ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 12 N. 1 (2023), e29081
eISSN: 2255-2863
DOI: https://doi.org/10.14201/adcaij.29081

Restricted Computations and Parameters in Type-Theory of Acyclic Recursion

Roussanka Loukanova

Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria
rloukanova@gmail.com

KEYWORD

Algorithms; Types; Parameters; Restrictor Operator; Reduction Calculi; Iterative Computations; Compiler Iterations;

ABSTRACT

The paper extends the formal language and the reduction calculus of Moschovakis type-theory of recursion, by adding a restrictor operator on terms with predicative restrictions. Terms with restrictions over memory variables formalise inductive algorithms with generalised, restricted parameters. The extended type-theory of restricted recursion (TTRR) provides computations for algorithmic semantics of mathematical expressions and definite descriptors, in formal and natural languages.

The reduction calculi of TTRR provides a mathematical foundation of the work of compilers for reducing recursive programs to iterative ones. The type-theory of acyclic recursion (TTAR) has a special importance to syntax-semantics interfaces in computational grammars.

1. Introduction

This paper is part of the author's work on development of a new type-theory of the mathematical notion of algorithms, its concepts, and potentials for applications to advanced, computational technologies, especially in Artificial Intelligence (AI). For the initiation of this new theory, see the original work on the formal languages of recursion (FLR) (Moschovakis, 1989; Moschovakis, 1993; Moschovakis, 1997). The formal languages of recursion FLR are untyped systems. The typed version of this new approach to algorithmic, acyclic computations was introduced, for the first time in (Moschovakis, 2006), by designating it with $\mathrm{L}^{\lambda}_{ar}$, and demonstrating it by its applications to computational semantics of natural language. For more recent developments of the language and theory of acyclic algorithms $\mathrm{L}^{\lambda}_{ar}$, see, e.g., (Loukanova, 2019b; Loukanova, 2019c; Loukanova, 2020b).

The class L^{λ}_{r}, of the typed formal languages and theories of full recursion, is an extension of (Gallin, 1975) logic TY_2, and thus, of Montague Intensional Logic (IL), (Thomason, 1974). In this paper, we

extend the formal language and reduction calculus of L^{λ}_{ar} and L^{λ}_{r}, by incorporating restrictions on terms designating objects and algorithmic computations, which are required to satisfy the corresponding restrictions.

Two standard, distinctive approaches for representing definite descriptions by logic expressions were introduced by (Russell, 1905) and (Strawson, 1950). These two different interpretations were reconsolidated by using Situation Theory to represent each of them, see (Loukanova, 2001), by using situated, restricted parameters.

At first, we extend the type-theory of acyclic recursion L^{λ}_{ar}, by adding a restrictor operator and corresponding restricted L^{λ}_{ar}-terms, their denotational semantics, and reduction rules.

We demonstrate the application of the extended L^{λ}_{ar}, by representing the denotational and algorithmic semantics of various kinds of definite descriptors, by rendering them into restricted L^{λ}_{ar}-terms.

The definite descriptors may have interpretations that can be:

(1) Uniqueness conditions over objects, similar to (Ludlow, 2021), while expressed in type-theory L^{λ}_{ar}
(2) Restricted references to objects, the restricted values of which are computed in context
(3) In either case, the type-theory L^{λ}_{ar} allows the values of the definite descriptors to be erroneous, by flagging them with designated values for errors

The introduction of restricted memory (i.e., recursion) variables and restricted L^{λ}_{ar}-terms, introduced in this paper, is a new, algorithmic approach, which pertains to syntax-semantics of programming languages and other specification languages widely used in Computer Science and Artificial Intelligence (AI).

The reduction calculi of L^{λ}_{ar} reduces every L^{λ}_{ar}-term to its canonical form, which provides a mathematical foundation of the work of compilers for reducing recursive programs to iterative ones. The type-theory L^{λ}_{ar} has a special importance to computational syntax-semantics of human language, i.e., natural language

Restricted variables designate memory slots for storing information constrained to satisfy propositional restrictions. The definite descriptor provides restrictions for imposing properties over data and in algorithms with restricted parameters. The algorithmic restrictor is specifically important for computational semantics of formal languages, e.g., in programming, specification languages in data and Artificial Intelligence (AI), and computational syntax-semantics of natural language. In particular, we present alternative possibilities for algorithmic semantics of singular, nominal expressions. For example, a Noun Phrase (NP), which is a definite description, such as "the cube", can designate an object in a semantic domain, via identifying it as the unique object satisfying the property denoted by the description "cube". Definite descriptors are abundant in the languages of mathematics, as specialised, subject delimited fragments of human language, which can be intermixed with mathematical expressions, e.g., "the natural number n, such that $n + 1 = 2$".

This paper is part of the author's work on the development of type-theory of algorithms by targeting versatile applications, especially in advanced technology with Natural Language Processing (NLP) in AI. For instance, a sequence of papers (Loukanova, 2011a; Loukanova, 2011b; Loukanova, 2011c; Loukanova, 2012b; Loukanova, 2012a; Loukanova, 2013c; Loukanova, 2013b; Loukanova, 2013a) use L^{λ}_{ar} to represent fundamental semantic notions in human language. The research presented in (Loukanova, 2016) is on the applications of L^{λ}_{ar} in the important topic of quantifier scope ambiguities, while (Loukanova, 2020b) is on formalisation of binding arguments of recursive properties, represented by functions, across mutual recursion assignments in recursion terms and modeling neural receptors.

Figure 1 depicts two approaches for algorithmic semantics of natural language (NL) used in the author's work. By (1a), the computational grammar of NL delivers syntactic analyses of NL expressions. Afterwards, the syntactic analyses are used to render them into terms of L^{λ}_{ar}, including the extended L^{λ}_{rar} introduced in this paper, to represent their algorithmic semantics. This approach is useful when algorithmic semantics of NL is the focus without providing syntactic analyses (in details), e.g., as in this paper. By (1b), the computational grammar of NL provides syntax-semantics analyses of NL, in a compositional mode (Loukanova, 2019a). Figure 2 gives the general scheme of the internal to L^{λ}_{rar} syntax-semantics relation between the syntax of the terms of L^{λ}_{rar} (L^{λ}_{ar}, L^{λ}_{r}) and their algorithmic and denotational semantics.

Recent work (Loukanova, 2019b; Loukanova, 2019c) extends the reduction calculus of type-theory of algorithms in useful ways, by targeting practical applications in advanced technologies, in line with the topic of this paper. A fundamental rule and conversion, η-conversion, of classic λ-calculi, was investigated for L^{λ}_{ar} (Loukanova, 2020a), by providing a version that grasps recursion assignments in canonical forms of L^{λ}_{ar} terms. A broader work (Loukanova, to appear) presents an algorithmic η-rule and η-reduction in L^{λ}_{ar}.

To the best of the author's knowledge, a precise, mathematical representation of the semantic notion of restricted parameters, as per se semantic objects, in type-theoretic, situated models was introduced by (Loukanova, 1991). The mathematical, semantic models introduced in that work are mathematical structures that include situated, dependent types defined hierarchically, by primitive types and recursion on situated propositions. The mathematical structures are based on relations, without coding them by one-argument functions, i.e., without currying encoding. At a semantic level, the restricted, typed objects of Situation Theory, were used to model both Russell and Strawson definite descriptors (Loukanova, 2001). In contrast to that work on Situation Theory, in this paper, we introduce syntactic counterparts of the semantic objects that are restricted parameters — restricted, memory (recursion) variables — as a special case of new, restricted L^{λ}_{ar}-terms.

The generalised restrictor operator in Def. 2, (6a)–(6e), was introduced with a focus on algorithmic semantics, by extending the type-theory of algorithms L^{λ}_{ar} (Loukanova, 2021). The current paper presents this restrictor operator, by the syntax and semantics of type-theory of parametric algorithms, via covering syntax-semantics interrelations in L^{λ}_{ar} and L^{λ}_{r}. In this extended paper, we emphasise the significance of the theoretical features of L^{λ}_{ar}, which we have been developing by targeting advanced applications with new intelligent technologies, primarily through distributed computing in AI.

In Sect. 2, we introduce the formal syntax of the type-theory of algorithms with acyclic recursion and restrictor, L^{λ}_{ar} and L^{λ}_{rar}, by pointing to the corresponding versions with full recursion, L^{λ}_{r} and L^{λ}_{rr}. This new restrictor operator, for propositional restrictions, is at the object level of the formal language. In Sect. 3, we add the definition of the denotational semantics of L^{λ}_{ar}, by including the denotation of the restrictor terms of the extended L^{λ}_{rar}. Section 4 provides the full definition of the canonical forms of the terms of the extended type-theory L^{λ}_{rar}. This definition has distinctive significance because the

$$\underbrace{\text{Computational Syntax of NL}}_{\text{Computational Grammar}} \xrightarrow{\text{render}} L^{\lambda}_{ar} \,/\, L^{\lambda}_{rar} \tag{1a}$$

$$\underbrace{\text{Computational Syntax of NL} \xleftrightarrow{\text{render}} L^{\lambda}_{ar} \,/\, L^{\lambda}_{rar}}_{\text{Computational Grammar: Including Syntax-Semantics Interface}} \tag{1b}$$

Figure 1: Algorithmic Syntax-Semantics Interface in Computational Grammar of Natural Language

canonical forms of the proper terms determine the algorithmic semantics of the type-theory of parametric algorithms, in their respective versions, L^{λ}_{ar}, L^{λ}_{rar}, L^{λ}_{r}, and L^{λ}_{rr}. Section 5 introduces the most significant ingredient of the type-theory of algorithms, the reduction calculus, by covering terms with restrictor operator. The calculus is equipped with a set of reduction rules that define the system of reducing each term to its canonical form. In Sect. 6, we introduce the notion of generalised variables, as restricted memory slots, by using the restrictor operator. Then we investigate their major role in the theory of algorithms, as specialised memory slots for storing information that is computed algorithmically and restricted by conditions, also computed algorithmically. We demonstrate their reduction to basic memory variables which are irreducible terms. In Sect. 7, we present the significant role of algorithmic semantics in the type-theories of acyclic and full recursion, L^{λ}_{rar} and L^{λ}_{rr} , with respect to their denotational semantics. The restrictor operator in the formal languages L^{λ}_{ar}, L^{λ}_{rar}, L^{λ}_{r}, and L^{λ}_{rr} have algorithmic meanings, which is demonstrated in Sect. 8.

Definite descriptors play an important role in the many formal languages of logic, in many different applications in computer science, in computational grammars of natural languages, in Natural Language Processing (NLP), by corresponding reflections to AI. In Sect. 9, we provide possibilities for the application of restricted algorithms to represent definite descriptors, which are abundant in data of various forms, including in domain specific areas. For clarity, we exemplify them with expressions from human language, i.e., natural language. Sect. 9, covers various options for algorithmic semantics of descriptors by L^{λ}_{rar}. In Sect. 10, we summarise the significance of the restrictor operator, some of the existing, forthcoming, and future work.

2. Terms for Algorithms with Acyclic Recursion and Restrictor

Definition 1 (Gallin Types). See (Gallin, 1975) logic TY_2:

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

We shall designate the type of the state dependent objects of type $(s \to \tau)$ by (2), see Sect.3. The semantic objects in the domain $\mathbb{T}_{(s\to\tau)}$ are called Carnap's Intensions:

$$\widetilde{\tau} \equiv (s \to \tau), \qquad \text{for } \tau \in \text{Types} \tag{2}$$

The set of the *constants* Consts *(also, shortly denoted by K)* are given by denumerable (finite) sets of typed constants:

$$\text{Consts}_\tau = \{c^\tau_0, \ldots, c^\tau_k\} \quad (k \geq 0) \quad \text{for all } \tau \in \text{Types}, K_\tau, \quad \text{Consts} = \bigcup_\tau \text{Consts}_\tau \tag{3}$$

L^{λ}_{ar} has *Pure Variables*, given by denumerable sets of typed pure variables:

$$\text{PureV}_\tau = \{v^\tau_0, \ldots\} \quad \text{for all } \tau \in \text{Types}, K_\tau, \quad \text{PureV} = \bigcup_\tau \text{PureV}_\tau \tag{4}$$

L^{λ}_{ar} has *Recursion (Memory) Variables* given by denumerable sets of typed recursion (memory) variables: $\text{RecV} = \bigcup_\tau \text{RecV}_\tau$, for $\text{RecV}_\tau = \{r^\tau_0, \ldots\}$. The vocabulary is without intersections: $\text{Consts} \neq \text{RecV} \neq \text{PureV}$, and the set of all variables is $\text{Vars} = \text{PureV} \bigcup \text{RecV}$.

$$\text{RecV}_\tau = \{r^\tau_0, \ldots\} \quad \text{for all } \tau \in \text{Types}, K_\tau, \quad \text{RecV} = \bigcup_\tau \text{RecV}_\tau \tag{5}$$

In this section, we extend the formal language of $\mathrm{L}_{ar}^{\lambda}$ by adding a constant for a restrictor operator (such that), for the formation of new terms, with restrictions. The extended formal language of $\mathrm{L}_{rar}^{\lambda}$ has the same set of types, constants and variables, as $\mathrm{L}_{ar}^{\lambda}$. Sometimes, when it is clear from the context, we shall write $\mathrm{L}_{ar}^{\lambda}$ instead of $\mathrm{L}_{rar}^{\lambda}$, by assuming that $\mathrm{L}_{ar}^{\lambda} = \mathrm{L}_{rar}^{\lambda}$. The Terms are defined by adding one more rule (6e), for the restrictor operator (a special operator constant) such that. Then, we extend the definition of the canonical forms and reduction calculus of $\mathrm{L}_{ar}^{\lambda}$.

Here, we present the recursive rules of the definition of the set of $\mathrm{L}_{rar}^{\lambda}$-terms by Def. 2, (6a)–(6e), using an extended, typed Backus-Naur Form (TBNF) style, with the assumed types given as superscripts.

We use the typical notations for type assignments, $A : \tau$, and A^{τ}, to express that A is a term of type τ.

