Gamma-star Reduction in the Type-theory of Acyclic Algorithms

Roussanka Loukanova
Department of Philosophy, Stockholm University, Stockholm, Sweden

Keywords: Mathematics Of Algorithms, Recursion, Types, Semantics, Algorithmic Semantics, Denotation, Canonical Computations.

Abstract: The paper extends a higher-order type theory of acyclic algorithms by adding a reduction rule, which results in a stronger reduction calculus. The new reduction calculus determines a strong algorithmic equivalence between formal terms. It is very useful for simplifying terms, by eliminating sub-terms having superfluous lambda abstraction and corresponding spurious functional applications.

1 INTRODUCTION

In a sequence of papers, see (Moschovakis, 1989; Moschovakis, 1994; Moschovakis, 1997), Yiannis Moschovakis introduced a new approach to the mathematical notion of algorithm, by the concept of recursion and mutually recursive computations. The approach, which we call Theory of Moschovakis Recursion, uses a formal language of recursion and fine-grained semantic distinctions between denotations of formal terms and algorithms for computing the denotations, correspondingly by two semantic layers: denotational and algorithmic semantics. The initial work on Moschovakis Recursion was on untyped theory of algorithms and is the basis of (Moschovakis, 2006), which introduced typed theory L_{ar}^{λ} of acyclic recursion, as formalization of the concepts of algorithmic meaning in typed models. The theory L_{ar}^{λ} was introduced in (Moschovakis, 2006) by considering its potentials for applications to algorithmic semantics of human language, analogously to semantics of programming languages, where a given denotation can be computed by different algorithms.

Our ongoing work on extending the expressiveness of L_{ar}^{λ} develops a class of formal languages and theories of typed acyclic recursion, which cover various computational aspects of the mathematical concept of algorithm. We target development of formal systems and calculi, which have applications in contemporary intelligent systems. In particular, we develop a class of type theories of algorithms, which have more adequate computational applications in AI, by covering context dependent algorithms that depend on AI agents and other contextual parameters. For instance, such work was initiated in (Loukanova, 2011d; Loukanova, 2013a; Loukanova, 2016c).

Collectively, the classes of typed formal languages and theories of Moschovakis acyclic, and, respectively, full recursion are formal systems, which we call *typed theory of acyclic recursion* (TTofAR), and, respectfully, *typed theory of (full) recursion* (TTofR).

TTofAR has potentials for applications to algorithmic semantics of formal and natural languages. Among the formal languages, we consider applications of typed theory of recursion to semantics of programming languages, formalisation of compilers, languages used in database systems, and many areas of Artificial Intelligence (AI), including robotics. The untyped theory of recursion (Moschovakis, 1997; Moschovakis, 1989) is applied in (Hurkens et al., 1998) to model reasoning. The potentials of L_{ar}^{λ}, with typed acyclic recursion, have been demonstrated for various applications, in particular, to computational semantics of human language. Application of L_{ar}^{λ} to logic programming in linguistics and cognitive science is given in (Hamm and van Lambalgen, 2004).

A sequence of papers initiated extending the original L_{ar}^{λ}, and provide applications of L_{ar}^{λ} to computational semantics and computational syntax-semantics interface of human language, see (Loukanova, 2013c; Loukanova, 2013b; Loukanova, 2013a; Loukanova, 2012a; Loukanova, 2012b; Loukanova and Jiménez-López, 2012; Loukanova, 2011a; Loukanova, 2011c; Loukanova, 2011d; Loukanova, 2011e; Loukanova, 2011f; Loukanova, 2011g; Loukanova, 2011b), (Loukanova, 2016b; Loukanova, 2016a; Loukanova, 2016c; Loukanova, 2015b). By adding polymorphism, the work in (Loukanova, 2016a) offers potentials for varieties of applications with polymorphic, or otherwise parametric types. The work in (Loukanova, 2014; Loukanova, 2015b; Loukanova, 2015a; Loukanova, 2017b) provides a formal tech-

Loukanova, R.
Gamma-star Reduction in the Type-theory of Acyclic Algorithms.
DOI: 10.5220/0006662802310242
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 231-242
ISBN: 978-989-758-275-2

nique for applications of L^{λ}_{ar} and its extended versions to data science, for representation of factual and situated content of underspecified and partial information. The work (Loukanova, 2017a) extends L^{λ}_{ar} and provides a basis for applications to computational neuroscience for mathematical medeling of neural structures and connections.

In this paper, at first, we present the original reduction calculus of the type theory of acyclic recursion, which effectively reduces terms to their canonical forms. This reduction calculus determines a strict algorithmic equivalence between L^{λ}_{ar}-terms. In the rest of the paper, we present our contribution. We extend the reduction calculus of L^{λ}_{ar} to a stronger γ^*-*reduction calculus*. We call, and denote it also *gamma-star reduction calculus*. The new γ^*-reduction system is very useful for simplifying terms, by eliminating subterms having superfluous lambda abstraction and corresponding spurious functional applications. We give motivation by using abstract examples, because the theory has broad applications in technologies. For better understanding, we give supplementary examples that render expressions of human language to L^{λ}_{ar}-terms. In addition, the theory has direct potentials for applications to computerised processing of human language, including large-scale, computational grammars of human language and NLP for AI.

2 SYNTAX AND SEMANTICS OF TYPED-THEORY OF ACYCLIC RECURSION

2.1 Syntax

Basic Types: $\mathsf{BTypes} = \{\mathsf{e}, \mathsf{t}, \mathsf{s}\}$
The basic type e is the type of the entities in the semantic domains and the expressions denoting entities; t of the truth values and corresponding expressions, s of the states.

Types: the set Types (i.e., the set of type terms) is the smallest set defined recursively, by using the Backus-Naur Form (BNF) notation as follows:

$$\theta :\equiv \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid \sigma \mid (\tau \to \sigma) \tag{1}$$

The type terms $(\tau \to \sigma)$ are the types of functions from objects of type τ to objects of type σ, and of expressions denoting such functions.

$$\mathsf{e} \to \mathsf{t}, \quad \text{the type of characteristic functions of sets of entities} \tag{2a}$$

$$\widetilde{\mathsf{e}} \equiv (\mathsf{s} \to \mathsf{e}), \quad \text{of state dependent entities} \tag{2b}$$

$$\widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t}), \quad \text{of state dependent truths} \tag{2c}$$

$$\widetilde{\tau} \equiv (\mathsf{s} \to \tau), \quad \text{of state dependent objects of type } \tau \tag{2d}$$

The vocabulary of L^{λ}_{ar} consists of

Constants:
$K = \bigcup_{\tau \in \mathsf{Types}} K_\tau$,
where, for each $\tau \in \mathsf{Types}$, $K_\tau = \{\mathsf{c}_0{}^\tau, \ldots, \mathsf{c}_k^\tau, \ldots\}$

Pure variables:
$\mathsf{PureVars} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{PureVars}_\tau$,
where, for $\tau \in \mathsf{Types}$, $\mathsf{PureVars}_\tau = \{\mathsf{v}_0, \mathsf{v}_1, \ldots\}$

Recursion variables:
$\mathsf{RecVars} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{RecVars}_\tau$,
where, for each $\tau \in \mathsf{Types}$, $\mathsf{RecVars}_\tau = \{\mathsf{r}_0, \mathsf{r}_1, \ldots\}$

