Acyclic Recursion with Polymorphic Types and Underpecification

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

Keywords: Semantics, Algorithms, Intension, Denotation, Recursion, Types, Underspecification, Subtypes, Polymorphism.

Abstract: The paper extends Moschovakis higher-order type theory of acyclic recursion by adding type polymorphism. We extend the type system of the theory to model parametric information that pertains to underspecified types. Different kinds of type polymorphism are presented via type variables and recursion constructs for alternative, disjunctive type assignments. Based on the new type system, we extend the reduction calculus of the theory of acyclic recursion. We motivate the type polymorphism with examples from English language.

1 INTRODUCTION

A sequence of papers (Moschovakis, 1989; Moschovakis, 1994; Moschovakis, 1997) introduced a new approach to the notion of algorithm, by the mathematical concept of recursion. The approach uses a formal language of recursion and fine semantic distinctions between denotations and algorithms computing the denotations. That work was on untyped theory of algorithms and was the basis of (Moschovakis, 2006), which introduced typed theory $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ of acyclic recursion, as formalization of the concepts of algorithmic meaning in typed models. $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ was introduced for primary application to algorithmic semantics of human language, by considering semantics of programming languages, where a given denotation can be computed by different algorithms.

Work on extending the expressive power of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ was initiated in (Loukanova, 2016; Loukanova, 2013a; Loukanova, 2011b). The class of formal languages and theories of typed acyclic recursion, collectively, is a formal system, which we call the *typed theory of acyclic recursion* (TTofAR).

TTofAR has potentials for applications to algorithmic semantics of formal and natural languages. What makes the work highly demanding is one of the distinctive features of human languages — semantic ambiguities and underspecification. Ambiguous language expressions are interpreted depending on context information, and often, even in context, specific interpretations can be parametric. Thus, intelligent language processing requires computational integration of syntax with semantics that allows underspecified representations. Availability of computational models of context are essential for specific interpretations and disambiguation. However, it is important to provide representations in underspecified forms, before context information can select specific interpretations. Thus, it is important to develop computational theory that represents parametric information as such. Furthermore, parametric components can be about unknown types of objects, not only underspecified objects of fully known types. This paper is on development of a theory for computational semantics that maintains parametric types.

Among the formal languages, we consider applications of TTofAR to semantics of programming languages, languages used in database systems and various areas of Artificial Intelligence (AI). There is realization in contemporary developments and technologies, related to language processing, that they are more successful by using language constructs with parametric or otherwise underspecified types. In programming languages and formal languages of type theory, this is provided by techniques classified as polymorphism. See, e.g., the classic work (Cardelli and Wegner, 1985). In large-scale computational grammars of human language, underspecification has become important subject, see e.g., (Copestake et al., 2005), work which, while not fully formalized, has been used in implementations. For providing such applications, our paper extends $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ by adding polymorphism to its type system.

Semantic ambiguities are among the core, open problems in language processing. While a broad spectrum of semantic theories have been seeking so-

Loukanova, R.
Acyclic Recursion with Polymorphic Types and Underpecification.
DOI: 10.5220/0005749003920399
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2*, pages 392-399
ISBN: 978-989-758-172-4

lutions, here we only mention theories that, according to us, have underlain research and applications in the areas related to processing of human language that includes computational semantics. Montague's Intensional Logic (IL), see (Thomason, 1974), introduced the first significant approach to computational semantics, which also included representation of semantical ambiguities. IL was extended by (Gallin, 1975) to handle hidden quantification over possible wolds and times, which was the basis of extended, Montague style grammars, e.g., (Muskens, 1995), and more recently (Villadsen, 2010) . On the side of comprehensive, logical approaches to information and semantics, (Barwise, 1981) was the first to introduce a computational approach to the concepts of partiality, semantic parameters, and restricted parameters in logic approaches to information. While TTofAR, including L^{ta}_{ar} introduced here, extend (Gallin, 1975), and by that Montague IL, TTofAR delimit Situation Theory to functional systems. This is done by representing set-theoretic relations with their characteristic functions, via standard Curry encodings (when possible and reasonable), and to acyclic recursion, i.e., to computations that close-off, via mutual recursion that ends after certain number of steps. TTofAR essentially extend Gallin logic, and logic systems based on it, by adding expressiveness for algorithmic semantics and semantic denotations, i.e., denotational and algorithmic semantics. By L^{ta}_{ar}, we extend the expressiveness of L^{λ}_{ar} for handling underspecified information by type polymorphism, which is also acyclic with respect to computations. The system L^{ta}_{ar} offers unique formalization of underspecification in information and semantics, via integration of polymorphism with computations that close-off. This makes it valuable theory, as a mathematical model of algorithms that can be underspecified, and for applications.

