Binding Operators in Type-Theory of Algorithms for Algorithmic Binding of Functional Neuro-Receptors

Roussanka Loukanova  
Stockholm University  
Email: rloukanova@gmail.com

Abstract—This paper is on a new approach to mathematics of the notion of algorithm. We extend the higher-order, type-theory of acyclic recursion, i.e., of typed, state-dependent algorithms, which was originally introduced by Moschovakis in [1]. We introduce the concept of recursive λ-binding of argument slots across a sequence of mutually recursive assignments. The primary applications of the extended theory are to computational semantics of formal and natural languages, and to computational neuroscience. We investigate some properties of algorithmic equivalence of functions and relations that bind argument slots of other functions and relations across the recursion operator acting via mutually recursive assignments.

I. INTRODUCTION

THE IDEAS of the new approach to the mathematical notion of algorithm, by a theory of formal languages of functional recursion, were introduced by Moschovakis [2]. The initial steps for extending the approach in [2] to a typed theory \( L^\lambda_{ac} \) of acyclic algorithms, were introduced by Moschovakis in [1]. The theory \( L^\lambda_{ac} \) and its formal language, also denoted by \( L^\lambda_{ac} \), use terms formed under an acyclicity condition restricting the theory to acyclic algorithms that always terminate their calculations after finite number of steps. In addition, \( L^\lambda_{ac} \) uses currying coding of functions and relations that have multiple arguments, via sequences of unary functions and corresponding terms denoting them. The idea of such coding was initially given by Gottlob Frege. Later, Schönfinkel re-introduced it by mathematical precision. Then, Curry [3] developed the coding into a fully formalised technique, nowadays popularly named as currying. The type theory \( L^\lambda \) of algorithms with full recursion, i.e., of algorithms that are not necessarily acyclic, is under development along with \( L^\lambda_{ac} \).

The type theory \( L^\lambda \), including its sub-theory \( L^\lambda_{ac} \), extends Gallin λ-calculus and its logic \( TY_2 \) (see Gallin [4]), in various aspects. Similarly to traditional λ-calculi, \( L^\lambda \) and \( L^\lambda_{ac} \) employ function application and λ-abstraction for construction of complex terms that denote composite functions with components that can involve other functions. E.g., if \( f \) is a constant denoting a unary function, then \( \lambda(x)f(x^3) \) is a term denoting another unary function. The theories \( L^\lambda \) and \( L^\lambda_{ac} \) extend traditional λ-calculi, by adding a specialised recursion operator designated by the constant where. E.g., the formal terms (1b) and (1c) are constructed by using the constant where. The \( L^\lambda \) terms (1a)–(1c) denote the same function. The term (1c) represent the algorithm for computing the denotation of these terms stepwise. At first, the function that is the denotation of the term \( \lambda(x)f(x^3) \) is computed, e.g., as a table of argument values and corresponding function values, and saved in the memory slot \( p \). After that, the denotation of \( \lambda(x)f(p(x)) \) is computed by using the data saved in the memory slot \( p \).

\[
\begin{align*}
\lambda(x)f(x^3) & \quad \text{(1a)} \\
\lambda(x)[f(p)] & \quad \{p := x^3\} \quad \text{(1b)} \\
\lambda(x)f(p(x)) & \quad \{p := \lambda(x)(x^3)\} \quad \text{(1c)}
\end{align*}
\]

In this way, \( L^\lambda \) and \( L^\lambda_{ac} \) extend the expressive power of λ-calculus. Actually, \( L^\lambda \) is a mathematical theory of the notion of algorithm, which is equivalent to modelling the notion of algorithm, e.g., by Turing machines. The sub-theory \( L^\lambda_{ac} \) models acyclic algorithms, i.e., computations that always end after a finite number of steps. Importantly, this is achieved by the recursion operator where, at the object level of \( L^\lambda \) for modelling algorithmic computations. The formal theories \( L^\lambda \) and \( L^\lambda_{ac} \) have reduction calculi, in various versions. By using the standard reduction calculus of \( L^\lambda \) and \( L^\lambda_{ac} \), the term (1a) can be reduced to (1c) (and even to a more basic term).

The type theory \( L^\lambda \) represents crucial semantic distinctions in formal and natural languages. We have demonstrated that \( L^\lambda_{ac} \) has major applications to computational semantics and computational syntax-semantics interfaces of human language. The work in this paper is on development of the mathematics of the notion of algorithms by targeting broad applications to Artificial Intelligence and Robotics. In Section II, we give an overview of related work on type-theory of situated algorithms and situated information. Primary applications of \( L^\lambda_{ac} \) have been achieved for computational semantics and computational syntax-semantics interfaces of human language. Development of computational syntax-semantics interfaces, by using \( L^\lambda_{ac} \), offers significant steps forward to computational representation of context-dependency and ambiguities in human language. In particular, recursion terms with free recursion variables, i.e., memory variables, represent parametric information and parametric algorithms.

This paper is on theoretical development of \( L^\lambda \) and \( L^\lambda_{ac} \). Section IV presents the syntax of an extended version \( L^\lambda_{aa} \) of \( L^\lambda_{ac} \), which has terms with components for restrictions. In the major Section V, we focus on some properties of generalised binding operators in the type theory of acyclic recursion.
We point out that the results presented in Section V, about the binding operators, hold for the language $L_{\lambda_{\mathcal{R}}}^\tau$, too. We target applications to computational neuroscience for modelling computational power of neural networks, e.g., as described in Section VI.

II. RELATED WORK

By providing the technical notion of binding acros recursive assignments, this paper is directly related to and extends the work in Loukanova [5]. For some more explanations, see the beginning of Section IV.

Terms with restrictions, as in Section V, were originally introduced for the first time in Loukanova [6]. That work is on the formalisation of major notions of algorithmic granularity and algorithmic underspecification defined inherently, at the object level of the languages of the typed theory of recursion $L^\lambda_\mathcal{R}$ and $L^\lambda_\mathcal{R}_\mathcal{R}$. Closely related to the work here, the paper [6] introduces two kinds of constraints on possible specifications of underspecified recursion variables by: (1) general acyclicity constraints, and (2) constraints that arise from specific applications. The theory of acyclic recursion is employed to represent semantic ambiguities in human language, which can not be resolved when only partial knowledge is available, even in specific contexts, with specific speakers and their references. The work in [6] takes the direction of formalisation of the notion of algorithmic underspecification carrying constraints, and fine-granularity specifications via syntax-semantics interfaces. For more details on representation of underspecification in semantics of human language, by using the type theory of acyclic recursion $L^\lambda_\mathcal{R}$, see Loukanova [7], [8], [9], [5].

