

Comparing the Use of Feature Structures in Nativism and in Database Semantics

Roland Hausser

Universität Erlangen-Nürnberg
Abteilung Computerlinguistik (CLUE)
rrh@linguistik.uni-erlangen.de

Abstract

Linguistics has always been a field with a great diversity of schools and sub-schools. This has naturally led to the question of whether different grammatical analyses of the same sentence are in fact equivalent or not. With the formalization of grammars as generative rule systems, beginning with the “Chomsky revolution” in the late nineteen fifties, it became possible to answer such questions in those fortunate instances in which the competing analyses were sufficiently formalized.

An early example is the comparison of Context-Free Phrase Structure Grammar (CF-PSG) and Bidirectional Categorical Grammar (BCG), which were shown to be weakly equivalent by Gaifman 1961. More recently, the question arose with respect to the language classes and the complexity hierarchies of Phrase Structure Grammar (PS-grammar) and of Left-Associative Grammar (LA-grammar), which were shown to be orthogonal to each other (TCS’92).

Here we apply the question to the use of feature structures in contemporary schools of Nativism on the one hand, and in Database Semantics (DBS) on the other. The practical purpose is to determine whether or not the grammatical analyses of Nativism based on constituent structure can be used in Database Semantics.

1 Introduction: Constituent Structure in Nativism

In contemporary linguistics, most schools are based on constituent structure analysis. Examples are GB (Chomsky 1981), LFG (Bresnan ed. 1982), GPSG (Gazdar et al. 1985), and HPSG (Pollard and Sag 1987, 1994). Related schools are DCG (Pereira and Warren 1980), FUG (Kay 1992), TAG (Vijay-Shanker and Joshi 1988), and CG (Kay 2002). For historical reasons and because of their similar goals and methods, these schools may be jointly referred to as variants of Nativism.¹

Constituent structure is defined in terms of phrase structure trees which fulfill the following conditions:

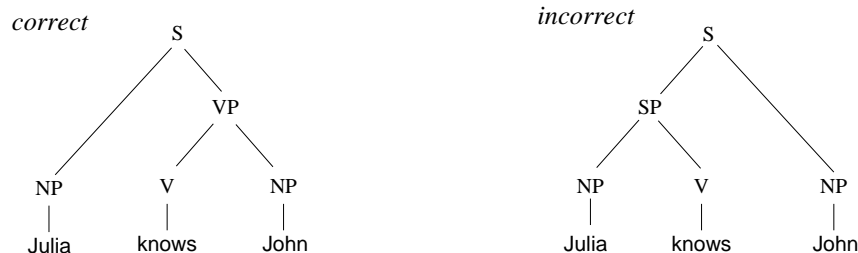
1.1 DEFINITION OF CONSTITUENT STRUCTURE

1. Words or constituents which belong together semantically must be dominated directly and exhaustively by a node.
2. The lines of a constituent structure may not cross (non-tangling condition).

¹Nativism is so-called because it aims at characterizing the speaker-hearer’s *innate* knowledge of language (competence) – excluding the use of language in communication (performance).

According to this definition, the first of the following two phrase structure trees is a linguistically correct analysis, while the second is not:

1.2 CORRECT AND INCORRECT CONSTITUENT STRUCTURE ANALYSIS

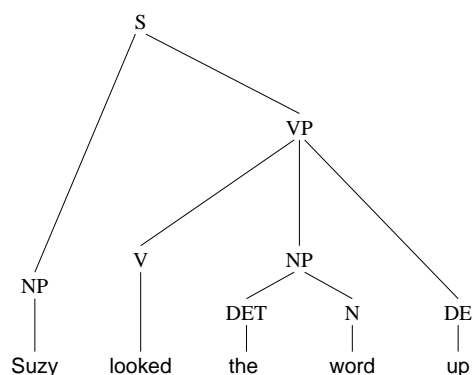


There is common agreement among Nativists that the words *knows* and *John* belong more closely together semantically than the words *Julia* and *knows*.² Therefore, only the tree on the left is accepted as a correct grammatical analysis.

Formally, however, both phrase structure trees are equally well-formed. Moreover, the number of possible trees grows exponentially with the length of the sentence.³ The problem is that such a multitude of phrase structure trees for the same sentence would be meaningless *linguistically*, if they were all equally correct.

It is for this reason that constituent structure as defined in 1.1 is crucial for phrase structure grammar (PS-grammar): constituent structure is the only known principle⁴ for excluding most of the possible trees. Yet it has been known at least since 1960 (cf. Bar-Hillel 1964, p. 102) that there are certain constructions of natural language, called “discontinuous elements,” which do not fulfill the definition of constituent structure. Consider the following examples:

1.3 CONSTITUENT STRUCTURE PARADOX: VIOLATING CONDITION 1



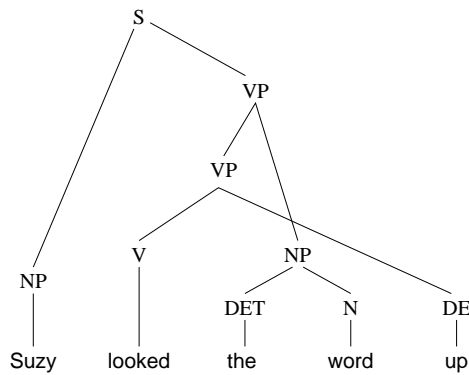
Here the lines do not cross, satisfying the second condition of Definition 1.1. The analysis violates the first condition, however, because the semantically related expressions *looked – up*, or rather the nodes V (verb) and DE (discontinuous element) dominating them, are not *exhaustively* dominated by a node. Instead, the node directly dominating V and DE also dominates the NP *the word*.