The untyped theory of recursion (Moschovakis, 1997; Moschovakis, 1989) is applied in (Hurkens et al., 1998) to model reasoning. The potentials of L^{λ}_{ar}, with typed recursion, have been used in various applications. Application of L^{λ}_{ar} to logic programming in linguistics and cognitive science is given in (Van Lambalgen and Hamm, 2004). A sequence of papers (Loukanova, 2016; Loukanova, 2013c; Loukanova, 2013b; Loukanova, 2013a; Loukanova, 2012a; Loukanova, 2012b; Loukanova, 2011b; Loukanova, 2011c; Loukanova, 2011d; Loukanova, 2011e; Loukanova, 2011a) and (Loukanova and Jiménez-López, 2012) along extending the original L^{λ}_{ar}, applied TTofAR to computational semantics and computational syntax-semantics interface of human language. By adding polymorphism, L^{ta}_{ar} offers potentials for varieties of applications.

The following sections are concise introduction to the theory of L^{ta}_{ar}. We give motivation from computational semantics of human language, because L^{ta}_{ar} has direct potentials for applications to computerized processing of human language, including large-scale, computational grammars of human language and NLP for AI.

2 MOTIVATION FOR UNDERSPECIFIED TYPES

In this section, we present some motivation for introducing parametric types in a type system, i.e., in this paper, by extending L^{λ}_{ar}. The general idea is that parametric types can be left underspecified when there is not enough information to designate fully specified types. Further specification (which still can be parametric) of a parametric type can be refined depending on environment. In order to reach readers from varying research areas, including AI, we present the usefulness of parametric types by examples from human language, with the special case of English. The motivation holds for other natural and formal languages, and applications, e.g., in databases and other information sciences, when there is underspecified data dependent on possibilities for further instantiations.

Human language is abundant of different kinds of ambiguities, e.g., see (Loukanova, 2010) for a classification along with representative examples. Some semantic ambiguities can be associated with corresponding syntactic categories and parsings, while there are purely semantic ambiguities that can not be distinguished syntactically. In any case, semantic interpretations need formal representations by expressions, i.e., terms of a formal theory that provides mathematical medium for semantic interpretations.

In formal and computational grammars of human language, the grammatical function of a modifier in syntax is associated with varieties of syntactic categories. Notoriously, a modifier can contribute to semantic ambiguities, some of which can correspond to syntactic ambiguities, e.g., as for the sentence (1a) in (1b)–(1d). The semantic difference between the two parsings (1b) (1c) is explicit. To represent the semantics of (1b), one needs a term, in which $[\text{in Pittsburgh}]_{\text{PP}}$ is rendered to a subterm of type $(\widetilde{e} \rightarrow \widetilde{e})$, which is appropriate for both nominal expressions (i.e., common nouns and more complex NOMs) and verb phrases. Depending on semantic theory and available information, one may assume that the semantics of (1c) and (1d) are approximately the same or, alternatively, that (1d) has no reasonable meaning. So, with this limitation, there is no need

of parametric type to represent the semantics of the sentence (1a), which can be rendered by using underspecified terms of L^{λ}_{ar}, but without involving parametric types.

John bought the car. (1a)

[John [[bought]$_{V}$ [the [car [in Pittsburgh]$_{PP}$]$_{NOM}$]$_{NP}$]$_{VP}$]$_{S}$ (1b)

[John [[[bought]$_{V}$ [the [car]$_{NOM}$]$_{NP}$]$_{VP}$[in Pittsburgh]$_{PP}$]$_{VP}$]$_{S}$ (1c)

[[John [[bought]$_{V}$ [the [car]$_{NOM}$]$_{NP}$]$_{VP}$]$_{S}$ [in Pittsburgh]$_{PP}$]$_{S}$ (1d)

Similarly to prepositional phrases (PPs), adverbs syntactically can be either verbal or sentence modifiers. E.g., the adverb "amusedly " in a sentence like "The comedian was presenting his new joke amusedly." can contribute to different parsings with corresponding, different semantic interpretations. The entire class of grammatical modifiers, across syntactic categories, would benefit from polymorphic type assignment to modifiers. In this paper, we present motivation for type polymorphism with the shorter sentence (2a), which does not involve tense, aspects, and transitive verbs in the VP, while shares a common pattern of grammatical modification. The adverb "disturbingly" in (2a) can be either a verbal modifier, e.g., as in the parsing (2b), or a sentence modifier, as in (2c).

Jerry barks disturbingly. (2a)

[[Jerry]$_{NP}$ [[barks]$_{VP}$ disturbingly]$_{VP}$]$_{S}$ (2b)

[[Jerry [barks]$_{VP}$]$_{S}$ disturbingly]$_{S}$ (2c)

Semantically, with the parsing (2b), the sentence can be used to describe a situation where the action of barking itself is explicitly disturbing as a physical action, e.g., Jerry himself may be disturbed and sound that way. In such cases, we can render the adverb "disturbingly" into a constant of type appropriate for modifiers of VP, i.e., $((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to (\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}))$, as in (10), and the sentence into a formal term of L^{λ}_{ar}, as in (11b)–(11f).

