Constraint Based Grammar and Semantic Representation with The Language of Acyclic Recursion*

Roussanka Loukanova

1 Introduction

Moschovakis [5] introduced the logical calculus $L^\lambda_{ar}$ of acyclic recursion for mathematical modelling of the concepts of meaning of natural language. This abstract relies on the accessibility to the original work of Moschovakis [5]. The final version of the paper will include an introduction to the language $L^\lambda_{ar}$ of Acyclic Recursion with necessary adjustments (e.g., the order in which the semantic arguments are “attached” to the function symbols) commentaries according to its application. There will be a section devoted to a discussion of: (1) the role of the two new kinds of variables: pure and recursion variables; (2) the terms that render natural language expressions with co-referential occurrences of names (proper names or descriptions) versus terms for representing anaphora–antecedent relations: some of these terms do not have any equivalent representation in classic type logic as Montague’s IL; (3) the semantic roles of recursion terms versus classic lambda-terms; (4) canonical forms of terms and their algorithmic role for computational semantics.

The paper considers a possibility for application of $L^\lambda_{ar}$ to computational syntax-semantics interface in Constraint Based Lexicalized Grammar (CBLG) of natural language. While, currently, HPSG framework is a distinctive representative of CBLG, this paper takes a general approach to computational syntax-semantics interface in CBLG, e.g., as introduced by Sag et al. [6], but not necessarily, and not necessarily to HPSG\(^1\). The grammar is defined by a CBLG type hierarchy, which interleaves various linguistic layers, e.g.: (1) lexicon, which consists of feature-value descriptions of basic lexical items and lexical rules for licensing feature-value structures of newly “generated” lexemes up to fully inflected words; (2) grammar rules expressed as constraints; (3) other constraints that define well-formed feature-value structures of phrases. The constraints of

---

\*I am enormously grateful to Manfred Sailer for his feedback and the many fruitful discussions, while I have been working on the paper. All examples in this paper are provided by him. Any errors in this abstract are my own.

\(^1\)In general, HPSG community does not consider $\lambda$-calculi for semantic representation included within it. This is not the case, as Sailer [8] demonstrated, by using a version of $TY_2$. 

1
the CBLG type hierarchy introduce a cross-layer for semantic representations in the feature-value structures of all lexical items, words and phrases.

This paper will define semantic representations by including the logic types of the formal language $L_{\lambda_{ar}}$ of acyclic recursion as a type sub-system of the CBLG type hierarchy. The feature-value descriptions of the natural language expressions include a “semantic” feature, sem, the value of which is a feature-value pair that encodes a canonical $L_{\lambda_{ar}}$ term. Thus, a feature-value description of a natural language expression includes its syntactic co-occurrence requirements and its semantic rendering into a $L_{\lambda_{ar}}$ term. The rendering operation is defined compositionally via the rules and constraints of CBLG. The paper will represent semantics of various lexical constructs and the major CBLG rules and principles, e.g., with CBLG, as introduced by Sag et al. [6]. By using lexical approach, The two rules in this abstract cover various syntactic constructs such as NPs, VPs, PPs, AdjPs, AdvPs, Sentences.

2 Semantic Representation in Major CBLG Rules

2.1 Feature-value Representation of Recursion Terms

We introduce features for representing logic types, terms and their parts. The value of the feature l-type is a logic type. The values of the features t-head and where are, respectively, a term and an acyclic system of assignments\(^2\) \{\(p_1 := A_1, \ldots, p_n := A_n\) \((n \geq 0)\) in the formal language of $L_{\lambda_{ar}}$.

Constraints associated with the grammar types of a CBLG restrict the values of these features so that a feature structure of the form

\[
\begin{bmatrix}
\text{L-TYPE} & \tau \\
\text{TERM} & \begin{bmatrix}
\text{T-HEAD} & A_0 \\
\text{WHERE} & \{p_1 := A_1, \ldots, p_n := A_n\}
\end{bmatrix}
\end{bmatrix}
\]

is well-defined iff \(A_0\) is an $L_{\lambda_{ar}}$ term of type \(\tau\) and the set \(\{p_1 := A_1, \ldots, p_n := A_n\}\), \(n \geq 0\), represented by the value of the feature where, is an acyclic system of assignments in $L_{\lambda_{ar}}$. The above feature structure represents the $L_{\lambda_{ar}}$ term \(A_0\) where \(\{p_1 := A_1, \ldots, p_n := A_n\}\), which, in general, is not necessarily in canonical form. And vice versa, any term \(A : \tau\) of $L_{\lambda_{ar}}$ can be represented by such feature structure.