Definition 2 (Terms). The set $\mathrm{Terms}_{\mathrm{L}_{rar}^{\lambda}}$ of the terms consists of the expressions defined by the following rules:

$$A :\equiv \mathrm{c}^{\tau} : \tau \mid x^{\tau} : \tau \qquad (\text{for } \mathrm{c}^{\tau} \in \mathrm{Consts}_{\tau},\ x^{\tau} \in \mathrm{PureV}_{\tau} \cup \mathrm{RecV}_{\tau}) \tag{6a}$$

$$\mid B^{(\sigma\to\tau)}(C^{\sigma}) : \tau \qquad (\textit{application term}) \tag{6b}$$

$$\mid \lambda(v^{\sigma})\,(B^{\tau}) : (\sigma \to \tau) \qquad (\lambda\textit{-term}) \tag{6c}$$

$$\mid [A_0^{\sigma_0} \text{ where } \{p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n}\}] \ : \sigma_0 \qquad (\textit{recursion term}) \tag{6d}$$

$$\mid (A_0^{\sigma_0} \text{ such that } \{C_1^{\tau_1}, \ldots, C_m^{\tau_m}\}) : \sigma_0' \qquad (\textit{restrictor / restriction term}) \tag{6e}$$

given that:

(1) $\mathrm{c} \in \mathrm{Consts}$, $x \in \mathrm{PureV} \cup \mathrm{RecV}$, $v^{\sigma} \in \mathrm{PureV}_{\sigma}$

(2) B, $C \in$ Terms of the respective types

(3) in (6d), for $n \geq 0$, $i = 0, \ldots, n$, $\sigma_i \in$ Types, $A_i \in \mathrm{Terms}_{\sigma_i}$;
for $i = 1, \ldots, n$, $p_i \in \mathrm{RecV}_{\sigma_i}$, are pairwise different recursion variables, of the same types as of the terms, correspondingly assigned to them

(4) the sequence of assignments in (6d) satisfies the AC, (8a)–(8c)

(5) in (6e), for $j = 1, \ldots, m$ ($m \geq 0$),
   (a) $C_j^{\tau_j} \in$ Terms, for $\tau_j \in$ Types
   (b) $\tau_j \in$ Types is either $\tau_j \equiv \mathrm{t}$, i.e., the type t of state-independent truth values, or $\tau_j \equiv \widetilde{\mathrm{t}} \equiv (\mathrm{s} \to \mathrm{t})$, i.e., the type $\widetilde{\mathrm{t}}$ of truth values that may depend on states

and the type σ_0' of the restrictor (restriction) term $A^{\sigma_0'}$ in (6e) is:

$$\sigma_0' \equiv \begin{cases} \sigma_0, & \text{if } \tau_i \equiv \mathrm{t}, \text{ for all } i \in \{1, \ldots, n\} \qquad (7a) \\ \sigma_0 \equiv (s \to \sigma), & \text{if } \tau_i \equiv \widetilde{\mathrm{t}}, \text{ for some } i \in \{1, \ldots, n\}, \text{ and} \qquad (7b) \\ & \quad \text{for some } \sigma \in \mathrm{Types},\ \sigma_0 \equiv (\mathrm{s} \to \sigma) \\ \widetilde{\sigma_0} \equiv (s \to \sigma_0), & \text{if } \tau_i \equiv \widetilde{\mathrm{t}}, \text{ for some } i \in \{1, \ldots, n\}, \text{ and} \qquad (7c) \\ & \quad \text{there is no } \sigma, \text{ s.th. } \sigma_0 \equiv (\mathrm{s} \to \sigma) \end{cases}$$

The type σ_0' of the restrictor (restriction) term $A^{\sigma_0'}$ in (6e) is that of $A_0^{\sigma_0}$, in the cases (7a)–(7b). In the case (7c), the type σ_0 of A_0, does not depend directly on states because there is no σ, such that

$\sigma_0 \equiv (s \to \sigma)$, while at least one of the restrictions C_i is of state-dependent type, $\tau_i \equiv \widetilde{\tau}$, for some $i \in \{1, \ldots, n\}$. The state-dependence of the entire restriction term A is reflected by lifting the type σ_0 of A_0 to its state dependent version, in the type σ_0' of $A : \sigma_0'$.

For each $\tau \in$ Types, Terms_τ is the set of the terms of type τ.

We call the terms A of the form (6d) *recursion terms*, and the terms of the form (6e) *restriction or restrictor terms* (sometimes, also, *restricted terms*).

Acyclicity Constraint (AC) For any $\sigma_i \in$ Types, $A_i \in \text{Terms}_{\sigma_i}$, and pairwise different recursion variables, $p_i \in \text{RecV}_{\sigma_i}$, for $i = 1, \ldots, n$:

$$\{p_1 := A_1, \ldots, p_n := A_n\}\ (n \geq 0) \text{ is an acyclic sequence}$$
$$\text{iff there is a function rank, such that:} \tag{8a}$$
$$\text{rank} : \{p_1, \ldots, p_n\} \to \mathbb{N} \tag{8b}$$
$$\text{for all } i, j \in \{1, \ldots, n\}, \quad p_j \in \text{FreeV}(A_i) \Longrightarrow \text{rank}(p_j) < \text{rank}(p_i) \tag{8c}$$

We call the sequence (8a) *acyclic system of assignments*.

The formal language of L^{λ}_{rar}, without the AC, (8a)–(8c), provides the version of the type-theory L^{λ}_{rr} of full recursion and restrictors.

Bound and Free Variables The sets FreeV(A) and BoundV(A), respectively of the free and bound variables of the L^{λ}_{rar}-terms A, are defined by structural recursion on A, in the usual way, as part of Def. 2, (6a)–(6e), for each clause, with the exception of the recursion and restriction terms, i.e.:

(BFV1) For any recursion term A of the form (6d), i.e., $A \equiv [A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\}]$, all the free occurrences of $p_1, \ldots, p_n$, in each of the terms $A_0, \ldots, A_n$, are *bound occurrences* in A. All *other free (bound) occurrences* of variables in $A_0, \ldots, A_n$ are also *free (bound)* in A, and:

$$\begin{aligned} \text{BoundV}(A) &= \text{BoundV}\left(A_0 \text{where}\left\{p_1 := A_1, \ldots, p_n := A_n\right\}\right) \\ &= \bigcup_{i=0}^{n}\left(\text{BoundV}(A_i)\right) \bigcup \left\{p_1, \ldots, p_n\right\} \end{aligned} \tag{9a}$$

$$\begin{aligned} \text{FreeV}(A) &= \text{FreeV}\left(A_0 \text{where}\left\{p_1 := A_1, \ldots, p_n := A_n\right\}\right) \\ &= \bigcup_{i=0}^{n}\left(\text{FreeV}(A_i)\right) - \left\{p_1, \ldots, p_n\right\} \end{aligned} \tag{9b}$$

(BFV2) For any restriction term A of the form (6e), i.e., $A \equiv (A_0^{\sigma_0} \text{ such that } \{C_1^{\tau_1}, \ldots, C_m^{\tau_m}\})$, all the free (bound) occurrences of variables in $A_0, C_1, \ldots, C_m$ are also free (bound) in A, and:

$$\text{BoundV}(A) = \text{BoundV}(A_0) \cup \bigcup_{i=1}^{m}\left(\text{BoundV}(C_i)\right) \tag{10a}$$

$$\text{FreeV}(A) = \text{FreeV}(A_0) \cup \bigcup_{i=1}^{m}\left(\text{FreeV}(C_i)\right) \tag{10b}$$

All occurrences of constants in any L^{λ}_{rar}-term are free.

Notation 1. Often, abbreviations for sequences are useful, e.g., for $i = 1, \ldots, n$, $j = 1, \ldots, m$ $(n, m \geq 0)$, $p_i \in \text{RecV}_{\tau_i}$, $A_i \in \text{Terms}_{\tau_i}$, υ_j , $u_i \in \text{PureV}$, $H \in \text{Terms}$, of suitable types:

$$\overrightarrow{p} := \overrightarrow{A} \equiv p_1 := A_1, \ldots, p_n := A_n \quad (n \geq 0) \tag{11a}$$

$$H(\overrightarrow{u}) \equiv [\ldots H(u_1) \ldots] (u_n) \quad (n \geq 0) \tag{11b}$$

$$\begin{aligned} \lambda(\overrightarrow{\upsilon})(H(\overrightarrow{u})) &\equiv \lambda(\upsilon_1)(\ldots \lambda(\upsilon_m)([\ldots H(u_1) \ldots](u_n)) \ldots) \\ &\equiv \lambda(\upsilon_1) \ldots (\upsilon_m)(H(u_1) \ldots (u_n)) \\ &\equiv \lambda(\upsilon_1, \ldots, \upsilon_m)(H(u_1, \ldots, u_n)) \quad (n, m \geq 0) \end{aligned} \tag{11c}$$

3. Denotational Semantics of L^{λ}_{ar} and L^{λ}_{rar}

A *standard semantic structure* is a tuple $\mathfrak{A}$ (Consts) $= \langle \mathbb{T}, \mathcal{I} \rangle$ that satisfies the following conditions:

- $\mathbb{T} = \{\mathbb{T}_\sigma \mid \sigma \in \mathit{Types}\}$ is a *frame*, i.e., a set of sets of typed objects, such that:
  - $\{0, 1, er\} \subseteq \mathbb{T}_t \subseteq \mathbb{T}_e$ $(er_t \equiv er_e \equiv er \equiv error)$
  - $\mathbb{T}_s \neq \varnothing$ (the set of the *states*)
  - $\mathbb{T}_{(\tau_1 \to \tau_2)} = (\mathbb{T}_{\tau_1} \to \mathbb{T}_{\tau_2}) = \{f \mid f : \mathbb{T}_{\tau_1} \to \mathbb{T}_{\tau_2}\}$ (standard frame)
  - Optionally, L^{λ}_{ar} may designate different objects as the erroneous values, which are typed: $er_{(\tau_1 \to \tau_2)} = h$, is a function, such that, for every $c \in \mathbb{T}_{\tau_1}$, $h(c) = er_{\tau_2}$, i.e., designated typed erroneous objects as values
- $\mathcal{I}$: Consts $\to \cup\, \mathbb{T}$ is the *interpretation function*, respecting the types: for every $\text{c} \in \text{Consts}_\sigma$, $\mathcal{I}(\text{c}) \in \mathbb{T}_\sigma$ there is some $c \in \mathbb{T}_\tau$, such that $\mathcal{I}(\text{c}) = c$
- $\mathfrak{A}$ has the set G of all variable valuations G, respecting the types:

$$G = \{g \mid g : \text{PureV} \cup \text{RecV} \to \cup\mathbb{T}, \text{ and for every } x\text{: } \sigma,\ g(x) \in \mathbb{T}_\sigma\} \tag{12}$$

Definition 3 (Denotation Function). There is a unique *denotation function* $\text{den}^{\mathfrak{A}}$:

$$\text{den}^{\mathfrak{A}}\text{: Terms} \to \{f \mid f : G \to \cup\mathbb{T}\} \tag{13}$$

defined by structural induction (i.e., by recursion) on the terms, by (D1)–(D5). For any given, fixed semantic structure $\mathfrak{A}$, we write $\text{den} \equiv \text{den}^{\mathfrak{A}}$.

(D1) (a) $\text{den}(x)(g) = g(x)$, for all $x \in \text{PureV} \cup \text{RecV}$
(b) $\text{den}(\text{c})(g) = \mathcal{I}(\text{c})$. for all $\text{c} \in \text{Consts}$

(D2) $\text{den}(A(B))(g) = \text{den}(A)(g)(\text{den}(B)(g))$

(D3) $\text{den}(\lambda x(B))(g)(t) = \text{den}(B)(g\{x := t\})$, for every $x \in \text{Vars}_\tau$, $t \in \mathbb{T}_\tau$

(D4) $\text{den}(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})(g) =$
$\text{den}(A_0)\,(g\{p_1 := \overline{p}_1, \ldots, p_n := \overline{p}_n\})$
where the values $\overline{p}_i \in \mathbb{T}_{\tau_i}$ are defined by recursion on $\text{rank}(p_i)$:

$$\overline{p}_i = \text{den}(A_i)\,(g\{p_{k_1} := \overline{p}_{k_1}, \ldots, p_{k_m} := \overline{p}_{k_m}\}) \tag{14}$$

given that $p_{k_1}, \ldots, p_{k_m}$ are all of the recursion variables $p_j \in \{p_1, \ldots, p_n\}$, such that $\text{rank}(p_j) < \text{rank}(p_i)$ Informally, $\text{den}(A_1)(g), \ldots, \text{den}(A_n)(g)$ are computed recursively and stored in $p_1, \ldots, p_n$, respectively; the denotation $\text{den}(A_0)(g)$ may depend on values stored in $p_1, \ldots, p_n$.

(D5) The stipulation of the denotation of the restricted terms A of the form (6e) is in two cases, with respect to possible state dependent types of the components of A:

$$A \equiv (A_0^{\sigma_0} \text{ such that } \{C_1^{\tau_1}, \ldots, C_m^{\tau_m}\}) \equiv (A_0^{\sigma_0} \text{ s.t. } \{\vec{C}\}) \tag{15}$$

Case 1: for all $i \in \{1, \ldots, n\}$, $C_i : \mathrm{t}$

For every $g \in G$:

$$\text{den}\left(A_0^{\sigma_0} \text{s.t.} \{\vec{C}\}\right)(g) = \begin{cases} \text{den}(A_0)(g), & \text{if, for all } i \in \{1,\ldots,n\}, \\ & \text{den}(C_i)(g) = 1 \\ er_{\sigma_0} & \text{if, for some } i \in \{1,\ldots,n\}, \\ & \text{den}(C_i)(g) = 0 \text{ or} \\ & \text{den}(C_i)(g) = er \end{cases} \tag{16}$$

Case 2: for some $i \in \{1, \ldots, n\}$, $C_i : \tilde{\mathrm{t}}$ (state dependent proposition)

For every $g \in G$, and every state $s \in \mathbb{T}_s$:

$$\text{den}\,(A_0^{\sigma_0} \text{ s.t. } \{\vec{C}\})\,(g)(s) \tag{17}$$

$$= \begin{cases} \text{den}(A_0)(g)(s), & \text{if den}(C_i)(g) = 1, \text{ for all } i \text{ s.th. } C_i : \mathrm{t}, \text{ and} \\ & \text{den}(C_i)(g)(s) = 1, \text{ for all } i \text{ s.th. } C_i : \tilde{\mathrm{t}}, \text{ and} \\ & \sigma_0 \equiv (\mathrm{s} \to \sigma) \\ \text{den}(A_0)(g), & \text{if den}(C_i)(g) = 1, \text{ for all } i \text{ s.th. } C_i : \mathrm{t}, \text{ and} \\ & \text{den}(C_i)(g)(s) = 1, \text{ for all } i \text{ s.th. } C_i : \tilde{\mathrm{t}}, \text{ and} \\ & \text{there is no } \sigma \in \text{Types, such that } \sigma_0 \equiv (\mathrm{s} \to \sigma) \\ er_{\sigma_0'}, & \text{otherwise, for } \sigma_0' \equiv \begin{cases} \sigma, \text{ such that } \sigma_0 \equiv (\mathrm{s} \to \sigma) \\ \sigma_0, \text{ if there is no } \sigma \in \text{Types, such that} \\ \qquad \sigma_0 \equiv (\mathrm{s} \to \sigma) \end{cases} \end{cases}$$

Informally, the denotation of a term A of the form (6e) is the denotation of A_0, in case all C_i are true, otherwise it is error, i.e., it is error in the following cases:

- the denotation of at least one of the terms A_0, C_i $(i = 1, \ldots, m)$ is error, or
- the denotation of at least one of the terms C_i is false $(i = 1, \ldots, m)$

Immediate terms do not carry algorithmic sense; their denotations are obtained by the variable valuations. For details on the immediate terms, see (Moschovakis, 2006) and (Loukanova, 2019b). Here, we provide their definition because they play an essential role in the computational notions of L^{λ}_{ar} and L^{λ}_{rar}, and for self-containment of the paper.