²To someone not steeped in Nativist linguistics, these intuitions may be difficult to follow. They are related to the substitution tests of Z. Harris, who was Chomsky’s teacher.

³If loops like $A \rightarrow \dots A$ are permitted in the rewrite rules, the number of different trees over a finite sentence is infinite!

⁴Historically, the definition of constituent structure is fairly recent, based on the movement and substitution tests of American Structuralism in the nineteen thirties and forties.

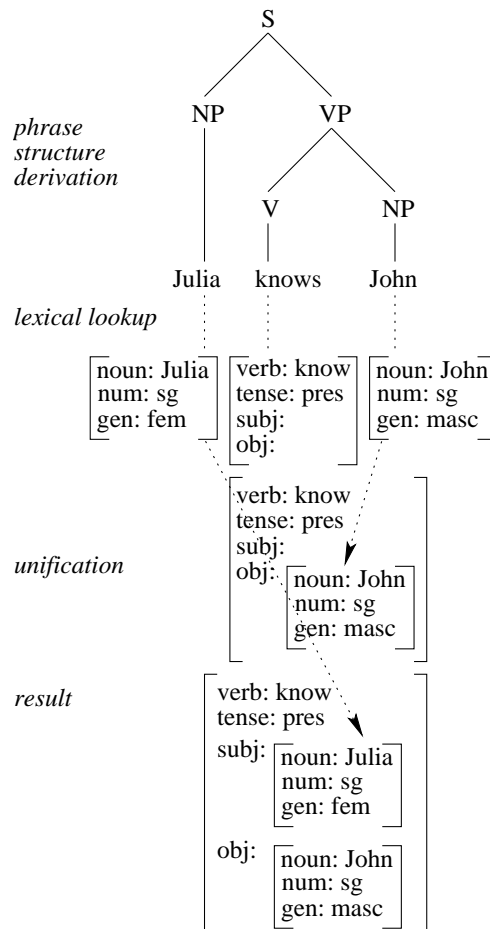
1.4 CONSTITUENT STRUCTURE PARADOX: VIOLATING CONDITION 2



Here the semantically related subexpressions *looked* and *up* are dominated directly and exhaustively by a node, thus satisfying the first condition of Definition 1.1. The analysis violates the second condition, however, because the lines in the tree cross.

Rather than giving up constituent structure as the intuitive basis of their analysis, the different schools of Nativism extended the formalism of context-free phrase structure with additional structures and mechanisms like transformations (Chomsky 1965), f-structures (Bresnan ed. 1982), meta-rules (Gazdar et al. 1985), constraints (Pollard and Sag 1987, 1994), the adjoining of trees (Vijay-Shanker and Joshi 1988), etc. In recent years, these efforts to extend the descriptive power of context-free phrase structure grammar have converged in the widespread use of recursive feature structures with unification. Consider the following example, which emphasizes what is common conceptually to the different variants of Nativism.

1.5 RECURSIVE FEATURE STRUCTURES AND UNIFICATION



As in 1.2 (correct tree), the analysis begins with the start symbol *S*, from which the phrase structure tree is derived by substituting NP and VP for *S*, etc., until the terminal nodes *Julia*, *knows*, and *John* are reached (*phrase structure derivation*). Next the terminal nodes are replaced by feature structures via *lexical lookup*. Finally, the lexical feature structures are unified (indicated by the dotted arrows), resulting in one big recursive feature structure (*result*). The order of unification mirrors the derivation of the phrase structure tree.

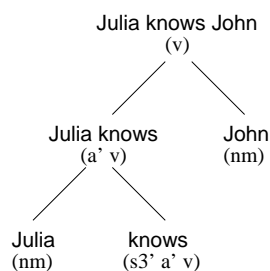
On the one hand, the use of feature structures provides for many techniques which go beyond the context free phrase structure tree, such as a differentiated lexical analysis, structure sharing (a.k.a. token identity), a truth-conditional semantic interpretation based on lambda calculus, etc. On the other hand, this method greatly increases the mathematical complexity from polynomial to exponential or undecidable. Also, the constituent structure paradox, as a violation of Definition 1.1, remains.

2 Elimination of Constituent Structure in LA-grammar

Instead of maintaining constituent structure analysis when it is possible (e.g. 1.2, correct tree) and taking exception to it when it is not (e.g. 1.3), Left-Associative Grammar completely abandoned constituent structure as defined in 1.1 by adopting another, more basic principle. This principle is the time-linear structure of natural language – in accordance with de Saussure’s 1913/1972 second law (*principe seconde*). Time-linear means linear like time and in the direction of time.