In the $L_{\lambda_{ar}}$ language and its calculus, recursive terms are essential and require sets of assignments in the scope of the recursion operator, represented by the constant where. Since in the recursive terms, i.e. the where-terms, the acyclic systems of assignments are sequences, the congruence relation between terms is closed over reordering of the assignments, i.e. over permutation. On the other

\(^2\)See, Moschovakis [5] and the full version of this paper for this notion. Intuitively, a system of assignments is acyclic if it does not allow assignments that lead to “loops”: \(p := \ldots A(p)\), i.e., it closes-off.
hand, in HPSG frameworks, lists are typically used instead of sets in values of features.

The operations \textit{append}, \textit{concatenation} and \textit{union} are different, and are defined over different objects, i.e., lists, sequences and sets, respectively. This nuisance can be avoided by a formal constraint-based grammar, in which the value of the feature \textit{where} is either a set or a sequence of assignments, and imposing closure under permutation over the order of the assignments, similarly to the permutation closure of the relation of congruence between terms in $L_{ar}^\lambda$. To comply with the tradition of HPSG, we could formulate the rules and the lexicon by stating that the value of the feature \textit{where} is a list of assignments, which is interpreted as a set by imposing indifference with respect to permutation of the list elements. Because such details are subject of the formal foundations of constraint based grammar, which is not in the topic of this paper, to simplify the exposition, we assume that the values of the feature \textit{where} is a set of assignments.

The typical CBLG grammar rules define co-occurrence restrictions, which correspond to syntactic combinations. The syntactical combination of a CBLG grammar rule and the logical types that are values of the features \textit{L-type} in the daughter’s feature structures determine which of the $L_{ar}^\lambda$ syntactic constructions need to be used. For example, a VP can be such that it requires a subject NP. The HSR combines the VP with an appropriate NP to form a sentence. In case when the term associated with the VP is $C : \bar{c} \rightarrow \bar{t}$ and the term associated with the subject NP is $D : \bar{c}$, the term $A$ associated with the mother S node is $A \equiv C(D) : \bar{c}$. This means that $A \approx \text{cf}(A) \equiv \text{cf}(C(D)) : \bar{c}$. The reduction calculus of $L_{ar}^\lambda$ provides effective procedure for reducing $A$ to its canonical form $\text{cf}(A)$ (modulo congruence): $A \Rightarrow \text{cf}(A)$. However, given the canonical forms $\text{cf}(C)$ and $\text{cf}(D)$ the canonical form $\text{cf}(C(D))$ can be determined directly from the parts of $\text{cf}(C)$ and $\text{cf}(D)$ and the definition of the canonical forms of the terms of $L_{ar}^\lambda$ (13.3, Moschovakis [5]).

In this paper, we assume that the lexicon (lexical entries and lexical rules) are formulated to handle feature-value representations of $L_{ar}^\lambda$ terms in canonical forms. This means that each lexical entry, of grammar type lexeme or word, has a feature structure which, by the values in the feature-value pairs $[\text{t-head} \ A_0]$ and $[\text{where} \ \{p_1 := A_1, \ldots, p_n := A_n\}]$ ($n \geq 0$), represents a $L_{ar}^\lambda$ term $A$ in canonical form: i.e., the term $A \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$ is in canonical form. We are targeting a grammar where all grammar rules are defined by distributing the parts of canonical forms among the features $\text{t-head}$ and $\text{where}$, in all daughter and mother feature structures. The term $A$ associated with the mother is computed from the terms associated with the daughters by respecting the logical types given as values of the feature \textit{L-type}, which, when matching, should determine the syntactic combination of the terms. The values of the features $\text{t-head}$ and $\text{where}$ of the mother node are determined from the corresponding values of these features in the daughters’ feature structures, by using the syntactical definition of the $L_{ar}^\lambda$ terms and the definition of the canonical form $\text{cf}(A)$ of each term $A$ of $L_{ar}^\lambda$ (13.3, Moschovakis [5]). This provides a proof by induction that the well-formed tree structures licenced by
the considered rules represent properly typed terms of $L^{\lambda}_{ar}$ in canonical forms.