Definition 4 (Immediate and Proper Terms). The set of the immediate terms is defined recursively, e.g., by the style of typed BNF notation:

$$\mathrm{ImT}^{\tau} :\equiv X^{\tau} \mid \underbrace{Y^{(\tau_1 \to \cdots \to (\tau_m \to \tau))}\left(v_1^{\tau_1}\right)\cdots\left(v_m^{\tau_m}\right)}_{(\text{immediate applicative terms})} \qquad \text{(ImAp)} \qquad (18a)$$

$$\mathrm{Im}\,\mathrm{T}^{(\sigma_1 \to \cdots \to (\sigma_n \to \tau))} :\equiv \underbrace{\lambda\left(u_1^{\sigma_1}\right)\cdots\lambda\left(u_n^{\sigma_n}\right)\left[Y^{(\tau_1 \to \cdots \to (\tau_m \to \tau))}\left(v_1^{\tau_1}\right)\cdots\left(v_m^{\tau_m}\right)\right]}_{(\text{immediate } \lambda\text{-terms})} \qquad \text{(Im}\lambda\text{)} \qquad (18b)$$

for $n \geq 1$, $m \geq 0$; $u_i, \upsilon_j \in$ PureV, for $i = 1, \ldots, n$, $j = 1, \ldots, m$; $X \in$ PureV, $Y \in$ RecV

A term A is *proper* if it is not immediate, i.e., the set PrT of the proper terms of L^{λ}_{rar} consists of all terms that are not in ImT:

$$\mathrm{PrT} = (\mathrm{Terms} - \mathrm{ImT}) \qquad (19)$$

4. Canonical Forms

For every term $A \in$ Terms, there is a term cf(A), a *canonical form* of A, defined by structural recursion on the term A, according to Definition 5. We extend the corresponding definition given in (Moschovakis, 2006), by adding a clause for cf(A), for terms A of the form (6e).

Definition 5 (Canonical Forms of Terms). In each of the following clause, we assume that the bound recursion variables are distinct for the distinct terms, and from the free recursion variables, because the bound variables can be renamed, by the congruence relation, see Def. 6.

(CF1) For every $c \in \mathrm{Consts}_{\tau}$, $\mathrm{cf}(c) :\equiv c \equiv c$ where $\{\}$
For every $x \in$ PureV $\cup$ RecV, $\mathrm{cf}(x) :\equiv x \equiv x$ where $\{\}$

(CF2) Assume that $A \in$ Terms, and

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\}\ (n \geq 0) \qquad (20a)$$

$$\equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\} \qquad (20b)$$

(CF2a) If X is an immediate term, then

$$\mathrm{cf}(A(X)) :\equiv A_0(X) \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\} \qquad (21)$$

(CF2b) If B is a proper term and

$$\mathrm{cf}(B) \equiv B_0 \text{ where } \{q_1 := B_1, \ldots, q_m := B_m\}\ (m \geq 0) \tag{22a}$$

$$\equiv B_0 \text{ where } \{\overrightarrow{q} := \overrightarrow{B}\} \tag{22b}$$

then

$$\mathrm{cf}(A(B)) :\equiv A_0(q_0) \text{ where } \{\overrightarrow{p} := \overrightarrow{A}, q_0 := B_0, \overrightarrow{q} := \overrightarrow{B}\} \tag{23}$$

where $q_0 \in \mathrm{RecV}$ is a fresh recursion variable

(CF3) Assume that $A \in \mathrm{Terms}$, and

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\} \tag{24}$$

Then, for every $u \in \mathrm{PureV}_\tau$, and fresh $p_1', \ldots, p_n' \in \mathrm{RecV}$:

$$\mathrm{cf}(\lambda(u)A) :\equiv \lambda(u)A_0' \text{ where } \{p_1' := \lambda(u)A_1', \ldots, p_n' := \lambda(u)A_n'\} \tag{25}$$

where A_i' is the result of the simultaneous replacement of all the free occurrences of $p_1, \ldots, p_n$ in A_i with $p_1'(u), \ldots, p_n'(u)$, respectively

(CF4) Assume that $A \in \mathrm{Terms}$ is such that

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\} \tag{26}$$

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

$$\mathrm{cf}(A_i) \equiv A_{i,0} \text{ where } \{p_{i,1} := A_{i,1}, \ldots, p_{i,k_i} := A_{i,k_i}\} \tag{27a}$$

$$\equiv A_{i,0} \text{ where } \{\overrightarrow{p_i} := \overrightarrow{A_i}\} \tag{27b}$$

Then, for fresh $p_1, \ldots, p_n \in \mathrm{RecV}$:

$$\mathrm{cf}(A) :\equiv A_{0,0} \text{ where } \{\overrightarrow{p_0} := \overrightarrow{A_0}, \tag{28a}$$

$$p_1 := A_{1,0}, \overrightarrow{p_1} := \overrightarrow{A_1}, \tag{28b}$$

$$\vdots$$

$$p_n := A_{n,0}, \overrightarrow{p_n} := \overrightarrow{A_n}\} \tag{28c}$$

(CF5) Assume that $j = 1, \ldots, m$ $(m \geq 0)$, $i = 1, \ldots, k$ $(k \geq 0)$, and:

(1) $A, C_1^{\tau_1}, \ldots, C_m^{\tau_m}, R_1^{\sigma_1}, \ldots, R_k^{\sigma_k} \equiv \overrightarrow{R^\sigma} \in \mathrm{Terms}$
(2) for $j = 1, \ldots, m$ $(m \geq 0)$, $C_j^{\tau_j}$, and for $i = 1, \ldots, k$ $(k \geq 0)$, $\overrightarrow{R^\sigma}$ have types of truth values, i.e., $\tau_j \equiv \mathrm{t}$ or $\tau_j \equiv \tilde{\mathrm{t}}$; and $\sigma_i \equiv \mathrm{t}$ or $\sigma_i \equiv \tilde{\mathrm{t}}$
(3) C_j are proper

(4) $\overrightarrow{R^{\sigma}}$ are immediate
(5) $c_j \in \mathrm{RecV}_{\tau_j}$ are fresh with respect to $A, \overrightarrow{C^{\tau}}, \overrightarrow{R^{\sigma}}$
(6) the canonical forms of A, $C_1^{\tau_1}, \ldots, C_m^{\tau_m} \in$ Terms are as in (29a)–(29b) (without writing the type superscripts, which are not per se part of the terms):

$$\begin{aligned} \mathrm{cf}(A) &\equiv_{\mathrm{C}} A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \quad (n \geq 0) \\ &\equiv A_0 \text{ where } \{\overrightarrow{p} := \overrightarrow{A}\} \end{aligned} \tag{29a}$$

$$\begin{aligned} \mathrm{cf}(C_j) &\equiv_{\mathrm{C}} C_{j,0} \text{ where } \{c_{j,1} := C_{j,1}, \ldots, c_{j,kj} := C_{j,kj}\} \\ &\equiv C_{j,0} \text{ where } \{\overrightarrow{c_j} := \overrightarrow{C_j}\}, \quad j = 1, \ldots, m \ (m \geq 0) \end{aligned} \tag{29b}$$

given that

(a) no $\overrightarrow{c_{j'}}$ occurs freely in any $\overrightarrow{C_{j''}}$, for $j' \neq j''$, and in any A_i $(j', j'' = 0, \ldots, m, i = 0, \ldots, n)$, which is guaranteed by the congruence
(b) no c_j occurs freely in any C_l $(l = 0, \ldots, m)$, no c_j occurs freely in any A_i $(i = 0, \ldots, n)$, because c_j are fresh, and guaranteed by the congruence

(CF5a) If A_0 is an immediate term, then:

$$\mathrm{cf}(A \text{ such that } \{C_1, \ldots, C_m, \overrightarrow{R}\}) \tag{30a}$$

$$\equiv (A_0 \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \tag{30b}$$

$$\begin{aligned} &\text{where } \{\overrightarrow{p} := \overrightarrow{A}, \\ &\quad c_1 := C_{1,0}, \overrightarrow{c_1} := \overrightarrow{C_1}, \\ &\quad \vdots \\ &\quad c_m := C_{m,0}, \overrightarrow{c_m} := \overrightarrow{C_m}\} \end{aligned} \tag{30c}$$

(CF5b) If A_0 is a proper term, then, for fresh $p_0, c_j \in \mathrm{RecV}$, $j = 1, \ldots, m$:

$$\mathrm{cf}(A \text{ such that } \{C_1, \ldots, C_m, \overrightarrow{R}\}) \tag{31a}$$

$$\equiv (p_0 \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \tag{31b}$$

$$\begin{aligned} &\text{where } \{p_0 := A_0, \overrightarrow{p} := \overrightarrow{A}, \\ &\quad c_1 := C_{1,0}, \overrightarrow{c_1} := \overrightarrow{C_1}, \\ &\quad \vdots \\ &\quad c_m := C_{m,0}, \overrightarrow{c_m} := \overrightarrow{C_m}\} \end{aligned} \tag{31c}$$

5. Reduction Calculus of L_{ar}^{λ} and L_{rar}^{λ}

The formal language L_{ar}^{λ} has reduction rules that reduce its terms to their unique, up to congruence, canonical forms. Here, after we provide the original set of reduction rules, given in (Moschovakis, 2006), we extend it by adding two *Restriction Reduction Rules*, (st1)–(st2), for reducing terms having occurrences of the restrictor operator.

5.1 Congruence Relation

Definition 6 (Congruence). The *congruence* relation, denoted by $\equiv_c$, is the smallest relation between L_{ar}^{λ}-terms ($A \equiv_c B$), i.e., $\equiv_c \subseteq \text{Terms} \times \text{Terms}$, that is closed under:

(1) reflexivity, symmetricity, transitivity
(2) the term formation rules of L_{ar}^{λ}
    - constants, variables
    - application
    - λ-abstraction
    - acyclic recursion
    - restriction term
(3) renaming bound, pure and recursion, variables without causing variable collisions
(4) re-ordering of the assignments within the acyclic sequences of assignments of the recursion terms
(5) re-ordering of the restriction sub-terms in the restriction terms

5.2 Reduction Rules of L_{rar}^{λ}

5.2.1 Original Reduction Rules of the Theory of Acyclic Recursion

The set of the L_{ar}^{λ}-reduction rules is as follows:

Congruence

$$\text{If } A \equiv_c B, \text{ then } A \Rightarrow B \qquad \text{(cong)}$$

Transitivity

$$\text{If } A \Rightarrow B \text{ and } B \Rightarrow C, \text{ then } A \Rightarrow C \qquad \text{(trans)}$$

Compositionality

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

$$\text{If } A \Rightarrow B, \text{ then } \lambda(u)\,(A) \Rightarrow \lambda(u)\,(B) \qquad (\lambda\text{-comp})$$

$$\text{If } A_i \Rightarrow B_i, \text{ for } i = 0, \ldots, n, \text{ then } A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \qquad \text{(wh-comp)}$$

$$\Rightarrow B_0 \text{ where } \{p_1 := B_1, \ldots, p_n := B_n\}$$

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

$$A_0 \text{ such that } \{C_1, \ldots, C_n\} \qquad \text{(st-comp)}$$

$$\Rightarrow B_0 \text{ such that } \{R_1, \ldots, R_n\}$$

The Head Rule

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

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

The Bekič-Scott Rule

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

given that no q_j occurs freely in any A_i, for $i = 0, \ldots, n, j = 1, \ldots, m$

The Recursion-Application Rule

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

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

The Application Rule

$$A(B) \Rightarrow A(p) \text{ where } \{p := B\} \qquad \text{(ap)}$$

given that B is proper and p is a fresh (recursion) memory variable

The λ-Rule

$$\lambda(u)\,(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})$$
$$\Rightarrow \lambda(u)\,(A_0' \text{ where } \{p_1' := \lambda(u)\,A_1', \ldots, p_n' := \lambda(u)\,A_n'\}$$
$$\equiv \lambda(u)\,A_0\left\{\overrightarrow{p} := \overrightarrow{p'(u)}\right\} \text{ where } \left\{\overrightarrow{p'} := \overrightarrow{\lambda(u)\,A\left\{\overrightarrow{p} := \overrightarrow{p'(u)}\right\}}\right\} \qquad (\lambda)$$

where:

(1) $u \in \text{PureV}_\sigma$
(2) for all $i = 1, \ldots, n$, $p_i, \in \text{RecV}_{\sigma_i}$,
$p_i' \in \text{RecV}_{(\sigma \to \sigma_i)}$ is a fresh recursion variable

(3) for all $i = 0, \ldots, n, A_i \in \text{Terms}_{\sigma_i}$,
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.:

$$A'_i :\equiv A_i\{p_1 :\equiv p'_1(u), \ldots, p_n :\equiv p'_n(u)\} \tag{33a}$$

$$A'_i :\equiv A_i\{\overrightarrow{p} :\equiv \overrightarrow{p'(u)}\} \tag{33b}$$

5.2.2 Extending the Original Reduction by Rules for Restriction Terms

Restriction / Restrictor Rules of $\mathbf{L}^{\lambda}_{rar}$ (st1) / (st2)

Given that, for $j = 1, \ldots, m$ $(m \geq 0)$, $i = 1, \ldots, k$ $(k \geq 0)$:

(1) Each $C^{\tau_j}_j \in$ Terms is proper
(2) Each $R^{\sigma}_i \in$ Terms in $\overrightarrow{R^{\sigma}}$ is immediate
(3) $C^{\tau_j}_j$ and R^{σ}_i have types of truth values, i.e., $\tau_j \equiv \mathrm{t}$ or $\tau_j \equiv \tilde{\mathrm{t}}$; and $\sigma_i \equiv \mathrm{t}$ or $\sigma_i \equiv \tilde{\mathrm{t}}$ there are two reduction rules for the restrictor (restricted) terms:

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

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

for fresh $c_j \in$ RecV $(j = 1, \ldots, m)$

(st2) A_0 is a proper term

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

for fresh $a_0, c_j \in$ RecV $(j = 1, \ldots, m)$

Informally, the restriction parts in (st1) / (st2) are proposition terms.

Definition 7 (Irreducible Terms). We say that a term $A \in$ Terms is *irreducible* in $\mathbf{L}^{\lambda}_{rar}$ iff

$$\text{for all } B \in \text{Terms}, \quad A \Rightarrow B \quad \Longrightarrow \quad A \equiv_c B \tag{36}$$

Theorem 1 (Criteria for Irreducibility). *With respect to the extended set of reduction rules:*

(1) Every $A \in$ Consts $\cup$ Vars is irreducible
(2) $A(B)$ is irreducible iff B is immediate and A is explicit and irreducible

(3) $\lambda(x)(A)$ *is irreducible iff* A *is explicit and irreducible*
(4) $[A_0$ where $\{\overrightarrow{p} := \overrightarrow{A}\}]$ *is irreducible iff all parts* A_i $(i = 0, \ldots, n)$ *are explicit and irreducible*
(5) $(A_0$ such that $\{\overrightarrow{C}\})$ *is irreducible iff all parts* A_0, C_i $(i = 0, \ldots, n)$ *are immediate*

Proof. By structural induction on terms and checking the reduction rules. □

6. Algorithmically Restricted Memory Parameters

Theorem 2 (Basic, Restricted Memory Variables). *Assume that, for any* $n \geq 1$*:*

- $\overrightarrow{R_j}$ *are immediate terms* $(j = 1, \ldots, n)$*, and*
- $p_i \in$ RecV $(i = 2, \ldots, n)$ *are fresh with respect to* $p_1, \overrightarrow{R_j}$

Then:

$$((\ldots((p_1 \text{ s.t. } \overrightarrow{R_1}) \text{ s.t. } \overrightarrow{R_2}) \ldots) \text{ s.t. } \overrightarrow{R_n}) \tag{37a}$$

$$\Rightarrow (p_n \text{ s.t. } \overrightarrow{R_n}) \text{ where } \{p_n := (p_{n-1} \text{ s.t. } \overrightarrow{R_{n-1}}), \tag{37b}$$

$$\ldots,$$

$$p_2 := (p_1 \text{ s.t. } \overrightarrow{R_1})\} \tag{37c}$$

Proof. The proof is by induction on $n \geq 1$.