Consider the following reanalysis of Example 1.2 within Left-Associative Grammar (LA-grammar) as presented in NEWCAT’86:

2.1 TIME-LINEAR ANALYSIS OF *Julia knows John* IN LA-GRAMMAR

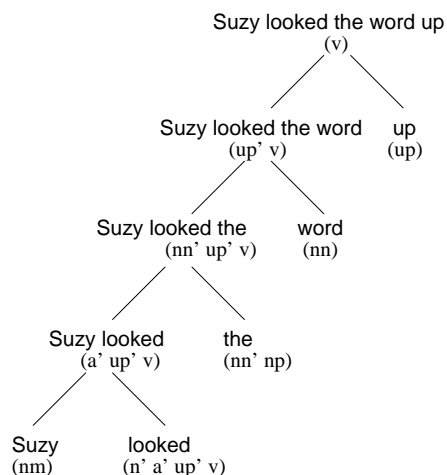


Given an input sentence or a sequence of input sentences (text), LA-grammar always combines a “sentence start,” e.g. *Julia*, and a “next word,” e.g. *knows*, into a new sentence start, e.g. *Julia knows*. This time-linear procedure starts with the first word and continues until there is no more next word available in the input.

In LA-grammar, the intuitions about what “belongs semantically together” (which underlie the definition of constituent structure 1.1) are reinterpreted in terms of *functor-argument* structure and coded in categories which are defined as lists of one or more category segments. For example, in 2.1 the category segment *nm* (for name) of *Julia* cancels the first valency position *s3'* (for nominative singular third person) of the category *(s3' a' v)* of *knows*, whereby *Julia* serves as the argument and *knows* as the functor. Then the resulting sentence start *Julia knows* of the category *(a' v)* serves as the functor and *John* as the argument. The result is a complete sentence, represented as a verb without unfilled valency positions, i.e., as the category *(v)*.

Next consider the time-linear reanalysis of the example with a discontinuous element (cf. 1.3 and 1.4):

2.2 TIME-LINEAR ANALYSIS OF Suzy looked the word up



Here the discontinuous element *up* is treated like a valency filler for the valency position *up'* in the lexical category (*n' a' up' v*) of *looked*. Note the strictly time-linear addition of the “constituent” *the word*: the article *the* has the category (*nn' np*) such that the category segment *np* cancels the valency position *a'* in the category (*a' up' v*) of *Suzy looked*, while the category segment *nn'* is added in the result category (*nn' up' v*) of *Suzy looked the*. In this way, the obligatory addition of a noun after the addition of a determiner is ensured.

The time-linear analysis of LA-grammar is based on formal rules which compute *possible continuations*. Consider the following example (explanations in italics):

2.3 EXAMPLE OF AN LA-GRAMMAR RULE APPLICATION

(i) <i>rule name</i>	(ii) <i>ss</i>	(iii) <i>nw</i>	⇒	(iv) <i>ss'</i>	(v) <i>RP</i>	
Nom+Fverb:	(NP)	(NP' X V)		(X V)	{Fverb+Main, ...}	<i>matching and binding</i>
	(nm)	(s3' a' v)		(a' v)		
	Julia	knows		Julia knows		

An LA-grammar rule consists of (i) a *rule name*, here *Nom+Fverb*, (ii) a pattern for the sentence start *ss*, here (NP), (iii) a pattern for the next word *nw*, here (NP' X V), (iv) a pattern for the resulting sentence start *ss'*, here (X V), and (v) a rule package *RP*, here {Fverb+Main, ...}. The patterns for (ii) *ss*, (iii) *nw*, and (iv) *ss'* are coded by means of restricted variables, which are matched and vertically bound with corresponding category segments of the language input. For example, in 2.3 the variable NP at the rule level is bound to the category segment nm at the language level, the variable NP' is bound to the category segment s3', etc.

If the matching of variables fails with respect to an input (because a variable restriction is violated), the rule application fails. If the matching of variables is successful, the *categorial operation* (represented by (ii) *ss*, (iii) *nw*, and (iv) *ss'*) is performed and a new sentence start is derived. That the categorial operation defined at the rule level can be executed at the language level is due to the *vertical binding* of the rule level variables to language level constants. After the successful application of an LA-grammar rule, the rules in its (v) rule package *RP* are applied to the resulting sentence start (iv) *ss'* and a new next word.

A crucial property of LA-grammar rules is that they have an *external interface*, defined in terms of the rule level variables and their vertical matching with language level category segments. This is in contradistinction to the rewrite rules of phrase structure grammar: they do not have any external interface because all phrase structure trees are derived from the same initial S node, based on the principle of possible substitutions.

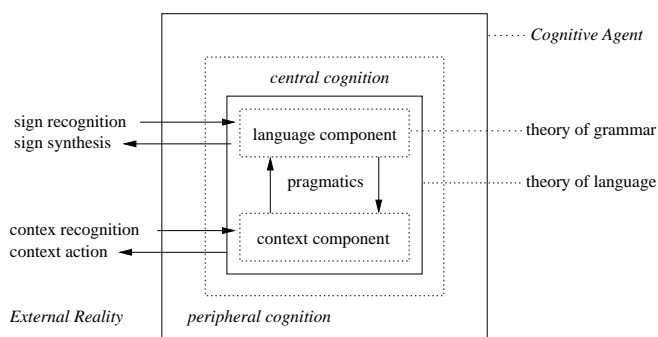
3 From LA-grammar to Database Semantics

The external interfaces of LA-grammar rules, originally introduced for computing the possible continuations of a time-linear derivation, open the transition from a sign-oriented approach to an agent-oriented approach of natural language analysis.⁵ While a sign-oriented approach analyses sentences in isolation, an agent-oriented approach analyses sentences as a means to transfer information from the mind of the speaker to the mind of the hearer. In Database Semantics, LA-grammar is used for an agent-oriented approach to linguistics which aims at building an artificial cognitive agent (talking robot).