The idea of generalised, restricted parameters were originally, for the first time, introduced by Barwise and Perry [10]. An early, more precise mathematical introduction of restricted parameters was given by Loukanova and Cooper [11], and then by Loukanova [12], [13], [14], [15]. Restricted parameters, as semantic objects, in relational semantic domains of mathematical structures, were presented more officially, i.e., mathematically, in Loukanova [16]. The first introduction of formal language of restricted parameters is given by Loukanova [17], which introduces a higher-order, type-theoretical formal language of information content that is partial, parametric, underspecified, dependent on situations, and recursive. The formal system is extended by Loukanova [18]. While the formal syntax of that language is relational and semantically designates relational semantic structures, it is the first, original formalisation of the semantic concept of generalised, restricted parameters and parametric networks. The terms of that formal language represent situation-theoretic objects. The language has specialised terms for constrained computations by mutual recursion. It introduces terms representing nets of parameters that are simultaneously constrained to satisfy restrictions. The restricted terms presented here in Section V are close in their formal structure to corresponding terms in the formal languages in [17], [18]. In this paper, we limit the formal language and theory to functional structures of typed functions, via Curry coding, see Curry [3].

III. OVERVIEW OF THE TYPE-THEORY OF ACYCLIC RECURSION

Here we give a brief overview of $L^\lambda_\mathcal{R}$ to facilitate the exposition in the rest of the paper. For details, see Moschovakis [1]. and Loukanova [5], [19].

A. Syntax of $L^\lambda_\mathcal{R}$

a) The set $\text{Types}^{\lambda}_{\mathcal{R}}$ of $L^\lambda_\mathcal{R}$ is the smallest set defined recursively by the following rules in Backus-Naur form (BNF):

$$\tau ::= e \mid t \mid s \mid (\tau_1 \to \tau_2) \quad (2)$$

The type $e$ is for primitive objects that are entities of the semantic domains, as well as for the terms of $L^\lambda_\mathcal{R}$ denoting such entities. The type $s$ is for states consisting of context information, e.g., possible worlds (situations), time and space locations, speakers, listeners; $t$ is the type of the truth values. The type $(\tau_1 \to \tau_2)$ is for functions from objects of type $\tau_1$ to objects of type $\tau_2$. The type (3) is for functions on $n$-arguments of corresponding types $\tau_1, \ldots, \tau_n$ that take values of type $\tau$, by currying coding.

$$\left(\tau_1 \to \cdots \to (\tau_n \to \tau)\right) \quad \sigma, \tau_i \in \text{Types}, \ n \geq 0 \quad (3)$$

The formal language $L^\lambda_\mathcal{R}$ has typed vocabulary. For each type $\tau \in \text{Types}$:

Constants $K$: denumerable set of typed constants

$$K_{\tau} = \{c^\tau_0, \ldots, c^\tau_k, \ldots\} \quad (4a)$$

$$K = \bigcup_{\tau} K_{\tau} \quad (4b)$$

Pure variables $PV$: denumerable set of typed pure variables

$$PV_{\tau} = \{v_0, v_1, \ldots\} \quad (5a)$$

$$PV = \bigcup_{\tau} PV_{\tau} \quad (5b)$$

Reursion (memory) variables $RV$: denumerable set of typed recursion (memory) variables

$$RV_{\tau} = \{r_0, r_1, \ldots\} \quad (6a)$$

$$RV = \bigcup_{\tau} RV_{\tau} \quad (6b)$$

Variables:

$$\vars_{\tau} = PV_{\tau} \cup RV_{\tau} \quad (7a)$$

$$\vars = PV \cup RV \quad (7b)$$

In addition to the terms the typical $\lambda$-calculi, the language $L^\lambda_\mathcal{R}$ has new ones formed by using the facility of the recursion, i.e., memory, variables and a new operator for term construction, which we call recursion operator, designated by the operator constant where, in infix notation.

The recursive rules for generating the set of $L^\lambda_\mathcal{R}$-terms are given in (8a)–(8e), by using the extended, typed Backus-Naur (TBNF) form, with the assumed types given as superscripts. We also use the typical notation for type assignments: $A : \tau$, to express that $A$ is a term of type $\tau$. 
Definition 1. The set Terms_{LB,T} of the terms of $L^T_{\alpha}$ consists of the expressions generated by the following rules, in Typed Backus-Naur Form (TBNF):

\[
A ::= e^T : \tau \quad \quad (8a)
\]

\[
x^T : \tau \quad \quad (8b)
\]

\[
B^{(\sigma_1 \to \tau)}(C^\sigma) : \tau \quad \quad (8c)
\]

\[
\lambda(v^\sigma)(B^T) : (\sigma \to \tau) \quad \quad (8d)
\]

\[
A_n^T \text{ where } \{ p_1^{a_1} := A_1^{a_1}, \ldots, p_n^{a_n} := A_n^{a_n} \} : \sigma \quad (8e)
\]

where $A_1 : \sigma_1$, $\ldots$, $A_n : \sigma_n$ are in Terms; $p_1 : \sigma_1$, $\ldots$, $p_n : \sigma_n$ ($n \geq 0$) are pairwise different recursion variables of the types of the assigned terms, such that the sequence of assignments $\{ p_1^{a_1} := A_1^{a_1}, \ldots, p_n^{a_n} := A_n^{a_n} \}$ satisfies the following Acyclicity Constraint (AC):

Acyclicity Constraint (AC): the sequence of assignments $\{ p_1 := A_1, \ldots, p_n := A_n \}$ is acyclic iff there is a function rank : $\{p_1, \ldots, p_n\} \rightarrow \mathbb{N}$ such that, for all $p_i, p_j \in \{p_1, \ldots, p_n\}$,

\[
\text{if } p_j \text{ occurs freely in } A_i \text{ then } \text{rank}(p_j) < \text{rank}(p_i) \quad (9)
\]

Types$_{\alpha}$ is the set of the types of $\tau$. For each $\tau \in$ TYPE. We call the terms of the form (10) recursion terms, or alternatively where-terms:

\[
[A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \}] \quad (10)
\]

We say that a term $A$ is explicit if the constant where does not occur in it.

Notation 1. We shall use the abbreviation (11a) for state-dependent types sigma, and (11b) for state-dependent truth values:

\[
\overline{\sigma} \equiv s \Rightarrow \sigma \quad (11a)
\]

\[
\overline{t} \equiv s \Rightarrow t \quad (11b)
\]

We may use the following abbreviations and similar variants:

Notation 2.

\[
\overline{p} := \overline{A} \equiv p_1 := A_1, \ldots, p_n := A_n \quad (n \geq 0) \quad (12a)
\]

Notation 3.

\[
H(\overline{x}) \equiv H(x_1) \ldots (x_n) \quad (13)
\]

\[
\lambda(\overline{v}) \equiv \lambda(v_{j_1}, \ldots, v_{j_l}) \equiv \lambda(v_{j_1}) \ldots \lambda(v_{j_l}) \quad (14)
\]

We use the typical notation $\mathbb{N}$ of the set of the natural numbers.