Induction Basis: $n = 1$

$(p_1 \text{ s.t. } \overrightarrow{R_1}) \Rightarrow (p_1 \text{ s.t. } \overrightarrow{R_1})$ is trivially true.

Induction Hypothesis: Assume (37a)–(37c), for an arbitrary $n \geq 1$.

Induction Step: We reduce the term (38a) to the canonical form (38g)–(38i), by applying the reduction rules (compositionally), as follows:

$$(\underbrace{((\ldots(p_1 \text{ s.t. } \overrightarrow{R_1})\ldots) \text{s.t. } \overrightarrow{R_n})}_{p_{n+1}} \text{s.t. } \overrightarrow{R_{n+1}}) \tag{38a}$$

by (st2):

(38b)

$$\Rightarrow (p_{n+1} \text{ s.t. } \overrightarrow{R_{n+1}}) \text{where} \tag{38c}$$

$$\underbrace{\{p_{n+1} := ((\ldots(p_1 \text{ s.t. } \overrightarrow{R_1})\ldots) \text{ s.t. } \overrightarrow{R_n})\}}$$

by ind.hyp.; (wh-comp):

$$\Rightarrow (p_{n+1} \text{ s.t. } \overrightarrow{R_{n+1}}) \text{ where } \{p_{n+1} := [(p_n \text{ s.t. } \overrightarrow{R_n}) \text{ where } \{ \tag{38d}$$

$$p_n := (p_{n-1} \text{ s.t. } \overrightarrow{R_{n-1}}), \tag{38e}$$

$$\ldots,$$

$$p_2 := (p_1 \text{ s.t. } \overrightarrow{R_1})\}] \tag{38f}$$

by (B-S):

$$\Rightarrow (p_{n+1} \text{ s.t. } \overrightarrow{R_{n+1}}) \text{ where } \{p_{n+1} := (p_n \text{ s.t. } \overrightarrow{R_n}), \tag{38g}$$

$$p_n := (p_{n-1} \text{ s.t. } \overrightarrow{R_{n-1}}), \tag{38h}$$

$$\ldots,$$

$$p_2 := (p_1 \text{ s.t. } \overrightarrow{R_1})\} \tag{38i}$$

Therefore, (37a)–(37c) hold, for every $n \geq 1$.

Theorem 3 (Restricted Memory Variables). *Assume that, for* $n \geq 1$, $j = 1, \ldots, n$, *and* $i = 2, \ldots, n$:

- $\overrightarrow{C_j}$ *are proper terms, and* $\overrightarrow{R_j}$ *are immediate*
- $p_i \in \text{RecV}$ *and* $c_j \in \text{RecV}$ *are fresh with respect to* $p_1, \overrightarrow{C_j}, \overrightarrow{R_j}$

Then:

$$V \equiv ((\ldots((p_1 \text{ s.t. } \{\overrightarrow{C_1}, \overrightarrow{R_1}\}) \text{ s.t. } \{\overrightarrow{C_2}, \overrightarrow{R_2}\}) \ldots) \text{ s.t. } \{\overrightarrow{C_n}, \overrightarrow{R_n}\}) \tag{39a}$$

$$\Rightarrow (p_n \text{ s.t. } \{\overrightarrow{c_n}, \overrightarrow{R_n}\}) \text{ where } \{p_n := (p_{n-1} \text{ s.t. } \{\overrightarrow{c_{n-1}}, \overrightarrow{R_{n-1}}\}), \tag{39b}$$

$$\ldots,$$

$$p_2 := (p_1 \text{ s.t. } \{\overrightarrow{c_1}, \overrightarrow{R_1}\}), \tag{39c}$$

$$\overrightarrow{c_1} := \overrightarrow{C_1}, \ldots, \overrightarrow{c_n} := \overrightarrow{C_n}\} \tag{39d}$$

The term (39b)–(39d) *is in canonical form, i.e., it is the canonical form* cf(*V*) *of the term V in* (39a), *iff the terms in* $\overrightarrow{C_i}$ *are explicit (i.e., without occurrences of the constant* where*) and irreducible, for all* $i = 1, \ldots, n$.

Proof. by induction on $n \geq 1$ and using the reduction rules.

This theorem and it's proof are generalization of the Theorem 2. The proof is long, we do not include it here.

We call the terms of the form (39a) and (39b)–(39d), and also (37a), (37b)–(37c), *restricted memory variables*. Optionally, we call the terms (37b)–(37c) and (39b)–(39d) *memory networks of restricted memory locations* $p_i \in \text{RecV}$, for all $i = 1, \ldots, n$.

Note 1. It is essential that Theorem 2 and Theorem 3 provide *generalised, restricted memory variables / locations / slots*: the memory variables $p_i \in$ RecV, for all $i = 1, \ldots, n$. Each one of them is recursively linked to the others, by the corresponding recursion term.

(M1) The memory (recursion) variable $p_1 \in$ RecV is free in both of the terms (37a), (37b)–(37c), and restricted by $\overrightarrow{R_1}$.
(M2) For all $i = 2, \ldots, n$, the memory variables $p_i \in$ RecV are both restricted by $\overrightarrow{C_i}, \overrightarrow{R_i}$ and bound by the recursion operator where, in the recursion term (37b)–(37c).
(M3) The memory, i.e., recursion, variable $p_1 \in$ RecV is free in both of the terms (39a) and (39b)–(39d), while it is restricted by $\overrightarrow{C_1}, \overrightarrow{R_1}$.
(M4) For all $i = 2, \ldots, n$, the memory variables $p_i \in$ RecV are both restricted by $\overrightarrow{C_i}, \overrightarrow{R_i}$ and bound by the recursion operator where, in the recursion term (39b)–(39d).

In the special case of $\overrightarrow{C_i} = \emptyset$, for all $i = 1, \ldots, n$, the memory variables $p_i \in$ RecV ($i = 1, \ldots, n$) are called *basic, restricted memory variables*, or *basic, restricted memory (variables)*. Restricted parameters were introduced, as generalised parameters in situated structures of relational, partial information, in (Loukanova, 2002; Loukanova, 2014; Loukanova, 2017).

Definition 8 (Reduction Relation, $\Rightarrow$). The reduction rules of $\mathrm{L}^{\lambda}_{rar}$, which we often designate as $\mathrm{L}^{\lambda}_{ar}$, define a reduction relation between terms, $A \Rightarrow^{*} B$:

For every $A, B \in \mathrm{L}^{\lambda}_{rar}$

$$A \Rightarrow^{*} B \iff A \equiv A_0 \Rightarrow \cdots \Rightarrow A_n \equiv B, \text{ for some } A_i \in \mathrm{L}^{\lambda}_{rar},\ i = 0, \ldots, n\ (n \geq 0) \tag{40a}$$

by application of reduction rules of $\mathrm{L}^{\lambda}_{rar}$, given in Sect. 5.2

$$A \Rightarrow^{*} B \equiv A \Rightarrow B \text{ by abbreviation} \tag{40b}$$

Typically, we shall write $A \Rightarrow B$, instead of $A \Rightarrow^{*} B$, i.e., as in (40b), when there is no confusion.

Often, we write $\mathrm{L}^{\lambda}_{ar}$ instead of $\mathrm{L}^{\lambda}_{rar}$

Note 2. The Canonical Form Theorem 4 encompasses the extended language and reduction calculus of $\mathrm{L}^{\lambda}_{rar}$, i.e., it covers terms that have occurrences of the constant of the restriction operator such that. Such terms are formed by Def. 2, blending (6e) with all the other formation rules (6a)–(6e).

Theorem 4 (Canonical Form Theorem). *For every* $A \in$ Terms*, there is a unique up to congruence, irreducible term* B*, such that* (41)*:*

$$B \equiv \mathrm{cf}\,(A) \in \text{Terms} \tag{41}$$

The term cf (A) *is called the* canonical form of A*, It has the properties (1)–(5):*

(1) For some explicit, irreducible $A_0, \ldots, A_n \in$ Terms *($n \geq 0$):*

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \tag{42}$$

(2) *A and its canonical form* cf(*A*) *have the same constants,* (43a)*, and the same free (pure and memory) variables,* (43b)*:*

$$\text{Consts}(A) = \text{Consts}(\text{cf}(A)) \tag{43a}$$
$$\text{FreeV}(A) = \text{FreeV}(\text{cf}(A)) \tag{43b}$$

(3) *Cannonical irreducibility, modulo congruence:*

$$A \text{ is irreducible} \quad \Longrightarrow \quad A \equiv_{\text{c}} \text{cf}(A) \tag{44}$$

(4) *Uniqueness of the canonical forms, in reduction sequences, up to congruence:*

$$A \Rightarrow B \quad \Longrightarrow \quad \text{cf}(A) \equiv_{\text{c}} \text{cf}(B) \tag{45}$$

(5) cf (*A*) *is the unique, up to congruence,* irreducible *term, i.e., for every B,*

$$A \Rightarrow B \text{ and } B \text{ is irreducible} \quad \Longrightarrow \quad \text{cf}(A) \equiv_{\text{c}} \text{cf}(B) \tag{46}$$

Proof. The proof is by structural induction on formation of the term *A*, according to Def. 2, and using Def. 5 of the canonical forms cf(*A*), the reduction rules in Sects. 5.2.1–5.2.2, and by the Criteria for Irreducibility, Theorem 1.

The full proof is long and we do not include it.

For example, the proof of item (1), for the case in the indiction step, in which *A* is a restriction term, i.e., of the form (6e), in Def. 2, uses Def. 5 of the canonical forms cf(*A*), (CF5) (30a)–(30c). □

Corollary 4.1 (Canonical Form of Restricted Terms). *Assume that* $D \in$ Terms *is a restriction term of the form*

$$D \equiv (A \text{ such that } \{\overrightarrow{C}, \overrightarrow{R}\}) \tag{47}$$

given that, for $j = 1, \ldots, m$ $(m \geq 0)$, $i = 1, \ldots, k$ $(k \geq 0)$*:*

(1) *Each* $C_j^{\tau_j} \in$ Terms *is proper*
(2) *Each* $R_i^{\sigma} \in$ Terms *in* $\overrightarrow{R^{\sigma}}$ *is immediate*
(3) $C_j^{\tau_j}$ *and* R_i^{σ} *have types of truth values, i.e.,* $\tau_j \equiv \text{t}$ *or* $\tau_j \equiv \tilde{\text{t}}$*; and* $\sigma_i \equiv \text{t}$ *or* $\sigma_i \equiv \tilde{\text{t}}$

Then (48) *follows:*

$$\text{cf}(D) \equiv (\text{A}' \text{ such that } \{\overrightarrow{c}, \overrightarrow{R}\}) \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \tag{48}$$

for some immediate $A' \in$ Terms*, some explicit, irreducible* $A_1, \ldots, A_n \in$ Terms $(n \geq 0)$*, and memory variables* c_j, $p_i \in$ RecV $(j = 1, \ldots, m,\ m \geq 0,\ i = 1, \ldots, n)$*, such that* $\overrightarrow{c} \subseteq \overrightarrow{p}$ *(i.e., for all j,* $c_j \in \{p_1, \ldots, p_n\}$*).*

Theorem 5 (Effective Reduction Calculus). *For every term* $A \in$ Terms*, its canonical form* cf(*A*) *is effectively computed, by using the reduction calculus. That is, A is effectively reduced to its canonical form* cf(*A*)*, by a finite number of applications of the reduction rules:*

$$A \Rightarrow_{\text{cf}} \text{cf}(A) \tag{49}$$

Proof. The proof is by structural induction on the term A, by Def. 2, and using Def. 5 of the cf(A), the reduction rules in Sects. 5.2.1–5.2.2, and by the Criteria for Irreducibility, Theorem 1.

The full proof is long. We shall prove the case in the indiction step, in which A is a restriction term, i.e., of the form (6e), in Def. 2

Induction Hypothesis Assume that $D \in$ Terms is a restriction term of the form (50a)–(50c):

$$D \equiv (A \text{ such that } \{\overrightarrow{D^{\vartheta}}\}) \tag{50a}$$

$$A \Rightarrow \text{cf}(A), \qquad D_i \Rightarrow \text{cf}(D_i) \qquad \text{(Induction Hypothesis)} \tag{50b}$$

$$\text{for } \overrightarrow{D^{\vartheta}} \text{ of types of truth values, i.e., } \vartheta_i \equiv \text{t or } \vartheta_i \equiv \tilde{\text{t}},\ i = 1, \ldots, k\ (k \geq 0) \tag{50c}$$

Induction Step We shall prove that $D \Rightarrow_{\text{cf}} \text{cf}(D)$.

By the congruence, Def. 6, the term parts in the scope of the operator such that can be reordered as in (51):

$$D \equiv_{\text{c}} (A \text{ such that } \{\overrightarrow{C}, \overrightarrow{R}\}) \tag{51}$$

with the term parts as in Corollary 4.1

Case 1 $A \in$ Terms is immediate

Then, by applying the reduction rule (st1) to D in (51), and taking $A' \equiv A$, we get, (52a), for fresh recursion (memory) variable $c_j \in$ RecV ($j = 1, \ldots, m$):

$$\begin{aligned} \text{D} &\Rightarrow (A \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\ &\quad \text{where } \{c_1 := C_1, \ldots, c_m := C_m\} \qquad (52a) \\ &\qquad \text{from } D \text{ in (51), by (st1), for } A' \equiv A \text{ immediate} \\ &\Rightarrow (A \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\ &\quad \text{where } \{c_1 := \text{cf}(C_1), \ldots, c_m := \text{cf}(C_m)\} \qquad (52b) \\ &\qquad \text{from (52a), by Induction Hypothesis (50b) and (wh-comp)} \\ &\Rightarrow (A \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\ &\quad \text{where } \{c_1 := C_{1,0}, \overrightarrow{c_1} := \overrightarrow{C_1}, \\ &\qquad\qquad \vdots \\ &\qquad\qquad c_m := C_{m,0}, \overrightarrow{c_m} := \overrightarrow{C_m}\} \qquad (52c) \\ &\qquad \text{from (52b), by the Bekič-Scott Rule (B-S)} \\ &\equiv \text{cf}(D) \quad \text{from (52a) and (50b), by Def. 5 of the canonical forms for } D, \text{cf}(D) \qquad (52d) \end{aligned}$$

Case 2 $A \in$ Terms is proper

Then, by applying the reduction rule (st2) to A in (47), we have:

$$\begin{aligned}
&\mathrm{D} \Rightarrow (A \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\
&\qquad \text{where } \{c_1 := C_1, \ldots, c_m := C_m\} && (53a)\\
&\Rightarrow (p_0 \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\
&\qquad \text{where } \{(p_0 := \mathrm{cf}(A), c_1 := \mathrm{cf}(C_1), \ldots, c_m := \mathrm{cf}(C_m)\} && (53b)\\
&\qquad\qquad \text{from (53a), by Induction Hypothesis (50b) and (wh-comp)} \\
&\Rightarrow (p_0 \text{ such that } \{c_1, \ldots, c_m, \overrightarrow{R}\}) \\
&\qquad\qquad \text{where } \{p_0 := A_{0,0}, \overrightarrow{p_0} := \overrightarrow{A_0}, \\
&\qquad\qquad\qquad c_1 := C_{1,0}, \overrightarrow{c_1} := \overrightarrow{C_1}, \\
&\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad c_m := C_{m,0}, \overrightarrow{c_m} := \overrightarrow{C_m}\} && (53c)\\
&\qquad\qquad\qquad \text{from (53b), by the Bekič-Scott Rule (B-S)} \\
&\equiv \mathrm{cf}(D) \quad \text{from (53a) and (50b), by Def. 5 of the canonical forms for } D, \mathrm{cf}(D) && (53d)
\end{aligned}$$

□

7. Algorithmic Syntax-Semantics in $\mathrm{L}^{\lambda}_{ar}$ and Memory Restrictions

In this section, we emphasise the significance of the algorithmic semantics in the type-theory of acyclic recursion $\mathrm{L}^{\lambda}_{rar}$, with respect to its denotational semantics.