This requires the design of (i) *interfaces* for recognition and action, (ii) a *data structure* suitable for storing and retrieving content, and (iii) an *algorithm* for (a) reading content in during recognition, (b) processing content during thought, and (c) reading content out during action. Moreover, the data structure must represent non-verbal cognition at the context level as well as verbal cognition at the language level. Finally, the two levels must interact in such a way as to model the speaker mode (mapping from the context level to the language level) and the hearer mode (mapping from the language level to the context level).

Consider the representation of these requirements in the following schema:

3.1 STRUCTURING CENTRAL COGNITION IN AGENTS WITH LANGUAGE



The interfaces of recognition and action are based on pattern matching. At the context level, the patterns are defined as concepts, which are also used for coding and storing content. At the language level, the concepts of the context level are reused as the literal meanings of content words. In this way, the lexical semantics is based on procedurally defined concepts rather than the metalanguage definitions of a truth-conditional semantics (cf. NLC’06, Chapter 2 and Section 6.2).

The data structure for coding and storing content at the context level is based on flat (non-recursive) feature structures called *proplets* (in analogy to “droplets”). Proplets are so-called because they serve as the basic elements of concatenated *propositions*. Consider the following example showing the simplified proplets representing the content resulting from an agent perceiving a barking dog (recognition) and running away (action):

3.2 CONTEXT PROPLETS REPRESENTING *dog barks. (I) run.*

$\left[\begin{array}{l} \text{sur:} \\ \text{noun: } \textit{dog} \\ \text{fnc: bark} \\ \text{prn: 22} \end{array} \right]$	$\left[\begin{array}{l} \text{sur:} \\ \text{verb: } \textit{bark} \\ \text{arg: dog} \\ \text{nc: 23 run} \\ \text{prn: 22} \end{array} \right]$	$\left[\begin{array}{l} \text{sur:} \\ \text{verb: } \textit{run} \\ \text{arg: moi} \\ \text{pc: 22 bark} \\ \text{prn: 23} \end{array} \right]$
---	--	--

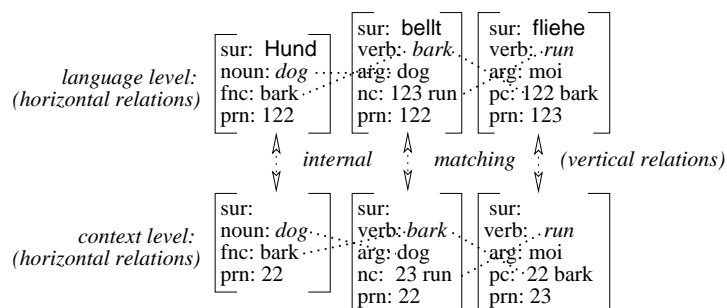
⁵Clark 1996 distinguishes between the *language-as-product* and the *language-as-action* traditions.

The semantic relation between the first two proplets is intrapropositional functor-argument structure, and is coded as follows: The first proplet with the core feature [noun: *dog*] specifies the associated functor with the intrapropositional continuation feature [fnc: *bark*], while the second proplet with the core feature [verb: *bark*] specifies its associated argument with [arg: *dog*] (bidirectional pointering). That the first and the second proplet belong to the same proposition is indicated by their having the same prn (proposition number) value, namely 22.

The semantic relation between the second and the third proplet is extrapropositional coordination. That these two proplets belong to different propositions is indicated by their having different prn values, namely 22 and 23, respectively. Their coordination relation is coded in the second proplet by the extrapropositional continuation feature [nc: 23 run] and in the third proplet by [pc: 22 bark], whereby the attributes nc and pc stand for “next conjunct” and “previous conjunct,” respectively. The values of the nc and pc attributes are the proposition number and the core value of the verb of the coordinated proposition.

By coding the semantic relations between proplets solely in terms of attributes and their values, proplets can be stored and retrieved according to the needs of one’s database, without any of the graphical restrictions induced by phrase structure trees. Furthermore, by using similar proplet at the levels of language and context, the matching between the two levels during language interpretation (hearer mode) and language production (speaker mode) is structurally straightforward. Consider the following example in which the context level content of 3.2 is matched with corresponding language proplets containing German surfaces:

3.3 MATCHING BETWEEN THE LANGUAGE AND THE CONTEXT LEVEL



The proplets at the language and the context level are alike except that the SUR (surface) attributes of context proplets have an empty value, while those of the language proplets have a language-dependent surface, e.g. *Hund*, as value.

On both levels, the intra- and extrapropositional relations are coded by means of attribute values (horizontal relations, indicated by dotted lines). The reference relation between corresponding proplets at the two levels, in contrast, is based on matching (vertical relations, indicated by double arrows). Simply speaking, the matching between a language and a context proplet is successful if they have the same attributes and their values are compatible.