### 2.2 Head Specifier Rule

The Head Specifier Rule (HSR) defines rendering into application terms\(^3\) of $L^{\lambda}_{ar}$, for the typical simple cases of saturation of the specifier, when the daughter feature structures do not introduce type incompatibility or semantic underspecification (which can arise, for example, in expressions with multiple occurrences of quantifiers):

\[
\begin{align*}
\begin{bmatrix}
\text{phrase} \\
\text{syn} \\
\text{sem}
\end{bmatrix}
& 
\begin{Bmatrix}
\begin{bmatrix}
\text{val} \\
\text{spr} \\
\text{comps}
\end{bmatrix}
\end{Bmatrix}
\begin{bmatrix}
\text{term} \\
\text{t-head} \\
\text{where} \\
\text{U}
\end{bmatrix}
\begin{bmatrix}
\text{t-head} A_i, \text{spr} \langle \rangle \\
\text{where} \text{U}_i
\end{bmatrix}
\begin{bmatrix}
\text{term} \\
\text{t-head} A_j, \text{spr} \langle \rangle \\
\text{where} \text{U}_j
\end{bmatrix}
\end{align*}
\]

where:

(2) If $T_i \equiv (\sigma \rightarrow \tau)$ and $T_j \equiv \sigma$, for $i, j \in \{1, 2\}$ and $i \neq j$, then $T \equiv \tau$ and

$A_0 \text{ where } U \equiv \text{cf}([A_i,0 \text{ where } U_i][A_j,0 \text{ where } U_j])$.

There are two sub-cases for the term $[A_0 \text{ where } U]$ in\(^4\) (2) that are determined by the case (CF2) of the definition of the canonical forms of the terms of $L^{\lambda}_{ar}$ (3.13, Moschovakis [5]):

(3a) If $A_j,0$ is immediate, then $A_0 \equiv A_i,0(A_j,0)$ and $U \equiv U_i \cup U_j$;

(3b) otherwise, $A_0 \equiv A_i,0(q_0)$ and $U \equiv \{q_0 := A_j,0\} \cup U_1 \cup U_2$.

The values of features T-HEAD and WHERE of the left hand side of the HCR rule are the parts of a canonical form, which is determined by: the types $T_1$ and $T_2$; the values of the features T-HEAD and WHERE in the daughters’ feature structures on the right hand side; and the definition of the canonical form cf($A$) of each term $A$ (see the definition in 3.13, Moschovakis [5]). To avoid binding and free variable clashes, we also assume that, in each application of the grammar rules, the representations of the $L^{\lambda}_{ar}$ terms are such that all bound location variables are distinct and distinct from all the free locations (variables and constant), by making appropriate renaming substitutions, if needed.

### 2.2.1 An Example for a Definite Description

(4) The dog barks.

---

\(^3\)We use extra brackets and sizes of parentheses for easier comprehension of terms.

\(^4\)The additional sub-index 0 in $A_{i,0}$ and $A_{j,0}$ is unnecessary in these formulations, but allows easier re-formulations of the rules with constructs inside $U_1$ and $U_2$. 

---

4
2.2.2 An Example for a NP Quantifier in Subject Position

(5) Every dog barks.
(6a) $\text{every}(\text{dog})(\text{bark}) \Rightarrow c_f \text{every}(d)(b)$ where $\{b := \text{bark}, d := \text{dog}\}$;
(6b) $c_f(\text{every}) \equiv \text{every where } \{\}; c_f(\text{dog}) \equiv \text{dog where } \{\}$;
(6c) $c_f(\text{bark}) \equiv \text{bark where } \{\}$;
(6d) $c_f(\text{every}(\text{dog})) \equiv \text{every}(d)$ where $\{d := \text{dog}\}$
(by (CF2), from (6b));
(6e) $c_f(\text{every}(\text{dog})(\text{bark})) \equiv \text{every}(d)(b)$ where $\{d := \text{dog}, b := \text{bark}\}$
(by (CF2), from (6c) and (6d)).

The canonical form of the term $\text{every}(\text{dog})(\text{bark})$ is the following recursion term (modulo congruence, i.e., up to renaming of bound locations and reordering of the assignments):

$$c_f(\text{every}(\text{dog})(\text{bark}) \equiv c_f \text{every}(d)(b) \text{ where } \{b := \text{bark}, d := \text{dog}\}.$$

A reduction to a canonical form, like that in (6a), can be effectively found by the rules of the reduction calculus of $L_{\lambda ar}$. But $c_f(\text{every}(\text{dog})(\text{bark}))$ can be “derived” step-by-step in the two applications of the HSR with the case (3b), i.e., by (CF2) of the definition of canonical form.