Immediate Terms. In case A is an immediate term of $\mathrm{L}^{\lambda}_{ar}$, and thus of $\mathrm{L}^{\lambda}_{rar}$, see Def. 4, e.g., of the form $\lambda(u_1) \ldots \lambda(u_n)\,(p(\upsilon_1) \ldots (\upsilon_m))$, as in (18b), it has no algorithmic sense. The value $\mathrm{den}(A)(g)$ is given by the valuations $g(p)$ and $g(\upsilon_i)$, and abstracting away from the values u_j.

Proper Terms. In case A is a proper term, i.e., non-immediate, there is an algorithm $\mathrm{alg}(A)$ for computing $\mathrm{den}(A)(g)$. The algorithm $\mathrm{alg}(A)$ is determined by the canonical form $\mathrm{cf}(A)$ of A, (54).

By the Canonical Form Theorem 4 and (42), the canonical form $\mathrm{cf}(A)$ is unique up to congruence:

$$\begin{aligned}
&\mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \\
&\qquad \text{for some explicit, irreducible } A_i \in \text{ Terms}, i = 1, \ldots, n\ (n \geq 0) && (54)
\end{aligned}$$

The canonical form $\mathrm{cf}(A)$ of a proper term A determines the algorithm for computing the denotational value $\mathrm{den}(A)(g) = \mathrm{den}(\mathrm{cf}(A))(g)$ from the components $\mathrm{den}(A_i)(g)$ of the canonical form $\mathrm{cf}(A)$, by Def. 3.

For every algorithmically meaningful, i.e., proper (non-immediate) term A, $A \in$ Terms, its canonical form $\mathrm{cf}(A)$ determines the algorithm $\mathrm{alg}(A)$ for computing $\mathrm{den}(A)$.

- The denotational semantics of $\mathrm{L}^{\lambda}_{rar}$ ($\mathrm{L}^{\lambda}_{ar}$) is by induction on term structure
- The type theories $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$ have *effective reduction calculi*, see Theorem 5 for $\mathrm{L}^{\lambda}_{rar}$. For $\mathrm{L}^{\lambda}_{ar}$, see (Moschovakis, 2006) and (Loukanova, 2019b; Loukanova, 2019c; Loukanova, 2020b). For

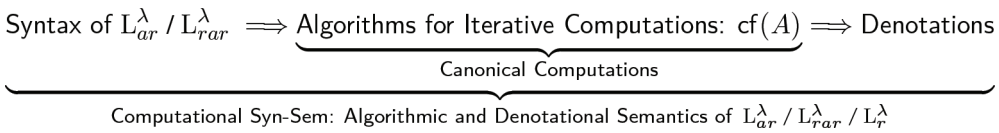

Figure 2: Computational Syntax-Semantics Interface for Parametric Computations in $\mathrm{L}^{\lambda}_{rr}$

every $A \in$ Terms, there is a unique, up to congruence, canonical form cf(A), which can be obtained from A, by a finite number of reductions:

$$A \Rightarrow_{\mathrm{cf}} \mathrm{cf}(A) \tag{55}$$

- For a given, fixed semantic structure $\mathfrak{A}$ and valuations G, for every *algorithmically meaningful* $A \in \mathrm{Terms}_{\sigma}$, the algorithm alg($A$) for computing den($A$) is determined by cf($A$), so that:

$$\mathrm{den}(A)(g) = \mathrm{den}(\mathrm{cf}(A))(g), \text{ for } g \in G \tag{56a}$$
$$\mathrm{alg}(A) = \mathrm{alg}(\mathrm{cf}(A)) \colon G \to \mathbb{T}_{\sigma} \tag{56b}$$

The denotation $\mathrm{den}(A)(g) = \mathrm{den}(\mathrm{cf}(A))(g)$ of a proper term A is computed stepwise, iteratively, according to the ranking $\mathrm{rank}(p_i)$, $i = 1, \ldots, n$ ($n \geq 0$). By starting with i of the lowest value among $\mathrm{rank}(p_1) \ldots, \mathrm{rank}(p_n)$, and moving in increasing order, the algorithm computes the denotational values of the parts $\mathrm{den}(A_i)(g)$ and saves them in the corresponding memory variables p_i. Then, these values are used for computing the denotation:

$$\mathrm{den}(A)(g) = \mathrm{den}(\mathrm{cf}(A))(g) = \mathrm{den}(A_0)(g) \tag{57}$$

Figure 2 gives the general scheme of the relation between the syntax of the terms of $\mathrm{L}^{\lambda}_{rar}$ ($\mathrm{L}^{\lambda}_{ar}$, L^{λ}_{r}) and their algorithmic and denotational semantics, via *computational syntax-semantics interface* within $\mathrm{L}^{\lambda}_{rar}$ ($\mathrm{L}^{\lambda}_{ar}$, L^{λ}_{r}).

7.1 Formalization of the Notion of Iterative Algorithm

The type-theory of acyclic recursion is a formalization of the mathematical notion of algorithm, for computing values of recursive functions designated by *recursion terms*. The values of the functions, when denoted by meaningful, i.e., proper, recursive terms, are computed iteratively by algorithms determined by *the canonical forms of the corresponding terms*. In this paper, we have introduced $\mathrm{L}^{\lambda}_{rar}$, which is an enrichment of $\mathrm{L}^{\lambda}_{ar}$, by restricted memory variables (to save parametric data). I.e., the concept of algorithm, with memory parameters, which are constrained to store data, which in addition to being typed, can be restricted to satisfy properties. The algorithms are expressed by terms in canonical forms, which carry data components computed and stored in restricted memory variables, i.e., memory slots. The restrictions of the memory variables, are expressed algorithmically by the canonical forms as well.

Thus, the concept of algorithm, with restricted memory parameters, is formalised at the object level of the syntax and reduction calculus of $\mathrm{L}^{\lambda}_{rar}$.

For a term $A \in \mathrm{L}^{\lambda}_{rar}$, such that A has occurrences of the restrictor operator, the canonical form cf(A) has occurrences of the restrictor only in sub-terms that represent restricted immediate terms of the form (P such that $\{C_1, \ldots, C_m\}$), for immediate terms $P, C_1, \ldots, C_m$.

8. Algorithmic Syntax-Semantics and Constants

For each algorithmically meaningful term A, the denotation of the term $\mathrm{den}(A) = \mathrm{den}(\mathrm{cf}(A))$ is computed by stepwise, iterative computations, according to the algorithm determined by its canonical form cf (A), by the increasing rank of the term parts A_i. In the computational process, the values of the parts A_i of cf (A) are computed in the iterative steps and stored in corresponding memory slots p_i.

In this section, we shall provide examples of applications of the type-theory $\mathrm{L}_{rar}^{\lambda}$ of restricted algorithms to algorithmic semantics of basic arithmetic expressions.

The algorithms for computing the values of the expressions are expressed by terms in canonical forms. Some of the terms are restricted terms that have occurrences of restricted recursion variables, i.e., restricted memory locations for saving data satisfying restrictions.

For this purpose, we shall use simple examples from arithmetics, to explicate the algorithmic concepts.

In the terms in this section, we use the symbol "/" for denoting the division operation, in ratio terms like n/d.

8.1 Canonical Terms of Arithmetic Expressions without Restrictions

Example 8.1. The terms A, B, C, respectively in (58a), (59a), (60), designate different algorithms for computing the same numerical value 40, which is the denotation of each of these terms. The corresponding algorithms are determined by the canonical forms $\mathrm{cf}(A)$, $\mathrm{cf}(B)$, $\mathrm{cf}(C)$, given in (58b), (59b), (60):

(1) The term $\mathrm{cf}(A)$ in (58b) determines the algorithm for computing $\mathrm{den}(A)$:

$$A \equiv (200 + 40) / 6 \tag{58a}$$

$$\Rightarrow_{\mathrm{cf}} \underbrace{n / d \text{ where } \{n := (a_1 + a_2),}_{\text{algorithmic pattern}} \quad \underbrace{a_1 := 200,\ a_2 := 40,\ d := 6\}}_{\text{algorithmic instantiation of memory slots}} \equiv \mathrm{cf}(A) \tag{58b}$$

(2) The term $\mathrm{cf}(B)$ in (59b) determines the algorithm for computing $\mathrm{den}(B)$:

$$B \equiv (120 + 120) / 6 \tag{59a}$$

$$\Rightarrow \underbrace{n / d \text{ where } \{n := (a_1 + a_2)}, \underbrace{a_1 := 120,\ a_2 := 120,\ d := 6\}} \equiv \mathrm{cf}(B) \tag{59b}$$

(3) The term $C \equiv \mathrm{cf}(C)$ in (60) determines the algorithm for computing $\mathrm{den}(C)$:

$$C \equiv \underbrace{n / d \text{ where } \{n := (a + a)}, \underbrace{a := 120,\ d := 6\}} \equiv \mathrm{cf}(C) \tag{60}$$

Here, as an example, we assume that the numerical denotations are with respect to the decimal number system (base 10).

The terms A, B, C, in this example, have the same denotations as their canonical forms, respectively, $\mathrm{cf}(A)$, $\mathrm{cf}(B)$, $\mathrm{cf}(C)$.

The canonical forms $\mathrm{cf}(A)$, $\mathrm{cf}(B)$, $\mathrm{cf}(C)$ of the terms A, B, C, respectively in (58b), (59b), (60), determine different algorithms that compute the same number, which is, in addition, denoted by the numeral constant 40, as their denotational values. But, each of their denotations, while equal as denotational

values, den(A) = den(B) = den(C) = den(40), is computed by different algorithms, expressed, correspondingly, by their canonical forms cf(A), cf(B), cf(C), den(40), (61a)–(61b).

These numerical constants have numerical values, which are computed by algorithms that depend on the base of the number system. An important feature of the theory of recursion $\mathrm{L}_{ar}^{\lambda}$, is that some constants by themselves may have denotations computed by, more or less complex, algorithms. By the term cf(B) in (59b), the same value of the constant 120 gets computed twice and stored in two different memory slots: $a_1 := 120$ and $a_2 := 120$.

$$\mathrm{den}(A) = \mathrm{den}(B) = \mathrm{den}(C) = \mathrm{den}(40) \quad \text{(decimal number system)} \tag{61a}$$

$$\mathrm{alg}(A) \neq \mathrm{alg}(B) \neq \mathrm{alg}(C) \neq \mathrm{alg}(40) \tag{61b}$$

8.2 Parametric Algorithms for Arithmetic Expressions with Restrictions

Example 8.2. Recursion terms with restrictor operator designated by the operator constant such that:

- Restrictor terms having satisfied restrictor partssuch that:

$$D_1 \equiv \underbrace{\left(n \,/\, d \text{ such that } \left\{n,\, d \in \mathbb{N},\, d \neq 0\right\}\right)}_{\text{restrictor term R}} \text{ where } \{ \tag{62a}$$

$$n := (a_1 + a_2),\, d := (d_1 \times d_2), \tag{62b}$$

$$a_1 := 200,\, a_2 := 40,\, d_1 := 2,\, d_2 := 3\} \tag{62c}$$

- A term having the restrictor unsatisfied:

$$E_1 \equiv \underbrace{\left(n \,/\, d \text{ such that } \left\{n,\, d \in \mathbb{N},\, d \neq 0\right\}\right)}_{\text{restrictor term R}} \text{where} \{ \tag{63a}$$

$$n := (a_1 + a_2),\, d := (d_1 \times d_2), \tag{63b}$$

$$a_1 := 200,\, a_2 := 40,\, d_1 := 2,\, d_2 := 0\} \tag{63c}$$

- cf(D_1) determines the algorithm alg(D_1) for computing the value den(40)
- cf(E_1) determines the algorithm alg(E_1) for computing the value den(E_1)

$$\mathrm{alg}(D_1) \text{ computes } \mathrm{den}(D_1) = \mathrm{den}(40) \qquad \text{(decimal base)} \tag{64a}$$

$$\mathrm{alg}(E_1) \text{ computes } \mathrm{den}(E_1) = \mathit{Error} \equiv \mathit{er} \tag{64b}$$

Example 8.3. Restricted memory locations, i.e., restricted variables

- The constant such that designates a restrictor operator:
  $R \approx \mathrm{cf}(R)$ and r_0 designate parametric, restricted algorithms

$$R \equiv \underbrace{\left(n \,/\, d \text{ such that } \left\{\left(n \in \mathbb{N}\right), \left(d \in \mathbb{N}\right), \left(d \neq 0\right)\right\}\right)}_{\text{restrictor term } R} \tag{65a}$$

$$\Rightarrow R_1 \equiv \Big[\underbrace{\big(a_0 \text{ such that} \{z_n, z_d, d_0\}\big)}_{\text{restrictor memory variable } r_0} \text{ where } \{ \tag{65b}$$

$$a_0 := n / d,\ z_n := (n \in \mathbb{N}),\ z_d := (d \in \mathbb{N}),$$

$$d_0 := \neg p,\ p := (d = 0)\}] \tag{65c}$$

- r_0, in (65b), and R_1, in (65b)–(65c), are restricted memory variables
- R_1 instantiates r_0 via parametric (underspecified) assignments (65c)

- $D \in$ Terms instantiates the restrictor R_1, by (66a)–(66b)

$$D \equiv r \text{ where } \{r := R_1,\ n := (a_1 + a_2),\ d := (d_1 \times d_2), \tag{66a}$$

$$a_1 := 200,\ a_2 := 40,\ d_1 := 2,\ d_2 := 3\} \tag{66b}$$

- cf(D) designates the algorithm alg(D) for computing the value: den(D) = den(40) (in decimal number system)

Example 8.4. Restricted memory locations, i.e., restricted variables and restriction with negation operator

- $R_1 \approx$ cf(R_1) designate the parametric, restricted algorithm alg(R_1) represented by cf(R_1)

$$R_1 \Rightarrow_{\text{cf}} \Big[\ \underbrace{\big(a_0 \text{ such that} \{z_n, z_d, d_0\}\big)}_{\text{restrictor memory variable } r_0} \text{ where } \{ \tag{67a}$$

$$a_0 := n / d,$$

$$z_n := (n \in \mathbb{N}),\ z_d := (d \in \mathbb{N}),$$

$$d_0 := \neg p,\ p := (d = n_0),\ n_0 := 0\}] \tag{67b}$$

- $D \in$ Terms instantiates the memory variables R_1, cf(R_1), r

$$D \Rightarrow r \text{ where } \{r := \Big[\ \underbrace{\big(a_0 \text{such that} \{z_n, z_d, d_0\}\big)}_{\text{restrictor memory variable } r_0} \text{ where } \{ \tag{68a}$$

$$a_0 := n / d, \tag{68b}$$

$$z_n := (n \in \mathbb{N}),\ z_d := (d \in \mathbb{N}), \tag{68c}$$

$$d_0 := \neg p,\ p := (d = n_0),\ n_0 := 0\}], \tag{68d}$$

$$n := (a_1 + a_2),\ d := (d_1 \times d_2), \tag{68e}$$

$$a_1 := 200,\ a_2 := 40,\ d_1 := 2,\ d_2 := 3\} \tag{68f}$$

$$\Rightarrow_{\text{cf}} r \text{ where } \{r := \underbrace{\big(a_0 \text{such that} \{z_n, z_d, d_0\}\big)}_{\text{restrictor memory variable } r_0}, \tag{68g}$$

$$a_0 := n \,/\, d, \tag{68h}$$

$$z_n := (n \in \mathbb{N}),\ z_d := (d \in \mathbb{N}), \tag{68i}$$

$$d_0 := \neg p,\ p := (d = n_0),\ n_0 := 0, \tag{68j}$$

$$n := (a_1 + a_2),\ d := (d_1 \times d_2), \tag{68k}$$

$$a_1 := 200,\ a_2 := 40,\ d_1 := 2,\ d_2 := 3\} \equiv \mathrm{cf}(D) \tag{68l}$$

from (68a)–(68f), by (B-S)

- cf (D) designates the algorithm alg(D) for computing the value: den(D) = den(40) (in decimal number system)
- The term cf (D) is a restricted memory location r, in which the algorithm alg(D) computes and "stores" the value den(D) = den(40)

9. Application to Algorithmic Semantics of Definite Descriptors

Definite descriptors are present in data of various forms in specific domains of information. For the purposes of general demonstration, we use the natural language definite descriptions, consisting of noun phrases (NPs) formed by the definite determiner "the".