Definition 2 (Immediate terms). The set of the immediate terms, which we denote by ImT, is defined as follows:

\[
\text{ImT}^T \equiv X \mid Y(v_1) \ldots (v_m) \quad (15a)
\]

\[
\text{ImT}(\sigma_1 \to \ldots (\sigma_n \to \tau)) \equiv \lambda(u_1) \ldots \lambda(u_n)Y(v_1) \ldots (v_m) \quad (15b)
\]

where $n \geq 0$, $m \geq 0$; $u_i \in PV_{\sigma_i}$, for $i = 1, \ldots, n$; $v_j \in PV_{\tau_j}$, for $j = 1, \ldots, m$; $X \in PV$, $Y \in RV_{(\tau_1 \to \ldots (\tau_n \to \tau))}$.

Definition 4 (Proper terms). A term $A$ is proper if it is not immediate, e.g. the set PrT of the proper terms of $L^T_{\alpha}$ consists of all terms that are not in ImT:

\[
\text{PrT} = (\text{Terms} \ \text{ImT}) \quad (16)
\]

B. Reduction Calculus

a) Reduction Rules:

Congruence: If $A \equiv_c B$, then $A \Rightarrow B$ (con)

Transitivity: If $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$ (t)

Compositionality:

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

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

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

\[
A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \} \Rightarrow B_0 \text{ where } \{ p_1 := B_1, \ldots, p_n := B_n \} \quad (c-r)
\]

Head rule:

\[
(A_0 \text{ where } \{ \overline{p} := \overline{A} \}) \Rightarrow (A_0 \text{ where } \{ \overline{p} := \overline{B} \}) \quad (h)
\]

Bekić-Scott rule:

\[
A_0 \text{ where } \{ p := (B_0 \text{ where } \{ \overline{q} := \overline{B} \}) \}, \overline{p} := \overline{A} \}
\]

\[
\Rightarrow A_0 \text{ where } \{ p := B_0, \overline{q} := \overline{B}, \overline{p} := \overline{A} \}
\]

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

Recursion-application rule:

\[
(A_0 \text{ where } \{ \overline{p} := \overline{A} \})(B) \Rightarrow A_0(B) \text{ where } \{ \overline{p} := \overline{A} \}
\]

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

Application rule:

\[
A(B) \Rightarrow A(p) \text{ where } \{ p := B \}
\]

given that $B$ is a proper term and $p$ is a fresh recursion variable $\lambda$-rule:

\[
\lambda(u)(A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \}) \Rightarrow \lambda(u)(A_0) \text{ where } \{ p'_1 := \lambda(u)A'_1, \ldots \}
\]

\[
p'_n := \lambda(u)A'_n \}
\]

where for all $i = 1, \ldots, n$, $p'_i$ is a fresh recursion variable and $A'_i$ is the result of the replacement of the free occurrences of $p_1, \ldots, p_n$ in $A_i$ with $p'_1(u), \ldots, p'_n(u)$, respectively, i.e.:

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

for all $i \in \{1, \ldots, n\}$.
Definition 5 (Reduction Relation). The reduction relation in $L_{ar}$ is the smallest relation, denoted by $\Rightarrow$, between terms that is closed under the reduction rules.

Definition 6 (Term Irreducibility). We say that a term $A \in \text{Terms}_e$ is irreducible if and only if

$$for \ all \ B \in \text{Terms}, \ if \ A \Rightarrow B, \ then \ A \equiv_c B \quad (21)$$

Here we shall present some of the major results that are essential for algorithmic semantics and which have direct relevance to this paper.

Theorem 1 (Canonical Form Theorem: existence and uniqueness of the canonical forms). (See Moschovakis [1], § 3.1.) For each term $A$, there is a unique, up to congruence, irreducible term $C$, denoted by $cf(A)$ and called the canonical form of $A$, such that:

1) $cf(A) \equiv A_0$ where $\{ p_1 := A_1, \ldots, p_n := A_n \}$, for some explicit, irreducible terms $A_1, \ldots, A_n$ ($n \geq 0$)

2) $A \Rightarrow cf(A)$

3) if $A \Rightarrow B$ and $B$ is irreducible, then $B \equiv_c cf(A)$, i.e., $cf(A)$ is the unique, up to congruence, irreducible term to which $A$ can be reduced.

Theorem 2 (Referential Synonymy Theorem). (For the original theorem, see Moschovakis [1]) Two terms $A$ and $B$ are algorithmically synonymous, i.e., algorithmically equivalent, $A \approx B$, if and only if there are explicit, irreducible terms of corresponding types: $A_0 : \sigma_0$, $A_n : \sigma_n$, and $B_0 : \sigma_0, \ldots, B_n : \sigma_n$ ($n \geq 0$), such that:

$$A^\sigma \Rightarrow cf \ A_0^\sigma \quad (22a)$$

$$B^\sigma \Rightarrow cf \ B_0^\sigma \quad (22b)$$

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

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

Thus, $A$ and $B$ are algorithmically synonymous, $A \approx B$, if and only if

1) either $A$ and $B$ are proper terms that have the same denotations computed by the same algorithm

2) or $A$ and $B$ are immediate and have the same denotations

When $A \approx B$, we also say that $A$ and $B$ are referentially synonymous, in case we refer to the algorithms they designate.

Theorem 3 (Compositionality Theorem for algorithmic synonymy). For all $A \in \text{Terms}_e$, $B, C \in \text{Terms}_e$, $x \in PV_e$, such that the substitutions $A\{x \equiv B\}$, and $A\{x \equiv C\}$ are free, i.e., do not cause variable collisions:

$$B \approx C \quad \Rightarrow \quad A\{x \equiv B\} \approx A\{x \equiv C\} \quad (24)$$

Proof. See Moschovakis [1], § 3.22.

Corollary 1. For all explicit, irreducible terms $A : \sigma$ and $B : \sigma$.

$$A \approx B \quad \text{iff} \quad \text{den}(A)(g) = \text{den}(B)(g), \ \text{for all} \ g \in G \quad (25)$$

Corollary 2. For every explicit, irreducible term $A : (\sigma \rightarrow \tau)$ and $x \in PV_e, x : \sigma$, such that $x$ does not occur in $A$:

$$\lambda(x)(A(x)) \approx A \quad (26)$$

IV. SEQUENTIAL BINDERS

Loukanova [5] renders sentences of human language, which contain several quantifiers with multiple scope interpretations, into underspecified $L_{ar}$ terms. These terms contain quantifier expressions $Q_i$, e.g., for $i = 1, 2, 3$, that can have multiple scope distributions over a joint core relation $h$, depending on context. The common characteristics of such terms is that regardless of the specified scope distribution of the quantifier subterms $Q_i$, each $Q_i$ binds a fixed argument slot of $h$, i.e., $i$-th argument of $h$.

Recursion terms in canonical forms provide a very sophisticated and elegant representation of scope distributions. They display all the common factors across multiple scope distributions corresponding to a given sentence $A$ with several quantifiers. By factoring out the differences, the canonical forms of the $L_{ar}$ terms representing different scopes give a common underspecified term that represents the set of all scope distributions for $A$. Such a term has free recursion variables that can be instantiated to specific scope distributions. The technique is based on formal linking of each of the quantifiers $Q_i$ with the corresponding $i$-th argument slot of $h$ that it binds, across $\lambda$-abstractions, recursion assignments, and reduction steps.

