Semantics with The Language of Acyclic Recursion in Constraint-Based Grammar

Roussanka Loukanova

1 Introduction

Computational semantics in general, and especially for contemporary grammar theories of NL, is an active research area. Largely, incorporating computational semantics into formal syntax, and in particular, in computational grammar systems for natural language processing, is one of the foremost lines of research on natural and artificial languages. Moschovakis [11] introduced the logical calculus L_{ar}^{λ} of acyclic recursion for mathematical modelling of the concepts of meaning of natural language. This paper relies on familiarity with the syntax and semantics of the formal language L_{ar}^{λ}. For an extended introduction to the type theory L_{ar}^{λ}, see the original work of Moschovakis [11]. An introduction to the language L_{ar}^{λ} of acyclic recursion is given also in Loukanova [10].

While, currently, HPSG framework is a distinctive representative of Constraint-Based Lexicalized Grammar (CBLG), this paper takes a general perspective on an emerging constraint-based approach to computational grammar. For demonstration purpose, we shall use CBLG as introduced by Sag et al. [15], which is closest to, but is not necessarily HPSG. Originally, HPSG was developed by Pollard and Sag, in [12] and [13], based on the original ideas of Situation Theory and Situation Semantics, see Barwise and Perry [1] and Devlin [6], [7]. By its constraint and lexicalized approach to grammar of NL, where a head component drives propagation of grammatical information, [13] became and remains a classic introduction to HPSG. The CBLG introduced by Sag et al. in [15] inherits some of the major ideas in [13], and, similarly, is primarily about the prospects of formalizing linguistic concepts by constraints into generalized grammar, which bland syntax and lexicon. Semantic representations are used only occasionally, in a rudimentary form, as minimally needed for some syntactic constructions that are tightly related to semantics, and to demonstrate the ideas of taking semantics into formal grammar. The feature-value descriptions that are used for embedding semantic content within syntactic constructions, retain the original situation semantics presentation.

Typically, logic type-theories with λ-calculi languages have not been considered as native to HPSG for semantic representation. This is not always the case, as Sailer [17] demonstrated with a version of Gallin's typed logic TY_2 (Gallin [9]). L_{ar}^{λ} properly extends the language and theory of TY_2 in a way that offers more expressiveness, not only from the point of mathematical modeling of algorithms, but also for semantics of natural language. This paper takes the task to show that L_{ar}^{λ} can be used for representation of semantics of natural language, in a compositional syntax-semantics mode with formal grammar. The formal grammar is conceived as defined by a CBLG type-hierarchical system, which interweaves various linguistic layers, e.g., lexicon, grammar rules and linguistic principals. The target is generalized, computational grammar that amalgamates syntax and semantics.

Lexicon The lexicon is a subsystem of the formal grammar, which consists of feature-value descriptions of basic lexical items and lexical rules. The lexical rules license (well-formed) feature-value structures of newly "generated" lexemes up to fully inflected words.

Grammar Rules In contrast to classic rewriting grammars, the grammar rules of CBLG are expressed as constraints, which define linguistic co-occurrence requirements, according to linguistic types. Rewriting grammars for NL (e.g., classic context-free grammars or even more advanced phrase structure grammars) define well-formed phrases according to their syntactic categories with generative phrase structure rules. The rules classify each syntactic category, e.g., the categories of sentences (S), noun phrases (NP), verb phrases (VP), adjective phrases (AP), etc., by specifying detailed co-occurrence information about the combinations between syntactic categories. The grammar rules of CBLG are defined as very general constraints, typically expressed by using feature-value descriptions, which provide linguistic classification across syntactic phrasal categories.

Principles Grammar principles and other constraints that define well-formed feature-value structures of phrases are also expressed by using a language of feature-value descriptions.

CBLG Type System The lexicon, grammar rules and principles are all integrated into a general grammar system by using a type hierarchy. The constraints of 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 defines semantic representations of natural language expressions by including the logic types of the formal language L^{λ}_{ar} of acyclic recursion as a type sub-system of the CBLG type hierarchy. The CBLG type system declares that feature structures of the natural language expressions require a "semantic" feature, SEM, the value of which is a feature-value pair that encodes a L^{λ}_{ar} term. Thus, a feature-value description of a natural language expression can represent[1] not only its syntactic structure and co-occurrence requirements, but also a *rendering* into a L^{λ}_{ar} term that represents its semantics. The render operation is defined compositionally via the rules and constraints of CBLG. The paper will represent semantics of various lexical items, phrasal constructs and some of the major syntactic rules, by using a version of CBLG, as introduced by Sag et al. [15]. In particular, a lexical approach to grammar is demonstrated by incorporating semantic representations with two rules that can be used for

[1] Some versions of HPSG include the phonological or orthographical form of the expressions in their feature structures.

saturation of the syntactic co-occurrence requirements of lexical items belonging to various parts of speech classes: verbs, nouns, adjectives, adverbs, prepositions. These rules cover the saturation of syntactic and semantic arguments in various syntactic constructs such as NPs, VPs, PPs, AdjPs, AdvPs, sentences.

Some Notations and Agreements. In general, through this paper, we follow closely Moschovakis [11] in notations, terminology, and style of presentation of all matters related to the formal language L^{λ}_{ar}. In particular, we use "$\equiv$" for the orthographical identity relation between the syntactic objects of L^{λ}_{ar}, i.e., terms and types. The equation symbol "$=$" is used as the identity predicate symbol in the language L^{λ}_{ar}, and as denotational equality between terms. The symbol "$:\equiv$" is a meta-symbol (i.e., not among the symbols of L^{λ}_{ar}) that is used in definitions of objects, i.e., for strict syntactic identity "by definition". The symbol "$:=$" is the assignment symbol in the recursion terms. It is also used in the definition of the updates of the variable function, which assigns values to free variables.

We shall diverge from Moschovakis [11], by using a variant of the class of languages L^{λ}_{ar}, with respect to the order of currying in the λ-terms. In Moschovakis [11], the currying order for types, functions and function symbols, of two and more arguments, follows the tradition, which is used in λ-calculi for mathematics and computer science, i.e.: $\tau_1 \times \tau_2 \to \tau \equiv (\tau_1 \to (\tau_2 \to \tau))$, where $\tau_1, \tau_2, \tau \in \mathit{Types}$, and, $A(B_1, B_2) \equiv [A(B_1)](B_2) : \tau$, where A, B_1 and B_2 are terms such that $A : (\tau_1 \to (\tau_2 \to \tau))$, $B_1 : \tau_1$ and $B_2 : \tau_2$. This implies that the function symbol *like*, which represents the corresponding relation, applies at first to the rendering of the verbal subject and then to the rendering of the verbal complement:

$$\text{John likes Mary.} \tag{1a}$$

$$\xrightarrow{\text{render}} \mathit{like}(\mathit{john})(\mathit{mary}) \equiv \mathit{like}(\mathit{john}, \mathit{mary}) \tag{1b}$$

According to English syntax, the transitive verbs combine at first with their complements to form a VP, and then, with their subject to form a sentence. This combination order can be achieved in the corresponding semantic rendering too, by using λ-abstraction and rendering the transitive verbs, similar to "like", by appropriate λ-terms, instead by constants:

$$\text{likes} \xrightarrow{\text{render}} \lambda y \lambda x\, \mathit{like}(x)(y) \tag{2a}$$

$$\text{John likes Mary.} \xrightarrow{\text{render}} \tag{2b}$$

$$[\lambda y \lambda x\, \mathit{like}(x)(y)](\mathit{mary})(\mathit{john}) \equiv [\lambda y \lambda x\, \mathit{like}(x, y)](\mathit{mary})(\mathit{john}) \tag{2c}$$

$$\equiv [\lambda y \lambda x\, \mathit{like}(x, y)](\mathit{mary}, \mathit{john}) \tag{2d}$$

Now, β-reduction of L^{λ}_{ar} is in restricted version, and does not imply the referential synonymy: $[\lambda y \lambda x\, \mathit{like}(x)(y)](\mathit{mary})(\mathit{john}) \approx \mathit{like}(\mathit{john})(\mathit{mary})$. Such synonymy is not justified also by our intuitions about the correspondence between English syntax and semantics. I.e.,

$$[\lambda y \lambda x\, \mathit{like}(x)(y)](\mathit{mary})(\mathit{john}) \not\approx \mathit{like}(\mathit{john})(\mathit{mary})$$

which is justified formally and intuitively.

From this point of this paper, unless otherwise stated, we use currying order that complies with the typical order of saturating syntactic arguments in linguistic theories of NL: the verbal head combines at first with its complements, in order to form a phrase with all complements saturated, before saturating its specifier argument. This is motivated at least by the syntax of English language across syntactic categories: S, VP, NP, AP, PP, etc. The subject argument of a verb is its linguistic specifier feature. By such currying order, the semantic representation of the lexicon is more simple and natural.

$$\tau_1 \times \tau_2 \to \tau \equiv (\tau_2 \to (\tau_1 \to \tau)), \text{ where } \tau_1, \tau_2, \tau \in \mathit{Types} \tag{3a}$$
$$A(B_1, B_2) \equiv [A(B_2)](B_1) : \tau \tag{3b}$$
$$\text{where } A : (\tau_2 \to (\tau_1 \to \tau)),\ B_1 : \tau_1,\ B_2 : \tau_2 \tag{3c}$$

While changing the order of currying is not absolutely necessary, because a desirable order of the arguments can be achieved by using λ-abstraction, the lexical items for transitive verbs are more simply rendered by setting the order of the arguments as in (3a)–(3b). Thus,

$$\text{likes} \xrightarrow{\text{render}} \mathit{like} \tag{4a}$$
$$\text{John likes Mary.} \xrightarrow{\text{render}} \tag{4b}$$
$$\mathit{like}(\mathit{mary})(\mathit{john}) \equiv \mathit{like}(\mathit{john}, \mathit{mary}) \tag{4c}$$

2 Semantics within Major CBLG Rules

2.1 Recursion Terms in Feature-value Representation

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^{λ}_{ar}.