The terms of L^{λ}_{ar}: The set Terms of L^{λ}_{ar} is defined by recursion expressed by using a typed variant of BNF, with the type assignments given either as superscripts or with column sign:

$$A :\equiv \mathsf{c}^\tau \mid x^\tau \mid \tag{3a}$$

$$\left[B^{(\sigma\to\tau)}(C^\sigma)\right]^\tau \mid \tag{3b}$$

$$\left[\lambda v^\sigma (B^\tau)\right]^{(\sigma\to\tau)} \mid \tag{3c}$$

$$\left[A_0^{\sigma_0} \text{ where } \{\, p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n} \,\}\right]^{\sigma_0} \tag{3d}$$

where for $n, m \geq 0$, $\mathsf{c}^\tau \in K_\tau$ is a constant; $x^\tau \in \mathsf{PureVars}_\tau \cup \mathsf{RecVars}_\tau$ is a pure or recursion variable; $v^\sigma \in \mathsf{PureVars}_\sigma$ is a pure variable; $A, B, A_i^{\sigma_i} \in \mathsf{Terms}$ $(i = 0, \ldots, n)$ are terms of the respective types; $p_i \in \mathsf{RecVars}_{\sigma_i}$ $(i = 1, \ldots, n)$ are pairwise different recursion variables; and, in the expressions of the form (3d), the subexpression $\{p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n}\}$ is a sequence of assignments that satisfies the *acyclicity constraint*:

Acyclicity Constraint AC 1. For any given terms $A_1 : \sigma_1, \ldots, A_n : \sigma_n$, and recursion variables $p_1 : \sigma_1, \ldots, p_n : \sigma_n$, the sequence $\{p_1 := A_1, \ldots, p_n := A_n\}$ is an *acyclic system of assignments* iff there is a ranking function $\mathsf{rank} : \{p_1, \ldots, p_n\} \longrightarrow \mathbb{N}$ such that, for all $p_i, p_j \in \{p_1, \ldots, p_n\}$,

$$\begin{array}{l} \text{if } p_j \text{ occurs freely in } A_i \\ \text{then } \mathsf{rank}(p_j) < \mathsf{rank}(p_i) \end{array} \tag{4}$$

Usually, the type assignments in the term expressions are skipped. The terms of the form (3d), are called *recursion terms*:

$$[A_0^\sigma \text{ where } \{p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n}\}]^{\sigma_0} \tag{5a}$$

$$\equiv (A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}) \tag{5b}$$

$$\equiv A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\} \tag{5c}$$

2.2 Dynamic Specifications in Context by Agents

We shall render the pronoun "his" into a simple recursion variable h for the purpose of the demonstration. More adequate treatment of the pronouns is not in the subject of this paper. Here we consider general underspecification of terms via free recursion variables occurring in $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ -terms. Using recursion variables allows the recursion terms that contain them, like (6c), to be expended by adding assignments $h := H$, for some appropriate term H. What is the term H depends on context, and can be specified dynamically, when more detailed information is provided, e.g., by discourse or explicitly by users. Here we do not treat details of such contextual contributions, by focusing on the possibility to represent underspecification by "underspecified" $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms containing free recursion variables. Using free recursion variables in terms gives potentials for its algorithmic resolving of underspecification by adding assignments to canonical terms.

The reductions in this section can be done by using the reduction rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, which are given in Section (3).

Example 2.1 (Underspecified pronouns)**.**

$$\text{John greeted his wife.} \tag{6a}$$

$$\xrightarrow{\text{render}} \mathit{greeted}(\mathit{wife\text{-}of}(h))(\mathit{john}) \tag{6b}$$

$$\Rightarrow_{\mathsf{cf}} \mathit{greeted}(w)(j) \text{ where } \{\, j := \mathit{john},\ w := \mathit{wife\text{-}of}(h)\,\} \tag{6c}$$

The recursion variable h in (6c) is left free, which we use to represents semantic underspecification of the sentence (6a), in absence of context with an agent that lacks information to interpret the pronoun. If left unbound by any assignment, h can be interpreted as a deictic pronoun obtaining its denotational referents by the agents' references provided by context information, e.g., by adding $h := \mathit{tom}$ to the assignments in the term (7c)–(7e). The result is the dynamic specification of the underspecified term (7b), i.e., (6c), and its canonical form (7c)–(7e), to the sully specified term (7f)–(7i).

$$\text{John greeted his wife.} \tag{7a}$$

$$\xrightarrow{\text{render}} \mathit{greeted}(\mathit{wife\text{-}of}(h))(\mathit{john}) \tag{7b}$$

$$\Rightarrow_{\mathsf{cf}} \mathit{greeted}(w)(j) \text{ where } \{ \tag{7c}$$

$$j := \mathit{john}, \tag{7d}$$

$$w := \mathit{wife\text{-}of}(h)\,\} \tag{7e}$$

$$\Rightarrow_{\mathit{context}} \mathit{greeted}(w)(j) \text{ where } \{ \tag{7f}$$

$$j := \mathit{john}, \tag{7g}$$

$$w := \mathit{wife\text{-}of}(h), \tag{7h}$$

$$h := \mathit{tom}\,\} \tag{7i}$$

An alternative, anaphoric reading of (6a) can be obtained by adding the assignment $h := j$ to the term (6b), and thus to the system of assignments in its canonical form (6c), e.g., dynamically, after the agent has obtained relevant information. The result is the specified term (8c)–(8f):

$$\text{John greeted his (own) wife.} \tag{8a}$$

$$\xrightarrow{\text{render}} \mathit{greeted}(\mathit{wife\text{-}of}(h))(\mathit{john}) \tag{8b}$$

$$\Rightarrow_{\mathit{context}} \mathit{greeted}(w)(j) \text{ where } \{ \tag{8c}$$

$$j := \mathit{john}, \tag{8d}$$

$$w := \mathit{wife\text{-}of}(h), \tag{8e}$$

$$h := j\,\} \tag{8f}$$

2.3 Semantics

Denotational Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$: An L^{λ}_{ar} *semantic structure*, also called *model*, is a tuple $\mathfrak{A} = \langle \mathbb{T}, I \rangle$, satisfying the following conditions (S1)–(S4):

(S1) $\mathbb{T}$ is a set, called a *frame*, of sets

$$\mathbb{T} = \{\mathbb{T}_{\sigma} \mid \sigma \in \mathit{Types}\} \tag{9}$$

where $\mathbb{T}_e \neq \varnothing$ is a nonempty set of entities, $\mathbb{T}_t = \{0, 1, er\} \subseteq \mathbb{T}_e$ is the set of the *truth values*, $\mathbb{T}_s \neq \varnothing$ is a nonempty set of objects called *states*

(S2) $\mathbb{T}_{(\tau_1 \to \tau_2)} = \{\, p \mid p\colon \mathbb{T}_{\tau_1} \longrightarrow \mathbb{T}_{\tau_2}\,\}$

(S3) I is a function $I\colon K \longrightarrow \mathbb{T}$, called the *interpretation function* of $\mathfrak{A}$, such that for every $\mathsf{c} \in K_{\tau}$, $I(\mathsf{c}) = c$ for some $c \in \mathbb{T}_{\tau}$

(S4) The set G of the variable assignments for the semantic structure $\mathfrak{A}$ is:
$G = \{\, g \mid g\colon \mathit{PureVars} \cup \mathit{RecVars} \longrightarrow \mathbb{T}$ and $g(x) \in \mathbb{T}_{\sigma}$, for every $x\colon \sigma\,\}$

Definition 1 (Denotation Function)**.** The *denotation function* den, when it exists, of the semantics structure $\mathfrak{A}$, is a function:

$$\mathsf{den}\colon \mathit{Terms} \longrightarrow \{\, f \mid f\colon G \longrightarrow \mathbb{T}\,\} \tag{10}$$

which is defined, for each $g \in G$, by induction on the structure of the terms, as follows:

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

(D2) for application terms

$$\mathsf{den}([B^{(\sigma\to\tau)}(C^{\sigma})]^{\tau})(g) = \mathsf{den}(B)(g)(\mathsf{den}(C)(g)) \tag{11}$$

(D3) for λ-terms
$\mathsf{den}([\lambda v^{\sigma}(B^{\tau})]^{(\sigma\to\tau)})(g) : \mathbb{T}_{\tau} \longrightarrow \mathbb{T}_{\sigma}$,
where $x : \tau$ and $B : \sigma$, is the function such that, for every $t \in \mathbb{T}_{\tau}$:

$$[\mathsf{den}([\lambda v^{\sigma}(B^{\tau})]^{(\sigma\to\tau)})(g)](t) \tag{12}$$

$$= \mathsf{den}(B)(g\{x := t\}) \tag{13}$$

(D4) for recursion terms

$$\mathsf{den}([A_0^{\sigma_0} \text{ where } \{\, p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n} \,\}]^{\sigma_0}) \tag{14a}$$

$$= \mathsf{den}(A_0)(g\{p_1 := \overline{p}_1, \ldots, p_n := \overline{p}_n\}) \tag{14b}$$

where for all $i \in \{1,\ldots,n\}$, $\overline{p}_i \in \mathbb{T}_{\tau_i}$ are defined by recursion on $\mathsf{rank}(p_i)$, so that:

$$\overline{p_i} = \mathsf{den}(A_i)(g\{p_{k_1} := \overline{p}_{k_1}, \ldots, p_{k_m} := \overline{p}_{k_m}\}) \tag{15}$$

where $p_{k_1}, \ldots, p_{k_m}$ are all the recursion variables $p_j \in \{p_1, \ldots, p_n\}$ such that $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$

Intuitively, a system $\{p_1 := A_1, \ldots, p_n := A_n\}$ defines recursive computations of the values to be assigned to the locations $p_1, \ldots, p_n$. When p_j occurs freely in A_i, the denotational value of A_i, which is assigned to p_i, may depend on the values of the variable p_j, as well as on the values of the variables p_k having lower rank than p_j. Requiring a ranking function rank, such that $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$, i.e., an acyclic system guarantees that computations end after finite number of steps. Omitting the acyclicity condition gives an extended type system $\mathrm{L}_{\mathrm{r}}^{\lambda}$, which admits full recursion. This is not in the subject of this paper.

Algorithmic Semantics: The notion of algorithmic meaning (algorithmic semantics) in the languages of recursion covers the most essential, computational aspect of the concept of meaning. The *algorithmic meaning*, $\mathsf{Int}(A)$, 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 $A_1, \ldots, A_n$ in the system of assignments of its canonical form (see the next sections) $\mathsf{cf}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$. Intuitively, for each meaningful term A, the algorithmic meaning $\mathsf{Int}(A)$ of A, is the mathematical *algorithm* for computing the denotation $\mathsf{den}(A)$.

Two meaningful expressions A and B are algorithmically equivalent, $A \approx B$ i.e., algorithmically synonymous iff their recursors $\mathsf{Int}(A)$ and $\mathsf{Int}(B)$ are naturally isomorphic, i.e., they are the same algorithms. Thus, the formal languages of recursion offer a formalisation of central computational aspects: denotation, with at least two semantic "levels": *algorithmic meanings* and *denotations*. The terms in canonical form represent the algorithmic steps for computing semantic denotations.

3 REDUCTION CALCULUS

Definition 2 (Congruence Relation)**.** For any terms $A, B \in \mathsf{Terms}$, A and B are congruent, $A \equiv_{\mathsf{c}} B$, if and only if one of them can be obtained from the other by renaming bound variables and reordering assignments in recursion terms.

3.1 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:

If $A \Rightarrow A'$ and $B \Rightarrow B'$, then $A(B) \Rightarrow A'(B')$ (c-ap)

If $A \Rightarrow B$, then $\lambda(u)(A) \Rightarrow \lambda(u)(B)$ (c-λ)

If $A_i \Rightarrow B_i$, for $i = 0, \ldots, n$, then

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

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 $i = 1, \ldots, n$, $j = 1, \ldots, m$

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_i occurs free in any A_j, for $i = 1, \ldots, n$, $j = 1, \ldots, m$

Recursion-application rule: (recap)

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

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)

$$\lambda(u)(A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}) \Rightarrow \lambda(u)A_0' \text{ where } \{\, p_1' := \lambda(u)A_1', \ldots, p_n' := \lambda(u)A_n' \,\}$$

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.:

$$A'_i \equiv A_i\{p_1 :\equiv p'_1(u), \ldots, p_n :\equiv p'_n(u)\} \quad \text{for all } i \in \{1, \ldots, n\} \tag{20}$$

Definition 3. The *reduction relation* is the smallest relation between terms that is closed under the reduction rules.

The reduction relation is denoted by $\Rightarrow$. That is, for any two terms A and B, A reduces to B, denoted by $A \Rightarrow B$, iff B can be obtained from A by 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

$$\text{for all } B \in \mathsf{Terms}, \text{ if } A \Rightarrow B, \text{ then } A \equiv_{\mathsf{c}} B \tag{21}$$

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

Theorem 1 (Canonical Form Theorem: existence and uniqueness of the canonical forms)**.** *(Moschovakis, 2006) 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* $B \equiv_{\mathsf{c}} \mathsf{cf}(A)$*, i.e.,* $\mathsf{cf}(A)$ *is the unique, up to congruence, irreducible term to which A can be reduced.*

Theorem 2 (Referential Synonymy Theorem)**.** *(See (Moschovakis, 2006)) Two terms* A, B *are algorithmically equivalent, i.e., synonymous,* $A \approx B$*, if and only if there are explicit, irreducible terms of corresponding types,* $A_0 : \sigma_0, \ldots, A_n : \sigma_n$, $B_0 : \sigma_0, \ldots, B_n : \sigma_n$ $(n \geq 0)$*, such that:*

$$A^{\sigma_0} \Rightarrow_{\mathsf{cf}} A_0^{\sigma_0} \text{ where } \{ p_1 := A_1^{\sigma_1}, \ldots, \tag{22a}$$

$$p_n := A_n^{\sigma_n} \} \tag{22b}$$

$$B^{\sigma_0} \Rightarrow_{\mathsf{cf}} B_0^{\sigma_0} \text{ where } \{ p_1 := B_1^{\sigma_1}, \ldots, \tag{22c}$$

$$p_n := B_n^{\sigma_n} \} \tag{22d}$$

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

$$\mathsf{den}(A_i)(g) = \mathsf{den}(B_i)(g), \quad \textit{for all } g \in G \tag{23}$$

4 ALGORITHMIC PATTERNS AND λ-ABSTRACTIONS

In this section we demonstrate the technique of underspecified, parametric algorithm, i.e., algorithmic patterns that represent classes of specified algorithm in reduction steps . We use the technique with some examples to motivate the γ^*-reduction introduced in the second part of the paper.