The details of the formalisation of linking the quantifiers to the respective argument slots that they bind across recursive assignments are left open in [5].

The rest of this paper elaborates the formalisation of bind concepts for a broad class of terms that bind argument slots. The class of these terms includes terms denoting quantifiers and other binding relations and functions.

For sake of rigour and clarity, in Theorem 4, we provide detailed assumptions, the formal types (33a)–(33h), and extra
subterms in (32a)-(32e) and (34b)-(34i). These details are important for the proof. They can be ignored for understanding the essence of the theorem. Similarly, we provide such details in other theorems and results presented in this paper.

**Theorem 4** (Reduction of Strong Binders 4). Let $T_{m+1}$ be the term (32a)-(32e):

$$T_{m+1} \equiv Q_{m+1} \left[ \lambda x_{i+1} Q_{i+1} \left[ \lambda x_{i} Q_{i} \left[ \lambda x_{i-1} Q_{i-1} \left[ \ldots \lambda x_{1} Q_{1} \left[ \lambda x_{1} H(x_{1}) \ldots (x_{n}) \right] \right] \right] \right] \right]$$

where we assume that:

- $m, n \in \mathbb{N}, m, n \geq 1$
- $x_{1}, \ldots, x_{n} \in PV$ are pure variables of types $\sigma_{i}$, i.e., $(x_{i} : \sigma_{i})$, for $\sigma_{i}$ in Types, $i = 1, \ldots, n$
- $x_{1}, \ldots, x_{n} \in PV$ are pure variables of types $\sigma_{i}$, i.e., $(x_{i} : \sigma_{i})$, for $\sigma_{i}$ in Types, $i = 1, \ldots, m$
- $Q_{1}, \ldots, Q_{m}$ are terms of the corresponding types

$$H : (\sigma_{1} \rightarrow \cdots \rightarrow (\sigma_{n-1} \rightarrow (\sigma_{n} \rightarrow \sigma)))$$

The types are such that $T_{m+1}$ in (32a)-(32e) is a well-formed term.

In addition, assume the following:

1. $H$ is a proper, i.e., not immediate, term
2. $m \leq n$
3. $x_{1}, \ldots, x_{n} \in PV$ are pairwise different, pure variables
4. $x_{1}, \ldots, x_{n} \in PV$ are pairwise different, pure variables
5. $\{x_{1}, \ldots, x_{n}\} \subseteq \{x_{1}, \ldots, x_{n}\}$, i.e., $\{x_{i}, \ldots, x_{i}\} \subseteq \{1, \ldots, n\}$

Then, the term $T_{m+1}$ in (32a)-(32e) can be reduced to the term $R_{m+1}$ in (34b)-(34i):

$$T_{m+1} \Rightarrow R_{m+1} \equiv$$

$$Q_{m} \left[ R_{m}^{0} \right] \quad \text{where} \quad$$

$$R_{m}^{0} := \lambda(x_{m})Q_{i(m-1)} \left[ R_{i(m-1)}^{1} (x_{i(m-2)}) \right], \quad (34a)$$

$$R_{i(m-1)}^{1} := \lambda(x_{i(m-1)}) \lambda(x_{i(m-2)}) \left[ R_{i(m-2)}^{2} (x_{i(m-1)}) \right], \quad (34b)$$

$$R_{i(m-2)}^{2} := \lambda(x_{i(m-1)}) \lambda(x_{i(m-2)})Q_{i(m-1)} \left[ R_{i(m-1)}^{3} (x_{i(m-2)}) \right], \quad (34c)$$

$$\ldots$$

$$R_{i(j+1)}^{m-j} := \lambda(x_{i(j+1)})Q_{j+1} \left[ R_{i(j+1)}^{m-j-1} (x_{j+1}) \right], \quad (34d)$$

$$R_{i(j+1)}^{m-j} := \lambda(x_{i(j+1)}) \lambda(x_{i(j+1)}Q_{j+1} \left[ R_{i(j+1)}^{m-j-1} (x_{j+1}) \right], \quad (34e)$$

for some fresh recursion variables $R_{k}^{0} \in RV$ of the types

$$R_{i(j+1)}^{m-j} := \lambda(x_{i(j+1)})Q_{j+1} \left[ R_{i(j+1)}^{m-j-1} (x_{j+1}) \right], \quad (34f)$$

Proof. The proof is by induction on the number of the terms $Q_{1}, \ldots, Q_{m}$. It uses the reduction rules of $L_{\sigma}^{\lambda}$ and verifies the types.

The superscripts of the variables $R_{k}^{m-j}$ are counters of the number of the applications of the $\lambda$-rule ($\lambda$).

Note 1. The types of the terms $Q_{1}, \ldots, Q_{m}$, $H$ do not need to be such that $Q_{1}, \ldots, Q_{m}$ can denote quantifiers over arguments of a range denoted by $H$, which are in the focus of the work in Loukanoi [5]. In this paper, we investigate the broader class of terms $Q_{1}, \ldots, Q_{m}$, $H$, such that $Q_{1}, \ldots, Q_{m}$ can bind argument slots of the term $H$.

Note 2. The requirements (4)-(5) in Theorem 4 guarantee that there is binding of existing argument slots of $H$. I.e., the bindings in the term $T_{m+1}$ in (32a)-(32e) are strong, not vacuous. Therefore, the chained bindings by $R_{m+1}$ in (34b)-(34i) are strong too. The term $H$ may still denote a function that does not depend essentially on some of its arguments, including such that are bound by some $Q_{j}$. The requirements (4)-(5) in Theorem 4 can be removed in a general term $T_{m+1}$ of the same form (32a)-(32e), while the reduction to the term $R_{m+1}$ in (34b)-(34i) holds.
Theorem 5. The term $R_{m+1}$ in (34b)–(34i) is algorithmically synonymous (equivalent) with the term $R'_{m+1}$ in (36b)–(36i). 

Thus, the terms $R_{m+1}$ and $R'_{m+1}$ are algorithmically equivalent. 

Definition 7 (Recursive Distance). Let $T$ be the a term of the form: 

$$T \equiv A_0 \text{ where } \{ p_0 : A_n, \ldots, p_i : A_{i+1}, \ldots, p_{t+1} : A_{t+1}, p_t : A_t, \}$$

The recursive distance $\text{Rdist}(p_s, H, A_1) = Rdist(p_s, H, p_t) = Rdist(A_n, H, A_1) = Rdist(A_n, H, p_t)$ of $A_s$, (or its $p_t$), from a subterm $H$ of a term $A_1$ (or its recursion memory $p_t$), in a recursion term $T$ of the form (41a)–(41c) (modulo congruence with respect to the order of the assignments), is defined by induction:

$$\text{Rdist}(p_i, H, A_1) = Rdist(A_i, H, A_1)$$

$$= Rdist(p_i, H, p_t) = 0,$$

if $H$ occurs in $A_i$

$$\text{Rdist}(p_{i+1}, H, A_1) = Rdist(A_{i+1}, H, A_1)$$

$$= Rdist(p_{i+1}, H, p_t) = Rdist(A_{i+1}, H, p_t)$$

$$= \min \{ \text{Rdist}(p_i, H, p_t) \mid p_i \text{ occurs in } A_{i+1} \} + 1,$$

Note: $\text{Rdist}(p_n, H, A_1)$ is a partial function.

