, $V \in \mathsf{Vars}_{(\tau_1 \to \dots \to (\tau_m \to \tau))}$.

The above TBNF in Definition 6 can be given without mentioning the actual type assignments:

Definition 7 (Abbreviated CImT, without explicating types)**.**

$$T :\equiv V \mid V(v_1) \dots (v_m) \mid \qquad (23a)$$
$$\lambda(u_1) \dots \lambda(u_n) V(v_1) \dots (v_m) \qquad (23b)$$
$$m, n \geq 0,\ u_j, v_i, \in \mathsf{PureVars}, \qquad (23c)$$
$$V \in \mathsf{Vars} \qquad (23d)$$

Note that, in Definitions 6–7, T, X, V, u_i, v_i are metavariables. We can present the TBNF Definition 7 of the set CImT of the canonically immediate terms, by using a simple Context-free Grammar (CFG) in BNF, without the relevant type assignment:

Definition 8 (Canonically Immediate Terms, CImT: BNF)**.**

$$\mathsf{I}_{\mathsf{ci}} ::= \mathsf{A}_{ap} \mid \mathsf{A}_{\lambda} \qquad (24a)$$
$$\mathsf{A}_{ap} ::= \mathsf{V} \mid \mathsf{A}_{ap}(\mathsf{X}) \qquad (24b)$$
$$\mathsf{A}_{\lambda} ::= \lambda(\mathsf{X})(\mathsf{A}_{\lambda}) \mid \lambda(\mathsf{X})(\mathsf{A}_{ap}) \qquad (24c)$$
$$\mathsf{V} ::= \mathsf{X} \mid \mathsf{R} \qquad (24d)$$
$$\mathsf{R} ::= r, \text{ for each } r \in \mathsf{RecVars} \qquad (24e)$$
$$\mathsf{X} ::= x, \text{ for each } x \in \mathsf{PureVars} \qquad (24f)$$

In Definitions 8–9, the symbols I_{ci}, A_{ap}, etc., are nonterminals naming the corresponding to syntactic categories.

Definition 9 (Generalised Immediate Terms, GImT)**.** The set of the *generalised immediate terms*, GImT, is generated by a grammar, which is presented in BNF style notation (without the necessary type assignments), consisting of the rules in (25) by adding the ones from Definition 8.

$$\mathsf{I}_{\mathsf{gi}} ::= \mathsf{I}_{\mathsf{ci}} \mid \mathsf{I}_{\mathsf{gi}}(\mathsf{X}) \mid \lambda(\mathsf{X})(\mathsf{I}_{\mathsf{gi}}) \tag{25}$$

The generalised immediate terms are formed from variables of both kinds, in each type, and the operations application and λ abstraction, so that, recursively, the applicants are generalised terms, the arguments are pure variables, and the λ-abstraction over proper variables.

Example 5.1.

$$\Big[\lambda(x_1)\lambda(x_2)\dots\big(V(y_1)\dots(y_1)\big)\Big](z_{k_1})\dots(z_{k_m}) \tag{26a}$$

$$\lambda(\vec{u})\Big(\Big(\lambda(\vec{x})\big(X(\vec{y})\big)\Big)(\vec{z})\Big) \tag{26b}$$

$$\big(\lambda(u)C\big)(v) \tag{26c}$$

$$\big(\lambda(\vec{u})C\big)(\vec{v}) \tag{26d}$$

$$\big[\big(\lambda(\vec{u})C\big)(\vec{v})\big](\vec{z}) \tag{26e}$$

where $C \in \mathsf{GImT}$ is a generalised immediate term

Often, we shall say that a term A is immediate term when it is a generalised immediate term, i.e., $A \in \mathsf{GImT}$.

Definition 10 (Proper Terms). A term A is *proper* if it is not immediate:

$$\mathsf{PrT} = (\mathsf{Terms} - \mathsf{GImT}) \tag{27}$$

5.2 i-Beta Reduction

In this section, we extend the reduction calculus of L^{λ}_{ar} by

- Replacing the (ap)-rule with a new version for the new notion of immediate terms, i.e., generalised immediate terms
- Adding the following restricted iβ-rule to the reduction rules in Sect. 3.

  Restricted iβ-rule: For any $u, v \in \mathsf{PureVars}$ and any generalised immediate term $C \in \mathsf{GImT}$ such that the replacement $C\{u :\equiv v\}$ is free

$$\big(\lambda(u)(C)\big)(v) \Rightarrow_{\mathsf{i}\beta} C\{u :\equiv v\} \tag{iβ}$$

Definition 11 (iβ Reduction Relation, $\Rightarrow_{\mathsf{i}\beta}$).