In other contexts, a parsing corresponding to (2c), with the adverb "disturbingly" as a sentence modifier, requires a different type $(\widetilde{\mathsf{t}} \to \widetilde{\mathsf{t}})$, as in (12), which is appropriate for semantics of sentences, represented by (13a)–(13f). E.g., this is needed when the sentence is used to describe a situation that is disturbing, not necessarily because the barking action is disturbing per se, but in other aspects. E.g., Jerry himself may not be disturbed, and his action of barking may not be physically disturbing by itself at all, while the situation described by the component sentence [Jerry [barks]$_{VP}$]$_{S}$ is disturbing for other participants, i.e., agents, in the described environment.

For these examples and their representation in L^{λ}_{ar}, see (Loukanova, 2012a), which initiated representation om grammatical modifiers by underspecified recursion. Here we need to stress that the technique for using underspecified L^{λ}_{ar}-terms, where the underspecification covers only the presence of free, i.e., underspecified, recursion variables of L^{λ}_{ar}, as presented in (Loukanova, 2012a), requires extending the theory L^{λ}_{ar} by adding parametric types and extending the entire reduction system of L^{λ}_{ar}. This is the topic of this paper.

We introduce the extended formal system in the following sections. Then, in Section 7, we demonstrate the superiority of the extended theory L^{λ}_{ar}. At first, we give alternative formal representations of the semantics of (2b)–(2b) by using the original L^{λ}_{ar}, without parametric types. Then, we present the semantic ambiguity of (2a) as semantic underspecification, by terms of the extended L^{λ}_{ar}.

In this paper, we show that the use of the technique of underspecification, extended by adding parametric, underspecified types, is representative for handling semantic ambiguity. In particular, we apply it for representing underspecified ambiguities of the grammatical function of modification in human language. Instead of rendering a sentence into alternative formal terms, for all possible interpretations, when it is not known which one may be appropriate in different contexts, and with respect to different agents, it is better to render all of them into a single underspecified term with underspecified recursion variables and parametric types. Then, the parameters can be instantiated suitably depending on contexts and agents having additional information.

We shall provide renderings of this sentence in Section 7, after we present the necessary definitions of the extended, polymorphic, formal system L^{λ}_{ar}.

3 TYPES

We extend the type system of L^{λ}_{ar}, by adding a formal sub-language $\mathsf{Types}_{L^{\lambda}_{ar}}$ (abbreviated as Types).
Basic Type Constants: $\mathsf{BTypes} = \{\mathsf{e}, \mathsf{t}, \mathsf{s}\}$,
where e is for *entities* (*individuals*); t for truth values; s for states, i.e., context information
Type Variables: $\mathsf{TV} = \{\zeta_i \mid i = 0, 1, \dots\}$
Types $\mathsf{Types}_{L^{\lambda}_{ar}}$**:** defined recursively (in BNF)

$$\theta :\equiv \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid \sigma \mid (\tau \to \sigma) \mid \quad (3a)$$

$$\tau_0 \text{ where } \{\vartheta_1 :\overset{\text{type}}{=} T_1, \dots, \vartheta_m :\overset{\text{type}}{=} T_m\} \quad (3b)$$

where $\tau, \sigma, \tau_0 \in \mathsf{Types}$; $\vartheta_j \in \mathsf{TV}$; T_j are nonempty, finite sets of types ($j = 0, \dots, m$), such that $\{\vartheta_1 :\overset{\text{type}}{=}$

$T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\}$ satisfies the Acyclicity Constraint AC 1.

Acyclicity Constraint AC 1. For any pairwise different $\vartheta_j \in \mathsf{TV}$ and nonempty, finite sets T_j of types $(j=0,\ldots,m)$, $\{\vartheta_1 :\overset{\text{type}}{=} T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\}$ is *acyclic* iff there is $\mathsf{rank}\colon \{\vartheta_1,\ldots,\vartheta_m\} \longrightarrow \mathbb{N}$, such that

- if ϑ_j occurs freely in T_i then $rank(\vartheta_j) < rank(\vartheta_i)$.

We call the type terms that have free occurrences of type variables *underspecified types*. Some useful notations:

$$\mathsf{e}\to\mathsf{t}, \text{ the type of characteristic functions of sets of individuals} \tag{4a}$$

$$\widetilde{\mathsf{e}} \equiv (\mathsf{s}\to\mathsf{e}), \text{ the type of state dependent entities, i.e., entity "concepts"} \tag{4b}$$

$$\widetilde{\mathsf{t}} \equiv (\mathsf{s}\to\mathsf{t}), \text{ the type of state dependent truth, i.e., of state dependent propositions}$$

$$\widetilde{\tau} \equiv (\mathsf{s}\to\tau), \text{ for } \tau\in\mathsf{TV} \tag{4c}$$

The type (5) is for curryed functions from objects of type $\tau_1,\ldots,\tau_n$ to objects of type σ; (5) has type variables as components, i.e. it is underspecified.