Theorem 6 (Binding Across Recursion 6). Let $R_{m+1}$ be a term of the form (43b)–(43h), as in Theorem 5.

$$R_{m+1} \equiv Q_{i_{m+1}}[\lambda(x_{i_{m+1}})R_0^0(x_{i_{m+1}})] \text{ where }$$

$$R_0^0 := \lambda(x_{i_{m+1}})Q_{i_{m+1}}[x_{i_{m+1}}],$$

$$R_{i_{m+1}}^1 := \lambda(x_{i_{m+1}})\lambda(x_{i_{m+1}})Q_{i_{m+1}}[x_{i_{m+1}}],$$

etc. Then $R_{m+1}$ and $R'_{m+1}$ are algorithmically equivalent.
Then, for every \( i, j \in \{1, \ldots, m\} \), (1) the term \( Q_{ij} \) binds the \( i, j \)-th argument slot of \( H \), and (2) \( R_{ij}^{m-j} \) is \( \lambda \)-abstraction \( \lambda(x_{i1}) \cdots \lambda(x_{ij+1}) \lambda(x_{ij}) \) over the \( i, m \)-th, \( i, j \)-th argument slots of \( H \) in this specific order.

Proof. The proof is by induction on \( j = 1, \ldots, m \), i.e., on the recursive distance of \( R_{ij}^{m-j} \) from \( H \) in \( R_{ij}^{m-1} \).

**Theorem 7** (Binding Across Recursion 7). Let \( R_{m+1} \) be a term of the form (44b)-(44h), as in Theorem 4:

\[
R_{m+1} = \begin{cases} 
Q_{im}[R_{im}^0] & \text{where } \\
R_{im}^0 := \lambda(x_{im})Q_{(m-1)}[R_{(m-1)-j}^0(x_{im})] & \\
R_{im}^1 := \lambda(x_{im})\lambda(x_{(m-1)})Q_{(m-2)}[R_{(m-2)-j}^2(x_{im})], & \text{for } j = 1, \ldots, 2, \\
R_{im}^{m-j+1} := \lambda(x_{im})\ldots\lambda(x_{ij+1})Q_{ij} & \\
R_{im}^{m-j} := \lambda(x_{im})\ldots\lambda(x_{ij+1})\lambda(x_{ij})Q_{ij-1} & \\
R_{im}^{m-j-1}(x_{im}) \ldots (x_{ij})] & \\
\text{for } j = m, \ldots, 1, \\
R_{im}^{m-2} := \lambda(x_{im})\ldots\lambda(x_{ij})Q_{ij} & \\
R_{im}^{m-3}(x_{im}) \ldots (x_{ij})] & \\
R_{im}^{m-1} := \lambda(x_{im})\ldots\lambda(x_{ij})H(\tilde{x}) & \text{(44i)}.
\end{cases}
\]

Then, for every \( i, j \in \{1, \ldots, m\} \), (1) \( Q_{ij} \) binds the \( i, j \)-th argument slot of \( H \), and (2) \( R_{ij}^{m-j} \) is \( \lambda \)-abstraction \( \lambda(x_{i1}) \cdots \lambda(x_{ij+1}) \lambda(x_{ij}) \) over the \( i, m \)-th, \( i, j \)-th argument slots of \( H \) in this specific order.

Proof. The proof is by induction on \( j = 1, \ldots, m \), i.e., on the recursive distance of \( R_{ij}^{m-j} \) from \( H \) in \( R_{ij}^{m-1} \).

Now, we show that Theorem 4 holds by weakening the requirement (3).

**Theorem 8** (Reduction of Strong Binders 8). Let \( T_{m+1} \) be the term (45a)-(45e):

\[
T_{m+1} = \begin{cases} 
Q_{im}[R_{im}^0] & \text{where } \\
Q_{im} := \lambda(x_{im})Q_{(m-1)}[R_{(m-1)-j}^1(x_{im})] & \\
\lambda x_{(m-2)} Q_{(m-3)}[\lambda x_{(m-3)} Q_{(m-4)}] & \\
\cdots & \\
\lambda x_{(j+1)} Q_{ij} \lambda x_{ij} Q_{ij-1}[\lambda x_{ij-1} Q_{ij-2}] & \\
\cdots & \\
\lambda x_{ij} Q_{ij} & \text{(45d)}
\end{cases}
\]

where we assume that:

- \( m, n \in \mathbb{N}, m, n \geq 1 \)
- \( x_{i1}, \ldots, x_{im} \in PV \) are pure variables of types \( (x_{ij} : \sigma_{ij}) \), for \( \sigma_{ij} \in \text{Types}, i, j \in \mathbb{N}, j = 1, \ldots, m \)
- \( Q_{ij}, Q_{ij-1}, Q_{ij-2}, H \) are terms of the corresponding types in (33a)-(33h):

In addition, assume the following:

- \( H \) is a proper, i.e., not immediate, term \( (2 \ m \leq n) \)
- \( v_1, \ldots, v_k \in PV \) are pure variables, not necessarily pairwise different.
- \( x_{i1}, \ldots, x_{im} \in PV \) are pairwise different, pure variables.
- \( \{ x_{i1}, \ldots, x_{im} \} \subseteq \{ v_1, \ldots, v_k \} \), i.e., \( x_{i1}, \ldots, x_{im} \subseteq \{1, \ldots, n\} \)

Then, the term \( T_{m+1} \) in (45a)-(45e) can be reduced to the term \( R_{m+1} \) in (46a)-(46i):

\[
T_{m+1} \Rightarrow S_{m+1} \equiv R_{m+1} \equiv (46a) \\
Q_{im}[R_{im}^0] & \text{where } \\
R_{im}^0 := \lambda(x_{im})Q_{(m-1)}[R_{(m-1)-j}^1(x_{im})] & \text{(46b)} \\
R_{im}^1 := \lambda(x_{im})\lambda(x_{(m-1)})Q_{(m-2)}[R_{(m-2)-j}^2(x_{im})] & \text{(46c)} \\
R_{im}^{m-j} := \lambda(x_{im})\ldots\lambda(x_{ij+1})\lambda(x_{ij})Q_{ij-1} & \\
R_{im}^{m-j-1}(x_{im}) \ldots (x_{ij})] & \text{(46d)} \\
R_{im}^{m-2} := \lambda(x_{im})\ldots\lambda(x_{ij})Q_{ij} & \\
R_{im}^{m-3}(x_{im}) \ldots (x_{ij})] & \text{(46e)} \\
R_{im}^{m-1} := \lambda(x_{im})\ldots\lambda(x_{ij})H(\tilde{x}) & \text{(46f)}
\]

where we assume that:

- \( m, n \in \mathbb{N}, m, n \geq 1 \)
- \( x_{i1}, \ldots, x_{im} \in PV \) are pure variables of types \( (x_{ij} : \sigma_{ij}) \), for \( \sigma_{ij} \in \text{Types}, i, j \in \mathbb{N}, j = 1, \ldots, m \)
- \( Q_{ij}, \ldots, Q_{im}, H \) are terms of the corresponding types in (33a)-(33h):

\[
T_{m+1} \equiv Q_{im}[R_{im}^0] \Rightarrow S_{m+1} \equiv R_{m+1} \equiv (46a) \\
Q_{im} := \lambda(x_{im})Q_{(m-1)}[R_{(m-1)-j}^1(x_{im})] & \text{(46b)} \\
\lambda x_{(m-2)} Q_{(m-3)}[\lambda x_{(m-3)} Q_{(m-4)}] & \text{(46c)} \\
\cdots & \\
\lambda x_{(j+1)} Q_{ij} \lambda x_{ij} Q_{ij-1}[\lambda x_{ij-1} Q_{ij-2}] & \text{(46d)} \\
\cdots & \\
\lambda x_{ij} Q_{ij} & \text{(46e)}
\]

where we assume that:

- \( m, n \in \mathbb{N}, m, n \geq 1 \)
- \( x_{i1}, \ldots, x_{im} \in PV \) are pure variables of types \( (x_{ij} : \sigma_{ij}) \), for \( \sigma_{ij} \in \text{Types}, i, j \in \mathbb{N}, j = 1, \ldots, m \)
- \( Q_{ij}, \ldots, Q_{im}, H \) are terms of the corresponding types in (33a)-(33h):

\[
T_{m+1} \equiv Q_{im}[R_{im}^0] \Rightarrow S_{m+1} \equiv R_{m+1} \equiv (46a) \\
Q_{im} := \lambda(x_{im})Q_{(m-1)}[R_{(m-1)-j}^1(x_{im})] & \text{(46b)} \\
\lambda x_{(m-2)} Q_{(m-3)}[\lambda x_{(m-3)} Q_{(m-4)}] & \text{(46c)} \\
\cdots & \\
\lambda x_{(j+1)} Q_{ij} \lambda x_{ij} Q_{ij-1}[\lambda x_{ij-1} Q_{ij-2}] & \text{(46d)} \\
\cdots & \\
\lambda x_{ij} Q_{ij} & \text{(46e)}
\]

Proof. The proof is similar to that of Theorem 4.

Similarly, Theorem 5 holds by weakening the requirement (3), but we do not present it here for sake of space.

**Theorem 9** (Reduction of Strong Binders 9). Let \( T_{m+1} \) be the term (47a)-(47e):

\[
T_{m+1} \equiv Q_{im}[R_{im}^0] \rightarrow S_{m+1} \equiv R_{m+1} \equiv (47a) \\
\lambda x_{(m-2)} Q_{(m-3)}[\lambda x_{(m-3)} Q_{(m-4)}] & \text{(47b)} \\
\cdots & \\
\lambda x_{(j+1)} Q_{ij} \lambda x_{ij} Q_{ij-1}[\lambda x_{ij-1} Q_{ij-2}] & \text{(47c)} \\
\cdots & \\
\lambda x_{ij} Q_{ij} & \text{(47d)}
\]

The difference from Theorem 8 is that now \( H \) is immediate, i.e., not proper.
\[ \lambda x_{t(m-2)}Q_{t(m-3)}[\lambda x_{t(m-3)}Q_{t(m-4)}[ \ldots \lambda x_{t(j+1)}Q_{t(j)}[\lambda x_{t(j)}Q_{t(j-1)}[\lambda x_{t(j-1)}Q_{t(j-2)}[ \ldots \lambda x_{t(1)}Q_{t(0)}[ \lambda x_{1}H(v_1) \ldots (v_k)]]]]] \]  

where we assume that:

- \( m, n \in \mathbb{N}, m, n \geq 1 \)
- \( x_1, \ldots, x_m \in PV \) are pure variables of types \( (x_i : \sigma_i) \), for \( \sigma_i \in \text{Types} \), \( i \in \mathbb{N}, j = 1, \ldots, m \)
- \( Q_1, \ldots, Q_m, H \) are terms of the corresponding types in (33a)–(33b).

In addition, assume the following:

1. \( H \) is an immediate term
2. \( m \leq n \)
3. \( v_1, \ldots, v_k \in PV \) are pure variables, not necessarily pairwise different.
4. \( x_1, \ldots, x_m \in PV \) are pairwise different, pure variables
5. \( \{x_1, \ldots, x_m\} \subseteq \{v_1, \ldots, v_k\} \), i.e., \( \{x_1, \ldots, x_m\} \subseteq \{1, \ldots, n\} \)

Then, the term \( T_{m+1} \) in (47a)–(47c) can be reduced to the term \( R_{m+1} \) in (48a)–(48h):

\[ T_{m+1} \Rightarrow R_{m+1} \]

\[ Q_{t(m)}[R_{t(m)}] \quad \text{where} \]

\[ R_{t(0)} := \lambda x_{t(0)}Q_{t(0)}[R_{t(1)}(x_{t(0)})], \]

\[ R_{t(1)} := \lambda x_{t(1)}\lambda x_{t(1)}Q_{t(0)}[R_{t(2)}(x_{t(1)})], \]

\[ R_{t(2)} := \lambda x_{t(2)}\lambda x_{t(2)}Q_{t(1)}[R_{t(3)}(x_{t(2)})], \]

\[ \ldots \]

\[ R_{t(j+1)} := \lambda x_{t(j+1)} \ldots \lambda x_{t(j+1)}Q_{t(j)}[R_{t(j+2)}(x_{t(j+1)})], \]

\[ R_{t(j)} := \lambda x_{t(j)} \ldots \lambda x_{t(j)}Q_{t(j-1)}[R_{t(j+1)}(x_{t(j)})], \]

\[ \quad | \text{for } j = m, \ldots, 1, \]

\[ R_{t(1)} := \lambda x_{t(1)} \ldots \lambda x_{t(1)}Q_{t(0)}[R_{t(2)}(x_{t(1)})], \]

\[ \quad \text{for some fresh recursion variables } R_{t(k)} \in RV \text{ of the types (35a)–(35f)}. \]

**Proof.** The proof is similar to that of Theorem 8. \( \square \)

A result as in Theorem 5 corresponding to Theorem 9 can be obtained for any immediate term \( H \), but we do not present it here for sake of space. Theorem 4, 8, 9 still hold after removing the requirements (4)–(5), by allowing vacuous binding.