3 Head Complement Rule: Version 1

As in the above statement of the HSR, in the basic cases, with no type incompatibility or under-specification (which can arise for multiple occurrences of quantifiers), the Head Complement Rule (HCR) determines rendering into application terms of $L_{\lambda ar}^h$:

$$\begin{align*}
\text{phrase} & \quad \text{syn} \quad \text{val} \quad \text{comps} \quad \text{rest} \quad \text{first} \\
\text{syn} & \quad \text{val} \quad \text{comps} \quad \text{rest} \quad \text{first} \\
\text{sem} & \quad \text{term} \quad \text{t-head} \quad \text{where} \quad U \\
\text{sem} & \quad \text{term} \quad \text{t-head} \quad \text{where} \quad U
\end{align*}$$

where the types $T, T_1, T_2$ and the terms $A_0, A_{1,0}, A_{2,0}$ and $U$ are defined as in (2) with its both sub-cases, (3a) and (3b).

Note that this version of the HCR saturates the list of the complements one at a time. This means that the grammar type phrase should not introduce a constraint $[\text{COMPS }\langle\rangle]$. Intuitively, any feature structure, that marks a node of a parse tree associated with the grammar rules, and is not of type word, is of type phrase. I.e., a feature structure is of type phrase, if it has partly or entirely saturated COMPS list. Such version of the HCR introduces “semi-phrases” that are not distinguished from proper phrases, for which $[\text{COMPS }\langle\rangle]$. In all cases of complete or partial saturation of the complement list, the statement of the
We have rendered typical transitive extensional verbs, like “read, hug, kiss” into terms headed by a transitive verb with a NP quantifier in its complement position.

In this subsection, we consider the special case of sentences where the head VP refers to (2).

3.0.3 Transitive Verbs with Proper Names in Subject and Complement Positions

(7) Kim hugged Maja.

4 Head Complement Rule: Version 2

4.1 Transitive Verbs with Quantifier NP in Complement Position

In this subsection, we consider the special case of sentences where the head VP is headed by a transitive verb with a NP quantifier in its complement position. We have rendered typical transitive extensional verbs, like “read, hug, kiss” into \( L_{at} \) constants of type \((\bar{e} \rightarrow (\bar{e} \rightarrow \bar{t}))\). On the other hand, NP quantifiers are rendered into terms of type \((\bar{e} \rightarrow \bar{t}) \rightarrow \bar{t}\). If the head of a VP is transitive extensional verb and its complement position is occupied by a NP quantifier, then there is type mismatch between the semantic representations of the head verb and its complement, and, the HCR rule can not be used with the statement...
in (2) and its sub-cases (associated with the HCR: Version 1) For such cases of the HCR, we define its semantic representations as follows (here we repeat the HCR for clarity):

\[
\begin{array}{c}
\begin{bmatrix}
\text{phrase} \\
\text{syn} \\
\text{val} \\
\text{comps} [1] \\
\text{term} \\
\text{t-head} A_0 \text{ where } U \\
\end{bmatrix}
\end{array}
\rightarrow
\begin{array}{c}
\begin{bmatrix}
\text{SYN} \\
\text{VAL} \\
\text{COMPS} [2] \\
\text{FIRST} [4] \\
\text{REST} [5] \\
\text{SEM} \\
\text{TERM} \\
\text{T-HEAD} A_0 \text{ where } U \\
\end{bmatrix}
\end{array}
\end{array}
\]

where \( T_2 \equiv (\overline{e} \rightarrow (\overline{\sigma} \rightarrow \overline{t})) \), \( T_1 \equiv ((\overline{e} \rightarrow \overline{t}) \rightarrow \overline{t}) \), and either of the following cases:\(^5\):