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

$$\begin{bmatrix} \text{L-TYPE} & \tau \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_0 \\ \text{WHERE} & \{p_1 := A_1, \ldots, p_n := A_n\} \end{bmatrix} \end{bmatrix} \tag{5}$$

is well-defined iff A_0 is an L^{λ}_{ar} term of type τ and the set of assignments $\{p_1 := A_1, \ldots, p_n := A_n\}$, $n \geq 0$, represented by the value of the feature WHERE, is an

[2] See, Moschovakis [11] for the notion of acyclicity. Intuitively, a system of assignments is acyclic if it does not allow assignments that lead to "loops": $p :\equiv \ldots A(p)$, i.e., it closes-off.

acyclic system of assignments in L_{ar}^{λ}. The above feature structure represents the L_{ar}^{λ} term $A_0 \,\mathsf{where}\, \{p_1 := A_1, \ldots, p_n := A_n\}$, which, in general, is not necessarily in canonical form. And vice versa, any term $A : \tau$ of L_{ar}^{λ} can be represented by such feature structure. In this paper, the formulations of the rules are such that the mother's semantic representation corresponds to a L_{ar}^{λ} term in a canonical form, given that the L_{ar}^{λ} terms corresponding to the daughters nodes are in canonical forms.

In the L_{ar}^{λ} language and its calculus, recursion terms are essential and have sequences of assignments in the scope of the recursion operator, represented by the constant WHERE. In the recursive terms, i.e. the WHERE-terms, the acyclic systems of assignments are sequences, where the order of the assignments is irrelevant because the congruence relation between terms is closed over reordering (which is formalized by permutation) of the assignments. Thus, the sequences of assignments in the WHERE-terms are interpreted as sets. In HPSG frameworks, lists, defined similarly to the list data structure objects in programming languages, are typically used in values of features. The operations *append*, *concatenation* and *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 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}^{λ}. To comply with the tradition of HPSG, we could formulate the rules and the lexicon by stating that the value of the feature 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 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 L-TYPE in the daughter's feature structures determine which of the L_{ar}^{λ} 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 $C : \widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}$, and the term associated with the subject NP is $D : \widetilde{\mathsf{e}}$, the term A associated with the mother S node is $A \equiv C(D) : \widetilde{\mathsf{t}}$. This means that $A \approx \mathsf{cf}(A) \equiv \mathsf{cf}(C(D)) : \widetilde{\mathsf{t}}$. The reduction calculus of L_{ar}^{λ} provides effective procedure for reducing A to its canonical form $\mathsf{cf}(A)$ (modulo congruence): $A \Rightarrow \mathsf{cf}(A)$. However, given the canonical forms $\mathsf{cf}(C)$ and $\mathsf{cf}(D)$ the canonical form $\mathsf{cf}(C(D))$ can be determined directly from the parts of $\mathsf{cf}(C)$ and $\mathsf{cf}(D)$, and the definition of the canonical forms of the terms of L_{ar}^{λ} (Moschovakis [11]).

In this paper, we assume that the lexicon (lexical entries and lexical rules) are formulated to handle feature-value representations of L_{ar}^{λ} 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 [T-HEAD A_0] and [WHERE $\{p_1 := A_1, \dots, p_n := A_n\}$] ($n \geq 0$), represents a L^{λ}_{ar} term A in canonical form: i.e., the term $A \equiv A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\}$ 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 T-HEAD and 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 L-TYPE, which, when matching, should determine the syntactic combination of the terms. Exceptions may require alternative rules, as demonstrated in this paper.. The values of the features T-HEAD and 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^{λ}_{ar} terms and the definition of the canonical form $\mathsf{cf}(A)$ of each term A of L^{λ}_{ar} (Moschovakis [11]). This provides a proof by induction that the well-formed tree structures licenced by the considered rules represent properly typed terms of L^{λ}_{ar} in canonical forms.

2.2 Head Specifier Rule

The Head Specifier Rule (HSR) defines rendering into application terms[3] of L^{λ}_{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):

$$\begin{bmatrix} \textit{phrase} & \\ \text{SYN} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{SPR} & \langle\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_0 \\ \text{WHERE} & U \end{bmatrix} \end{bmatrix} \end{bmatrix} \longrightarrow \boxed{1} \begin{bmatrix} \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T_1 \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_{1,0} \\ \text{WHERE} & U_1 \end{bmatrix} \end{bmatrix} \end{bmatrix} \ \text{H} \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{SPR} & \langle \boxed{1} \rangle \\ \text{COMPS} & \langle\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T_2 \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_{2,0} \\ \text{WHERE} & U_2 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where:

$$T_i \equiv (\sigma \to \tau) \text{ and } T_j \equiv \sigma, \text{ for } i, j \in \{1, 2\} \text{ and } i \neq j, \ T \equiv \tau \text{ and}$$
$$A_0 \text{ where } U \equiv \mathsf{cf}\big([A_{i,0} \text{ where } U_i]([A_{j,0} \text{ where } U_j])\big). \tag{6}$$

There are two sub-cases for the term $[A_0 \text{ where } U]$ in[4] (6) that are determined by the case (CF2) of the definition of the canonical forms of the terms of L^{λ}_{ar} (3.13, Moschovakis [11]):

$$\text{If } A_{j,0} \text{ is immediate, then } A_0 \equiv A_{i,0}(A_{j,0}) \text{ and } U \equiv U_i \equiv U_1 \cup U_2; \tag{7a}$$
$$\text{otherwise, } A_0 \equiv A_{i,0}(q_0) \text{ and } U \equiv \{q_0 := A_{j,0}\} \cup U_1 \cup U_2. \tag{7b}$$

[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.

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 $\mathsf{cf}(A)$ of each term A (see the definition in 3.13, Moschovakis [11]). To avoid binding and free variable clashes, we also assume that, in each application of the grammar rules, the representations of the L^{λ}_{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.

2.3 An Example for a Definite Description

A graph that represents the feature-value description of the syntax of the sentence (8), by including semantic representation, is given in Figure 1, as a tree-like structure.

$$\text{The dog barks.} \tag{8}$$

Technically, co-labeled feature-structure descriptions, e.g., those co-labeled by $\boxed{1}$ (or $\boxed{2}$, etc.), designate unique, single feature-structures. In graphical representations of phrasal analyses, the nodes are labeled by feature-structures. Thus, co-labeled feature-structure descriptions are feature-structures labeling single nodes in labeled graphs that actually are not trees. As a tradition, and for practical reasons of depicting the analyses, the labeled graphs are visualized as trees, by splitting nodes and co-labeling them with the extra box-labels $\boxed{n}$.

2.4 An Example for a NP Quantifier in Subject Position

A graph that represents the feature-value description of the syntax of the sentence (9), by including semantic representation, is given in Figure 2, as a tree-like structure.

$$\text{Every dog barks.} \tag{9}$$

A reduction to a canonical form, like that in (10) modulo congruence, i.e., up to renaming of bound locations and reordering of the assignments, can be effectively found by the rules of the reduction calculus of L^{λ}_{ar}.

$$\mathit{every}(\mathit{dog})(\mathit{bark}) \Rightarrow_{\mathsf{cf}} \mathit{every}(d)(b) \text{ where } \{b := \mathit{bark},\ d := \mathit{dog}\}; \tag{10}$$

On the other hand, the term in canonical form $\mathsf{cf}\big(\mathit{every}(\mathit{dog})(\mathit{bark})\big)$ can be derived step-by-step in the two applications of the HSR with the case (7b), i.e., by (CF2) of the definition of canonical forms.

$$\mathsf{cf}(\mathit{every}) :\equiv \mathit{every} \text{ where } \{\} \tag{11a}$$
$$\mathsf{cf}(\mathit{dog}) :\equiv \mathit{dog} \text{ where } \{\}; \tag{11b}$$
$$\mathsf{cf}(\mathit{bark}) :\equiv \mathit{bark} \text{ where } \{\}; \tag{11c}$$