$$(\tau_1\to\cdots\to(\tau_n\to\sigma)) \quad \sigma,\tau_i\in\mathsf{TV}, n\geq 0 \tag{5}$$

The type variables, e.g., $\tau\in\mathsf{TV}$, which we use as a specialized kind of recursion variables, introduced in the following section, represent "unknown" or "partly known" types, for which no, or only some, information is available. E.g.:

$$\Upsilon_1 \equiv (\tau_1\to\tau_2) \text{ where } \{\tau_1 :\overset{\text{type}}{=} \mathsf{e},\ \tau_2 :\overset{\text{type}}{=} \mathsf{t}\} \tag{6a}$$

$$\Upsilon_2 \equiv \tau \text{ where } \{\tau :\overset{\text{type}}{=} (\tau_1\to\tau_2),\ \tau_1 :\overset{\text{type}}{=} \mathsf{e},\ \tau_2 :\overset{\text{type}}{=} \mathsf{t}\} \tag{6b}$$

4 SYNTAX

$\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$ has typed vocabulary, i.e., for each $\tau\in\mathsf{Types}$:
Constants K**:** $K_\tau = \{\mathsf{c}_0{}^\tau,\ldots,\mathsf{c}_k^\tau,\ldots\}$,
Pure variables PV: $\mathsf{PV}_\tau = \{\mathsf{v}_0,\mathsf{v}_1,\ldots\}$,
Recursion variables RV: $\mathsf{RV}_\tau = \{\mathsf{r}_0,\mathsf{r}_1,\ldots\}$,

Pure and recursion variables can be of underspecified types, by type variables and alternative options via sets of types assigned to type variables, in (3b).
The terms $\mathsf{Terms}_{\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}}(\mathsf{Terms})$**:** Recursively, in BNF:

$$A :\equiv \mathsf{c}^\tau \mid x^\tau \mid \tag{7a}$$

$$\big[B^{(\sigma\to\tau)}(C^\sigma)\big]^\tau \mid \tag{7b}$$

$$\big[\lambda v^\sigma (B^\tau)\big]^{(\sigma\to\tau)} \mid \tag{7c}$$

$$\big[A_0^{\sigma_0} \text{ where } \{\vartheta_1 :\overset{\text{type}}{=} T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\} \{p_1^{\sigma_1} := A_1^{\sigma_1},\ldots, p_n^{\sigma_n} := A_n^{\sigma_n}\}\big]^{\sigma_0} \tag{7d}$$

where for $n,m\geq 0$, $\mathsf{c}^\tau\in K_\tau$; $x^\tau\in\mathsf{PV}_\tau\cup\mathsf{RV}_\tau$; $v^\sigma\in\mathsf{RV}_\sigma$; $A,B,A_i^{\sigma_i}\in\mathsf{Terms}$ $(i=0,\ldots,n)$; $p_i\in\mathsf{RV}_{\sigma_i}$ $(i=1,\ldots,n)$; $\vartheta_j\in\mathsf{TV}$; $T_j\subset\mathsf{Types}$ are nonempty, finite sets of types $(j=1,\ldots,m)$; such that the sequences:
$\{\vartheta_1 :\overset{\text{type}}{=} T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\}$ and
$\{p_1^{\sigma_1} := A_1^{\sigma_1},\ldots,p_n^{\sigma_n} := A_n^{\sigma_n}\}$ satisfy the Acyclicity Constraint AC 2.

Acyclicity Constraint AC 2. the sequences
$\{\vartheta_1 :\overset{\text{type}}{=} T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\}$ and
$\{p_1^{\sigma_1} := A_1^{\sigma_1},\ldots,p_n^{\sigma_n} := A_n^{\sigma_n}\}$ are (jointly) *acyclic* iff for a function $\mathsf{rank}\colon \{\vartheta_1,\ldots,\vartheta_m,p_1,\ldots,p_n\}\longrightarrow\mathbb{N}$,

1. if p_j occurs freely in A_i, then $rank(p_j) < rank(p_i)$
2. if ϑ_j occurs freely in T_i, then $rank(\vartheta_j) < rank(\vartheta_i)$
3. for $p_i^{\sigma_i} := A_i^{\sigma_i}$, if ϑ_j occurs freely in σ_i, then $rank(\vartheta_j) < rank(p_i)$

Informally, $\{p_1^{\sigma_1} := A_1^{\sigma_1},\ldots,p_n^{\sigma_n} := A_n^{\sigma_n}\}$ defines recursive step-wise computations of the values of $A_i^{\sigma_i}$ (i.e., of their denotations), which are saved in p_i. The requirement $rank(p_j) < rank(p_i)$, allows the value of $A_i^{\sigma_i}$ saved in $p_i^{\sigma_i}$ to depend on the value of $A_j^{\sigma_j}$ saved in $p_j^{\sigma_i}$, which occurs freely in $A_i^{\sigma_i}$, and on the values of p_l, ϑ_k with rank lower than $\mathsf{rank}(p_j)$. By the acyclicity AC 2, the computations defined by the recursion assignments $\{p_1^{\sigma_1} := A_1^{\sigma_1},\ldots,p_n^{\sigma_n} := A_n^{\sigma_n}\}$, along with the type constraints $\{\vartheta_1 :\overset{\text{type}}{=} T_1,\ldots,\vartheta_m :\overset{\text{type}}{=} T_m\}$, terminate after a finite number of steps.