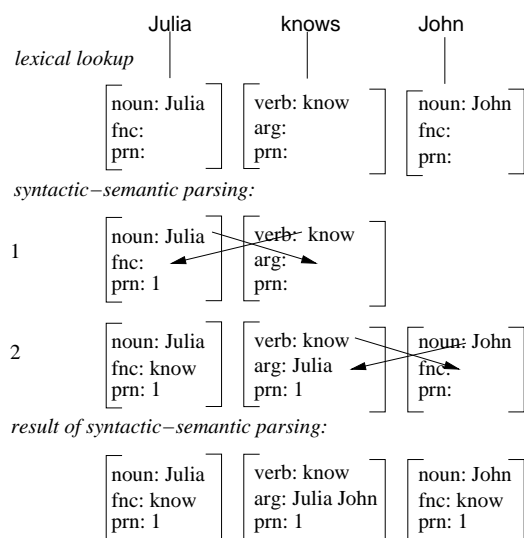
Even though the vertical matching takes place between individual proplets, the horizontal semantic relations holding between the proplets at each of the two levels are taken into account as well. Assume, for example, that the noun proplet *dog* at the language level has the fnc value *bark*, while the corresponding proplet at the context level had the fnc value *sleep*. In this case, the two proplets would be vertically incompatible – due to their horizontal relations to different verbs, coded as different values of their respective fnc attributes.

Having described the data structure of Database Semantics, let us turn next to its algorithm. For natural language communication, the time-linear algorithm of LA-grammar is used in three different variants: (i) in the hearer mode, an *LA-hear* grammar interprets sentences of natural language as sets of proplets ready to be stored in the database of the cognitive agent,

(ii) in the think mode, an *LA-think* grammar navigates along the semantic relations between proplets, and (iii) in the speaker mode an *LA-speak* grammar verbalizes the proplets traversed in the think mode as surfaces of a natural language.

Consider the following LA-hear derivation of *Julia knows John* in Database Semantics.

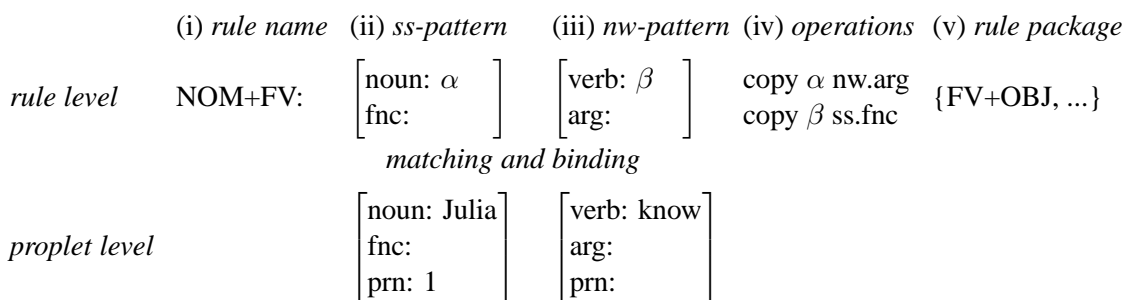
3.4 TIME-LINEAR HEARER-MODE ANALYSIS OF *Julia knows John*



This derivation is similar to 2.1 in that it is strictly time-linear. The differences are mostly in the format. While 2.1 must be read bottom up, 3.4 starts with the lookup of lexical proplets and must be read top down. Furthermore, while the *ss* and *nw* in 2.1 each consist of a surface and a category defined as a list, the *ss* and *nw* in 3.4 consist of proplets. Finally, while the output of 2.1 is the whole derivation (like a tree in a sign-oriented approach), the output of 3.4 is a set of proplets (cf. *result*) ready to be stored in the database.

The rules of an LA-hear grammar have patterns for matching proplets rather than categories (as in 2.3). This is illustrated by the following example (explanations in italics):

3.5 EXAMPLE OF AN *LA-hear* RULE APPLICATION



This rule resembles the one illustrated in 2.3 in that it consists of (i) a rule name, (ii) a pattern for the *ss*, (iii) a pattern of the *nw*, and (v) a rule package. It differs from 2.3, however, in that the resulting sentence start (iv) *ss*' is replaced by a set of *operations*.

During matching, the variables, here α and β , of the rule level are vertically bound to corresponding values at the proplet level. This is the basis for executing the rule level operations at the proplet level. In 3.5, the operations code the functor-argument relation between the subject and the verb by copying the core value of the noun into the *arg* slot of the verb and the core value of the verb into the *fnc* slot of the noun. In the schematic derivation 3.4, the copying is indicated by the arrows. The result of the rule application 3.5 is as follows:

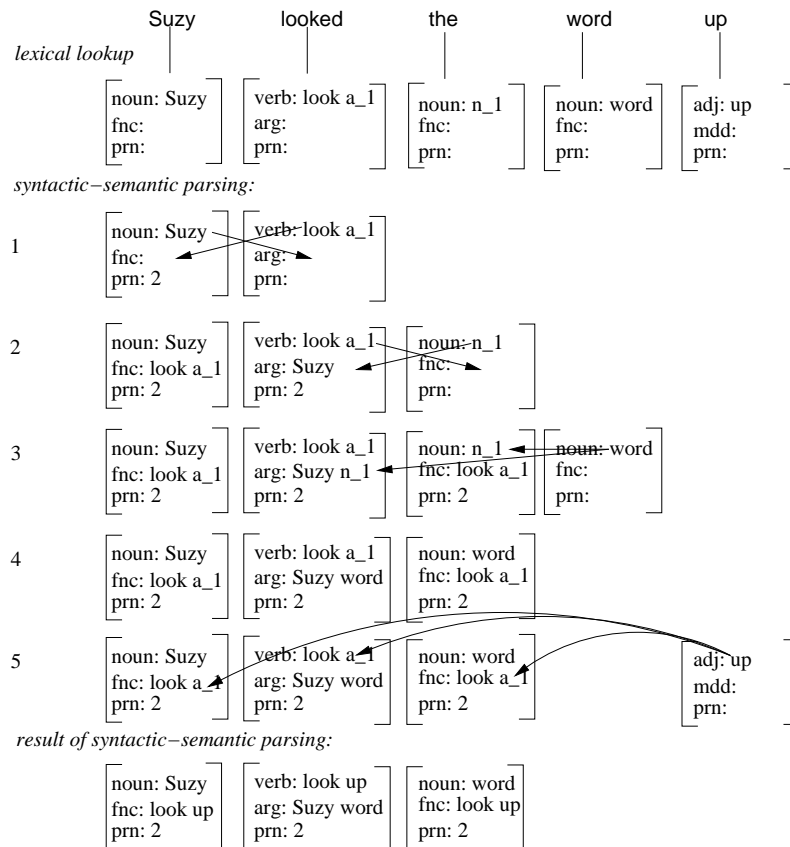
3.6 RESULT OF THE *LA-hear* RULE APPLICATION SHOWN IN 3.5

[noun: Julia fnc: know prn: 1]	[verb: know arg: Julia prn: 1]
--------------------------------------	--------------------------------------

In the next time-linear combination, the current result serves as the sentence start, while lexical lookup provides the proplet *John* as the next word (cf. 3.4, line 2).

The example with a discontinuous element (cf. 2.2 and 2.3) is reanalyzed in the hearer mode of Database Semantics as follows:

3.7 HEARER MODE ANALYSIS OF *Suzy looked the word up*



One difference to the earlier LA-grammar analysis 2.2 is the handling of the determiner *the*. In its lexical analysis, the core value is the substitution value *n_1*. In line 2, this value is copied into the *arg* slot of *look* and the core value of *look* is copied into the *fnc* slot of *the*. In line 3, the core value of *word* is used to substitute all occurrences of the substitution value *n_1*, after which the *nw* proplet is discarded. This method is called *function word absorption*.

An inverse kind of function word absorption is the treatment of the discontinuous element *up*. It is lexically analyzed as a standard preposition with the core attribute *adj* (cf. NLC'06, Chapter 15). In line 5, this preposition is absorbed into the verb, based on a suitable substitution value. Thus, a sentence consisting of five words is represented by only three proplets.

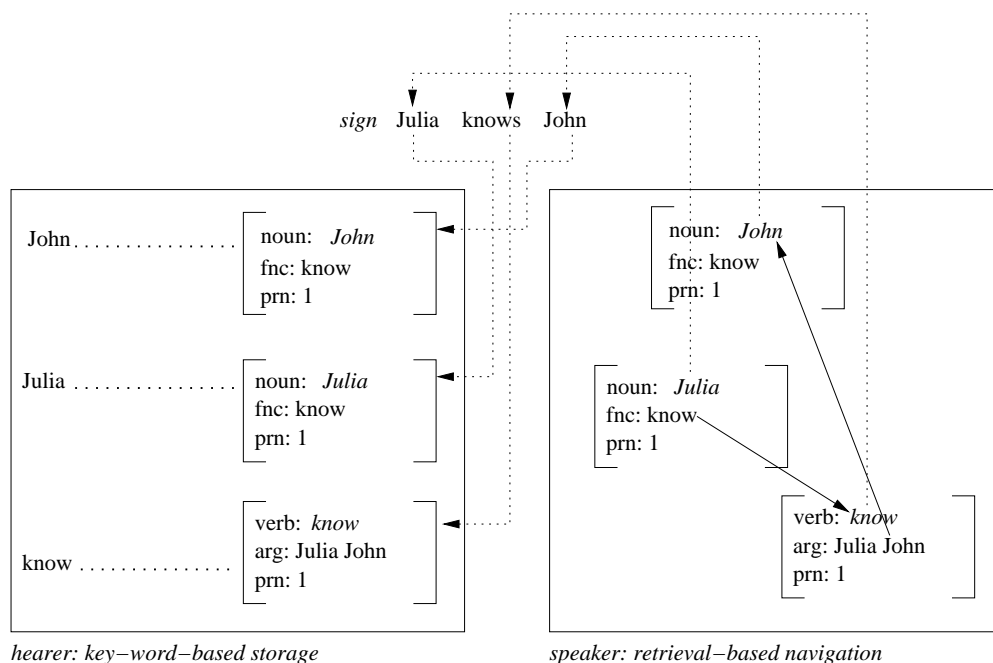
4 The Cycle of Natural Language Communication

In Database Semantics, the proplets resulting from an LA-hear derivation are stored in alphabetically ordered token lines, called a word bank. Each token line begins with a concept,

corresponding to the owner record of a classic network database, followed by all proplets containing this concept as their core value in the order of their occurrence, serving as the member records of a network database (cf. Elmasri and Navathe 1989).

Consider the following example.