First Order Logic (FOL) and Higher Order Logic (HOL) offer possibilities for rendering expressions of the definite descriptions that have the determiner "the" as a constituent.

We shall briefly present such possibilities because, firstly, while they have unsatisfactory aspects, they are in use in various domains of applications of FOL or classic HOL. We shall show that the classical logical representations of such descriptions in FOL and HOL are available in $\mathrm{L}^{\lambda}_{rar}$.

Then, in addition to the standard possibilities from FOL and HOL, we shall provide new, more adequate possibilities, by using the restricted terms in the extended type-theory of parametric algorithms *lrar*. The reduction calculi of $\mathrm{L}^{\lambda}_{rar}$ provide algorithmic patterns by canonical terms, for more adequate computational semantics of definite descriptors.

In this section, as an illustration of the restricted memory variables, we give one of several possible alternatives, for representation of descriptions formed with the definite determiner "the".

The terms with restricted parameters provide alternatives for computational semantics of definite descriptors, in particular the referential descriptors.

Alternative Definite Operators We shall present different alternative operators for representing the uniqueness property designated by some of the definite descriptors. Such optional representations are dependent on the context, in which descriptors occur. The formal language of $\mathrm{L}^{\lambda}_{rar}$ provides varieties of terms for such variations.

9.1 Classic Representations of the Definite Descriptors

We shall demonstrate some of the typical representations of the definite descriptors by examples from human language, by using the definite determiner "the", e.g., by the sentence Φ in (69):

$$\Phi \equiv \text{The cube is large} \tag{69}$$

9.1.1 First Order Logic (FOL)

The so called Russellian analysis of definite descriptions has been used for representing the determiner "the" in logic forms.

In First Order Logic (FOL), the sentence Φ in (69) can be rendered to the formula A in (70a). A is logically equivalent to the formulae that are the usual Russellian descriptions, i.e., logical forms of the definite determiner "the". The predicate symbol *isLarge*, which renders the verbal phrase (VP) "is large", is applied over the quantified variable.

For example, the logical forms of the definite descriptions are exemplified by the FOL meta-formula, i.e., FOL scheme of definite descriptors, like (70b), by replacing the meta-variables P and Q with specific predicate expressions, e.g., constants *cube* and *isLarge*, respectively:

$$\Phi \ \underline{\text{render}} \ A \equiv \exists x[\underbrace{\forall y(cube(y) \leftrightarrow x = y)}_{uniqueness} \wedge isLarge(x)] \tag{70a}$$

$$S \equiv \exists x[\underbrace{\forall y(P(y) \leftrightarrow x = y}_{uniqueness} \wedge Q(x)] \quad \text{meta-formula in FOL: scheme for FOL formulae} \tag{70b}$$

The predication expressed by a Russellian logic form, like (70b), and its specific case (70a), has the following features:

- Existential quantification as the direct, topmost predication
- Predication of the uniqueness: in conjunction with the predication by the VP ("is large")
- The logical form A of Φ has no referential force towards the object denoted by the noun phrase (NP) "the cube", (71):

$$[\text{the cube}]_{\text{NP}} \tag{71}$$

- There is no distinctive, separate component rendering of the definite descriptor "the cube", (71), i.e., of its compositional rendering, from the renderings of its constituents: the determiner "the" and the common noun "cube"
- There is no compositional analysis, i.e., no "derivation" of the rendering of A from the renderings of the linguistic components of the sentence Φ

Note 3. In FOL, (70b) is an expression in meta-language, i.e., a formula scheme for many well-formed FOL formulae, including for (70a).

In L^{λ}_{ar}, (70b) is a well-formed L^{λ}_{ar}-term, i.e., $S \in$ Terms, for $P, Q \in$ Vars, e.g., in each of the cases $P, Q \in$ PureV or $P, Q \in$ RecV.

9.1.2 Higher Order Logic (HOL)

The method of the generalised quantifiers was introduced by (Henkin, 1950) and (Mostowski, 1957). It can be used for the definite descriptions like Φ in (69). By generalised quantification that uses

the Russellian descriptions, the determiner "the" has lost its natural referential force of interpretation, e.g., in the following renderings (there are optional alternatives):

$$\text{the } \underline{\text{render}}\ T \equiv \left[\lambda P \lambda Q \left[\exists x \left[\underbrace{\forall y\left(P\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right]\right] \tag{72a}$$

$$\text{the cube } \underline{\text{render}}\ C \equiv T\left(cube\right)$$

$$C \equiv \left[\lambda P \lambda Q \left[\exists x \left[\underbrace{\forall y\left(P\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right]\right]\left(cube\right) \tag{72b}$$

$$D \equiv \lambda Q \left[\exists x \left[\underbrace{\forall y\left(cube\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right] \tag{72c}$$

(from (72b) by denotational β-conversion)

The rendering C in (72b) is obtained by the operation of functional application of T, from (72a), to the rendering of the common noun, *cube*, i.e.: $C \equiv T$ (*cube*), which is denotationally equivalent to $D \in$ Terms in (72c).

Then, the rendering of a sentence like Φ into a term $B \equiv D(isLarge)$ is the next application, as in (73b):

$$\Phi \equiv \text{The cube is large } \underline{\text{render}}\ B \equiv D\left(isLarge\right) \tag{73a}$$

$$B \equiv \left[\lambda Q \left[\exists x \left[\underbrace{\forall y\left(cube\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right]\right]\left(isLarge\right) \tag{73b}$$

$$\models A' \equiv \exists x \left[\underbrace{\forall y\left(cube\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge isLarge\left(x\right)\right] \tag{73c}$$

(from (73b) by denotational β-conversion)

The formula A', in (73c), is well-formed term of HOL, obtained by β-conversion from the term $B \equiv D$ (*isLarge*) in (73a). After that, A' is the same expression as the FOL formula A, in (70a).

Property 1 (Montague Intensional Logic (IL)). *The denotational equalities in* (72a)–(72b)–(72c)*, and* (73a)–(73c) *are valid in Montague Intensional Logic (IL) (and other classic λ-calculus).*

The stepwise, component analysis represented by T in (72a), and then by D, from T in (72b)–(72c), followed by using D in (73a)–(73c), to obtain A', can be put into a joint sequence of β-conversions, by using the compositionally of the denotational equality.

Property 2 (Montague Intensional Logic (IL)). *The denotational equalities in* (74a)–(74b)–(74c) *and* (74c)–(74d) *are valid in Montague Intensional Logic (IL) (and other classic λ-calculi with valid β-conversion). (Thomason, 1974).*

$$L \equiv [T\,(cube)](isLarge) \tag{74a}$$

$$\equiv \underbrace{\left[\left[\left[\lambda P \lambda Q\left[\exists x\left[\underbrace{\forall y\left(P\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right]\right]\left(cube\right)\right]\left(isLarge\right)\right]}_{\beta\text{-conversion}} \tag{74b}$$

$$\vDash \left[\lambda Q\left[\exists x\left[\underbrace{\forall y\left(cube\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge Q\left(x\right)\right]\right]\right]\left(isLarge\right) \tag{74c}$$

(from (74b), by denotational β-conversion and Denotational Replacement Property)

$$\vDash A' \equiv \left[\exists x\left[\underbrace{\forall y\left(cube\left(y\right) \leftrightarrow x = y\right)}_{uniqueness} \wedge isLarge\left(x\right)\right]\right] \tag{74d}$$

(from (74c), by denotational β-conversion)

Property 3 (In $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$). *Assume that:*

(i) $x, y \in \mathrm{PureV}_{\bar{e}}$, $P, Q \in \mathrm{PureV}_{(\bar{e}\to\bar{t})}$

(ii) $cube, isLarge \in \mathrm{Consts}_{(\bar{e}\to\bar{t})}$

Then:

(1) *the above expressions T, C, D, B, A' are well-formed $\mathrm{L}^{\lambda}_{ar}$-terms, i.e., $T, C, D, B, A' \in$ Terms, of respective types*

(2) *The above replacements* (74a)–(74b)–(74c) *and* (74c)–(74d) *are denotationally valid in $\mathrm{L}^{\lambda}_{ar}$, and also in $\mathrm{L}^{\lambda}_{rar}$*

Proof. Item (1) follows from Def. 2, (6a)–(6e) for the set Terms of $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$.

Item (2) follows from the denotational β-conversion and Denotational Replacement Property in L^{λ}_{ar}. see (Moschovakis, 2006) and (Loukanova, 2019b; Loukanova, 2019c) □

Question 1. *Are the above replacements* (74a)–(74b)–(74c) *and* (74c)–(74d) *algorithmically equivalent in* L^{λ}_{ar} *or not?*

To provide an answer of this question, we need to develop the addition of the standard quantifiers and quantified formulae, such as $\exists xA$ *and* $\forall xA$ *to the set of the logical operators for formation of formulae of* L^{λ}_{ar}. *Furthermore, there is need to develop the reduction rules to encompass formulae. This work is outside the scope of this paper, especially because it is space taking.*

9.2 Direct Interpretation of the Definite Determiner (Option 1)

The type-theory of acyclic recursion L^{λ}_{ar} provides good flexibility for representing the referential force of the definite descriptors.

The determiner "the" can be used in NPs for reference to specific entities that are subject to uniqueness requirement. We present rendering "the" to a referential constant of definitness:

$$\text{the } \underline{\text{render}}\ \textit{the} \in \text{Consts}_{\left(\left(\tilde{e}\to\tilde{t}\right)\to\tilde{e}\right)} \tag{75}$$

The following denotation of the constant *the* is provided by (Moschovakis, 2006):

$$\left[\left(\text{den}\left(\textit{the}\right)\right)\left(g\right)\right]\left(\overline{p}\right)\left(s_0\right) = \begin{cases} y, & \text{if, } y \text{ is the unique } y \in \mathbb{T}_{\tilde{e}}, \\ & \text{for which } \overline{p}\left(s \mapsto y\right)\left(s_0\right) = 1 \\ \text{er}, & \text{otherwise,} \\ & \text{i.e., there is no unique entity} \\ & \text{that has the property } \overline{p} \text{ in } s_0 \end{cases} \tag{76a}$$

$$\text{for every } \overline{p} \in \mathbb{T}_{(\tilde{e}\to\tilde{t})} \text{ and every } s_0 \in \mathbb{T}_s \text{ in the semantic domains} \tag{76b}$$

Then, a sentence like Φ can be rendered into a term $A_0 \approx \text{cf}(A_0)$, as in (77a)–(77c):

$$\text{The cube is large } \underline{\text{render}}\ \text{cf}\left(A_0\right) : \tilde{t} \tag{77a}$$

$$A_0 \equiv \textit{isLarge}(\textit{the}(\textit{cube})) \tag{77b}$$

$$\Rightarrow \textit{isLarge}(d) \text{ where } \{d := \textit{the}(c),\ c := \textit{cube}\} \equiv \text{cf}(A_0) \tag{77c}$$

from (77b) by (ap), (wh-comp), (ap), (B-S)

Instead of the direct denotation of the constant *the*, we shall consider several options by using restrictor terms, with such that. Towards that, at first, we introduce a constant designating the relation uniqueness.

9.3 Optional Properties of Uniqueness of Objects

Here, we shall consider several options for constants of L_{rar}^{λ} that express alternative properties of uniqueness of objects, by defining them by slight denotational differences. We shall select the constant *unique* defined by (86).

Example 9.1 (Option 2: Tentative Candidate for Combination with Definitness Determiner "the"). Here, we shall look into a constant $unique_0$ for uniqueness of any object y satisfying a property p in a state s_0, by (79).

$$unique_0 \in \text{Consts}_{((\bar{e}\to\bar{t})\to(\bar{e}\to\bar{t}))} \tag{78}$$

For every $\overline{p} \in \mathbb{T}_{(\bar{e}\to\bar{t})}$, $\overline{q} \in \mathbb{T}_{\bar{e}}$, and every $s_0 \in \mathbb{T}_s$, we can define the denotation den($unique_0$) by (79):

$$\left[\left(\text{den}\left(unique_0\right)\right)\right]\left(\overline{p}\right)\left(\overline{q}\right)\left(s_0\right) = \begin{cases} 1, & \overline{q}\left(s_0\right) \text{ is the unique } y \in \mathbb{T}_e \\ & \text{s.t. } \overline{p}\left(s \mapsto y\right)\left(s_0\right) = 1 \\ er, & \text{otherwise,} \end{cases} \tag{79}$$

The weakness of the constant $unique_0$ is that, by (79), both (80a)–(80b) are possible, for some $\overline{p}_0 \in \mathbb{T}_{(\bar{e}\to\bar{t})}$, $\overline{q}_0 \in \mathbb{T}_{\bar{e}}$, $s_0 \in \mathbb{T}_s$:

$$\overline{q}_0(s_0) = er \text{ and}$$

$$\overline{q}_0(s_0) = \text{is the unique } y \in \mathbb{T}_e \text{ s.t. } \overline{p}_0\ (s \to y)\ (s_0) = 1 \tag{80a}$$

$$[(\text{den}(unique_0))]\ (\overline{p}_0)(\overline{q}_0)(s_0) = 1 \tag{80b}$$

We shall consider better possibilities for the uniqueness property, instead of Option 2, to be used in definite descriptors.