A type τ can be underspecified in two ways. (1) τ can have free occurrences of type variables in it. An underspecified type τ_0 can be specified by adding specifications of some of its free variables, in the form of type constraints via type assignments: (2) Each type assignment $\vartheta_j :\overset{\text{type}}{=} T_j$ binds the type variable ϑ_j disjunctively, so it can still carry disjunctively underspecified information, when T_j has more than one element in it.

The terms (7d) of $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$ are essential for encoding two-fold semantic information: denotational values and algorithmic steps for computation of the denotational values. We leave the precise definitions of denotational and algorithmic semantics outside of the scope of this paper. Informally, the denotation of a term of the form (7d), is computed from the parts $A_i^{\sigma_i}$, by mutual recursion following the rank, saving the values on the way in $p_i^{\sigma_i}$.

5 REDUCTION CALCULUS

This section introduces the reduction system of L^{ta}_{ar}, which extends L^{λ}_{ar}.

Informally, *congruence* between terms (types) is the smallest relation $\equiv_c$ ($\overset{t}{\equiv}_c$) between terms (types) that is reflexive, symmetric, transitive, and closed under: term formation, renaming bound variables, reordering of assignments, congruence of types. A type $\tau \in \mathsf{Types}$ is *proper* iff it is not a type variable, i.e., $\tau \notin \mathsf{TV}$, otherwise it is *immediate.*

Reduction Rules for Types:

Congruence (t-cong)

If $\tau \equiv_c \sigma$, then $\tau \Rightarrow_t \sigma$

Transitivity (t-trans)

If $\tau \Rightarrow_t \sigma$ and $\sigma \Rightarrow_t \theta$ then $\tau \Rightarrow_t \theta$

Compositionality

If $\tau_1 \Rightarrow_t \tau_2$ and $\sigma_1 \Rightarrow_t \sigma_2$, then (t-rep12)
$(\tau_1 \to \sigma_1) \Rightarrow_t (\tau_2 \to \sigma_2)$

If $\tau_i \Rightarrow_t \sigma_i$, for $i = 0, \ldots, n$, then (t-rep3)
τ_o where $\{\vartheta_1 :\overset{\text{type}}{=} \tau_1, \ldots, \vartheta_n :\overset{\text{type}}{=} \tau_n\}$
$\Rightarrow_t \sigma_0$ where $\{\vartheta_1 :\overset{\text{type}}{=} \sigma_1, \ldots, \vartheta_n :\overset{\text{type}}{=} \sigma_n\}$

The Head Rule for Types (t-head)

$\left(\tau_0 \text{ where } \{\overrightarrow{\vartheta} :\overset{\text{type}}{=} \overrightarrow{\tau}\}\right)$ where $\{\overrightarrow{\theta} :\overset{\text{type}}{=} \overrightarrow{\sigma}\}$
$\Rightarrow_t \tau_0$ where $\{\overrightarrow{\vartheta} :\overset{\text{type}}{=} \overrightarrow{\tau},\ \overrightarrow{\theta} :\overset{\text{type}}{=} \overrightarrow{\sigma}\}$
given that no ϑ_i occurs free in any σ_j,
for $i = 1, \ldots, n$, $j = 1, \ldots, m$

The Bekič-Scott Rule for Types (t-B-S)

τ_0 where $\{\vartheta :\overset{\text{type}}{=} (\sigma_0 \text{ where } \{\overrightarrow{\theta} :\overset{\text{type}}{=} \overrightarrow{\sigma}\}),$
$\overrightarrow{\vartheta} :\overset{\text{type}}{=} \overrightarrow{\tau}\}$
$\Rightarrow_t \tau_0$ where $\{\vartheta :\overset{\text{type}}{=} \sigma_0,\ \overrightarrow{\theta} :\overset{\text{type}}{=} \overrightarrow{\sigma},\ \overrightarrow{\vartheta} :\overset{\text{type}}{=} \overrightarrow{\tau}\}$
given that no θ_i occurs freely in any τ_j,
for $i = 1, \ldots, n$, $j = 1, \ldots, m$

S-subtype Rule (S-subt)

For any type variables $t_1, \ldots, t_m \in \mathsf{TV}$ and any proper types $\tau_1, \ldots, \tau_n$,
$\{t_1, \ldots, t_m, \tau_1, \ldots, \tau_n\}$
$\Rightarrow_t \{t_1, \ldots, t_m, \vartheta_1, \ldots, \vartheta_n\}$ where $\{\vartheta_1 :\overset{\text{type}}{=} \tau,$
$\ldots, \vartheta_n :\overset{\text{type}}{=} \tau\}$

Reduction Rules for Terms: The set of the *reduction rules* for $\mathsf{Terms}_{L^{ta}_{ar}}$ extends the L^{λ}_{ar}-reduction (Moschovakis, 2006), by adding one more rule (type-r) defined here, to the L^{λ}_{ar} rules (cong), (trans), (rep1), (rep3), (rep3), (head), (B-S), (recap), (ap), (λ).