Fig. 1. A Definite Description

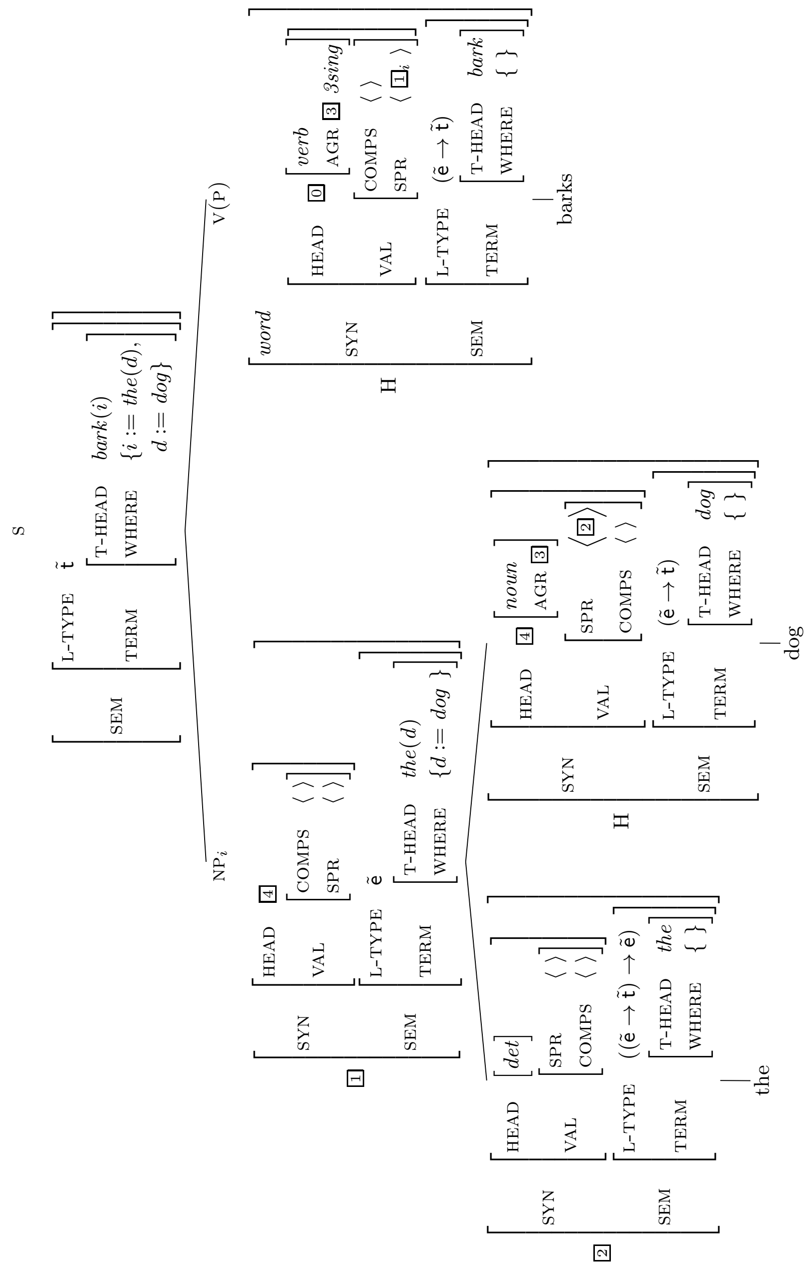
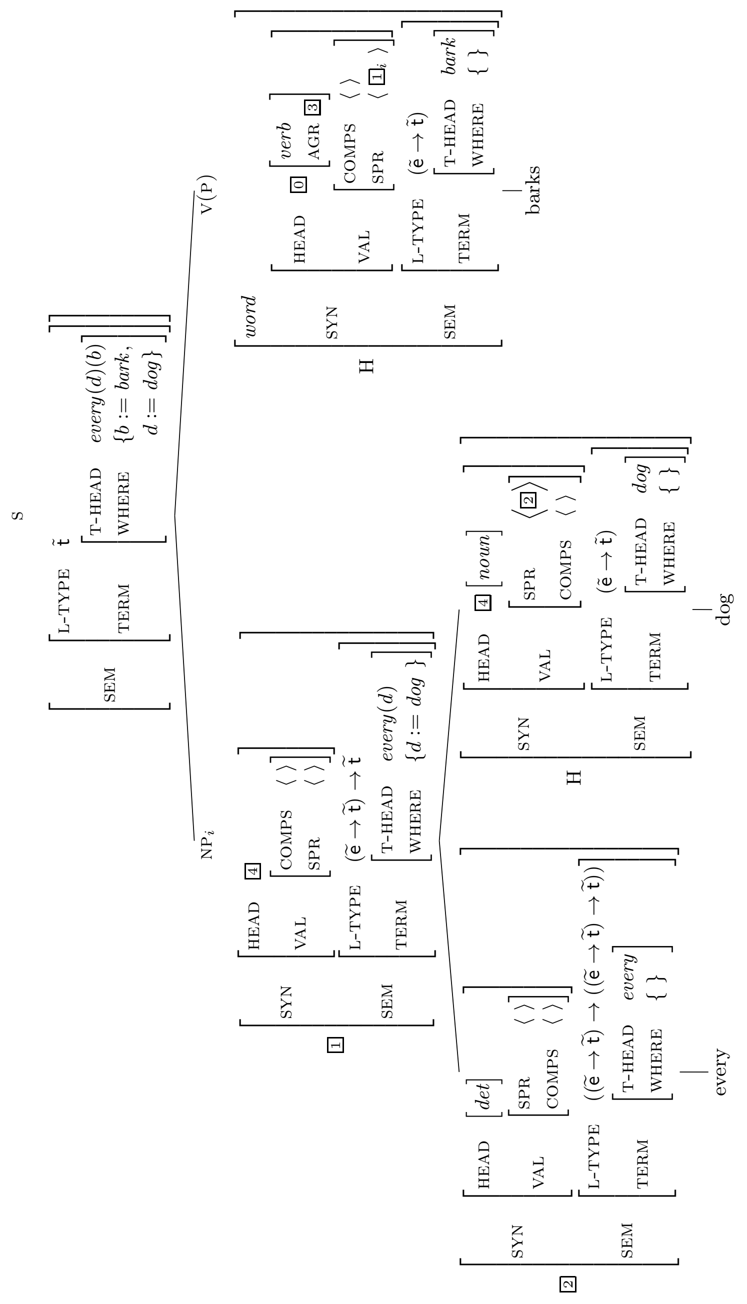


Fig. 2. NP Quantifier in Subject Position



$$\mathsf{cf}\big(\mathit{every}(\mathit{dog})\big) :\equiv \mathit{every}(d) \text{ where } \{d := \mathit{dog}\} \tag{12}$$
$$\text{(by (CF2), (11a) and (11b));}$$

$$\mathsf{cf}\big(\mathit{every}(\mathit{dog})(\mathit{bark})\big) :\equiv \mathit{every}(d)(b) \text{ where } \{d := \mathit{dog},\ b := \mathit{bark}\} \tag{13}$$
$$\text{(by (CF2), (11c) and (12)).}$$

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_{ar}^{λ}:

$$\begin{bmatrix} \mathit{phrase} & \\ \textsc{syn} & \begin{bmatrix} \textsc{val} & \begin{bmatrix} \textsc{comps} & \boxed{a} \end{bmatrix} \end{bmatrix} \\ \textsc{sem} & \begin{bmatrix} \textsc{l-type} & T \\ \textsc{term} & \begin{bmatrix} \textsc{t-head} & A_0 \\ \textsc{where} & U \end{bmatrix} \end{bmatrix} \end{bmatrix} \longrightarrow \text{H}\begin{bmatrix} \textsc{syn} & \begin{bmatrix} \textsc{val} & \begin{bmatrix} \textsc{comps} & \begin{bmatrix} \textsc{first} & \boxed{1} \\ \textsc{rest} & \boxed{a}\begin{bmatrix} \mathit{list} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{sem} & \begin{bmatrix} \textsc{l-type} & T_2 \\ \textsc{term} & \begin{bmatrix} \textsc{t-head} & A_{2,0} \\ \textsc{where} & U_2 \end{bmatrix} \end{bmatrix} \end{bmatrix} \boxed{1}\begin{bmatrix} \textsc{sem} & \begin{bmatrix} \textsc{l-type} & T_1 \\ \textsc{term} & \begin{bmatrix} \textsc{t-head} & A_{1,0} \\ \textsc{where} & U_1 \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{14}$$

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 (6) with its both sub-cases, (7a) and (7b).

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 [COMPS ⟨⟩]. 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*, when it has been "output" by some grammar rule, such as HSR or HCR, and may have partly or entirely saturated COMPS list. Such version of the HCR introduces "semi-phrases" that are not distinguished from proper phrases, for which [COMPS ⟨⟩]. In all cases of complete or partial saturation of the complement list, the statement of the semantic representation remains the same. Something more, the semantic representation in the statement associated with the HCR rule is the resembles that of HSR, (6), with its both sub-cases. In both rules, HCR and HSR, so far, the semantic representation is functional application dictated by respecting the logic types of the terms associated with the daughter's feature structures. Both rules HSR and HCR, as formulates above refer to (6). However, HCR, with the above definition (Version 1) covers only some cases of complement saturation. Thus, there is a need for further analyses, as exemplified in the following section.