A Parametric Algorithm: Now, we can use a more general term of an algorithmic pattern, as parametric algorithm. For any proper terms W, J, G_1, G_2 that to not contain free occurrences of the pure variables x_1, x_2, x_3, e.g., constants, the following reductions can be done by using the reduction rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, which are given in Section (3).

$$P_0 \equiv q(W(h))(J) \text{ where } \{ \tag{24a}$$

$$q := (G_1(x_1) + G_2(x_2)) \} \tag{24b}$$

$$\Rightarrow_{\mathsf{cf}} q(w)(j) \text{ where } \{$$

$$q := (q_1 + q_2),$$

$$j := J, \tag{24c}$$

$$w := W(h),$$

$$q_1 := G_1(x_1),\ q_2 := G_2(x_2) \}$$

The term P_0 in (24a)– (24c) can be preceded by a sequence of λ-abstractions, as in the term P_1 in (25a)–(25b). By using the reduction rules given in Section (3), P_1 can be reduced to the term (25d)–(25i).

$$P_1 \equiv \lambda(x_1)\lambda(x_2)\lambda(x_3)\big[q(W(h))(J) \text{ where } \{ \tag{25a}$$

$$q := (G_1(x_1) + G_2(x_2)) \}\big] \tag{25b}$$

$$\Rightarrow \lambda(x_1)\lambda(x_2)\lambda(x_3)\big[q(w)(j) \text{ where } \{$$

$$q := (q_1 + q_2),$$

$$j := J, \tag{25c}$$

$$w := W(h),$$

$$q_1 := G_1(x_1),\ q_2 := G_2(x_2) \}\big]$$

$$\Rightarrow \lambda(x_1)\lambda(x_2)\lambda(x_3)\Big[$$

$$\big[q'(x_1)(x_2)(x_3)\big]\big(w'(x_1)(x_2)(x_3)\big) \tag{25d}$$

$$(j'(x_1)(x_2)(x_3))\Big] \text{ where } \{$$

$$q' := \lambda(x_1)\lambda(x_2)\lambda(x_3)\Big[$$

$$(q'_1(x_1)(x_2)(x_3) + \tag{25e}$$

$$q'_2(x_1)(x_2)(x_3))\Big]$$

$$j' := \lambda(x_1)\lambda(x_2)\lambda(x_3)\Big[J\Big], \tag{25f}$$

$$w' := \lambda(x_1)\lambda(x_2)\lambda(x_3)\Big[W(h)\Big], \tag{25g}$$

$$q'_1 := \lambda(x_1)\lambda(x_2)\lambda(x_3)\Big[$$

$$G_1(x_1)\Big], \tag{25h}$$

$$q_2' := \lambda(x_1)\,\lambda(x_2)\,\lambda(x_3)\Big[\quad G_2(x_2)\Big]\ \} \tag{25i}$$

The term (25d)–(25i) has vacuous λ-abstractions, e.g., (25f), (25g), (25h), (25i), which denote constant functions, and corresponding applications, e.g., in (25d), that give the same values.

The γ^*-reduction, introduced in this paper, reduces such spurious sub-terms.

5 GAMMA-STAR REDUCTION

5.1 The γ^*-Rule

In the following sections, we give the definition of the γ^*-rule, see Table 1, and its major properties. Expanding the reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ with the γ^*-rule simplifies some terms, by reducing sub-terms with vacuous λ-abstractions, while maintaining closely the original algorithmic structure. By using the γ^*-rule, the canonical forms determine more efficient versions of algorithms, by maintaining the essence of the computational steps.

Definition 5 (Strong γ^*-condition)**.** A recursion term $A \in \mathsf{Terms}$ satisfies the strong γ^*-condition for an assignment $p := \lambda(\overrightarrow{u}^{\overrightarrow{\vartheta}})\,\lambda(v^{\vartheta})P^{\tau} : (\overrightarrow{\vartheta} \to (\vartheta \to \tau))$, with respect to $\lambda(v)$, if and only if A is of the form: (26a)–(26c):

$$A \equiv A_0 \text{ where } \{\ \overrightarrow{a} := \overrightarrow{A}, \tag{26a}$$
$$p := \lambda(\overrightarrow{u})\,\lambda(v)P, \tag{26b}$$
$$\overrightarrow{b} := \overrightarrow{B}\ \} \tag{26c}$$

with the sub-terms of correspondingly appropriate types, and which is such that the following holds:

1. The term $P \in \mathsf{Terms}_{\tau}$ does not have any (free) occurrences of v in it, i.e., $v \notin \mathsf{FreeV}(P)$
2. All the occurrences of p in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are occurrences in sub-terms $p(\overrightarrow{u})(v)$, modulo renaming the variables $\overrightarrow{u}, v$

In such a case, we also say that *the assignment* $p := \lambda(\overrightarrow{u})\,\lambda(v)P$ *satisfies the* γ^**-condition in the recursion term* A *in* (26a)–(26c).

6 THE γ^*-REDUCTION

Adding the $\Rightarrow_{\gamma^*}$ to the reduction rules of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ determines an extended reduction relation between terms as follows.

Table 1: The γ^*-rule

(γ^*)

$$A \equiv A_0 \text{ where } \{\ \overrightarrow{a} := \overrightarrow{A}, \tag{27a}$$
$$p := \lambda(\overrightarrow{u})\,\lambda(v)P, \tag{27b}$$
$$\overrightarrow{b} := \overrightarrow{B}\ \} \tag{27c}$$
$$\Rightarrow_{\gamma^*} A_0' \text{ where } \{\ \overrightarrow{a} := \overrightarrow{A}', \tag{27d}$$
$$p' := \lambda(\overrightarrow{u})P', \tag{27e}$$
$$\overrightarrow{b} := \overrightarrow{B'}\ \} \tag{27f}$$

where

- the term $A \in \mathsf{Terms}$ satisfies the (strong) γ^*-condition (in Definition 5) for $p := \lambda(\overrightarrow{u})\,\lambda(v)P$
- $p' \in \mathsf{RecVars}_{(\overrightarrow{\vartheta} \to \tau)}$ 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., of all occurrences of $p(\overrightarrow{u})(v)$ by $p'(\overrightarrow{u})$, in all parts X_i in (27d)–(27f), modulo renaming the variables $\overrightarrow{u}, v$

Definition 6 (γ^*-reduction)**.** The γ^**-reduction* relation is the smallest relation, $\Rightarrow^*_{\gamma^*} \subseteq \mathsf{Terms} \times \mathsf{Terms}$ (also denoted by $\Rightarrow_{\gamma^*}$), between terms that is closed under the $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-reduction rules, given in Section 3.1, and the γ^*-rule, given in Table 1.

We refer to the set of all $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ reduction rules extended with the γ^*-rule as the set of γ^**-reduction rules*.

In addition to the notations $\Rightarrow^*_{\gamma^*}$, and $\Rightarrow_{\gamma^*}$, for the γ^*-reduction, we also use the usual notation for reflexive and transitive closure of a relation, given in (28a). To specify that the γ^*-rule has been applied certain number of times (including zero times), possibly intervened by applications of some of the other reduction rules, we use the notation (28b)–(28c).

$$A \Rightarrow^n_{\gamma^*} B \iff A \Rightarrow^*_{\gamma^*} B$$
by n applications of reduction rules, possibly γ^* $(n \geq 0)$ (28a)

$$A \Rightarrow_{\gamma^*}{}^* B \iff A \Rightarrow^*_{\gamma^*\,[n]} B, \text{ for } n \geq 0$$
by using $\Rightarrow$-rules and n applications of the γ^*-rule (28b)

$$A \Rightarrow^+_{\gamma^*} B \iff A \Rightarrow^*_{\gamma^*\,[n]} B \text{ for } n \geq 1$$
by using $\Rightarrow$-rules and n applications of the γ^*-rule (28c)

Definition 7 (γ^*-irreducible terms)**.** We say that a

term $A \in \mathsf{Terms}$ is γ^*-*irreducible* if and only if

$$\text{for all } B \in \mathsf{Terms}, \quad A \Rightarrow^*_{\gamma^*} B \implies A \equiv_{\mathsf{c}} B \qquad (29)$$

Definition 8 (γ^*-irreducible recursion terms for a specific assignment $c := \lambda(\overrightarrow{u})\lambda(v)C$**).** We say that a recursion term

$$A \equiv A_0 \text{ where } \{\, \overrightarrow{p} := \overrightarrow{A},\ c := \lambda(\overrightarrow{u})\lambda(v)C,\ \overrightarrow{q} := \overrightarrow{B} \,\}$$

is γ^*-*irreducible* for the assignment $c := \lambda(\overrightarrow{u})\lambda(v)C$, with respect to $\lambda(v)$, if and only if the conditions for the γ^*-rule are not satisfied for it, i.e., either

(1) $v \in \mathsf{FreeV}(C)$, or

(2) $v \notin \mathsf{FreeV}(C)$, and not all of the occurrences of c in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are sub-occurrences in a term $c(\overrightarrow{u})(y)$, modulo congruence by renaming the variables $\overrightarrow{u}, y \in \mathsf{PureVars}$.