The Type Rule (type-r)

If $\tau \Rightarrow_t \sigma$, then $A^{\tau} \Rightarrow A^{\sigma}$

6 KEY THEORETICAL FEATURES OF L^{ta}_{ar}

We give some of the major results (with proofs in an extended paper) that are essential for the algorithmic semantics of the language L^{ta}_{ar}. We say that a type $\tau \in \mathsf{Types}$ is *irreducible* iff

$$\text{for all } \sigma \in \mathsf{Types}, \text{ if } \tau \Rightarrow \sigma, \text{ then } \tau \overset{t}{\equiv}_c \sigma \qquad (8)$$

We say that a term $A \in \mathsf{Terms}$ is *irreducible* iff

$$\text{for all } B \in \mathsf{Terms}, \text{ if } A \Rightarrow B, \text{ then } A \equiv_c B \qquad (9)$$

Theorem 1 (Canonical Form Theorem for Types)**.** *For each type term $\tau \in \mathsf{Types}$, there is a unique, up to congruence, irreducible type term $\mathsf{cf}(\tau)$, such that:*

1. *(a) $\mathsf{cf}(\tau) \equiv \tau$ or*
   (b) $\mathsf{cf}(\tau) \equiv \tau_0$ where $\{\vartheta_1 :\overset{\text{type}}{=} \tau_1, \ldots, \vartheta_n :\overset{\text{type}}{=} \tau_n\}$
2. *$\tau \Rightarrow_t \mathsf{cf}(\tau)$*
3. *if $\tau \Rightarrow_t \sigma$ and σ is irreducible, then $\sigma \overset{t}{\equiv}_c \mathsf{cf}(\tau)$, i.e., $\mathsf{cf}(\tau)$ is the unique up to congruence irreducible type to which τ can be reduced.*

Theorem 2 (Canonical Form Theorem)**.** *For each term A^{τ}, there are a unique, up to congruence, irreducible type $\mathsf{cf}(\tau)$ and a unique, up to congruence, irreducible term $\mathsf{cf}(A^{\tau})$, such that:*

1. *(a) $\mathsf{cf}(A^{\tau}) \equiv A^{\mathsf{cf}(\tau)}$ or*
   (b) $\mathsf{cf}(A^{\tau}) \equiv A_0^{\tau_0}$ where $\{p_1 := A_1^{\tau_1}, \ldots, p_n := A_n^{\tau_n}\}$, where $\mathsf{cf}(\tau) \equiv_c \tau_0 \equiv_c \mathsf{cf}(\tau_0)$ and $\tau_i \equiv_c \mathsf{cf}(\tau_i)$, for $i = 1, \ldots, n$.
2. *$A^{\tau} \Rightarrow \mathsf{cf}(A^{\tau})$*
3. *if $A^{\tau} \Rightarrow B^{\sigma}$ and B^{σ} is irreducible, then $B^{\sigma} \equiv_c \mathsf{cf}(A^{\tau})$, i.e., $\mathsf{cf}(A^{\tau})$ is the unique up to congruence irreducible term of irreducible type to which A^{τ} can be reduced.*

The proofs of the above theorems use induction by the definitions of $\mathsf{Types}_{L^{ta}_{ar}}$, $\mathsf{Terms}_{L^{ta}_{ar}}$, and the reduction rules. They are too long to be included in this paper, and will be provided in an extended work. See (Moschovakis, 2006) for the proof of the Canonical Form Theorem in L^{λ}_{ar} (without polymorphism), corresponding to Theorem 2.

The canonical forms have a distinguished feature that is part of their algorithmic role: they provide algorithmic patterns for computations of the denotations of all basic components, i.e., algorithms for semantic computations. The more general terms provide algorithmic patterns represented by sub-terms for the fundamental computational operators — functional application, λ-abstraction, component values that are saved in recursion variables upon the recursive computations. The assignments of values to the

recursion variables of lower ranks provide the specific basic data that feeds-up the successive computational steps.

The more general terms and sub-terms classify language expressions with respect to their semantics and determine the algorithms for computing their denotations. The canonical forms represent the decomposition of the algorithmic steps to the most basic ones.

Informally, two terms $A, B \in \mathsf{Terms}$ are algorithmically equivalent, i.e., algorithmically synonymous, $A \approx B$, iff (a) when A and B have algorithmic senses (i.e., A and B are proper), then their denotations (which are equal) are computed by the same algorithm; (b) otherwise (i.e., A and B are immediate), A and B have the same denotations.