Example 9.2. Now, we shall look into a possible constant $unique_1$ for uniqueness of $y \neq er$ satisfying a property p in a state s_0

$$\text{For every } \overline{p} \in \mathbb{T}_{(\bar{e}\to\bar{t})},\ \overline{q}_0 \in \mathbb{T}_{\bar{e}},\ s_0 \in \mathbb{T}_s,$$

$$\left[\left(\text{den}\left(unique_1\right)\right)\left(g\right)\right]\left(\overline{p}\right)\left(\overline{q}\right)\left(s_0\right) = \begin{cases} 1, & \text{if } \overline{q}\left(s_0\right) \text{ is the unique } y \in \mathbb{T}_e \\ & \text{such that } y \neq \text{er and} \\ & \qquad \overline{p}\left(s \mapsto y\right)\left(s_0\right) = 1 \\ er, & \text{otherwise} \end{cases} \tag{81}$$

By defining the constant $unique_1$, with (81), we have that

$$\text{den}(unique) \neq \text{den}(unique_1) \tag{82}$$

Then, by using (82): It is possible that there exist some $\overline{p}_0 \in \mathbb{T}_{(\bar{e}\to\bar{t})}$, $\overline{q}_0 \in \mathbb{T}_{\bar{e}}$, $s_0 \in \mathbb{T}_s$, such that:

$$\text{for all } x\ [[x \neq \text{er}\ \ \&\ \ \overline{p}_0(s \mapsto x)(s_0) = 1] \tag{83a}$$

ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 12 N. 1 (2023), e29081
eISSN: 2255-2863 - https://adcaij.usal.es
Ediciones Universidad de Salamanca - CC BY-NC-ND

$$\Longleftrightarrow x = \overline{q}_0(s_0)]$$

$$\overline{p}_0(s \mapsto er)(s_0) = 1 \tag{83b}$$

From the statement (83a), it follows that $\overline{q}_0(s_0) \neq er$ and $\overline{q}_0(s_0)$ is the unique $y \in \mathbb{T}_{\bar{e}}$, such that $y \neq er$ and $\overline{p}_0(s \mapsto y)\ (s_0) = 1$. Therefore, (84a) holds.

From (83a)–(83b), it follows that both $\overline{q}_0(s_0) \neq er$ and er have the property $\overline{p}_0$ in s_0, i.e., $\overline{q}_0(s_0)$ is not the unique $y \in \mathbb{T}_{\bar{e}}$ that has the property $\overline{p}_0$ in s_0. Thus, $\overline{q}_0(s_0) \neq er$, but $\overline{q}_0(s_0)$ is not the unique $y \in \mathbb{T}_{\bar{e}}$, such that $\overline{p}_0(s \mapsto y)\ (s_0) = 1$. Therefore, (84b).

$$(\text{den}(unique_1))\ (g)(\overline{p}_0)(\overline{q}_0)(s_0) = 1 \tag{84a}$$

$$(\text{den}(unique))\ (g)(\overline{p}_0)(\overline{q}_0)(s_0) = er \tag{84b}$$

Suggested Constant Designating Property of Uniqueness of Objects Now, we shall consider a constant *unique* denoting a property of uniqueness of the object $y = q(s_0)$ in s_0, for a given, state-dependent object q, satisfying any given property p in a state s_0.

Assume that $\mathrm{L}^{\lambda}_{ar}$ has a constant *unique* of the type in (85):

$$unique \in \text{Consts}_{((\bar{e}\to\bar{t})\to(\bar{e}\to\bar{t}))} \tag{85}$$

We can define the denotation of the constant *unique* by (86), for any given function $g \in G$, which is a variable valuation, i.e., provides values to all variables of $\mathrm{L}^{\lambda}_{ar}$. Note that, as a simplification, the variable valuation g can be suppressed, since it doesn't affect the value of the constant *unique* when applied to objects in the semantic domains. We can define

For every $\overline{p} \in \mathbb{T}_{(\bar{e}\to\bar{t})}$, $\overline{q} \in \mathbb{T}_{\bar{e}}$, and every $s_0 \in \mathbb{T}_s$:

$$\left[\left(\text{den}\left(unique\right)\right)\left(g\right)\right]\left(\overline{p}\right)\left(\overline{q}\right)\left(s_0\right) = \begin{cases} 1, & \text{if } \overline{q}_0\left(s_0\right) \neq er \text{ and,} \\ & \overline{q}\left(s_0\right) \text{ is the unique } y \in \mathrm{T}_e \\ & \text{such that } \overline{p}\left(s \mapsto y\right)\left(s_0\right) = 1 \\ & \\ er, & \text{otherwise,} \end{cases} \tag{86}$$

Therefore, $[(\text{den}(unique))(g)](\overline{p})(\overline{q})(s_0) = [(\text{den}(unique))]\ (\overline{p})(\overline{q})(s_0) = 1$ iff there exists (exactly one) entity y, $y \in \mathbb{T}_e$, such that $y = \overline{q}(s_0) \neq er$, and y is the unique object that has the property $\overline{p}$ in s_0, i.e., $\overline{p}(s \mapsto y)\ (s_0) = 1$. This esistence condition is expressed by (87a), and also by its instantiated version (87b). These expressions, (87a) and (87b), are in metalanguage, not in $\mathrm{L}^{\lambda}_{rar}$, and designate equivalent statements. Otherwise, the reference fails, i.e., $[(\text{den}(unique))\ (g)](\overline{p})(\overline{q})(s_0) = er$.

$$\text{exists } y\ [y = \overline{q}(s_0) \neq er\ \&\quad \text{for all } x\ [\overline{p}(s \mapsto x)(s_0) = 1 \Longleftrightarrow x = y]] \tag{87a}$$

$$\overline{q}(s_0) \neq er\ \&\quad \text{for all } x\ [\overline{p}(s \mapsto x)(s_0) = 1 \Longleftrightarrow x = \overline{q}(s_0)] \tag{87b}$$

9.4 Rendering of the Definite Determiner "the" via Underspecification

Syntactically, we take the definite determiner "the" to be a lexeme of the syntactic part of speech, category DET of the determiners.

For given memory (recursion) variables p, q of the types in (88c), the definite determiner "the" can be rendered into an underspecified term, as in (88a) for a property denoted by the memory variable $p : (\tilde{e} \rightarrow \tilde{t})$ and an entity $q : \tilde{e}$. Semantically, by the term (88a), the definite determiner "the" is treated as designating a restricted parameter. Any entity that is denoted by (88a) is the denotation of q restricted to be the unique object having an underspecified property p that would be contributed by the nominal complement of the determinar "the".

$$\text{the } \underline{\text{render}}\ A_1 \equiv \Big(q \text{ s.t.}\big\{unique\big(p\big)\big(q\big)\big\}\Big) : \tilde{e} \tag{88a}$$

$$\text{the } \underline{\text{render}}\ \text{cf}\big(A_1\big) \equiv \Big(q \text{ s.t.}\big\{U\big\}\Big) \text{ where } \big\{U := unique\big(p\big)\big(q\big)\big\} \tag{88b}$$

$$\text{for } p \in \text{RecV}_{(\tilde{e}\rightarrow\tilde{t})}, \quad q \in \text{RecV}_{\tilde{e}}, \quad U \in \text{RecV}_{\tilde{t}} \tag{88c}$$

The memory variable p is free, i.e., underspecified, in the term A_1, i.e., $p \in \text{FreeV}(A_1)$, in (88a), and also in the canonical form $\text{cf}(A_1)$, i.e., $p \in \text{FreeV}(\text{cf}(A_1))$.

Then, p can be specified when the determiner "the" is combined with the nominal head in NPs. In this example, we take simple common nouns, like "cube":

$$\text{the cube } \underline{\text{render}}\ A_2 \equiv \Big(q \text{ s.t.}\big\{unique\big(p\big)\big(q\big)\big\}\Big) \text{ where}\big\{p := cube\big\} \tag{89a}$$

$$\text{the cube } \underline{\text{render}}\ \text{cf}\big(A_2\big) \equiv \Big(q \text{ s.t.}\big\{U\big\}\Big) \text{ where}\big\{U := unique\big(p\big)\big(q\big),\ p := cube\big\} \tag{89b}$$

by (st1), (head), from (89a)

Algorithmic Pattern of Referential Force Both terms (88b) and (89b) have referential force, in distinction from existential predications involving uniqueness:

- The term (88b) expresses the underspecified, algorithmic pattern of reference to the unique object q having the underspecified (unknown) property p
- The term A_2 has a referential force, by also being a restricted variable constrained to be the unique, with already specified property, e.g., $p := cube$
- It is possible to render the definite determiner "the" into (88a), and then instantiate p, e.g., as in (89a), and then reduce to the canonical form $\text{cf}(A_2)$
- The underspecified term $\text{cf}(A_1)$ in canonical form, in (88b), is preferable for rendering the definite article "the". Then, (88b) can be instantiated by suitable assignment, e.g., as in (89b)

The canonical form cf (A_2) in (89b) represents an algorithmic pattern for semantic representation of a class of definite descriptions interpreted as reference to entities by the restrictor (q s.t. $\{U\}$). The subterm $U := unique(p)(q)$, In the term (89b), is for uniqueness, i.e., definiteness, for the property represented $p := cube$, in this example. The recursion variable q is free, and, thus underspecified, obtaining its denotation in a specific context. via assignments, e.g., by a "forthcoming" verb (V) or a verb phrase VP.

Then, a sentence having such a definite description in its subject position, can be rendered to the respective term A_3, or better directly to its canonical form cf(A_3) as in (90a)–(90d):

$$\text{The cube is large } \underline{\text{render}}\ \text{cf}\left(A_3\right) : \tilde{\text{t}} \tag{90a}$$

$$A_3 \equiv \mathit{isLarge}\ ((q \text{ s.t. } \{\mathit{unique}(p)(q)\}) \text{ where } \{p := \mathit{cube}\}) \tag{90b}$$

$$\Rightarrow \mathit{isLarge}(Q) \text{ where } \{$$

$$Q := [(q \text{ s.t. } \{\mathit{unique}(p)(q)\}) \text{ where } \{p := \mathit{cube}\}]\} \tag{90c}$$

by (ap), from (90b)

$$\Rightarrow_{\text{cf}} \text{cf}(A_3) \equiv \mathit{isLarge}(Q) \text{ where } \{Q := (q \text{ s.t. } \{U\}),$$

$$U := \mathit{unique}(p)(q), p := \mathit{cube}\} \tag{90d}$$

by (st1), (wh-comp), (head), (B-S), from (89b), (90c)

The term cf (A_3) in (90d) carries an algorithmic pattern for rendering predicative sentences, which have a subject NP that is a definite descriptor, into L^{λ}_{rar}. The reference by the descriptor is expressed by the memory variable Q, which in this specific example is instantiated by the assignments to Q, U, and p, but in other instances, the corresponding term parts can be different instantiations.

9.5 Algorithmic Pattern: Definite Descriptors (Option 3)

The canonical form $A \equiv \text{cf}(A)$ in (91a) is a generalization from (90d), by taking away the specific instantiations for the predicate constants *isLarge* and *cube*.

The term $A \equiv \text{cf}(A)$ in (91a) represents the computational algorithm for definite descriptors, e.g., such as composed with the definite article "the" and common nouns. The open predicative term $L(Q)$ designates predication of a property L to an underspecified object designated by the underspecified definite descriptor Q.

The restricted recursion (memory) variable $Q := (q \text{ s.t. } \{U\})$ is underspecified without a context.

$$\textit{the } \underline{\text{render}}\ A \equiv \text{cf}\left(A\right) \equiv L\left(Q\right) \text{ where } \left\{Q := \left(q \text{ s.t. } \left\{U\right\}\right), U := \mathit{unique}\left(p\right)\left(q\right)\right\} \tag{91a}$$

$$p, q, L \in \text{FreeV}(A), p \in \text{RecV}_{(\tilde{\text{e}} \to \tilde{\text{t}})}, q \in \text{RecV}_{\tilde{\text{e}}}, \tag{91b}$$

$$Q \in \text{RecV}_{\tilde{\text{e}}}, U \in \text{RecV}_{\tilde{\text{t}}}, L \in \text{RecV}_{(\tilde{\text{e}} \to \tilde{\text{t}})} \tag{91c}$$

Note that $q \in \text{RecV}_{\tilde{\text{e}}}$ is a memory variable, which is free in $A \in$ Terms, in (91a), i.e., $q \in \text{FreeV}(A)$. Its denotation den(q)(g) can be obtained by a variable assignment g satisfying the restriction of uniqueness in (91a).

9.6 Named Entities in Definite Descriptors and Predications

Here, we give an example of the definite descriptors as a direct *reference* by named entities in predicative sentences:

$$\text{The cube } n \text{ is large } \underline{\text{render}}\ \text{cf}\left(A_4\right) : \tilde{\text{t}} \tag{92a}$$

$$A_4 \equiv isLarge((q \text{ s.t. } \{unique(N)(q), p(q)\}) \text{ where } \{ \quad (92b)$$
$$q := n, p := cube, N := named\text{-}n\})$$
$$\Rightarrow_{cf} \text{cf}\,(A_4) \equiv isLarge(Q) \text{ where } \{Q := (q \text{ s.t. } \{U, C\}),$$
$$U := unique(N)(q), C := p(q), \quad (92c)$$
$$q := n, p := cube, N := named\text{-}n\}$$

The restriction about the uniqueness of the object named by a constant can be dropped out, by assuming that it is a part of the interpretation function on the constants, such as $n \in$ Consts. Thus, the same sentence can be rendered into the following, simpler term. in it, there is a direct reference; uniqueness and existence are consequences, not a direct part of the rendering term

$$\text{The cube } n \text{ is large } \underline{\text{render}}\ \text{cf}\left(A_5\right) : \tilde{\text{t}} \quad (93a)$$
$$A_5 \equiv isLarge((q \text{ s.t. } \{p(q)\}) \text{ where } \{q := n, p := cube\}) \quad (93b)$$
$$\Rightarrow_{cf} \text{cf}(A_5) \equiv isLarge(Q) \text{ where } \{Q := (q \text{ s.t. } \{C\}), C := p(q),$$
$$q := n, p := cube\} \quad (93c)$$

In Sects. 9.2–9.6, we provided renderings of descriptors and of sentences that include them, to L^{λ}_{rar} terms, which do not use any λ-abstractions and λ-applications. Instead, the renderings of the larger expressions combine the ones of the components, by the operator of term applications guided by types of the components, and / or specifications of underspecified memory variables, by adding assignments, in a compositional mode.

9.7 Rendering of the Definite Article "the" via λ-Abstraction

In this and the following Sects.9.7–9.8, we provide compositional renderings of expressions by combining smaller components into larger ones, by using λ-abstractions and λ-applications, by L^{λ}_{rar} terms and reduce them to canonical forms, by the reduction rules of L^{λ}_{rar}, from Sect. 5.2.