Transitive Verbs with Proper Names in Subject and Complement Positions A feature-value description of the syntax of the sentence (15), that includes semantic representations of the syntactic components, is given in Figure 3, as a tree-like structure.

$$\text{Kim hugged Maja.} \tag{15}$$

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^{λ}_{ar} constants of type $(\tilde{\mathsf{e}} \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}))$. On the other hand, NP quantifiers are rendered into terms of type $(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to \widetilde{\mathsf{t}}$. If the head of a VP is a 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 (6) 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):

$$\begin{bmatrix} \textit{phrase} & \\ \text{SYN} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{COMPS} & \boxed{a} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_0 \\ \text{WHERE} & U \end{bmatrix} \end{bmatrix} \end{bmatrix} \longrightarrow \text{H}\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{COMPS} & \begin{bmatrix} \text{FIRST} & \boxed{1}_x \\ \text{REST} & \boxed{a}\begin{bmatrix} \textit{list} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T_2 \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_{2,0} \\ \text{WHERE} & U_2 \end{bmatrix} \end{bmatrix} \end{bmatrix} \boxed{1}\begin{bmatrix} \text{SEM} & \begin{bmatrix} \text{L-TYPE} & T_1 \\ \text{TERM} & \begin{bmatrix} \text{T-HEAD} & A_{1,0} \\ \text{WHERE} & U_1 \end{bmatrix} \\ \text{INDEX} & x \end{bmatrix} \end{bmatrix} \tag{16}$$

where $T_2 \equiv (\tilde{\mathsf{e}} \to (\tilde{\sigma} \to \tilde{\mathsf{t}}))$, $T_1 \equiv ((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to \widetilde{\mathsf{t}})$, and either of the following cases[5]:

$$T \equiv (\tilde{\sigma} \to \tilde{\mathsf{t}}) \text{ and} \tag{17a}$$

$$A_0 \text{ where } U \equiv\ \lambda \boldsymbol{y} A'_{1,0}(p(\boldsymbol{y})) \text{ where } \{p := \lambda \boldsymbol{y}\, \lambda x A'_{2,0}(x)(\boldsymbol{y})\} \cup\ U'_1 \cup U'_2,$$

$$T \equiv (\tilde{\mathsf{q}} \to \tilde{\mathsf{t}}),\ \text{where } \tilde{\mathsf{q}} \equiv ((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to \widetilde{\mathsf{t}}) \tag{17b}$$

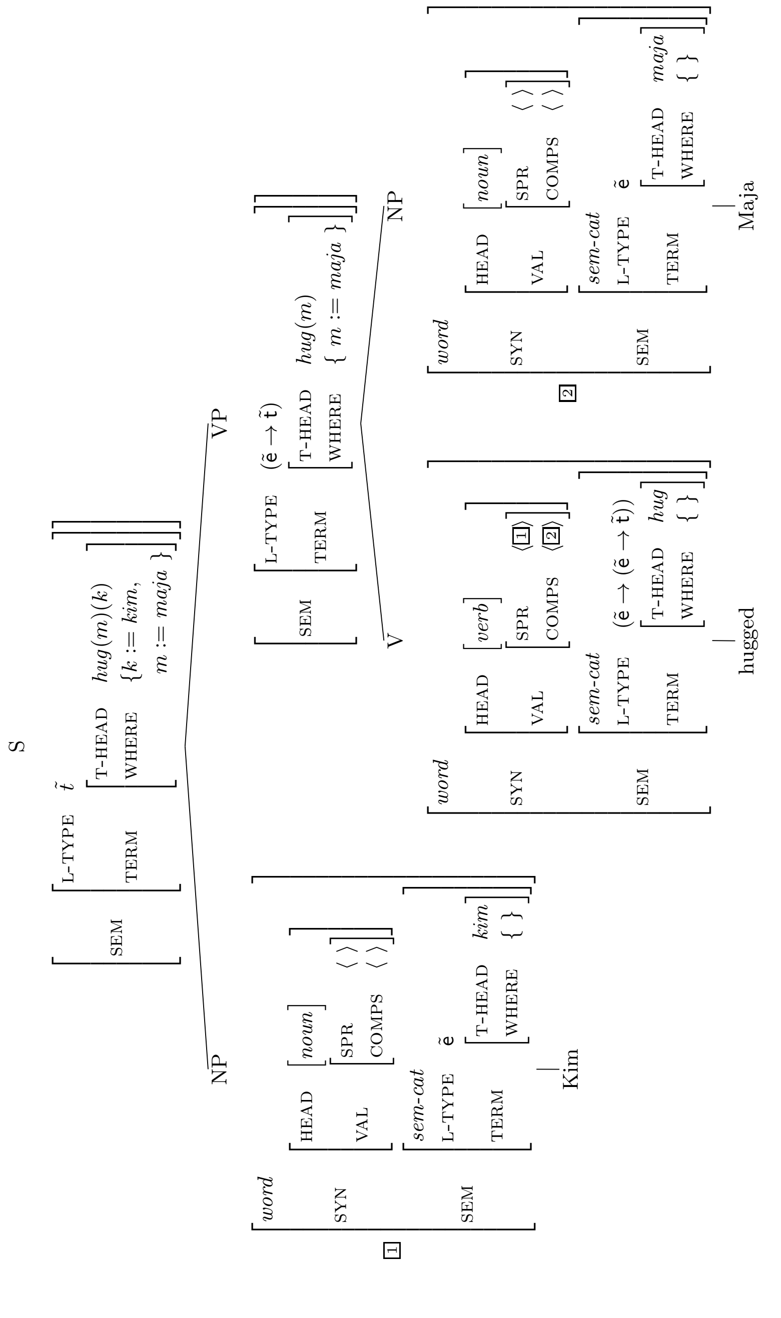
($\tilde{\mathsf{q}}$ is the type of unary quantifiers) and

$$\begin{aligned} A_0 \text{ where } U \equiv\ \lambda Y A'_{1,0}(p(Y)) \text{ where } \{ & p := \lambda Y\, \lambda x[Y(q(Y)(x))], \\ & q := \lambda Y\, \lambda x(\lambda \boldsymbol{y}[A'_{2,0}(x)(\boldsymbol{y})])\} \\ & \cup\ U'_1 \cup U'_2, \end{aligned}$$

where x and the variables in the sequence $\boldsymbol{y}$ 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

[5] We assume here that the terms $\lambda x[A_{2,0}(x)(\boldsymbol{y})]$ and $\lambda x[Y(\lambda \boldsymbol{y}[A_{2,0}(x)(\boldsymbol{y})])]$ are proper, i.e., not immediate. Technically, the special cases with immediate terms should be considered too, but would expand the definition, which is needless in this paper.

Fig. 3. A Transitive Verb with Proper Names as its Subject and Complement



specified in (CF3): i.e., each assignment $r := R$, is replaced by $r' := \lambda Y \lambda x R'$, where r' is a fresh location and the term R' is the result of the substitution

$$R' \equiv R\{r :\equiv r'(Y)(x) \ : \ \text{for all locations } r \text{ in the system of assignments}\}.$$

Note that the co-indexing of the value of the first complement $[\text{FIRST} \ \boxed{1}_x]$ of the head daughter binds, via λ-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:

$$T \equiv (\tilde{\sigma} \to \tilde{\mathsf{t}}) \text{ and} \tag{18a}$$
$$A_0 \ \mathsf{where}\ U \equiv \ \mathsf{cf}\Big(\lambda \boldsymbol{y}\ \big[A_{1,0}\big(\lambda x[A_{2,0}(x)(\boldsymbol{y})]\big) \ \mathsf{where}\ U_1 \cup U_2\big]\Big),$$
$$T \equiv (\tilde{\mathsf{q}} \to \tilde{\mathsf{t}}),\ \text{where } \tilde{\mathsf{q}} \equiv ((\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}}) \tag{18b}$$
$$(\tilde{\mathsf{q}} \text{ is the type of unary quantifiers) and}$$
$$A_0 \ \mathsf{where}\ U \equiv \ \mathsf{cf}\Big(\lambda Y\ \big[A_{1,0}\big(\lambda x[Y(\lambda \boldsymbol{y}[A_{2,0}(x)(\boldsymbol{y})])]\big) \ \mathsf{where}\ U_1 \cup U_2\big]\Big),$$
$$\text{so that in either case } A_{1,0} \text{ applies to } \lambda x\text{-indexed term over}$$
$$\text{the INDEX value of the quantifier } A_{1,0} :\ \boxed{1}(\text{INDEX}) \equiv x.$$

Assuming that the daughters are in canonical forms guarantees direct "derivation" of the canonical form of the L_{ar}^{λ} term represented in the mother feature structure.

Formally (e.g., in a proof of a statement that grammar rules provide canonical representations), the reduction rules that are predominantly in use are: (recap), (head), the compositionality rules, and the corresponding cases of the definition of the canonical forms, see Moschovakis [11], in particular, λ-case (CF3) of that definition. Note that indexing of the correct arguments is maintained by reducing to canonical forms via the λ-rule, respectively, the λ-case (CF3) in the definition of the canonical terms.

The type $T_2 \equiv (\tilde{\mathsf{e}} \to (\tilde{\sigma} \to \tilde{\mathsf{t}}))$ in the HCR Version 2, is the type of the head daughter, in the feature-structure descriptions in the rule. The subtypes $\tilde{\mathsf{e}}$ and $\tilde{\sigma}$ are the types of the arguments that need to be saturated to get to the semantic type of a full sentence, $\tilde{\mathsf{t}}$. When the head structure is a lexical head, which is a transitive verb, i.e. of syntactic type *word*, with HEAD information for the part of speech of appropriate transitive verbs, the leftmost subtype $\tilde{\mathsf{e}}$ in T_2 corresponds to the type of the direct complement, i.e., the first complement of the verbal head. The variable x in (17a) occupies the semantic argument slot for that first complement. More generally, in each application of the HCR, x is for the complement that is to be saturated by the current application of the HCR.