Theorem 3 (Criteria for γ^*-irreducibility)**.** *By structural induction:*

1. *If $A \in \mathsf{Const} \cup \mathsf{Vars}$, then A is γ^*-irreducible.*
2. *An application term $A(B)$ is γ^*-irreducible if and only if A is explicit and irreducible and B is immediate.*
3. *A λ-term $\lambda(x)A$ is γ^*-irreducible if and only if A is explicit and irreducible.*
4. *A recursion term A*

   $$A \equiv [A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}] \quad (n \geq 0)$$

   is γ^*-irreducible if and only if

   (a) all of the parts $A_0, \ldots, A_n$ are explicit and irreducible, and

   (b) A does not satisfy the γ^*-condition

Proof. By structural induction on terms and inspection of the γ^*-reduction rules. □

7 CANONICAL FORMS AND γ^*-REDUCTION

Theorem 4 (Extended γ^*-Canonical Form Theorem)**.** *For every $A \in \mathsf{Terms}$, the following holds:*

1. *(Existence of a γ^*-canonical form of A) There exist explicit, irreducible $A_0, \ldots, A_n \in \mathsf{Terms}$ $(n \geq 0)$ such that the term A_0 where $\{\, p_1 := A_1, \ldots, p_n := A_n \,\}$ is γ^*-irreducible, i.e., irreducible and does not satisfy the γ-condition, and*

   $$\mathsf{cf}_{\gamma^*}(A) \equiv A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\}, \qquad (30)$$

   Thus, $\mathsf{cf}_{\gamma^*}(A)$ is γ^*-irreducible.

2. *A constant $\mathsf{c} \in K$ or a recursion variable $p \in \mathsf{RecVars}$ occurs freely in $\mathsf{cf}_{\gamma^*}(A)$ if and only if it occurs freely in A.*
3. *$A \Rightarrow^*_{\gamma^*} \mathsf{cf}_{\gamma^*}(A)$*
4. *If A is γ^*-irreducible, then $\mathsf{cf}_{\gamma^*}(A) \equiv_c A$.*
5. *If $A \Rightarrow^*_{\gamma^*} B$, then $\mathsf{cf}_{\gamma^*}(A) \equiv_c \mathsf{cf}_{\gamma^*}(B)$*
6. *(Uniqueness of $\mathsf{cf}_{\gamma^*}(A)$ up to congruence) if $A \Rightarrow^*_{\gamma^*} B$ and B is γ^*-irreducible, then $B \equiv_c \mathsf{cf}_{\gamma^*}(A)$, i.e., $\mathsf{cf}_{\gamma^*}(A)$ is unique, up to congruence, γ^*-irreducible term. We write*

   $$A \Rightarrow_{\mathsf{gscf}} B \iff B \equiv_c \mathsf{cf}_{\gamma^*}(A) \qquad (31a)$$

   $$A \Rightarrow_{\mathsf{gscf}} \mathsf{cf}_{\gamma^*}(A) \qquad (31b)$$

Proof. The statement (1) is proved by induction on term structure, using the definition of the $\mathsf{cf}_{\gamma^*}(A)$. The statements (2) and (3) are proved by induction on term structure, using the criteria for γ^*-irreducibility 3. (4) is proved by induction on the definition of the γ^*-reduction relation. (5) follows from (3) and (4). □

Definition 9 (γ^*-equivalence (γ^*-synonymy) relation $\approx_{\gamma^*}$**).** For any $A, B \in \mathsf{Terms}$:

$$A \approx_{\gamma^*} B \iff \mathsf{cf}_{\gamma^*}(A) \approx \mathsf{cf}_{\gamma^*}(B) \qquad (32)$$

When $A \approx_{\gamma^*} B$, we say that A and B are γ^*-equivalent, alternatively, γ^*-synonymous.

Note 1. If we have added an additional restriction in the γ^*-condition of the γ^*-rule that all the occurrences of the sub-terms $p(\overrightarrow{u})(v)$ have to be in the scope of $\lambda(v)$ (modulo renaming congruence), the $\Rightarrow_{\gamma^*}$-rule would have preserved all the free variables of A in $\mathsf{cf}_{\gamma^*}(A)$, including the pure variables, not only the recursion variables, so that $\mathsf{FreeV}(\mathsf{cf}_{\gamma^*}(A)) = \mathsf{FreeV}(A)$ (see the γ^*-Canonical Form Theorem 4). In this strong γ^*-reduction, we refrain from adding such an extra restriction. Note also that the replacements $A_i\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$, $B_j\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$ in the γ^*-rule (Table 1) are not necessarily "free", in the inverse sense that the $\Rightarrow_{\gamma^*}$-rule may remove occurrences of v which are in the scope of $\lambda(v)$, in some parts, due to the clause (2) in the γ^*-condition (5).

8 SOME PROPERTIES OF THE γ^*-EQUIVALENCE

Theorem 5 (γ^*-Equivalence Theorem)**.** *Two terms A, B are algorithmically γ^*-synonymous, $A \approx_{\gamma^*} B$, if and only if there are explicit, irreducible terms of corresponding types, $A_i : \sigma_i$, $B_i : \sigma_i$ $(i = 0, \ldots, n)$, $(n \geq 0)$, such that:*

$$A \Rightarrow_{\mathsf{gscf}} A_0 \text{ where } \{\, p_1 := A_1, \ldots, p_n := A_n \,\} \equiv \mathsf{cf}_{\gamma^*}(A) \quad \textit{(i.e., } \gamma^*\textit{-irreducible)} \qquad (33a)$$

$$B \Rightarrow_{\mathsf{gscf}} B_0 \text{ where } \{\, p_1 := B_1, \ldots, p_n := B_n \,\} \equiv \mathsf{cf}_{\gamma^*}(B) \quad \textit{(i.e., } \gamma^*\textit{-irreducible)} \tag{33b}$$

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

$$\mathsf{den}(A_i)(g) = \mathsf{den}(B_i)(g), \quad \textit{for all } g \in G \tag{34}$$

Proof. The theorem follows from Definition 9 of γ^*-equivalence and Theorem 2. □

Definition 10 (Syntactic Synonymy (Equivalence) $\approx_s$)**.** For any $A, B \in \mathsf{Terms}$,

$$A \approx_s B \iff \mathsf{cf}(A) \equiv_{\mathsf{c}} \mathsf{cf}(B) \tag{35}$$

For more details about syntactic synonymy, see Moschovakis (Moschovakis, 2006). The difference between syntactic and algorithmic synonymies is that syntactic synonymy does not apply to denotationally equivalent constants and syntactic constructs such as λ-terms. For instance, assuming that *dog* and *canine* are constants, such that $\mathsf{den}(dog) = \mathsf{den}(canine)$, it holds that $dog \approx canine$ (by the Referential Synonymy Theorem 2), because both terms are in canonical forms, with the same denotations, i.e., they denote the same function obtainable by the same algorithm, determined by the interpretation function I of the semantics structure $\mathfrak{A} = \langle \mathbb{T}, I \rangle$. On the other hand, $dog \not\approx_s canine$, since $dog \not\equiv_{\mathsf{c}} canine$. Also, $\mathsf{den}(dog) = \mathsf{den}(\lambda(x)dog(x))$ (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 2), because both terms are in canonical forms. These two terms are syntactically different, $dog \not\approx_s \lambda(x)dog(x)$, because $dog \not\equiv_{\mathsf{c}} \lambda(x)dog(x)$.