### V. Algorithmic Computations with Restrictions

In this section, we give a very brief introduction of an extension of the formal language \( \mathcal{L}^{\lambda}_{\text{RaA}} \) to a formal language \( \mathcal{L}^{\lambda}_{\text{RaAA}} \)

of Type Theory of Restricted Acyclic Terms (TToRaAA).

If the acyclicity constraint AC (1) is dropped out, \( \mathcal{L}^{\lambda}_{\text{RaA}} \) is extended to a language \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) of typed, restricted algorithms with full recursion. The language and theory \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) is a mathematical model of the notion of algorithm, while \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) models the notion of acyclic algorithm.

In this paper, we focus on giving the syntax of \( \mathcal{L}^{\lambda}_{\text{RaAA}} \), with some intuitive explanations, for an introduction to potential applications of the results in the previous section to model neural connections as a result of binding functionality of receptors.

The denotational and algorithmic semantics of \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) (\( \mathcal{L}^{\lambda}_{\text{RaA}} \)), the reduction calculus, and various theoretical results are subject of other, forthcoming work. The language \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) is the result of extending \( \mathcal{L}^{\lambda}_{\text{RaA}} \) by adding a constant such that and corresponding terms having a component for restrictions, i.e., constraints. Such terms with restrictions designate a restricter over computations as an operator. The restricter operator, in combination with the recursion operator designated by where-terms, model algorithmic systems that are equipped with memory cells. Results of computations are stored in the memory cells. Memory cells are typed, i.e., they can hold only data of the respective type. In addition, some of the computations and the memory cells are restricted by propositional conditions. The propositional restrictions can represent properties of objects that have to be fulfilled. Our purpose is to use \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) and \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) for modelling functionality of biological entities that have valences or receptors for binding across sequences of structural entities with binding connections, such as neurons and neural networks.

The formal language \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) has the same set of types as \( \mathcal{L}^{\lambda}_{\text{RaA}} \) defined in (2), i.e., \( \text{Types}_{\lambda_{\text{RaA}}} = \text{Types}_{\lambda_{\text{RaAA}}} \). The vocabulary of \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) is the same as that of \( \mathcal{L}^{\lambda}_{\text{RaA}} \), i.e., \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) has typed constants and two kinds of typed variables: pure variables, for \( \lambda \)-abstraction operator; and, memory (recursion) variables, for storing information. The set \( \text{Terms}_{\lambda_{\text{RaAA}}} \) of the terms of \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) are defined by adding one more structural rule for the operator constant such that to (8a)–(8e).

**Definition 8.** The set \( \text{Terms}_{\lambda_{\text{RaAA}}} \) of the terms of \( \mathcal{L}^{\lambda}_{\text{RaAA}} \) consists of the expressions generated by the following rules in TBNF:

\[
A ::= c^* : \tau \quad (49a) \\
| x^* : \tau \quad (49b) \\
| B^{(\sigma \to \tau)}(C^n) : \tau \quad (49c) \\
| \lambda (v^\sigma)(B^\tau) : (\sigma \to \tau) \quad (49d) \\
| (A_0^{\sigma_0} \text{ where } \{ p_1^{\sigma_0} := A_1^{\sigma_0}, \ldots, \\
\quad \quad \quad p_m^{\sigma_0} := A_m^{\sigma_0} \} : \sigma_0) \quad (49e) \\
| (A_0^{\sigma_0} \text{ such that } \{ C_1^{\sigma_0}, \ldots, C_m^{\sigma_0} \}) : \sigma_0 \quad (49f)
\]
given that: $c$ is a constant; $x$ is a variable of either kind: $A_1: \sigma_1, \ldots, A_n: \sigma_n, C_1: \tau_1, \ldots, C_m: \tau_m \in \text{Terms}_{\mathcal{L}_\mathcal{M}}$; $p_i: \sigma_i$, $i = 1, \ldots, n$ ($n \geq 0$) in (49e) are pairwise different recursion variables of respective types, such that the sequence of assignments $\{ p_1^\alpha := A_1^\alpha, \ldots, p_n^\alpha := A_n^\alpha \}$ satisfies the Acyclicity Constraint (AC); and each $\tau_i$ is either the type $t$ of truth values, or the type $\tilde{t}$ of state dependent truth values, or (11b).

By extending the reduction calculus of $\mathcal{L}_{\mathcal{M}}$ with a rule for reducing terms with restrictions of the form (49f), the results in Section III and Section IV hold for the extended language $\mathcal{L}_{\mathcal{M}}$ of restricted algorithms. These properties are not in the subject of this paper because they require extensive mathematical work. They will be in a forthcoming paper devoted to them.

The rules (49a)–(49f) applied recursively provide very expressive formal terms that represent algorithmic computations that end after finite number of computational steps, because of the Acyclicity Constraint (AC).

In particular, combination of the rules (49e) and (49f), gives terms of recursion and restrictors of the forms (50a)–(50b) and (51a)–(51b):

$$\{ [A_0^\sigma \text{ such that } \{ C_1^\tau, \ldots, C_m^\tau \} ] \text{ where } \{ p_1^\tau := A_1^\tau, \ldots, p_n^\tau := A_n^\tau \} \} \quad \text{(50a)}$$

$$\{ [A_0^\sigma \text{ such that } \{ C_1^\tau, \ldots, C_m^\tau \} ] \text{ such that } \{ C_1^\tau, \ldots, C_m^\tau \} \} \quad \text{(50b)}$$

$$\{ [A_0^\sigma \text{ such that } \{ C_1^\tau, \ldots, C_m^\tau \} ] \text{ where } \{ p_1^\tau := A_1^\tau, \ldots, p_n^\tau := A_n^\tau \} \} \quad \text{(51a)}$$

$$\{ [A_0^\sigma \text{ where } \{ p_1^\tau := A_1^\tau, \ldots, p_n^\tau := A_n^\tau \} \text{ such that } \{ C_1^\tau, \ldots, C_m^\tau \} ] \} \quad \text{(51b)}$$

VI. MODELLING ALGORITHMIC NEURAL NETWORKS

A. Procedural and Declarative Neural Networks

We present the use of the recursion terms with constraints for modelling neural networks.

Neural systems (in peripheral and central nervous systems) of living organisms, even as simple as Drosophila melanogaster, have innate faculty of both procedural and declarative memory, see, e.g., Kandel et al. [20] and Squire and Kandel [21].

a) Neural Networks of Procedural Memory: Here, we propose to model procedural memory by terms having recursive assignments of the form (49e), while employing the entire range of term forms (49a)–(49f).