The subtype $\tilde{\sigma}$ is a sequence of types that consists of the types of the terms rendering the complements after the first one, or after the one saturated by the current application of the HCR, followed by the type of the rendering of the subject, which is the rightmost in $\tilde{\sigma}$. The sequence $\boldsymbol{y}$ in (17a) consists of variables that occupy the semantic argument slots for the complements after the one that is saturated by the current application of the HCR, with the subject slot at the rightmost end. In this paper we consider verbs with at most one complement, but the rule is formulated from the perspective of more general usages. Note that the HCR applies to other parts of speech, not only verbs, that require complements, e.g., prepositions, relational common nouns, adjectives, etc.

The sentence (19) is a representative for analyses of simple transitive verbs with one complement that is a quantifier NP.

$$\text{Kim hugged some dog.} \tag{19}$$

The graph (tree-like) representation of the feature-value description of (19), including semantic representations of the components, is given in Figure 4. In Figure 4, the variable sequence $\boldsymbol{y} \equiv y$ corresponds to the grammatical subject that is saturated by HSR, and the type $\tilde{\sigma} \equiv \tilde{\mathsf{e}}$.

If we apply the quantifier term represented by the node (n_4)NP to the term represented by the head verb in the node (n_3)N, which has been λ-indexed with the variable that is the value of the INDEX, i.e., $x_d \equiv \boxed{2}(\text{INDEX})$, the term is:

$$\lambda y\Big[\big[some(d) \text{ where } \{d := dog\}\big]\big(\lambda x_d\, hug(x_d)(y) \text{ where } \{\}\big)\Big] \tag{20a}$$

$$\Rightarrow \lambda y\Big[some(d)\big(\lambda x_d\, hug(x_d)(y)\big) \text{ where } \{d := dog\}\Big] \tag{20b}$$

$$\Rightarrow \lambda y\Big[some(d)\big(h\big) \text{ where } \{d := dog,\ h := \lambda x_d\, hug(x_d)(y)\}\Big] \tag{20c}$$

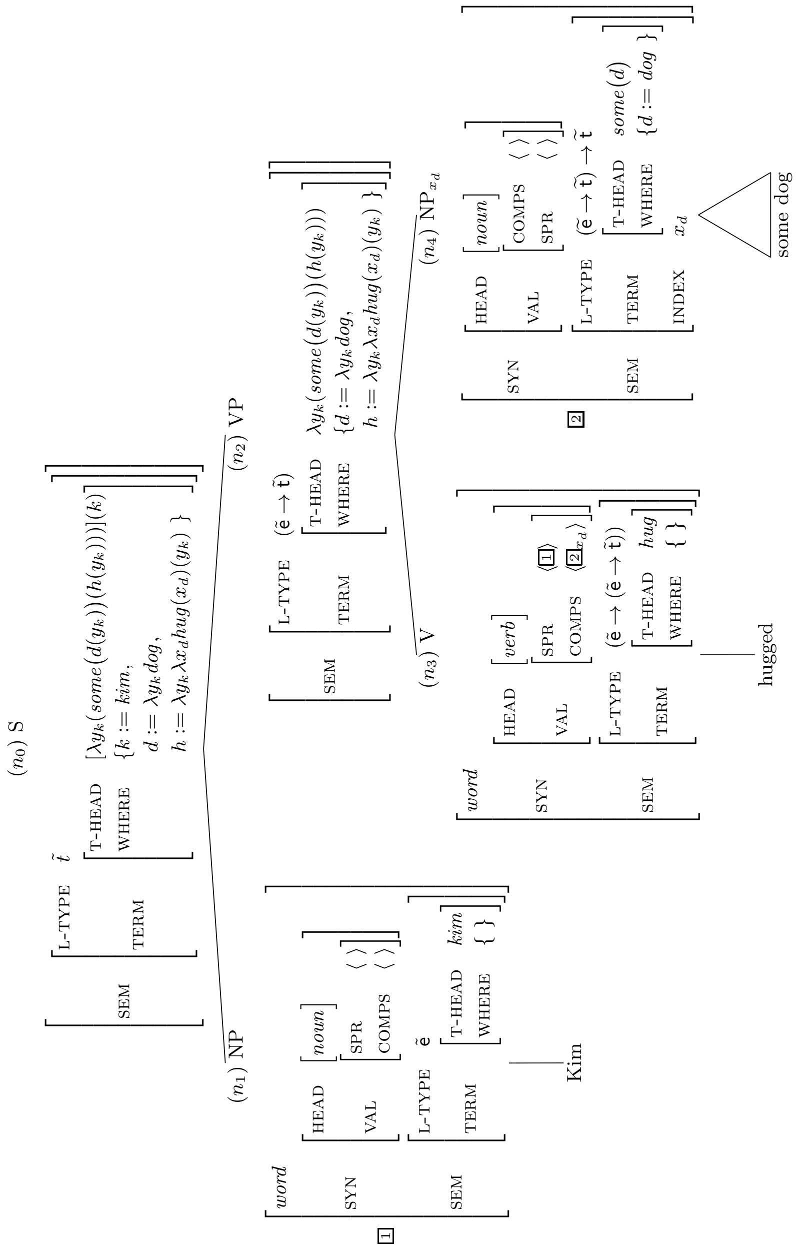
$$\Rightarrow_{\mathsf{cf}} \lambda y\, some(d'(y))\big(h'(y)\big) \tag{20d}$$
$$\text{where } \{d' := \lambda y\, dog,\ h' := \lambda y \lambda x_d\, hug(x_d)(y)\}$$
$$\text{by } (\lambda\text{-rule})$$

$$\equiv_c \lambda y_k\, some\big(d(y_k)\big)\big(h(y_k)\big) \tag{20e}$$
$$\text{where } \{d := \lambda y_k\, dog,\ h := \lambda y_k \lambda x_d\, hug(x_d)(y_k)\}$$
$$\text{by renaming replacement (i.e., congruence)}$$

$$\not\approx \lambda y\, some(d)\big(h(y)\big) \text{ where } \{d := dog,\ h := \lambda y \lambda x_d\, hug(x_d)(y)\} \tag{20f}$$

The term in (20f) is in a canonical form, but it is not referentially synonymous (by the reduction calculus) to any of the other terms (20a)-(20e). In particular, (20f) is not referentially synonymous to the term in (20e). The term (20e) 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: (17a) and (6), respectively. If the complement list had more elements to be saturated, next λ-indexing and quantifier binding can be done with another use of HCR. Note that the term $\lambda y_k\, dog : (\tilde{\mathsf{e}} \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}))$ may seem strange at first glance, but it has a technical role, from a computational perspective. It is resulted by the λ-rule of the reduction calculus. The λ-abstraction is over the variable y_k, which does not occur in the scope of the abstraction, i.e. in the

Fig. 4. Proper Name as Subject and NP Quantifier as Complement



sub-term $dog : (\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})$. Thus, the sub-term $d(y_k)$ of the head part in (20e), 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 (17a) determined by the L-TYPE values of the daughter's terms. Note that the term $[\lambda y_k(some\big(d(y_k)\big)(h(y_k)))](k)$ 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 $[\lambda y_k(some\big(d(y_k)\big)(h(y_k)))](k)$ after the basic components are computed and "stored" in locations k, d and h. In the above cases of the rules, the render relation is defined as compositional syntax-semantics relation.

In this paper, the actual value of $\tilde{\sigma}$, in the HCR, is simple and appropriate for simple transitive verbs, having just one complement, and the semantics of which is "extensional". E.g., the above formulations of the semantic representations do not cover lexical items, such as intensional verbs, that have semantics with attitudes. The semantic representation of verbs with more arguments, e.g., ditransitive ones, will use a sequence of variables $\boldsymbol{y}$ with more than one elements. Note that x is for the complement that is being saturated by this rule; $\boldsymbol{y}$ is a sequence of variables, where the first one (which may be the only one) stands for the next to be saturated. Other possibilities for $\tilde{\sigma}$ are subject of further work.

4.2 Transitive Verbs with Quantifier NPs in Subject and Complement Positions

In the following example, (21), the subject and complement positions of the sentence are filled by NP quantifiers that cause two different readings with different "scoping" of the quantifiers.