- The iβ-*reduction relation* between terms is the smallest relation, denoted by $\Rightarrow_{\mathsf{i}\beta}$, between terms that is closed under the extended reduction rules, including the updated (ap)-rule and $\Rightarrow_{\mathsf{i}\beta}$
- For any two terms A and B, A iβ-reduces to B, denoted by $A \Rightarrow_{\mathsf{i}\beta} B$, iff B can be obtained from A by a finite number of applications of reduction rules, from the extended reduction calculus

Definition 12 (Term Irreducibility in $\Rightarrow_{\mathsf{i}\beta}$). We say that a term $A \in \mathsf{Terms}$ is iβ *irreducible* if and only if

$$\text{for all } B \in \mathsf{Terms}, \text{ if } A \Rightarrow_{\mathsf{i}\beta} B, \text{ then } A \equiv_{\mathsf{c}} B \tag{28}$$

For every term $A \in L^{\lambda}_{ar}$, there is an iβ-irreducible term C, which we denote by $\mathsf{ibcf}(A)$ and call a iβ-*canonical form* of A, which satisfies Theorem 5.

Theorem 5 (iβ-Canonical Form). *For every $A \in$ Terms, the following properties hold:*

(1) (Existence of an iβ-canonical form of A) There exist explicit, iβ-irreducible terms $A_0, \dots, A_n$ ($n \geq 0$) such that

$$\mathsf{ibcf}(A) \equiv A_0 \text{ where } \{\, p_1 := A_1, \dots, p_n := A_n \,\} \tag{29}$$

and thus, $\mathsf{ibcf}(A)$ is iβ-irreducible

(2) A constant c or a recursion variable $r \in \mathsf{RecVars}$ occurs freely in $\mathsf{ibcf}(A)$ if and only if it occurs freely in A; some pure variables may occur in A, but not in $\mathsf{ibcf}(A)$: $\mathsf{FreeV}(\mathsf{ibcf}(A)) \cap \mathsf{PureVars} \subseteq \mathsf{FreeV}(A) \cap \mathsf{PureVars}$

(3) $A \Rightarrow \mathsf{ibcf}(A)$

(4) If A is iβ-irreducible, then $\mathsf{ibcf}(A) \equiv_{\mathsf{c}} A$

(5) If $A \Rightarrow_{\mathsf{i}\beta} B$, then $\mathsf{ibcf}(A) \equiv_{\mathsf{c}} \mathsf{ibcf}(B)$

(6) (Uniqueness of the iβ-canonical forms up to congruence) If $A \Rightarrow B$ and B is iβ-irreducible, then $B \equiv_{\mathsf{c}} \mathsf{ibcf}(A)$, i.e., $\mathsf{ibcf}(A)$ is the unique, up to congruence, iβ-irreducible term

Proof. By induction on term structure and using the extended system of reduction rules

(1) is proved by structural induction on formation of terms and using the definition of the $\mathsf{ibcf}(A)$

(2) and (3) are proved by induction on terms and using the reduction rules

(4) by induction on the definition of the iβ reduction relation

(5) follows from (3) and (4)

Note that for some terms A, there may be pure variables which have free occurrences in A, but not in $\mathsf{ibcf}(A)$, i.e., there maty be $x \in \mathsf{FreeV}$, such that $x \in \mathsf{FreeV}(A)$, while $x \notin \mathsf{FreeV}(\mathsf{ibcf}(A))$. □

We write

$$A \Rightarrow_{\mathsf{ibcf}} B \iff B \equiv_{\mathsf{c}} \mathsf{ibcf}(A) \tag{30a}$$

$$A \Rightarrow_{\mathsf{ibcf}} \mathsf{ibcf}(A) \tag{30b}$$

Theorem 6 (Canonically Immediate Terms: existence and uniqueness)**.** *For each generalised immediate term $A \in \mathsf{GImT}$, there is a unique, up to congruence,* $\mathsf{i}\beta$*-irreducible term C, such that:*

1. *$C \in \mathsf{CImT}$ is canonically immediate term*
2. *$A \Rightarrow_{\mathsf{i}\beta} C$*
3. *if $A \Rightarrow_{\mathsf{i}\beta} B$ and B is* $\mathsf{i}\beta$*-irreducible, then $B \equiv_c C$, i.e., C is the unique, up to congruence,* $\mathsf{i}\beta$*-irreducible term to which A can be reduced*
4. *$C \equiv_c \mathsf{ibcf}(A)$, i.e., $A \Rightarrow_{\mathsf{ibcf}} \mathsf{ibcf}(A)$*

Proof. By induction on the structure of $A \in \mathsf{GImT}$, using Definition 9, which extends Definition 8, and Theorem 5.