4.1 TRANSFER OF CONTENT FROM THE SPEAKER TO THE HEARER



The word bank of the agent in the hearer mode (left) shows the token lines resulting from the LA-hear derivation 3.4. Due to the alphabetical ordering of the token lines, the sequencing of the proplets resulting from the LA-hear derivation is lost. Nevertheless, the semantic relations between them are maintained, due to their common *prn* value and the coding of the functor-argument structure in terms of attributes and values.

The word bank of the agent in the speaker mode (right) contains the same proplets as the word bank on the left. Here a linear order is reintroduced by means of a navigation along the semantic relations defined between the proplets. This navigation from one proplet to the next serves as a model of thought and as the *conceptualization* of the speaker, i.e., as the specification of what to say and how to say it.

The navigation from one proplet to the next is powered by an *LA-think* grammar. Consider the following rule application:

4.2 EXAMPLE OF AN **LA-think** RULE APPLICATION

	(i) rule name	(ii) <i>ss</i> pattern	(iii) <i>nw</i> pattern	(iv) operations	(v) rule package
rule level	V_N_V:	$\begin{bmatrix} \text{verb: } \beta \\ \text{arg: X } \alpha \text{ Y} \\ \text{prn: k} \end{bmatrix}$	$\begin{bmatrix} \text{noun: } \alpha \\ \text{fnc: } \beta \\ \text{prn: k} \end{bmatrix}$	output position <i>ss</i> mark α <i>ss</i>	{V_N_V, ...}
		<i>matching and binding</i>			
proplet level		$\begin{bmatrix} \text{verb: know} \\ \text{arg: Julia John} \\ \text{prn: 1} \end{bmatrix}$			

By binding the variables β , α , and k to know, Julia, and 1, respectively, the next word pattern is specified at the rule level such that the retrieval mechanism of the database can retrieve (navigate to, traverse, activate, touch) the correct continuation at the proplet level:

4.3 RESULT OF THE LA-think RULE APPLICATION

[verb: know arg: !Julia John prn: 1]	[noun: Julia fnc: know prn: 1]
--	--

In order to prevent repeated traversal of the same proplet,⁶ the arg value currently retrieved is marked with “!” (cf. NLC’06, p. 44).

The autonomous navigation through the content of a word bank, powered by the rules of an LA-think grammar, is used not only for conceptualization in the speaker mode, but also for inferencing and reasoning in general. Providing a data structure suitable to (i) support navigation was one of the four main motivations for changing from the NEWCAT’86 notation of LA-grammar illustrated in 2.1, 2.2, and 2.3 to the NLC’06 notation illustrated in 3.4, 3.5, and 3.7. The other three motivations are (ii) the matching between the levels of language and context (cf. 3.3), (iii) a more detailed specification of lexical items, and (iv) a descriptively more powerful and more transparent presentation of the semantic relations, i.e., functor-argument structure, coordination, and coreference.

A conceptualization defined as a time-linear navigation through content makes language production relatively straightforward: If the speaker decides to communicate a navigation to the hearer, the core values of the proplets traversed by the navigation are translated into their language-dependent counterparts and realized as external signs. In addition to this language-dependent lexicalization of the universal navigation, the language production system must provide language-dependent

1. word order,
2. function word precipitation (as the inverse of function word absorption),
3. word form selection for proper agreement.

These tasks are handled by language-dependent LA-speak grammars in combination with language-dependent word form production.

As an example of handling word order consider the production of the sentence *Julia knows John* from the set of proplets derived in 3.4:

4.4 PROPLETS UNDERLYING LANGUAGE PRODUCTION

[verb: know arg: Julia John prn: 1]	[noun: Julia fnc: know prn: 1]	[noun: John fnc: know prn: 1]
---	--	---------------------------------------

Assuming that the navigation traverses the set by going from the verb to the subject noun to the object noun, the resulting sequence may be represented abstractly as VNN.

Starting the navigation with the verb rather than the subject is because the connection between propositions is coded by the nc and pc features of the verb (cf. 3.2 and NLC’06, Appendix A2). Assuming that n stands for a name, fv for a finite verb, and p for punctuation, the time-linear derivation of an abstract n fv n p surface from a VNN proplet sequence is based on the following incremental realization:

⁶Relapse, see *tracking principles*, FoCL’99, p. 454.

4.5 SCHEMATIC PRODUCTION OF *Julia knows John.*

	<i>activated sequence</i>	<i>realization</i>
i	V	
i.1	n	n
	V N	
i.2	fv n	n fv
	V N	
i.3	fv n n	n fv n
	V N N	
i.4	fv p n n	n fv n p
	V N N	

In line i.1, the derivation begins with a navigation from V to N, based on LA-think. Also, the N proplet is realized as the n *Julia* by LA-speak. In line i.2, the V proplet is realized as the fv *knows* by LA-speak. In line i.3, LA-think continues the navigation to the second N proplet, which is realized as the n *John* by LA-speak. In line i.4, finally, LA-speak realizes the p . from the V proplet. This method can be used to realize not only a subject–verb–object surface (SVO) as in the above example, but also an SOV and (trivially) a VSO surface. It is based on the following principles:

4.6 PRINCIPLES FOR REALIZING SURFACES FROM A PROPLET SEQUENCE

- *Earlier surfaces may be produced from later proplets.*

Example: The initial n surface is achieved by realizing the second proplet in the activated VN sequence first (cf. line i.1 in 4.5 above).