\begin{enumerate}
\item[(8a)] \( T \equiv (\overline{\sigma} \rightarrow \overline{t}) \) and
\[ A_0 \text{ where } U \equiv \lambda \overline{\gamma} A_{1,0}'(p(\overline{\gamma})) \text{ where } \{ p := \lambda \overline{\gamma} \lambda x A_{2,0}'(x)(\overline{\gamma}) \} \cup U'_1 \cup U'_2, \]
\item[(8b)] \( T \equiv (\overline{q} \rightarrow \overline{t}) \), where \( \overline{q} \equiv ((\overline{e} \rightarrow \overline{t}) \rightarrow \overline{t}) \)
\[ (\overline{q} \text{ is the type of unary quantifiers}) \text{ and } \]
\[ A_0 \text{ where } U \equiv \lambda Y A_{1,0}'(p(Y)) \text{ where } \{ p := \lambda Y \lambda x [Y(q(Y)(x))] \}, \]
\[ q := \lambda Y \lambda x (\lambda \overline{\gamma} [A_{2,0}'(x)(\overline{\gamma})]) \}
\[ \cup U'_1 \cup U'_2, \]
\end{enumerate}

where \( x \) and the variables in the sequence \( \overline{\gamma} \) are fresh pure variables; \( p \) and \( q \) are fresh location variables; \( A_{1,0}', A_{2,0}', U'_1 \) and \( U'_2 \) are obtained by the replacements specified in (CF3): i.e., each assignment \( r := R \), is replaced by \( r' := R' \), where \( r' \) is a fresh location and the term \( R' \) is the result of the substitution \( R' := R(r := r'(Y)(x)) \).

Note that the co-indexing of the value of the first complement [FIRST [4]] of the head daughter binds, via \( \lambda \)-abstraction indexing, the correct argument role in the quantifier application. The intuitions that are behind this statement are the reduction rules that are needed to reduce to canonical forms the term in the quantifier application. Instead of the above rule, we could have used the following one:

\(^5\)We assume here that the terms \( \lambda x[A_{2,0}'(x)(\overline{\gamma})] \) and \( \lambda x[Y(\lambda \overline{\gamma}[A_{2,0}'(x)(\overline{\gamma})])] \) are not immediate. These special cases will be considered in the full paper.
Assuming that the daughters are in canonical forms guarantees direct “derivation” of the canonical form of the $L^A_{n_x}$ term represented in the mother feature structure.

In the final version of the paper, we will consider other possibilities for $\tilde{\sigma}$. The semantic representation of verbs with more arguments, e.g., ditransitive ones, will use a sequence of variables $\tilde{y}$ with more than one element. Here we consider simple transitive verbs with one complement, i.e., $\tilde{\sigma} \equiv \tilde{e}$ and $\tilde{y} \equiv y$, as in the example:

(10) Kim hugged some dog.

If we apply the quantifier term represented by the node $(n_4)\text{NP}$ to the term represented by the head verb in the node $(n_3)\text{N}$, which has been $\lambda$-indexed with
the variable that is the value of the index, i.e., \( x_d \equiv \text{INDEX} \), the term is:

\[
(11a) \quad \lambda y \left[ \text{some}(d) \text{ where } \{d := \text{dog}\} \right] \left( \lambda x_d \text{hug}(x_d)(y) \text{ where } \{\}\right)
\]

\[
(11b) \quad \Rightarrow \lambda y \left[ \text{some}(d) \left( \lambda x_d \text{hug}(x_d)(y) \right) \text{ where } \{d := \text{dog}\} \right]
\]

\[
(11c) \quad \Rightarrow \lambda y \left[ \text{some}(d) \left( h \right) \text{ where } \{d := \text{dog}, \ h := \lambda x_d \text{hug}(x_d)(y)\} \right]
\]

\[
(11d) \quad \Rightarrow_{\text{cf}} \lambda y \text{some}(d'(y))(h'(y)) \quad \text{where } \{d' := \lambda y \text{dog}, \ h' := \lambda y \lambda x_d \text{hug}(x_d)(y)\}
\]

\[
(11e) \quad \equiv_c \lambda y_k \text{some}(d)(h(y_k)) \quad \text{where } \{d := \lambda y_k \text{dog}, \ h := \lambda y_k \lambda x_d \text{hug}(x_d)(y_k)\}
\]

\[
(11f) \quad \not\equiv \lambda y \text{some}(d)(h(y)) \text{ where } \{d := \text{dog}, \ h := \lambda y \lambda x_d \text{hug}(x_d)(y)\}
\]