The systems of assignments in terms of the form (49e) represent mutually recursive computations of the denotations of the terms $A_i^\alpha$, which are saved in the corresponding memory cells $p_i$. Procedural memory is modelled via the assignments $p_i^\alpha := A_i^\alpha$. The system of mutually recursive assignments (52b) models the following fundamental phenomena of functional, procedural neural networks:

1) The collection (52a) is a recursively linked network of memory cells $p_i: \sigma_i$ of corresponding types

2) The system (52b) has algorithmic, i.e., procedural, nature of a network of memory cells $p_i$:

Under completion of the computation of the data $A_i^\alpha$, i.e., of the denotation of $A_i^\alpha$, it is saved in the designated memory cell, i.e., in the neuron $p_i$.

3) The rank function, in accordance with the Acyclicity Constraint (AC), by (9), guarantees that the network (52b) has memorised the corresponding data pieces, after completing the algorithmic computations

$$p_1: \sigma_1, \ldots, p_n: \sigma_n \quad \text{(52a)}$$

$$\{ p_1^\alpha := A_1^\alpha, \ldots, p_n^\alpha := A_n^\alpha \} \quad \text{(52b)}$$

b) Declarative Information: Declarative information is modelled by terms of the form $A: \tau$, where $\tau$ is either the type $t$ of truth values, or the type $\tilde{t}$ of state dependent truth values (see (11b)).

c) Neural Networks of Declarative Memory: Here, we model declarative memory by the specialised networks, or sub-networks, of systems of assignments of the form (53a):

$$t_1^\alpha := P_1^\alpha, \ldots, t_n^\alpha := P_n^\alpha \quad \text{(53a)}$$

for $P_1: \tau_1$, where

$\tau_1$ is either the type $t$ of truth values, or

the type $\tilde{t}$ of state dependent truth values

Declarative memory is innately integrated into networks of procedural memory. That is, neural networks of declarative memory (53a) are typically integrated as recursive subsystems of more general procedural assignments (52b):

$$\{ t_1: \tau_1, \ldots, t_n: \tau_n \} \subseteq \{ p_1: \sigma_1, \ldots, p_n: \sigma_n \} \quad \text{(54)}$$

B. Algorithmic Binding of Functional Neuro-Receptors

A term $T_{m+1}$ of the form (55a)–(55f) represents a neural network. The head term (55a)–(55e) is a neural sub-network of sequentially bound neural cells (55c), which are sequentially linked by binding functionality. Each subterm $\lambda x_{i_j} Q_{i_{j-1}}$ models a neural cell. Its neural body $Q_{i_{j-1}}: \{ (\sigma_{i_{j-1}} \rightarrow \tau_{i_{j-2}}) \}$ has a receptor represented by its argument slot of the corresponding type $\{ \sigma_{i_{j-1}} \rightarrow \tau_{i_{j-2}} \}$. The $\lambda$-abstraction $\lambda x_{i_j} Q_{i_{j-1}}$ represents the axon of the neural cell $\lambda x_{i_j} Q_{i_{j-1}}$. Similarly, the $\lambda$-abstraction $\lambda x_{i_{j-1}} Q_{i_{j-2}}$, in $\lambda x_{i_{j-1}} Q_{i_{j-2}}$, represents the axon of $\lambda x_{i_{j-1}} Q_{i_{j-2}}$.

In the subsequently bound (linked) neural cells, represented by the subterm of the form (55c), $Q_{i_{j-1}}$, binds the axon $x_{i_{j-1}}$ of $\lambda x_{i_{j-1}} Q_{i_{j-2}}$, for each $j = 3, \ldots, (m - 3)$.

$$T_{m+1} \equiv Q_{i_m} \lambda x_{i_{(m-1)}} Q_{i_{(m-2)}} \lambda x_{i_{(m-2)}} Q_{i_{(m-3)}} \lambda x_{i_{(m-3)}} Q_{i_{(m-4)}} \ldots \lambda x_{i_{(3)}} Q_{i_{(2)}} \lambda x_{i_{(2)}} Q_{i_{(1)}} \lambda x_{i_{(1)}} Q_{i_{(0)}} \lambda x_{i_{(0)}} H(x_1) \ldots \ldots (x_n) \quad \text{(55a)}$$

$$\ldots \lambda x_{i_{(3)}} Q_{i_{(2)}} \lambda x_{i_{(2)}} Q_{i_{(1)}} \lambda x_{i_{(1)}} Q_{i_{(0)}} \lambda x_{i_{(0)}} H(x_1) \ldots \ldots (x_n) \quad \text{(55b)}$$

$$\ldots \lambda x_{i_{(3)}} Q_{i_{(2)}} \lambda x_{i_{(2)}} Q_{i_{(1)}} \lambda x_{i_{(1)}} Q_{i_{(0)}} \lambda x_{i_{(0)}} H(x_1) \ldots \ldots (x_n) \quad \text{(55c)}$$

$$\lambda x_{i_{(3)}} Q_{i_{(2)}} \lambda x_{i_{(2)}} Q_{i_{(1)}} \lambda x_{i_{(1)}} Q_{i_{(0)}} \lambda x_{i_{(0)}} H(x_1) \ldots \ldots (x_n) \quad \text{(55d)}$$

where $\overrightarrow{p} := \overrightarrow{A} \quad \text{(55e)}$
The term $T_{m+1}$ of the form $(55a)–(55f)$ represents a neural network in its ‘encapsulated’ form, where the algorithmic steps of binding axons by the corresponding receptors are ‘hidden’ below encapsulating membranes.

In the canonical form of $T_{m+1}$, the head term of $T_{m+1}$ representing the head neural sub-network, i.e., $(35a)–(35e)$, is reduced to a subterm $R_{m+1}$ that have the structural form of $R_{m+1}$ in $(34b)–(34i)$. The term $R_{m+1}$ represents the innate, inner algorithmic structure of the same neural sub-network of $T_{m+1}$, inside its encapsulating membrane. On the other hand, the neural network $R_{m+1}$ is algorithmically synonymous (equivalent) with the term $R_{m+1}$ in $(36b)–(36i)$, while they are structurally different.

VII. FORTHCOMING AND FUTURE WORK

The recursion assignments in Section IV include $\lambda$-terms binding argument slots of the ‘innermost’ subterm, sequentially by recursion within the scope of the recursion operator where. We have started the exposition by reducing the term $(32a)–(32e)$. That resulted the specific variables for the $\lambda$-abstracts. However, these terms are congruent to terms by renaming variables bound by the $\lambda$-operator. There are more interesting results related to linking of the bindings related to these $\lambda$-terms and renaming variables abstracted away with $\lambda$-operator. Such properties of the binding operators $Q_{hj}$ introduced in this paper are in our forthcoming work. Other forthcoming work is to relate the results in this paper with the reduction calculus in Loukanova [19] and rendering expressions of natural language, e.g., similar to the underspecified quantification presented in Loukanova [5], as well as with other extensions of $L^\lambda_m$.

Questions whether the approach presented in Ślęzak et al. [22] is comparable with the type theory of Moschovakis algorithms extended in this paper, and if yes, how, remains open work. Studying the shared ideas and differences in these approaches may provide mutual enrichments and developments.

REFERENCES