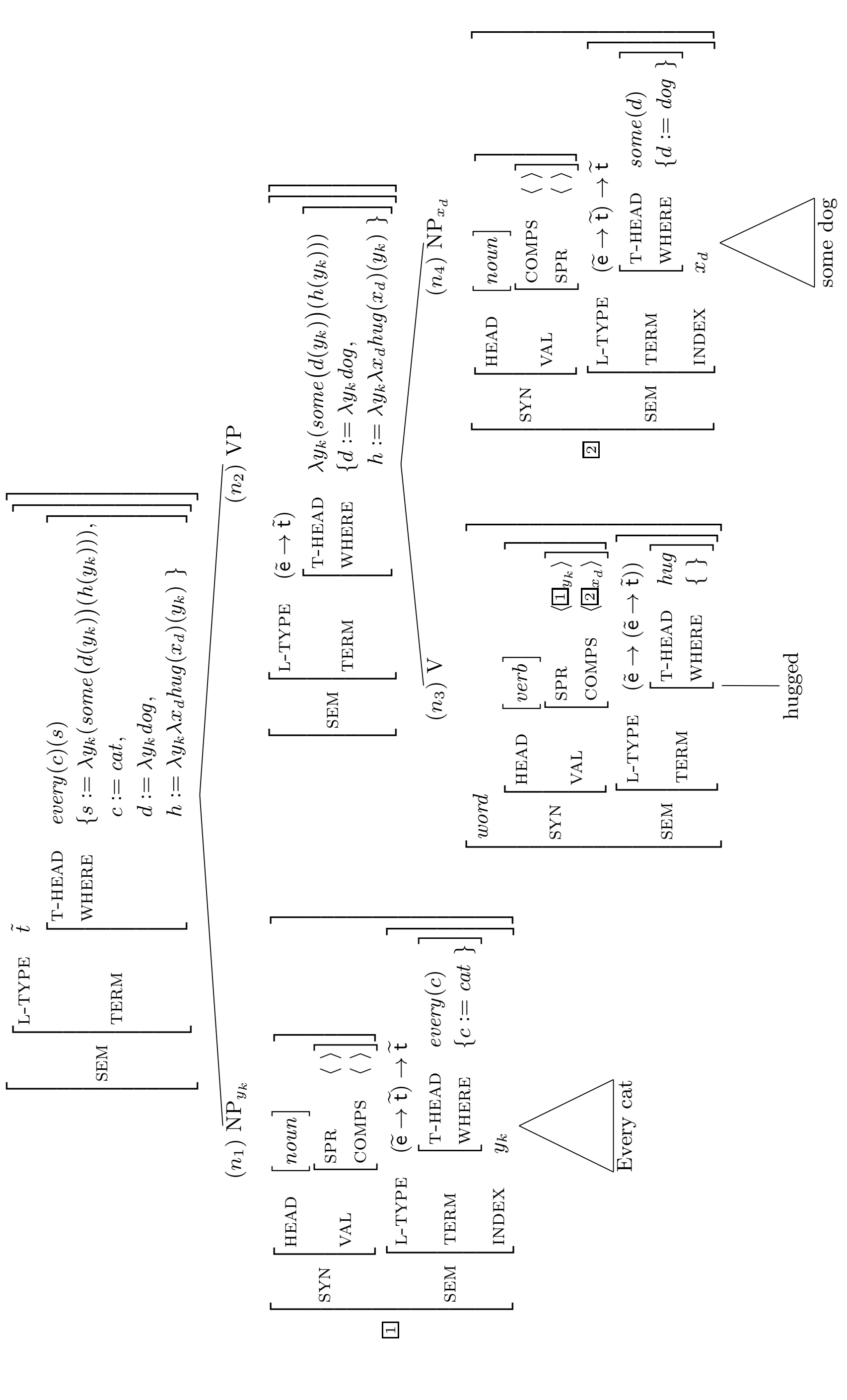
Figure 5 gives the tree-like graph representation of the feature-value syntactic description of (21), that includes semantic representations of the components, and produces the de-dicto reading of the sentence, in accordance with the constraint-based rules that are used. The values of the features L-TYPE, T-HEAD and WHERE, i.e., the parts of the L^{λ}_{ar} term associated with the node (n_2)VP are determined in the same way as the parts of the corresponding node (n_2)VP in the analysis of the previous sentence (19), i.e., by using the HCR with the rendering rule (17a). The parts of the L^{λ}_{ar} term represented by the node (n_0)S are determined by the rendering rule (6), in particular, by its case (7b).

The L^{λ}_{ar} term (22) is associated with the (n_0)S node of the graph in Figure 5. Both, the term (22) and the graph analysis represent the de-dicto reading of the sentence (21).

$\forall\exists$ *Rendering (de-dicto Reading)*

$$\text{Every cat hugged some dog.} \xrightarrow{\text{render}} A \tag{21}$$

Fig. 5. Quantifier NPs as Subject and Complement: de-dicto



(n_0) S
SEM L-TYPE $\tilde{t}$
TERM T-HEAD $every(c)(s)$
WHERE $\{s := \lambda y_k(some(d(y_k))(h(y_k))),$ $c := cat,$ $d := \lambda y_k dog,$ $h := \lambda y_k \lambda x_d hug(x_d)(y_k)\}$
(n_1) NP$_{y_k}$
$\boxed{1}$ SYN HEAD $noun$ VAL SPR $\langle\rangle$ COMPS $\langle\rangle$
SEM L-TYPE $(\tilde{e} \to \tilde{t}) \to \tilde{t}$ TERM T-HEAD $every(c)$ WHERE $\{c := cat\}$ INDEX y_k
Every cat
(n_2) VP
SEM L-TYPE $(\tilde{e} \to \tilde{t})$
TERM T-HEAD $\lambda y_k(some(d(y_k))(h(y_k)))$ WHERE $\{d := \lambda y_k dog,$ $h := \lambda y_k \lambda x_d hug(x_d)(y_k)\}$
(n_3) V
$word$ SYN HEAD $verb$ VAL SPR $\langle \boxed{1}_{y_k} \rangle$ COMPS $\langle \boxed{2}_{x_d} \rangle$
SEM L-TYPE $(\tilde{e} \to (\tilde{e} \to \tilde{t}))$ TERM T-HEAD hug WHERE $\{\}$
hugged
(n_4) NP$_{x_d}$
$\boxed{2}$ SYN HEAD $noun$ VAL COMPS $\langle\rangle$ SPR $\langle\rangle$
SEM L-TYPE $(\tilde{e} \to \tilde{t}) \to \tilde{t}$ TERM T-HEAD $some(d)$ WHERE $\{d := dog\}$ INDEX x_d
some dog

where A is the term

$$\begin{aligned} A \equiv every(c)(s) \text{ where } \{ & s := \lambda y_k (some\big(d(y_k)\big)(h(y_k))), \\ & c := cat, \\ & d := \lambda y_k\, dog, \\ & h := \lambda y_k \lambda x_d hug(x_d)(y_k)\} \end{aligned} \tag{22}$$

Figure 6 is the tree-like graph of the feature-value description of (23), which includes feature-value semantic representation of the de-re reading, in accordance with the constraint-based rules that are used. The L^{λ}_{ar} term associated with the (n_0)S node of the graph in Figure 6, which represents the de-re reading of the sentence, is given in (24).

$\exists\forall$ *Rendering (de-re Reading)*

$$\text{Every cat hugged some dog.} \xrightarrow{\text{render}} A \tag{23}$$

where A is the term

$$\begin{aligned} A \equiv \lambda E(some(d(E))(e(E)))(a) \text{ where } & \\ \{ & a := every(c), \\ & e := \lambda E \lambda x_d\, E(h(E)(x_d)), \\ & c := cat, \\ & d := \lambda E\, dog, \\ & h := \lambda E \lambda x_d \lambda y_k hug(x_d)(y_k)\} \end{aligned} \tag{24}$$

Now, given that *hug* is a constant of L^{λ}_{ar}, the following terms are referentially (i.e., algorithmically) synonymous, but they can not be reduced to each other:

$$hug \approx \lambda x_d \lambda y_k hug(x_d)(y_k) \tag{25}$$

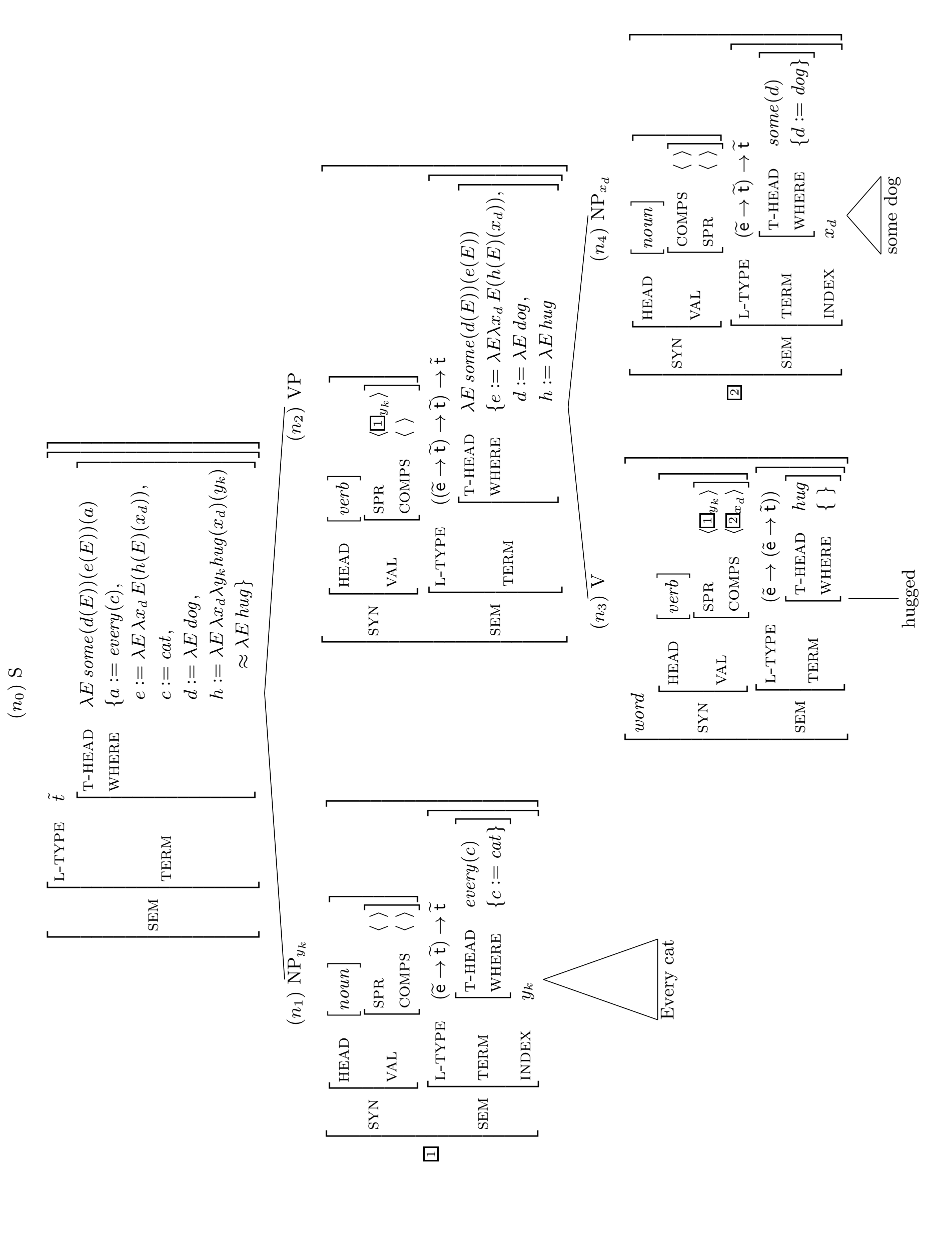
The referential synonymy (25) follows by the Referential Synonymy Theorem (see Moschovakis [11]), because the terms on both sides of the equivalence (25) are explicit, irreducible, and have the same denotations for all variable assignments. Furthermore, by the rules of the calculus of referential synonymy for L^{λ}_{ar} (see p. 30, Moschovakis [11]), it follows that