In (94a)–(94e), we give a possible rendering of the definite article "the", which is the essential component of many of the definite descriptors:

$$\text{the } \underline{\text{render}}\ B_1^{((\tilde{e}\to\tilde{t})\to\tilde{e})} \,/\, \text{cf}\left(B_1^{((\tilde{e}\to\tilde{t})\to\tilde{e})}\right), \text{ for } unique \in \text{Consts}_{((\tilde{e}\to\tilde{t})\to(\tilde{e}\to\tilde{t}))} \text{ as in (86)} \quad (94a)$$

$$B_1 \equiv \lambda(x)\,([q \text{ s.t. } \{unique(p)(q)\}] \text{ where } \{p := x\}) \quad (94b)$$
$$\Rightarrow \lambda(x)\,([q \text{ s.t. } \{U\}] \text{ where } \{U := unique(p)(q)\}]$$
$$\text{where } \{p := x\}) \quad (94c)$$

by (st1), (wh-comp), (λ-comp), from (94b)

$$\Rightarrow \lambda(x)\,([q \text{ s.t. } \{U\}] \text{ where } \{U := unique(p)(q), p := x\}) \quad (94d)$$

by (head), (λ-comp), from (94c)

$$\Rightarrow_{\text{cf}} \lambda(x)\,[q \text{ s.t. } \{U'(x)\}] \text{ where } \{U' := \lambda(x)\,[unique(p'(x))(q)],$$
$$p' := \lambda(x)(x)\} \tag{94e}$$

by (λ), from (94d)

$$\equiv \text{cf}(B_1) \tag{94f}$$

The rendering of a definite descriptor, like "the cube" in (95a)–(95e), combines the ones of its components, i.e., here we chose cf (B_1), for the definite article "the", from (94e), instead of B_1, and the term (i.e., the constant) *cube* rendering the noun "cube":

$$\text{the cube } \underline{\text{render}}\ \text{cf}\left(\text{cf}\left(B_1\right)\left(cube\right)\right) \equiv \text{cf}\left(B_2\right) : \tilde{\text{e}} \qquad \text{from (94e)} \tag{95a}$$

$$B_2 \equiv [\lambda(x)\,[q \text{ s.t. } \{U'(x)\}] \text{ where } \{$$
$$U' := \lambda(x)[unique(p'(x))(q)], \tag{95b}$$
$$p' := \lambda(x)(x)\}](cube)$$
$$\Rightarrow [\lambda(x)\,[q \text{ s.t. } \{U'(x)\}]](cube) \text{ where } \{$$
$$U' := \lambda(x)[unique(p'(x))(q)], \tag{95c}$$
$$p' := \lambda(x)(x)\}$$

by (recap), from (95b)

$$\Rightarrow [[\lambda(x)\,[q \text{ s.t. } \{U'(x)\}]](c) \text{ where } \{c := cube\}]$$
$$\text{where } \{U' := \lambda(x)[unique(p'(x))(q)], \tag{95d}$$
$$p' := \lambda(x)(x)\}$$

by (ap), (wh-comp), from (95c)

$$\Rightarrow_{\text{cf}} \text{cf}(B_2) \equiv [[\lambda(x)[q \text{ s.t. } \{U'(x)\}]](c)]$$
$$\text{where } \{U' := \lambda(x)\,[unique(p'(x))(q)],$$
$$p' := \lambda(x)(x), c := cube\}$$

by (head), (cong), from (95d)

9.8 The Definite Determiner "the" and Descriptors in Predicative Sentences

Computational grammar of natural language can implement compositional combinations of the renderings of the sentence components. In (96a)–(96d), the rendering of the verb phrase, a predicate term, e.g., *is Large*, is applied to that of a definite descriptor, like "the cube" from (95a)–(95e):

$$\text{The cube is large } \underline{\text{render}}\ isLarge\left(\text{cf}\left(B_2\right)\right) \equiv B_3 : \tilde{\text{t}} \quad \text{from (95e) by trem application} \tag{96a}$$
$$B_3 \equiv isLarge\left(\left[[\lambda(x)[q \text{ s.t. } \{U'(x)\}]](c)\right]\right.$$

$$\text{where}\{U' := \lambda(x)[unique(p'(x))(q)], \quad (96b)$$
$$p' := \lambda(x)(x),\ c := cube\}\big)$$
$$\Rightarrow isLarge(Q)\ \text{where}\{Q := \Big(\big[[\lambda(x)[q\ \text{s.t.}\ \{U'(x)\}]\big](c)\Big]$$
$$\text{where}\{U' := \lambda(x)[unique(p'(x))(q)], \quad (96c)$$
$$p' := \lambda(x)(x),\ c := cube\}\big)\}$$

by (ap), from (96b)

$$\Rightarrow_{\text{cf}} \text{cf}(B_3) \equiv isLarge(Q)\ \text{where}\ \{Q := [\lambda(x)[q\ \text{s.t.}\ \{U'(x)\}]](c),$$
$$U' := \lambda(x)\ [unique(p'(x))(q)], \quad (96d)$$
$$p' := \lambda(x)(x),\ c := cube\}$$

by (B-S), from (96c)

10. Conclusions and Outlook

In this paper, we have extended the theory of typed, acyclic recursion $\mathrm{L}^{\lambda}_{ar}$, by introducing terms with restrictions. The result is a formal language $\mathrm{L}^{\lambda}_{rar}$, which is a proper extension of the language $\mathrm{L}^{\lambda}_{ar}$ and its reduction calculus. The same extension applies to the version of the type-theory with full recursion L^{λ}_{r} without the acyclicity.

The two subclasses of the terms with restrictions, i.e., the basic restricted memory variables, and restricted memory variables, see Theorem 3, provide parametric algorithmic patterns for semantic representations of memory locations. The memory variables are typed, and thus, can be used to store data of the corresponding types. In addition, the restricted memory variables, introduced in this paper, can be used to store data, which is constrained to satisfy propositional restrictions. The restrictions are calculated recursively, by iterative algorithms determined by canonical forms of formal terms of $\mathrm{L}^{\lambda}_{rar}$. We have introduced a formalization of restricted algorithmic patterns for computational semantics of formal languages, e.g., in programming, and natural languages, by illustrations with mathematical expressions and definite descriptors of natural language, which are typical problems for Natural Language Processing (NLP) of a class of singular NPs that are definite descriptions.

Restrictor versus Conditionals This paper does not cover possible representations of the conditionals by $\mathrm{L}^{\lambda}_{rar}$. We shall only comment that topic, briefly in this note.

In the formal language of $\mathrm{L}^{\lambda}_{rar}$, the operator constant such that is essentially different from the conditional operator constant if … then …, by the definitions of syntax and semantics of term formation of restricted terms, see Def. 2, (6a)–(6e). Relatively complex or simple, it can be used for the same restrictor operator designated by the constant such that. Its syntactic and semantic roles are determined by the denotational and algorithmic semantics, and the reduction calculi. The restricted terms (6e) can be replicated only to certain extend by the traditional terms involving "if … then", and "if … then … else …". To some extend this is so, because $\mathrm{L}^{\lambda}_{ar}$, and $\mathrm{L}^{\lambda}_{rar}$, do not include constants (or terms) for the erroneous semantic values such as *er*. Secondly, restricted terms of the form (6e) carry referential force, which is demonstrated by the analysis of the definite descriptions in $\mathrm{L}^{\lambda}_{rar}$, presented in this paper.

An investigation of potential distinctions and similarities between the restrictor operator and the conditionals is a subject of other forthcoming work. This is more important, for an extended type-theory of full recursion, by allowing terms that do not obey the Acyclicity Constraint (AC) given on page 6.

This paper is part of such extended work. Our focus is on developments in two interrelated lines of work: type-theory of parametric algorithms and applications.

Theoretical Developments Here we shall briefly mention wider classes of formal languages and their algorithmic type theories. Recent theoretical work (Loukanova, 2019b; Loukanova, 2019c) extends the reduction calculi of $\mathrm{L}^{\lambda}_{rar}$ and L^{λ}_{r} . Extensive research on the algorithmic syntax-semantics of the type-theory of algorithms is in our ongoing and future work, by covering the following distinctions:

- The formal language L^{λ}_{r} of full recursion is similar to the formal language of $\mathrm{L}^{\lambda}_{ar}$, by Def. 2, (6a)–(6d), without the acyclicity constraint (AC), (8a)–(8c)
- The formal language $\mathrm{L}^{\lambda}_{rr}$ of full recursion and restrictors is similar to the formal language of $\mathrm{L}^{\lambda}_{rar}$, by Def. 2, (6a)–(6e), without the AC, (8a)–(8c)
- The classes of formal languages, with their corresponding reduction calculi and theories, L^{λ}_{r}, $\mathrm{L}^{\lambda}_{rr}$, come with similar sets of reduction rules, as $\mathrm{L}^{\lambda}_{ar}$, $\mathrm{L}^{\lambda}_{rar}$, respectively
- The reduction computations, $A \Rightarrow B$, are defined, for all terms A, B of the formal theories $\mathrm{L}^{\lambda}_{ar}$ / $\mathrm{L}^{\lambda}_{rar}$ / L^{λ}_{r} / $\mathrm{L}^{\lambda}_{rr}$, according to the corresponding set of reduction rules

Applications In our ongoing and future work on applications of type-theory of algorithms, with acyclic, $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$, and, respectively, with full recursion L^{λ}_{r} and $\mathrm{L}^{\lambda}_{rr}$, we have been maintaining several lines of applications. Our focus is on the reduction calculus of $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$, for acyclic algorithms, having terms with restrictor operator, and corresponding reduction rules introduced by this paper. The reason is that acyclic recursion and iteration guarantee termination of algorithms. This feature of $\mathrm{L}^{\lambda}_{ar}$ and $\mathrm{L}^{\lambda}_{rar}$ has great significance for practical applications of AI, e.g.:

- Applications to formal and natural languages
  - Computational Semantics
  - Computational Syntax-Semantics Interfaces
  - Semantics of programming and specification languages
  - Theoretical foundations of compilers
- Computational Neuroscience, e.g., by theoretical development, which has potential applications for linking receptors (Loukanova, 2020b)

Applications to Computational Grammar of Natural Language Applications to advanced, intelligent technologies, including AI areas, require integration of computational grammar of natural language, by covering syntax-semantics interfaces, e.g., (Loukanova, 2019a). In forthcoming and future work, we present syntax-semantic interfaces for phrasal varieties of natural language. For example, a broad class of noun phrases (NPs), will benefit and require adequate coverage of computational semantics, which is provided by the algorithmic approach of the extended type-theory of algorithms

L^{λ}_{rar} including the restrictor operator, presented by Def. 2, (6a)–(6e). Compositionally, the algorithmic semantics of such NPs propagates into the semantics of enclosing sentences and other expressions, by syntax-semantics in the computational grammar.

Definite Descriptors Descriptors are abundant in natural and formal languages, e.g., in specification languages in data bases and advanced software packages. Sentences that include definite descriptors, like Φ in (69), express prediction of a property, e.g., "is large", to the entity designated by the descriptor, e.g., "the cube". Classic logic, e.g., as in Sect. 9.1, represents such property predications by enclosing the corresponding property formulae in existential predications, which also include FOL formulae for uniqueness, e.g., by the FOL *A* in (70a). Introducing generalized quantifiers, e.g., as in Sect. 9.1.2, significantly improve the compositionally of the semantic representations of the syntactic components of the definite descriptors and sentences that include them, e.g., by the denotational equalities in (74a)–(74d).

Type-theory of algorithms L^{λ}_{ar} (L^{λ}_{r}) has formal terms that are also in FOL and HOL, including in Montague IL. This will facilitate upgrading of software that uses such classic semantics to advanced algorithmic semantics. Among the other advantages of the L^{λ}_{ar} terms, especially by the extended L^{λ}_{rar}, is that they express their denotations and also, their algorithmic semantics—the canonical forms of the L^{λ}_{ar} and L^{λ}_{rar} terms determine the respective algorithmic steps, i.e., by computational iteration of ranked algorithmic assignments.

In addition, L^{λ}_{ar} and L^{λ}_{rar} provide terms that express more subtle semantic distinctions, including for expressions of semantic reference force to the objects designated by definite descriptors. We investigated alternative options in Sects. 9.2–9.8.

10.1 Acknowledgements

This paper is essentially extended work from (Loukanova, 2021).

11. References

Gallin, D., 1975. *Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics*. North-Holland Publishing Company, Amsterdam and Oxford, and American Elsevier Publishing Company. ISBN 0 7204 0360 X.

Henkin, L., 1950. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15(2):81–91.

Loukanova, R., 1991. *Situation Semantical Analysis of Natural Language*. Ph.D. thesis, Faculty of Mechanics and Mathematics, Moscow State University (MGU), Moscow. (in Russian).

Loukanova, R., 2001. Russellian and Strawsonian Definite Descriptions in Situation Semantics. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing. CICLing 2001*, volume 2004 of *Lecture Notes in Computer Science*, pages 69–79. Springer, Berlin, Heidelberg. ISBN 978-3-540-44686-6.

Loukanova, R., 2002. Quantification and Intensionality in Situation Semantics. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing. CICLing 2002*, volume 2276 of *Lecture Notes in Computer Science*, pages 32–45. Springer, Berlin, Heidelberg. ISBN 978-3-540-45715-2.

Loukanova, R., 2011a. 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., 2011b. 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., 2011c. 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*, volume 228 of *Frontiers in Artificial Intelligence and Applications*, pages 215–236. IOS Press, Amsterdam; Berlin; Tokyo; Washington, DC. ISBN 978-1-60750-761-1.

Loukanova, R., 2012a. Algorithmic Semantics of Ambiguous Modifiers by the Type Theory of Acyclic Recursion. In *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference*, volume 3, pages 117–121. IEEE Computer Society, Los Alamitos, CA, USA.

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., 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., 2016. Relationships between Specified and Underspecified Quantification by the Theory of Acyclic Recursion. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 5(4):19–42. ISSN 2255-2863.

Loukanova, R., 2017. 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., 2019a. Computational Syntax-Semantics Interface with Type-Theory of Acyclic Recursion for Underspecified Semantics. In Osswald, R., Retoré, C., and Sutton, P., editors, *IWCS 2019 Workshop on Computing Semantics with Types, Frames and Related Structures. Proceedings of the Workshop*, pages 37–48. The Association for Computational Linguistics (ACL), Gothenburg, Sweden.

Loukanova, R., 2019b. Gamma-Reduction in Type Theory of Acyclic Recursion. *Fundamenta Informaticae*, 170(4):367–411. ISSN 0169-2968 (P) 1875-8681 (E).

Loukanova, R., 2019c. Gamma-Star Canonical Forms in the Type-Theory of Acyclic Algorithms. In van den Herik, J. and Rocha, A. P., editors, *Agents and Artificial Intelligence*, pages 383–407. Springer International Publishing, Cham.

Loukanova, R., 2020a. Algorithmic Eta-Reduction in Type-Theory of Acyclic Recursion. In Rocha, A., Steels, L., and van den Herik, J., editors, *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020)*, volume 2, pages 1003–1010. INSTICC, CITEPRESS – Science and Technology Publications, Lda. ISBN 978-989-758-395-7.

Loukanova, R., 2020b. Type-Theory of Acyclic Algorithms for Models of Consecutive Binding of Functional Neuro-Receptors. In Grabowski, A., Loukanova, R., and Schwarzweller, C., editors, *AI Aspects in Reasoning, Languages, and Computation*, volume 889, pages 1–48. Springer International Publishing, Cham. ISBN 978-3-030-41425-2.

Loukanova, R., 2021. Type-Theory of Parametric Algorithms with Restricted Computations. In Dong, Y., Herrera-Viedma, E., Matsui, K., Omatsu, S., González Briones, A., and Rodríguez González, S., editors, *Distributed Computing and Artificial Intelligence, 17th International Conference*, pages 321–331. Springer International Publishing, Cham. ISBN 978-3-030-53036-5.

Loukanova, R., to appear. Eta-Reduction in Type-Theory of Acyclic Recursion.

Ludlow, P., 2021. Descriptions. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition.

Moschovakis, Y. N., 1989. The formal language of recursion. *Journal of Symbolic Logic*, 54(4):1216–1252.

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

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

Moschovakis, Y. N., 2006. A Logical Calculus of Meaning and Synonymy. *Linguistics and Philosophy*, 29(1):27–89. ISSN 1573-0549.

Mostowski, A., 1957. On a generalization of quantifiers. *Fundamenta Mathematicae*, 1(44):12–36.

Russell, B., 1905. On Denoting. *Mind*, 14(56):479–493. ISSN 00264423, 14602113.

Strawson, P. F., 1950. On Referring. *Mind*, 59(235):320–344. ISSN 00264423, 14602113.

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