7 FORMAL PRESENTATION OF MOTIVATION

Now, we shall supplement the above formal definitions with examples and further motivation. The formal system allows flexible choices depending on application needs. For example, we can render the adverb "disturbingly" into a constant $disturbingly_1$ as in (10), where the type is specific without any parameters.

$$\text{disturbingly} \xrightarrow{\text{render}} disturbingly_1 : ((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to (\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})) \tag{10}$$

Then, with the rendering (10), we can render the sentence parsing (11a) into the terms in (11b)–(11f):

$$[[\text{Jerry}]_{\text{NP}}\ [\text{barks disturbingly}]_{\text{VP}}]_{\text{S}} \xrightarrow{\text{render}} \tag{11a}$$

$$[disturbingly_1(bark)](jerry) \tag{11b}$$

$$\Rightarrow [disturbingly_1(b) \text{ where } \{b := bark\}](jerry) \quad \text{(ap, rep1)} \tag{11c}$$

$$\Rightarrow disturbingly_1(b)(jerry) \text{ where } \{b := bark\} \quad \text{(recap)} \tag{11d}$$

$$\Rightarrow [disturbingly_1(b)(j) \text{ where } \{j := jerry\}] \text{ where } \{b := bark\} \quad \text{(ap, rep3)} \tag{11e}$$

$$\Rightarrow_{\mathsf{cf}} disturbingly_1(b)(j) \text{ where } \{j := jerry, b := bark\} \quad \text{(head)} \tag{11f}$$

Typically, in the existing systems of computational semantics and computational grammars that incorporate semantic representations, in order to represent alternative syntactic categories, type assignments, or meanings of a lexeme (word), that lexeme is represented in the system by a set of different lexical items, e.g., by marking a common lexeme sequence with some label or index, correspondingly. Similarly, we can render the adverb "disturbingly" into a second $\mathrm{L}^{\lambda}_{\mathrm{ar}}$-constant $disturbingly_2$ as in (12), which is needed for the same sentence with the alternative parsing in (13a) and its corresponding interpretation represented by (13b)–(13f). As in (10) and (11b)–(11f), the types of the constant $disturbingly_2$ in (12) and the terms in (13b)–(13f) are specific, without any parameters.

$$\text{disturbingly} \xrightarrow{\text{render}} disturbingly_2 : (\widetilde{\mathsf{t}} \to \widetilde{\mathsf{t}}) \tag{12}$$

The type of the rendering (12) is suitable for rendering of the sentence as in (13a)–(13f):

$$[[\text{Jerry } [\text{barks}]_{\text{VP}}\ ]_{\text{S}} \text{ disturbingly}]_{\text{S}} \xrightarrow{\text{render}} \tag{13a}$$

$$disturbingly_2\big(bark(jerry)\big) \quad \text{(ap)} \tag{13b}$$

$$\Rightarrow disturbingly_2(b) \text{ where } \{b := bark(jerry)\} \quad \text{(ap, rep3)} \tag{13c}$$

$$\Rightarrow disturbingly_2(b) \text{ where } \{b := [bark(j) \text{ where } \{j := jerry\}]\} \quad \text{(B-S)} \tag{13d}$$

$$\Rightarrow_{\mathsf{cf}} disturbingly_2(b) \text{ where } \{b := bark(j), j := jerry\} \tag{13e}$$

$$\approx disturbingly_2(b) \text{ where } \{j := jerry, b := bark(j)\} \tag{13f}$$

While, the above alternative renderings can be appropriate in specific environments, with respect to specific agents, such specific information may not be available. In such cases it is more adequate to render the sentence into a formal term that is underspecified and open to be instantiated when suitable information is available. We can not render the verb "disturbingly" into a parameter that could be instantiated either as $disturbingly_1$ or $disturbingly_2$ in the standard $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ since every parameter and every constant have to be of specific types as in (10) and (12). Our extended formal system TTofAR, with parametric types, formalizes such possibilities, e.g., to render the verb "disturbingly" to a constant *disturbingly* of parametric type, as in (14). The variable ζ is a type variable, which can be instantiated when there is suitable information available.

$$\text{disturbingly} \xrightarrow{\text{render}} disturbingly : (\widetilde{\zeta} \to \widetilde{\zeta}) \tag{14}$$

Note that, instead of assigning a simple type variable to the constant *disturbingly*, we have used the underspecified (i.e., parametric) type term $(\widetilde{\zeta} \to \widetilde{\zeta})$, since we "know" some information about the parametric type, which we can represent by parametric type expression. The type $(\widetilde{\zeta} \to \widetilde{\zeta})$ is functional and requires

that the term to be modified and the resulted, modified, term to be of the same "unknown" type $\widetilde{\zeta}$. This is characteristic for any modifier expressions in artificial and natural languages. In addition, $\widetilde{\zeta} \equiv (\mathsf{s} \to \zeta)$ is state dependent. One could just use simple, parametric modifier type $\zeta \to \zeta$).