$$\lambda E hug \approx \lambda E \lambda x_d \lambda y_k hug(x_d)(y_k) \tag{26}$$

and that the term A in (24), which renders the de-re reading of the sentence (23), is referentially (i.e., algorithmically) synonymous with the following term:

$$\begin{aligned} A \approx \lambda E\, some(d(E))(e(E))(a) \text{ where } \{ & a := every(c), \\ & e := \lambda E \lambda x_d\, E(h(E)(x_d)), \\ & c := cat, \\ & d := \lambda E\, dog, \\ & h := \lambda E\, hug\} \end{aligned} \tag{27}$$

Fig. 6. Quantifier NPs as Subject and Complement: de-re



Sub-terms like $\mathit{some}\big(d(y)\big)\big(h(y)\big)$, $\mathit{every}\big(d(y)\big)\big(h(y)\big)$, and, correspondingly, $\lambda y\, \mathit{some}\big(d(y)\big)\big(h(y)\big)$ and $\lambda y\, \mathit{every}\big(d(y)\big)\big(h(y)\big)$, as in (22), (24), and (27), respect the general patterns $Q\big(d(y)\big)\big(h(y)\big)$ and $\lambda y\, Q\big(d(y)\big)\big(h(y)\big)$, for any two-argument quantifier term Q. These terms represent the pattern of the general cases of higher order relations that are two-argument quantifiers Q, where it is possible for both arguments of Q to depend on a common object denoted by the variable y. Typically, in natural languages (i.e., those spoken by humans), renderings $\lambda y\, Q\big(d(y)\big)\big(h(y)\big)(a)$ of sentences with quantified NPs, as in the terms above, one of the arguments of Q does not depend essentially on the common variable y. E.g., in $\lambda E\, \mathit{some}(d(E))(e(E))$, the argument $d(E)$ does not depend strictly on E, because d is bound by $d := \lambda E\, \mathit{dog}$, where dog is a constant.

The following replacement property is valid for referential synonymy, but it is not sufficient to simplify the terms above, in a way that may be desirable, for example, by reducing λ-abstractions like $d\lambda E\, \mathit{dog}$, $\lambda E\, \mathit{hug}$, etc.

Statement 1 (Simple η-Replacement for Referential Synonymy). *For any pure variables $v : \sigma$ and $x : \sigma$, any recursion variable (location) $d : \sigma \to \tau$, and any explicit, irreducible term $D : \tau$ that does not have any (free) occurrences of the variables v and d:*

$$\lambda x d(x) \text{ where } \{d := \lambda v D\} \approx d \text{ where } \{d := \lambda v D\} \tag{28}$$

Proof. Since $d : \sigma \to \tau$ is a variable:

$$\mathsf{den}(\lambda x d(x)) = \mathsf{den}(d) \tag{29}$$

Indeed, by the definition 1.4. [11] of the denotational semantics of L^{λ}_{ar}, for any variable assignment g, $\mathsf{den}(\lambda x(d(x)))(g) = h$, where h is the function such that, for every $a \in \mathbb{T}_\sigma$:

$$\begin{aligned} h(a) &= \mathsf{den}(d(x))(g\{x := a\}) && \text{(by 1.4.D3 [11])} && \text{(30a)}\\ &= [\mathsf{den}(d)(g\{x := a\})](\mathsf{den}(x)(g\{x := a\})) && \text{(by 1.4.D2 [11])} && \text{(30b)}\\ &= [\mathsf{den}(d)(g)](a) && \text{(by 1.4.D1 [11])} && \text{(30c)} \end{aligned}$$

Therefore, $h = \mathsf{den}(d)(g) = \mathsf{den}(\lambda x(d(x)))(g)$, for every assignment g.

The referential synonymy (28) follows by the Referential Synonymy Theorem (see Moschovakis [11]), since the parts of the terms on both sides of (28) have the same denotations.

In fact, the terms on both sides of (28) denote the constant function $x \mapsto \mathsf{den}(D)(g)$, for any (fixed) assignment function g:

$$\mathsf{den}(\lambda x d(x) \text{ where } \{d := \lambda v D\})(g) = \tag{31a}$$

$$\mathsf{den}(d \text{ where } \{d := \lambda v D\})(g) = \tag{31b}$$

$$\mathsf{den}(d)(g\{d := \mathsf{den}(\lambda v D)(g)\}) = \mathsf{den}(\lambda v D)(g) = h, \tag{31c}$$

where $h(a) = \mathsf{den}(D)(g\{v := a\}) = \mathsf{den}(D)(g)$, for every individual a of type σ, since the term $D : \tau$ does not have any free occurrences of the variable $v : \sigma$.

The Simple η-Replacement (28) is not (directly) applicable to the terms that render the analyses of the above NL sentences. To achieve further simplifications that are intuitively desirable, we add the following rule to the set of reduction rules of the calculus of referential synonymy in L^{λ}_{ar} (see p. 30, Moschovakis [11]). Note that this rule does not preserve the referential synonymy.

***η-Reduction Rule** (η-Red). Let*

$$A_0 \text{ where } \{d := \lambda v D,\ p_1 := A_1, \ldots, p_n := A_n\}$$

be a term in canonical form, where $v : \sigma$ is a pure variable and $d : \sigma \to \tau$ is a recursion variable (location), such that

1. *the explicit, irreducible term $D : \tau$ does not have any (free) occurrences of the variables v and d,*
2. *all the occurrences of d in $A_0, \ldots, A_n$ are occurrences of the term $d(x)$, which are in the scope of λx (modulo appropriate renaming of x),*

Then, for any fresh recursion variable $d' : \tau$:

$$A_0 \text{ where } \{d := \lambda v D,\ p_1 := A_1, \ldots, p_n := A_n\} \tag{32a}$$
$$\Rightarrow_\eta A_0\{d(x) :\equiv d'\} \text{ where } \{d' := D,\ p_1 := A_1\{d(x) :\equiv d'\}, \ldots, p_n := A_n\{d(x) :\equiv d'\}\} \tag{32b}$$

where, for each $i \in \{0, \ldots, n\}$, $A_i\{d(x) :\equiv d'\}$ is the result of the replacement of all occurrences of $d(x)$ in A_i with d'.

In general, according to the Referential Synonymy Theorem, the above η-Reduction rule does not preserve referential synonymy. Indeed, the corresponding term parts, in the assignments $d := \lambda v D$ and $d' := D$, are not denotationally equal: $\mathsf{den}(D) \neq \mathsf{den}(\lambda v D)$. Nevertheless, it provides further simplification of canonical terms that are otherwise irreducible. Such simplifications accord with intuitions for preserving all essential computational steps of an algorithm, while simplifying it, by reducing unnecessary computations: vacuous abstraction $d := \lambda v D$, and then application $d(x)$, in case v does not occur in D. Such η-replacement is clearly useful in natural language processing, in particular for reducing complexity of parsing algorithms take care of semantic representations.

The natural isomorphism defined by Moschovakis [11]) is the strictest equivalence relation between recursors, i.e., the referential synonymy "$\approx$" is the strictest equivalence between terms representing algorithms for computing denotations. The set of the reduction rules of the referential synonymy, together with appropriately defined η-Reduction rules (e.g., like the above), induces another equivalence relation between recrsors and, correspondingly equivalence "$\approx_\eta$" between terms. Such equivalence between recursors would be just a little bit less stronger than the strongest intensional synonymy between recursors, but still far more stronger than denotational equivalence. Work on such developments is not in the subject of this paper. Here we have introduced $\Rightarrow_\eta$ to demonstrate the possibility for replacement properties that simplify semantic representations of natural language expressions so that:

1. The subterms that represent semantic components of larger natural language expressions do not use unnecessary abstractions and application operations.
2. The reductions preserve all the algorithmic steps in the compositional syntax-semantics interface, for the parses provided by grammar rules
3. The reductions maintain canonical forms.