Theorem 6. *For any* $A, B \in \mathsf{Terms}$,

$$A \Rightarrow B \implies A \approx_s B \tag{36a}$$
$$\implies A \approx B \tag{36b}$$
$$\implies A \approx_{\gamma^*} B \implies A \models\!\mid B \tag{36c}$$

Proof. By using the definitions. □

Theorem 7. *For any* $A, B \in \mathsf{Terms}$,

$$\mathsf{cf}(A) \approx_{\gamma^*} \mathsf{cf}(B) \iff \mathsf{cf}(A) \Rightarrow^*_{\gamma^*} A',\ \mathsf{cf}(B) \Rightarrow^*_{\gamma^*} B',\ \textit{and } A' \approx_{\gamma^*} B',\ \textit{for some } A', B' \in \mathsf{Terms} \tag{37a}$$

$$\mathsf{cf}(A) \approx_{\gamma^*} \mathsf{cf}(B) \iff A \approx_{\gamma^*} B \tag{37b}$$

Proof. The directions $\Longleftarrow$ are proved by using Definition 9, Referential Synonymy Theorem 2, and Extended γ^*-Canonical Form Theorem 4. □

Corollary 1. *For all* $A, B, C \in \mathsf{Terms}$,

$$A \Rightarrow B \Rightarrow^*_{\gamma^*} C \implies A \approx B \implies A \approx_{\gamma^*} B \approx_{\gamma^*} C \tag{38}$$

while there exist (many) terms $A, B, C \in \mathsf{Terms}$ *such that*

$$A \Rightarrow B \Rightarrow^*_{\gamma^*} C,\ C \not\approx B,\ \textit{and}\ C \not\approx A \tag{39}$$

Proof. (38) follows from Definition 9, the Canonical Form Theorems 1, and 4. □

By Definition 9 of γ^*-equivalence between two terms A, B as algorithmic synonymy between their γ^*-canonical forms, various properties of algorithmic synonymy are inherited by γ^*-equivalence, reflected, e.g., by the γ^*-Equivalence Theorem 5 and the compositionality of γ^*-equivalence, with the very restricted form of β-reduction.

Assume that the $(\gamma*)$-rule, see Table 1, is applied to a term A in canonical form, i.e., $A \equiv_{\mathsf{c}} \mathsf{cf}(A)$. By application of the $(\gamma*)$-rule until we obtain the γ^* canonical form $\mathsf{cf}_{\gamma^*}(A)$ of A. The corresponding parts in the assignments (27b)–(27e) are not denotationally equivalent, since they are not of the same type. By the γ^*-Equivalence Theorem 5, $A \approx \mathsf{cf}(A) \not\approx \mathsf{cf}_{\gamma^*}(A)$.

The γ^*-reduction calculus does not preserve per se the algorithmic synonymy between terms. That is, in general, it is possible that $A \approx B$, while $A \not\approx_{\gamma^*} B$.

Nevertheless, the γ^*-reduction relation $\Rightarrow^*_{\gamma^*}$ between terms is very useful. For any terms A and B, a γ^*-reduction $A \Rightarrow_{\gamma^*} \mathsf{cf}_{\gamma^*}(A)$ preserves the most essential algorithmic components of the canonical form $\mathsf{cf}(A)$ in $\mathsf{cf}_{\gamma^*}(A)$. It reduces vacuous λ-abstractions, which denote constant functions, and corresponding applications that give the constant values.

9 APPLICATIONS OF THE γ^*-RULE

In this section, we give pattern examples for possible renderenings of expressions in human language to $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms that can represent their algorithmic semantics. A definition of a rendering relation between human language expressions and their semantic representations by $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms is not in the subject of this paper. Rendering can be defined in a computational mode, via syntax-semantics interfaces, within a computational grammar, e.g., see (Loukanova, 2011f; Loukanova, 2017b). Typically, $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ offers alternative terms for representing algorithmic semantics of human language expressions. The choice would depend on applications.

Developments of new, hybrid machine learning techniques and statistical approaches for extraction of semantic information from text can provide more possibilities for rendering human language expressions to $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-terms.

Example 9.1.

$$\text{Kim hugs some dog} \xrightarrow{\text{render}} A \tag{40a}$$

$$A \equiv \Big[\lambda(y_k)\Big(some\big(dog\big) \big(\lambda(x_d)hug(x_d)(y_k)\big)\Big)\Big](kim) \tag{40b}$$

Proposition 1. *Given that A is the term in* (40b)*, its canonical and* γ^**-canonical forms,* $\mathsf{cf}(A)$ *and* $\mathsf{cf}_{\gamma^*}(A)$*, are as in* (41) *and* (42)*, correspondingly:*

$$\mathsf{cf}(A) \equiv \Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where } \{h := \lambda(y_k)\lambda(x_d)hugs(x_d)(y_k),\ d' := \lambda(y_k)dog,\ k := kim\} \tag{41}$$

$$\mathsf{cf}_{\gamma^*}(A) \equiv \big[\lambda(y_k)some(d)\big(h(y_k)\big)\big](k) \text{ where } \{h := \lambda(y_k)\lambda(x_d)hugs(x_d)(y_k),\ d := dog,\ k := kim\} \tag{42}$$

$$\mathsf{cf}(A) \not\approx \mathsf{cf}_{\gamma^*}(A) \tag{43a}$$

$$\mathsf{cf}(A) \approx_{\gamma^*} \mathsf{cf}_{\gamma^*}(A) \tag{43b}$$

Proof. The following reductions hold for the term A in (40b).

$$A \Rightarrow \ldots \tag{44a}$$

$$\Rightarrow \Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big) \text{ where } \{d' := \lambda(y_k)dog,\ h := \lambda(y_k)\lambda(x_d)hugs(x_d)(y_k)\}\Big](kim) \tag{44b}$$

$$\Rightarrow_{\mathsf{cf}} \Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where } \{h := \lambda(y_k)\lambda(x_d)hugs(x_d)(y_k),\ d' := \lambda(y_k)dog,\ k := kim\} \tag{44c}$$

$$\Rightarrow_{\gamma^*} \big[\lambda(y_k)some(d)\big(h(y_k)\big)\big](k) \text{ where } \{h := \lambda(y_k)\lambda(x_d)hugs(x_d)(y_k),\ d := dog,\ k := kim\} \tag{44d}$$

(43a) follows from Theorem 2 and (43b) from Theorem 5. □

The term in (44d), and thus, the term $\mathsf{cf}_{\gamma^*}(A)$ in (41) too, is in a canonical form, but it is not algorithmically equivalent to the term (44c), i.e., to $\mathsf{cf}(A)$ in (42) too, by the original reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ in Moschovakis (Moschovakis, 2006). The term (44d) is simpler than (44c), which has an extraneous, vacuous λ-abstraction over y_k in the assignment $d' := \lambda(y_k)dog$, while the term dog does not have any (free) occurrences of y_k, i.e., the values of $\lambda(y_k)dog$ stored in d' are constant and do not depend on $\lambda(y_k)$. The terms (44c) and (44d) denote very similar algorithms that are γ^*-equivalent, by applying the γ^*-rule. The term (44d) is in γ^*-canonical form.