The term in (11f) is in a canonical form, but it is not referentially synonymous (by the reduction calculus) to any of the other terms (11a)-(11e). In particular, (11f) is not referentially synonymous to the term in (11e). The term (11e) labels the node \((n_2)_{vp}\), as the result of the application of the rules HCR (for licencing \((n_2)_{vp}\)) and HSR (for licencing \((n_4)_{np}\)) and the semantic renderings associated with them: (8a) and (2), respectively. Note that the term \(\lambda y_k \text{dog} : (\emptyset \rightarrow (\emptyset \rightarrow \emptyset))\) may seem strange at first glance, but it has a technical role, from a computational perspective. It is resulted by the \(\lambda\)-rule of the reduction calculus. The \(\lambda\)-abstraction is over the variable \(y_k\), which does not occur in its scope, the sub-term \(\text{dog} : (\emptyset \rightarrow \emptyset)\). Thus, the sub-term \(d(y_k)\) of the head part in (11f), does not represent that \(y_k\) has the property of being a dog.

In this example, the values of the features L-TYPE, T-HEAD and WHERE in the node \((n_2)_{vp}\) are according to (8a) determined by the L-TYPE values of the daughter’s terms. Note that the term \([\lambda y_k \text{some}(d(y_k))(h(y_k))](k)\) that is the value of the feature T-HEAD, in the feature structure of the node \((n_0)\) \(S\), is explicit and irreducible. It can be considered, intuitively as stating that \(k\) has the property of having a \(d\) entity to which it does \(h\). The location \(d\) provides a property of entities; the location \(h\) provides an “action” of entities to entities. From mathematical point, these terms provide algorithmic steps of computing the denotation of \([\lambda y_k \text{some}(d(y_k))(h(y_k))](k)\) after the basic components are computed and “stored” in locations \(k\), \(d\) and \(h\).

4.1.1 Transitive Verbs with Quantifier NPs in Subject and Complement Positions

In the following example, the subject and complement positions of the sentence are filled by NP quantifiers that cause two different readings with different “scoping” of the quantifiers.
∃∀ Rendering

(12) Every cat hugged some dog.

The values of the features L-TYPE, T-HEAD and WHERE, i.e., the $L^\lambda_{ar}$ term parts associated with the node $(n_2)\text{VP}$ are determined in the same way as the parts of the corresponding node $(n_2)\text{VP}$, in the analysis of the previous sentence, i.e., by using the HCR with the rendering rule (8a). The parts of the $L^\lambda_{ar}$ term represented by the node $(n_0)S$ are determined by the rendering rule (2), in particular, by its case (3b).

∃∀ Rendering

(13) Every cat hugged some dog.
5 Comparisons and Conclusions

The values of the features T-head and WHERE, in the feature-value pairs [T-head A] and [WHERE \{p_1 := A_1, \ldots, p_n := A_n\} \ (n \geq 0)] represent \(L_A^\lambda\), terms A where \{p_1 := A_1, \ldots, p_n := A_n\} in a canonical form.

The language and theory of \(L_A^\lambda\) is a new and unique mathematical formalization of major concepts of language meaning, such as Frege’s notions of sense and denotation, and relations between them. Intuitively, the meaning of a language expression has two components: (1) a denotation, which can vary depending on particular states (worlds), and (2) a referential intension, which is the algorithm for “computing” the denotation in a given context. The referential intension of a term \(A\) is represented by its canonical form \(\text{cf}(A)\), which codifies all the basic facts from which the denotation \(A\) is computed. Thus, reducing terms to their canonical forms is important. A formal grammar that renders directly the canonical terms provides not only faithful semantic representation, but also efficiency. The calculus of \(L_A^\lambda\) is a proper extension of Gallin’s typed logic \(TY_2\) (Gallin [4]) and thus, of various \(\lambda\)-calculi, in particular, those that are embedded (interpreted) in \(TY_2\), as Montague’s Intensional Logic (IL). There are \(TY_2\)
terms that render NL sentences that do not have any corresponding terms in Montague’s IL, and represents faithfully their semantics.

There has been various approaches to semantic underspecification by using \(\lambda\)-calculi languages. A classic representative, with many variants, is Cooper’s semantics storage. There have been other approaches, more recently, by Bos [1], Copestake et al. [2], Egg [3], Richter and Sailer [7], and others. The language and calculus of \(L^\lambda_{ar}\) provides underspecification that is in the spirit of modeling meanings by algorithms, i.e., as abstract mathematical objects. The full paper will give an introduction to how this can be done by a brief comparison to the existing approaches. A comprehensive retrospective on this subject is a subject of further work, in particular, from the perspective of development of CBLG that takes type-theoretic grammar approach in full consideration.

References


13