□

The terms in Example (5.1) are not canonically immediate, while they are explicit, irreducible terms in canonical forms, in the original reduction calculus of L^{λ}_{ar}. Thus, they have algorithmic meanings. Their denotations are computed by the denotation function den over a sequence of λ-abstractions and applications. Some of these terms are algorithmically equivalent to simpler terms by the rule (βEq) in Table 2. In this paper we have extended the reduction calculus of L^{λ}_{ar}, to $\Rightarrow_{\mathsf{i}\beta}$-reduction. The purpose of this is effective reduction of such terms A, which are not per se immediate in the original system of L^{λ}_{ar}, to canonically immediate terms $\mathsf{ibcf}(A)$. By this, we reduce their structure to simpler, canonically immediate terms. The denotations of such terms, $\mathsf{den}(A) = \mathsf{den}(\mathsf{ibcf}(A))$, are obtained directly, immediately by $\mathsf{den}(\mathsf{ibcf}(A))$. This reduces the complexity of terms in which they occur as sub-terms.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced $\mathsf{i}\beta$-rule and $\Rightarrow_{\mathsf{i}\beta}$-reduction for the purpose of reducing complexity of algorithmic computations. The $\mathsf{i}\beta$ rule and its reduction system, $\Rightarrow_{\mathsf{i}\beta}$, reduces generalised immediate terms $A \in \mathsf{GImT}$, e.g., such as the ones in Example (5.1), and other terms in which they occur, to simpler terms.

An important purpose of the introduced canonically immediate and generalised immediate terms, by Definitions 6–9, concerns technical details. The calculus of algorithmic synonymy $\approx$ in Table 2, which is introduced by (Moschovakis, 2006), covers more than the original reduction calculus to canonical forms. The rules (eiEq) and the restricted β-reduction (βEq) provide algorithmic equivalence of limited, explicit irreducible terms, by appealing to finding their semantic denotations. The canonically immediate terms provide the denotations of respective generalised immediate terms, without loosing any essential algorithmic steps and declaratively, which remain in $\mathsf{i}\beta$-reductions. The $\mathsf{i}\beta$-reduction introduced here, complements the algorithmic semantics in this aspect. Technical details are beyond the scope of this paper and will be provided in extended work.

The $\Rightarrow_{\mathsf{i}\beta}$-reduction system is introduced in this work for the first time, up to our knowledge. The next direct line of work is to investigate more characteristics of the $\mathsf{i}\beta$-rule and $\Rightarrow_{\mathsf{i}\beta}$-reduction system.

In future, extended work, we shall investigate how the $\mathsf{i}\beta$ reduction rule can be incorporated with other extended reduction systems of L^{λ}_{ar}. Of particular interests is upcoming work on integrating the results from this paper on $\mathsf{i}\beta$-rule and $\Rightarrow_{\mathsf{i}\beta}$-reduction with work in (Loukanova, 2016a; Loukanova, 2016b; Loukanova, 2018).

The results in this paper are on theoretical developments for more efficient and adequate formalisation of computations based on formal and computer languages, for applications to advanced, intelligent technologies in AI.

REFERENCES

Gallin, D. (1975). *Intensional and Higher-Order Modal Logic*. North-Holland.

Gallin, D. (2011). *Intensional and higher-order modal logic: with applications to Montague semantics*, volume 19. Elsevier.

Loukanova, R. (2016a). Acyclic Recursion with Polymorphic Types and Underspecification. In van den Herik, J. and Filipe, J., editors, *Proceedings of the 8th International Conference on Agents and Artificial Intelligence*, volume 2, pages 392–399. SCITEPRESS — Science and Technology Publications, Lda.

Loukanova, R. (2016b). 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. (2018). Gamma-star Reduction in the Type-theory of Acyclic Algorithms. In Rocha, A. P. and van den Herik, J., editors, *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018)*, volume 2, pages 231–242. INSTICC, SciTePress — Science and Technology Publications, Lda.

Montague, R. (1973). The Proper Treatment of Quantification in Ordinary English. In Hintikka, J., Moravcsik, J., and Suppes, P., editors, *Approaches to Natural Language*, pages 221–242. D. Reidel Publishing Co., Dordrecht, Holland.

Moschovakis, Y. N. (1994). Sense and denotation as algorithm and value. In Oikkonen, J. and Vaananen, J., editors, *Lecture Notes in Logic*, number 2 in Lecture Notes in Logic, pages 210–249. Springer.

Moschovakis, Y. N. (2006). A logical calculus of meaning and synonymy. *Linguistics and Philosophy*, 29(1):27–89.