Example 9.2. Assume that the sentence (45) is rendered to a $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-term $B \equiv B_1$ that is given in (46a)–(46h).

$$\begin{aligned}&[\text{Jim}]_j \text{ sent Mia the article about}\\ &\text{the [discovery of Protein353 by [him}]_j]\\ &\xrightarrow{\text{render}} B\end{aligned} \tag{45}$$

Alternatively, depending on specific applications, B may be a term that is reduced to the term in (46a)–(46h).

$$B \Rightarrow B_1 \equiv \tag{46a}$$

$$\lambda(z)\Big[\lambda(x)\big[send(m^1)(a^1)(z) \text{ where} \tag{46b}$$

$$\{a^1 := the(r^1), \tag{46c}$$

$$r^1 := article\text{-}about(b^1), \tag{46d}$$

$$b^1 := the(d^1), \tag{46e}$$

$$d^1 := discovery\text{-}of\text{-}by(p^1)(z), \tag{46f}$$

$$p^1 := protein353, \tag{46g}$$

$$m^1 := mia\}\big]\Big](jim) \tag{46h}$$

The $\lambda(x)$ abstractions inside the assignments in (47a)–(47h) are the typical result of the (λ)-rule of the reduction calculus of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, in this case, to the term B_1.

$$B_1 \Rightarrow_{(\lambda)} B_2 \equiv \tag{47a}$$

$$\lambda(z)\Big[\lambda(x)send(m^2(x))(a^2(x))(z) \text{ where} \tag{47b}$$

$$\{a^2 := \lambda(x)the(r^2(x)), \tag{47c}$$

$$r^2 := \lambda(x)article\text{-}about(b^2(x)), \tag{47d}$$

$$b^2 := \lambda(x)the(d^2(x)), \tag{47e}$$

$$d^2 := \lambda(x)discovery\text{-}of\text{-}by(p^2(x))(z), \tag{47f}$$

$$p^2 := \lambda(x)protein353, \tag{47g}$$

$$m^2 := \lambda(x)mia\}\Big](jim) \tag{47h}$$

Another application of the (λ)-rule reduces the term B_2 to B_3 in (48a)–(48h).

$$B_2 \Rightarrow_{(\lambda)} B_3 \equiv \tag{48a}$$

$$\Big[\lambda(z)\lambda(x)send(m^3(z)(x))(a^3(z)(x))(z) \text{ where} \tag{48b}$$

$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (48c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b^3(z)(x)), \quad (48d)$$
$$b^3 := \lambda(z)\lambda(x)\mathit{the}(d^3(z)(x)), \quad (48e)$$
$$d^3 := \lambda(z)\lambda(x) \mathit{discovery\text{-}of\text{-}by}(p^3(z)(x))(z), \quad (48f)$$
$$p^3 := \lambda(z)\lambda(x)\mathit{protein353}, \quad (48g)$$
$$m^3 := \lambda(z)\lambda(x)\mathit{mia}\}\Big](\mathit{jim}) \quad (48h)$$

The term B_4 in (49a)–(49h) is the result of applying the Recursion-application rule (recap) to B_3 in (48a)–(48h).

$$B_3 \Rightarrow_{(\mathrm{recap})} B_4 \equiv \quad (49a)$$
$$\Big[\lambda(z)\lambda(x)\mathit{send}(m^3(z)(x))(a^3(z)(x))(z)\Big](\mathit{jim}) \quad (49b)$$
where
$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (49c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b^3(z)(x)), \quad (49d)$$
$$b^3 := \lambda(z)\lambda(x)\mathit{the}(d^3(z)(x)), \quad (49e)$$
$$d^3 := \lambda(z)\lambda(x) \mathit{discovery\text{-}of\text{-}by}(p^3(z)(x))(z), \quad (49f)$$
$$p^3 := \lambda(z)\lambda(x)\mathit{protein353}, \quad (49g)$$
$$m^3 := \lambda(z)\lambda(x)\mathit{mia}\} \quad (49h)$$

The term B_4 is reduced to the term B_5 in (50a)–(50h), by successive applications of the reduction rule (ap) to the head part in (49b), the Compositionality rule (c-rec) for recursion terms, the Head rule (head), and Congruence of the order of the recursion assignments.

$$B_4 \Rightarrow B_5 \equiv \quad (50a)$$
$$\Big[\lambda(z)\lambda(x)\mathit{send}(m^3(z)(x))(a^3(z)(x))(z)\Big](j) \quad (50b)$$
where
$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (50c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b^3(z)(x)), \quad (50d)$$
$$b^3 := \lambda(z)\lambda(x)\mathit{the}(d^3(z)(x)), \quad (50e)$$
$$d^3 := \lambda(z)\lambda(x) \mathit{discovery\text{-}of\text{-}by}(p^3(z)(x))(z), \quad (50f)$$
$$p^3 := \lambda(z)\lambda(x)\mathit{protein353}, \quad (50g)$$
$$m^3 := \lambda(z)\lambda(x)\mathit{mia},\ j := \mathit{jim}\} \quad (50h)$$

The denotations of the terms $\lambda(z)\lambda(x)\mathit{protein353}$ and $\lambda(z)\lambda(x)\mathit{mia}$, 'saved' respectively in p^3 and m^3, by (50g) and (50h), are constant functions that do not depend on the argument roles of the abstractions $\lambda(z)\lambda(x)$.

The term B_5 is reduced to B_6, by four successive applications of the γ^*-rule, for the assignments $p^3 := \lambda(z)\lambda(x)\mathit{protein353}$ and $m^3 := \lambda(z)\lambda(x)\mathit{mia}$.

$$B_5 \Rightarrow_{\gamma^*} B_6 \equiv \quad (51a)$$
$$\Big[\lambda(z)\lambda(x)\mathit{send}(m)(a^3(z)(x))(z)\Big](j) \quad (51b)$$
where
$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (51c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b^3(z)(x)), \quad (51d)$$
$$b^3 := \lambda(z)\lambda(x)\mathit{the}(d^3(z)(x)), \quad (51e)$$
$$d^3 := \lambda(z)\lambda(x) \mathit{discovery\text{-}of\text{-}by}(p)(z), \quad (51f)$$
$$p := \mathit{protein353}, \quad (51g)$$
$$m := \mathit{mia},\ j := \mathit{jim}\} \quad (51h)$$

Now, the term B_6 satisfies the γ^*-condition for the assignment (51f), with respect to $\lambda(x)$. Application of the γ^*-rule to B_6, reduces B_6 to B_7.

$$B_6 \Rightarrow_{\gamma^*} B_7 \equiv \quad (52a)$$
$$\Big[\lambda(z)\lambda(x)\mathit{send}(m)(a^3(z)(x))(z)\Big](j) \quad (52b)$$
where
$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (52c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b^3(z)(x)), \quad (52d)$$
$$b^3 := \lambda(z)\lambda(x)\mathit{the}(d(z)), \quad (52e)$$
$$d := \lambda(z)\mathit{discovery\text{-}of\text{-}by}(p)(z), \quad (52f)$$
$$p := \mathit{protein353}, \quad (52g)$$
$$m := \mathit{mia},\ j := \mathit{jim}\} \quad (52h)$$

Now, the term B_7 satisfies the γ^*-condition for the assignment (52e), with respect to $\lambda(x)$. Application of the γ^*-rule to B_7, reduces B_7 to B_8.