Example 1. By η-Reduction, the term (22), repeated below as (33a), simplifies to the term (33b). The explicit, irreducible term *dog* in the assignment $d := \lambda y_k\, dog$, in (33a), does not have any free occurrences of the pure variable y_k. We apply the η-Reduction for $d(y_k)$.

$$\begin{aligned} A \equiv\ & every(c)(s) \text{ where } \{s := \lambda y_k(some\big(d(y_k)\big)(h(y_k))), && (33a)\\ & \qquad c := cat,\ d := \lambda y_k\, dog,\\ & \qquad h := \lambda y_k \lambda x_d hug(x_d)(y_k)\}\\ \Rightarrow_\eta\ & every(c)(s) \text{ where } \{s := \lambda y_k(some\big(d\big)(h(y_k))), && (33b)\\ & \qquad c := cat,\ d := dog,\\ & \qquad h := \lambda y_k \lambda x_d hug(x_d)(y_k)\} \end{aligned}$$

Example 2. In the term (24) repeated below as (34a), the explicit, irreducible terms *dog* and $\lambda x_d \lambda y_k hug(x_d)(y_k)$ do not have any free occurrences of the pure variable E, in the assignments $d := \lambda E\, dog$ and $h := \lambda E \lambda x_d \lambda y_k hug(x_d)(y_k)$, respectively. (34b) is obtained by two applications of η-Reduction for $d(E)$ and $h(E)$, respectively. Furthermore, $\lambda x_d \lambda y_k hug(x_d)(y_k) \approx hug$, thus (34c) follows from (34b), by the rule for the compositionality of recursion, in the the calculus of referential synonymy for L^{λ}_{ar}:

$$\begin{aligned} A \equiv\ & \lambda E(some(d(E))(e(E)))(a) \text{ where} && (34a)\\ & \qquad \{a := every(c),\\ & \qquad e := \lambda E \lambda x_d\, E(h(E)(x_d)),\\ & \qquad c := cat,\ d := \lambda E\, dog,\\ & \qquad h := \lambda E \lambda x_d \lambda y_k hug(x_d)(y_k)\}\\ \Rightarrow_\eta\ & \lambda E(some(d)(e(E)))(a) \text{ where} && (34b)\\ & \qquad \{a := every(c),\\ & \qquad e := \lambda E \lambda x_d\, E(h(x_d)),\\ & \qquad c := cat,\ d := dog,\\ & \qquad h := \lambda x_d \lambda y_k hug(x_d)(y_k)\}\\ \approx\ & \lambda E(some(d)(e(E)))(a) \text{ where} && (34c)\\ & \qquad \{a := every(c),\\ & \qquad e := \lambda E \lambda x_d\, E(h(x_d)),\\ & \qquad c := cat,\ d := dog,\\ & \qquad h := hug \end{aligned}$$

Statement 2 (CBLG Compositionality Statement.). The values of the features T-HEAD and WHERE, in the feature-value pairs [T-HEAD A_0] and [WHERE $\{p_1 := A_1, \ldots, p_n := A_n\}$] ($n \geq 0$), that are associated with the SEM feature, in the well-formed graph (tree-like) structures of the CBLG analyses, represent L^{λ}_{ar} terms in canonical form: A where $\{p_1 := A_1, \ldots, p_n := A_n\}$.

Proof. Sketch of a proof, for a fully specified grammar: by structural induction over well formed tree structures. We assume that the grammar has rules like the ones stated in this paper, and that all lexical and grammar rules, principles, and constraints are formulated so that they preserve semantic representations in canonical forms. We also assume that the lexicon provides *word* feature structures with semantic representations that are L^{λ}_{ar} terms in canonical forms. The proof of this statement is by structural induction that uses the reduction rules and the definition of the canonical terms of L^{λ}_{ar}. Adding η-replacement, at appropriate stages, in rules, principles, or parses, the canonical forms of semantic representations would be maintained.

5 Comparisons and Conclusions

The languages of recursion and their theories, e.g., L^{λ}_{ar}, provide new, elegant mathematical formalization of major concepts of algorithms and language meaning. They introduce computational treatment of concepts that have been in the scope of cross-disciplinary research effort for years, including 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, situations), 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 $\mathsf{cf}(A)$, which codifies all the basic facts and the way they are combined for computing the denotation of A. Thus, reducing terms to canonical forms is important. A formal grammar of NL, that renders NL parsings directly to canonical terms, which represent their semantics, provides not only computational syntax-semantics interface, but also efficiency.

In addition to the algorithmic aspect, the theory of L^{λ}_{ar} provides faithful semantic representation of NL language expressions. The calculus of L^{λ}_{ar} is a proper extension of Gallin's typed logic TY_2 (Gallin [9]), and thus, of various λ-calculi, in particular, those that are embedded (interpreted) in TY_2, e.g., Montague's Intensional Logic (IL). There are natural language sentences that can be rendered into L^{λ}_{ar} terms representing them semantically more adequately, without any equivalent terms in Montague's IL.

There have been various approaches to semantic underspecification by using λ-calculi languages. A classic representative is the technique of Cooper's semantic storage (see [4]), which was developed into many variants. There have been other approaches, too, in particular, Muskens [14], which was followed more recently by Bos [2], Copestake et al. [5], Egg [8], Richter and Sailer [16], and others.

Bunt [3] gives a comprehensive overview of the field. A distinctive feature of the language and calculus of L_{ar}^{λ} is that it provides representation of language underspecification inherently, at its object level, that is also in the spirit of modeling meanings by algorithms, i.e., as abstract mathematical objects, which have syntactical representatives by terms in canonical forms. An introduction to how this can be done is given in Loukanova [10]. A comprehensive retrospective on this subject is for further work, in particular, for development of new approach to CBLG that takes type-theoretic grammar approach in full consideration.

References

1. Barwise, Jon and John Perry. *Situations and Attitudes.* Cambridge, MA:MIT press. Republished in 1999 by The David Hume Series of Philosophy and Cognitive Science Reissues. (1983)
2. Bos, Johan *Computational Semantics in Discourse: Underspecification, Resolution, and Inference.* Journal of Logic, Language and Information 13(2): 139–157. (2004)
3. Bunt, Harry. Semantic Underspecification: Which Technique for What Purpose? In: Harry Bunt and Reinhard Muskens (eds) *Computing Meaning.* Volume 3. p.55–85. Studies in Linguistics and Philosophy 83. Springer, Dordrecht. (2007)
4. Cooper, Robin. *Quantification and Syntactic Theory.* D. Reidel. Dordrecht, The Netherlands. (1983)
5. Copestake, Ann, Dan Flickinger, Carl Pollard, and Ivan A. Sag. *Minimal Recursion Semantics: an Introduction.* Research on Language and Computation 3.4: p.281–332. (2006)
6. Devlin, Keith. *Logic and information.* Cambridge University Press. (1991)
7. Situation theory and situation semantics. In *Handbook of the history of logic.* (eds) Gabbay, Dov and John Woods. Volume 7. p.601–664. Elsevier. (2008)
8. Egg, Markus and Gisela Redeker. *Underspecified discourse representation.* In: Anton Benz and Peter Kühnlein (eds), Constraints in Discourse, Amsterdam: Benjamins (Pragmatics & Beyond), 117–138. (2008)
9. Gallin, D. *Intensional and Higher-Order Modal Logic.* North-Holland. (1975)
10. Loukanova, Roussanka. *Typed Lambda Language of Acyclic Recursion and Scope Underspecification.* In: Reinhard Muskens. (ed.) Workshop on New Directions in Type-theoretic Grammars. August 6-10, 2007. Dublin, Ireland. Proceedings, p.73–89. (2007)
11. Moschovakis, Yiannis N. *A logical calculus of meaning and synonymy.* Linguistics and Philosophy, v. 29. p.27–89 (2006)
12. Pollard, Carl, and Ivan A. Sag. *Information-Based Syntax and Semantics.* Volume 1: Fundamentals. Stanford: CSLI Publications.
13. Pollard, Carl, and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* Chicago: University of Chicago Press. (1994)
14. Muskens, R. *Underspecified semantics.* In: Urs Egli and Klaus von Heusinger. (eds). *Reference and Anaphoric Relations.* Studies in Linguistics and Philosophy 72. p.311–338. Kluwer. (1999).
15. Sag, Ivan A., Thomas Wasow, and Emily M. Bender. *Syntactic Theory — A Formal Introduction.* 2nd Ed. Stanford: CSLI Publications. (2003)
16. Richter, Frank and Manfred Sailer. *Basic Concepts of Lexical Resource Semantics.* In: Beckmann, Arnold and Norbert Preining. (eds) ESSLLI 2003 – Course Material I. Collegium Logicum. Volume 5. Kurt Gödel Society Wien. p.87–143. (2004)

17. Sailer, Manfred. *Combinatorial Semantics and Idiomatic Expressions in Head-Driven Phrase Structure Grammar.* Doctoral dissertation. University of Tübingen. Appeared as Arbeitspapiere des SFB 340, Nr. 161. Universität Stuttgart and Universität Tübingen. (2003)