Then, the sentence can be rendered to an underspecified term as in (15b), where the type of the constant *disturbingly* is parametric and underspecified.

$$[\text{Jerry barks disturbingly}]_{\mathrm{S}} \xrightarrow{\text{render}} \tag{15a}$$

$$A \text{ where } \{j := jerry, B := bark, D := disturbingly\} \tag{15b}$$

When sufficient information is available, the type variable ζ can be instantiate suitable, either as $(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})$, as in (16), or as $\widetilde{\mathsf{t}}$. Thus, the already specified term (16) corresponds to the denotation of the terms in (11a)–(11f).

$$A \text{ where } \{\zeta :\overset{\text{type}}{=} (\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})\}\{j := jerry, B := bark, D := disturbingly\} \tag{16}$$

We have presented an example from human language, as a pattern of handling language ambiguities. Semantic ambiguities are abundant in human languages, and make language processing, understanding, and communications, via computerized system, one of the difficult areas in theories and applications. Representing semantic information is particularly difficult, because resolving multiple interpretations is highly context dependent, and not only inefficient, but also unpredictable without sufficient context information. In this section, we have presented these problems with a pattern example, and demonstrated formalization of ambiguous, multiple interpretations, computationally, with the type polymorphism of $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$. Specific interpretations can be obtained from the formal terms, via context and agents providing instantiations.

8 CONCLUSIONS AND FUTURE WORK

The notion of *referencial intension*, which we also call *algorithmic intension*, in the languages of recursion, including $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$ covers the computational aspect of the concept of meaning — the algorithmic steps by which denotations are computed. The *algorithmic intension*, $\mathsf{Int}(A)$, of a meaningful term A is the tuple of functions (a recursor) that is determined by its canonical form $\mathsf{cf}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$. Thus, the languages of recursion offer a formalisation of semantic layers: denotations and algorithmic steps for computation of denotations. When specific denotations depend on context information that is unknown, the algorithms can be represented by $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$ terms in canonical forms with free type and recursion variable.

$$\underbrace{\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}} \Longrightarrow \text{Type Specification} \Longrightarrow \text{Intensions (Algorithms)} \Longrightarrow \text{Denotations}}_{\text{Algorithmic Semantics}}$$

By $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$, we have extended, the concept of algorithm to a generalized notion of *underspecified algorithm*, with respect to polymorphism, and in addition, for acyclic computations, i.e., polymorphic algorithms that potentially close-off, which we consider unique among theoretic formalizations of semantic concepts.

We plan to extend the formal system presented in this paper in two directions. On the theoretical side, we work on extending the system with other reduction rules and calculi. Such work broadens the range of applications of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ to computational semantics of human languages, as well as to various areas of AI. E.g., we consider that it is important to add constraints over underspecified types, along other kinds of constraints over possible instantiations of free recursion variable, e.g., in the line of (Loukanova, 2013a).

On the application side, we target applications to computational syntax-semantics interface in natural language processing, e.g., along the lines of (Loukanova, 2011d; Loukanova, 2011e; Loukanova, 2011a; Loukanova, 2012b) and (Loukanova and Jiménez-López, 2012). Ambiguities and underspecification are characteristic for human languages. (Loukanova, 2012a) presented ambiguity among some human language modifiers. A vast range of ambiguous modifiers require to be rendered with terms of varying semantic types. The type system in this paper provides formalization of such underspecification. We work on extending furthermore the expressive power of the formal system $\mathrm{L}^{\mathrm{ta}}_{\mathrm{ar}}$ introduced in this paper, for covering other classes of ambiguities.

REFERENCES

Barwise, J. (1981). Scenes and other situations. *The Journal of Philosophy*, 78:369–397.

Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys (CSUR)*, 17(4):471–523.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. (2005). Minimal recursion semantics: an introduction. *Research on Language and Computation*, 3:281–332.

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

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. (2010). Computational Syntax-Semantics Interface. In Bel-Enguix, G. and Jiménez-López, M. D., editors, *Language as a Complex System: Interdisciplinary Approaches*, pages 111–150. Cambridge Scholars Publishing.

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

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

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

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

Loukanova, R. (2011e). 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 Recursion. 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. (2016). 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., De Tré, G., 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. 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*, 29:27–89.

Muskens, R. (1995). *Meaning and Partiality*. Studies in Logic, Language and Information. CSLI Publications, Stanford, California.

Thomason, R. H., editor (1974). *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, Connecticut.

Van Lambalgen, M. and Hamm, F. (2004). *The Proper Treatment Of Events*. Explorations in Semantics. Wiley-Blackwell, Oxford.

Villadsen, J. (2010). *Nabla: A Linguistic System based on Type Theory*, volume 3 of *Foundations of Communication and Cognition (New Series)*. LIT Verlag Münster-Hamburg-Berlin-Wien-London.