$$B_7 \Rightarrow_{\gamma^*} B_8 \equiv \quad (53a)$$
$$\Big[\lambda(z)\lambda(x)\mathit{send}(m)(a^3(z)(x))(z)\Big](j) \quad (53b)$$
where
$$\{a^3 := \lambda(z)\lambda(x)\mathit{the}(r^3(z)(x)), \quad (53c)$$
$$r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b(z)), \quad (53d)$$
$$b := \lambda(z)\mathit{the}(d(z)), \quad (53e)$$
$$d := \lambda(z)\mathit{discovery\text{-}of\text{-}by}(p)(z), \quad (53f)$$
$$p := \mathit{protein353}, \quad (53g)$$
$$m := \mathit{mia},\ j := \mathit{jim}\} \quad (53h)$$

In (54a), the term B_8 is reduced to B_9, i.e., $B_8 \Rightarrow_{\gamma^*} B_9$, by two successive applications of the γ^*-rule, at first for $r^3 := \lambda(z)\lambda(x)\mathit{article\text{-}about}(b(z))$, with respect to

$\lambda(x)$, and then for $a^3 := \lambda(z)\lambda(x)the(r(z))$, with respect to $\lambda(x)$.

$$B_8 \Rightarrow_{\gamma^*[2]} B_9 \equiv \tag{54a}$$

$$\Big[\lambda(z)\lambda(x)send(m)(a(z))(z)\Big](j) \tag{54b}$$

where

$$\{a := \lambda(z)the(r(z)), \tag{54c}$$

$$r := \lambda(z)article\text{-}about(b(z)), \tag{54d}$$

$$b := \lambda(z)the(d(z)), \tag{54e}$$

$$d := \lambda(z)discovery\text{-}of\text{-}by(p)(z), \tag{54f}$$

$$p := protein353, \tag{54g}$$

$$m := mia,\ j := jim\} \tag{54h}$$

10 FUTURE WORK

We work on applications of the type-theory of acyclic algorithms. For example, most promising results have been achieved in language processing of formal and natural languages. Specific applications are computational semantics and computational syntax-semantics interfaces. These lines of work continue.

A new direction of applications is to computational neuroscience, by algorithmic modelling of procedural, factual, and declarative memory, and dependencies between those, by mutual recursion.

Along such applications to advanced technologies and AI, we work on theoretical developments. The results in this paper are part of such long-term work.

REFERENCES

Hamm, F. and van Lambalgen, M. (2004). Moschovakis' notion of meaning as applied to linguistics. In *Logic Colloqium*, volume 1.

Hurkens, A. J. C., McArthur, M., Moschovakis, Y. N., Moss, L. S., and Whitney, G. T. (1998). The logic of recursive equations. *The Journal of Symbolic Logic*, 63(2):451–478.

Loukanova, R. (2011a). Constraint Based Syntax of Modifiers. *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, 3:167–170.

Loukanova, R. (2011b). 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 — New Interdisciplinary Paradigms*, volume 228 of *Frontiers in Artificial Intelligence and Applications*, pages 200–214. IOS Press, Amsterdam; Berlin; Tokyo; Washington, DC.

Loukanova, R. (2011c). Minimal Recursion Semantics and the Language of Acyclic Recursion. In Bel-Enguix, G., Dahl, V., and Puente, A. O. D. L., editors, *AI Methods for Interdisciplinary Research in Language and Biology*, pages 88–97, Rome. SciTePress — Science and Technology Publications.

Loukanova, R. (2011d). 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. Springer*, pages 265–274. Springer.

Loukanova, R. (2011e). 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. (2011f). 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. (2011g). Syntax-Semantics Interface for Lexical Inflection with the Language of Acyclic Recursion. In Bel-Enguix, G., Dahl, V., and Jiménez-López, M. D., editors, *Biology, Computation and Linguistics — New Interdisciplinary Paradigms*, volume 228 of *Frontiers in Artificial Intelligence and Applications*, pages 215–236. IOS Press, Amsterdam; Berlin; Tokyo; Washington, DC.

Loukanova, R. (2012a). Algorithmic Semantics of Ambiguous Modifiers by the Type Theory of Acyclic Recursion. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 3:117–121.

Loukanova, R. (2012b). Semantic Information with Type Theory of Acyclic Recursion. In Huang, R., Ghorbani, A. A., Pasi, G., Yamaguchi, T., Yen, N. Y., and Jin, B., editors, *Active Media Technology - 8th International Conference, AMT 2012, Macau, China, December 4-7, 2012. Proceedings*, volume 7669 of *Lecture Notes in Computer Science*, pages 387–398. Springer.

Loukanova, R. (2013a). Algorithmic Granularity with Constraints. In Imamura, K., Usui, S., Shirao, T., Kasamatsu, T., Schwabe, L., and Zhong, N., editors, *Brain and Health Informatics*, volume 8211 of *Lecture Notes in Computer Science*, pages 399–408. Springer International Publishing.

Loukanova, R. (2013b). Algorithmic Semantics for Processing Pronominal Verbal Phrases. In Larsen, H. L., Martin-Bautista, M. J., Vila, M. A., Andreasen, T., and Christiansen, H., editors, *Flexible Query Answering Systems*, volume 8132 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg.

Loukanova, R. (2013c). A Predicative Operator and Underspecification by the Type Theory of Acyclic Re-

cursion. In Duchier, D. and Parmentier, Y., editors, *Constraint Solving and Language Processing*, volume 8114 of *Lecture Notes in Computer Science*, pages 108–132. Springer Berlin Heidelberg.

Loukanova, R. (2014). Situation Theory, Situated Information, and Situated Agents. In Nguyen, N. T., Kowalczyk, R., Fred, A., and Joaquim, F., editors, *Transactions on Computational Collective Intelligence XVII*, volume 8790 of *Lecture Notes in Computer Science*, pages 145–170. Springer Berlin Heidelberg.

Loukanova, R. (2015a). Representing parametric concepts with situation theory. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, volume 5, pages 89–100. IEEE.

Loukanova, R. (2015b). Underspecified Relations with a Formal Language of Situation Theory. In Loiseau, S., Filipe, J., Duval, B., and van den Herik, J., editors, *Proceedings of the 7th International Conference on Agents and Artificial Intelligence*, volume 1, pages 298–309. SCITEPRESS — Science and Technology Publications, Lda.

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. (2016c). Specification of Underspecified Quantifiers via Question-Answering by the Theory of Acyclic Recursion. In Andreasen, T., Christiansen, H., Kacprzyk, J., Larsen, H., Pasi, G., Pivert, O., Tré, G. D., Vila, M. A., Yazici, A., and Zadrożny, S., editors, *Flexible Query Answering Systems 2015*, volume 400 of *Advances in Intelligent Systems and Computing*, pages 57–69. Springer International Publishing.

Loukanova, R. (2017a). Binding operators in type-theory of algorithms for algorithmic binding of functional neuro-receptors. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, volume 11, pages 57–66. IEEE.

Loukanova, R. (2017b). Typed Theory of Situated Information and its Application to Syntax-Semantics of Human Language. In Christiansen, H., Jiménez-López, M. D., Loukanova, R., and Moss, L. S., editors, *Partiality and Underspecification in Information, Languages, and Knowledge*, pages 151–188. Cambridge Scholars Publishing.

Loukanova, R. and Jiménez-López, M. D. (2012). On the Syntax-Semantics Interface of Argument Marking Prepositional Phrases. In Pérez, J. B., Sánchez, M. A., Mathieu, P., Rodríguez, J. M. C., Adam, E., Ortega, A., Moreno, M. N., Navarro, E., Hirsch, B., Lopes-Cardoso, H., and Julián, V., editors, *Highlights on Practical Applications of Agents and Multi-Agent Systems*, volume 156 of *Advances in Intelligent and Soft Computing*, pages 53–60. Springer Berlin / Heidelberg.

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

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. (1997). The logic of functional recursion. In *Logic and Scientific Methods*, pages 179–207. Kluwer Academic Publishers. Springer.

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