- *Later surfaces may be produced from earlier proplets.*

Example: The final punctuation p (full stop) is realized from the first proplet in the VNN sequence (cf. line i.4 in 4.5 above).

Next consider the derivation of *Suzy looked the word up.*, represented as an abstract n fv d nn de p surface, whereby n stands for a name, fv for a finite verb, d for a determiner, nn for a noun, de for a discontinuous element, and p for punctuation.

4.7 SCHEMATIC PRODUCTION OF *Suzy looked the word up.*

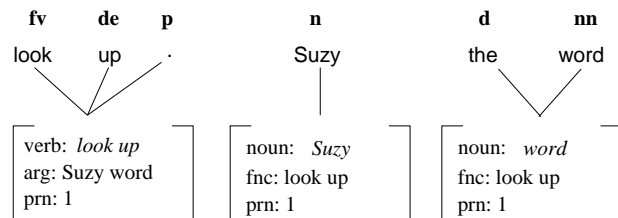
	<i>activated sequence</i>	<i>realization</i>
i	V	
i.1	n	n
	V N	
i.2	fv n	n fv
	V N	
i.3	fv n d	n fv d
	V N N	
i.4	fv n d nn	n fv d nn
	V N N	
i.5	fv de n d nn	n fv d nn de
	V N N	
i.6	fv de p n d nn	n fv d nn de p
	V N N	

This derivation of an abstract n fv d nn de p surface from an underlying VNN navigation shows two⁷ instances of function word precipitation: (i) of the determiner *the* from the second N proplet, and (ii) of the discontinuous element *up* from the initial V proplet.

5 “Constituent Structure” in Database Semantics?

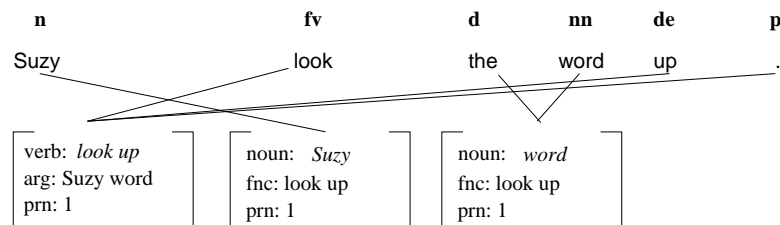
The correlation of the activated VNN sequence and the associated surfaces shown in line i.6 (left) of 4.7 may be spelled out more specifically as follows:

5.1 SURFACES REALIZED FROM PROPLETS IN A TRAVERSED SEQUENCE



This structure is like a constituent structure insofar as *what belongs together semantically* (cf. 1.1, condition 1) is realized from a single proplet. Like a deep structure in Chomsky 1965, however, the sequence fv de p n d nn of 5.1 does not constitute a well-formed surface. What is needed here is a transition to the well-formed surface sequence n fv d nn de p:

5.2 SURFACE ORDER RESULTING FROM AN INCREMENTAL REALIZATION



Instead of using a direct mapping like a transformation, Database Semantics establishes the correlation between the “deep” fv de p n d nn sequence 5.1 and the “surface” n fv d nn de p sequence 5.2 by means of a time-linear LA-think navigation with an associated incremental LA-speak surface realization, as shown schematically in 4.7 (for the explicit definition of the complete DBS1 and DBS2 systems of Database Semantics see NLC’06, Chapters 11–14).

Note, however, that this “rediscovery” of constituent structure in the speaker mode of Database Semantics applies to the *intuitions* supported by the substitution and movement tests by Bloomfield 1933 and Harris 1951 (cf. FoCL’99, p. 155 f.), but not to the formal Definition 1.1 based on phrase structure trees. Nevertheless, given the extensive linguistic literature within phrase-structure-based Nativism, let us consider the possibility of translating formal constituent structures into proplets of Database Semantics.

6 On Mapping Phrase Structure Trees into Proplets

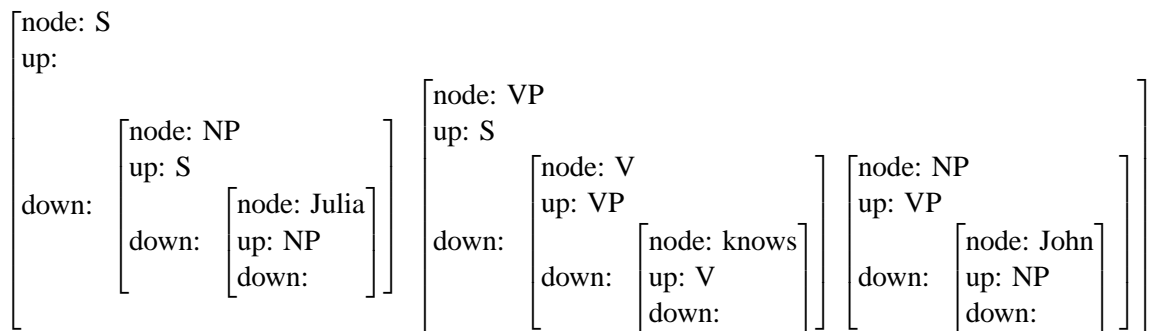
Any context-free phrase structure tree may be translated into a recursive feature structure. A straightforward procedure is to define each node in the tree as a feature structure with the attributes *node*, *up*, and *down*. The value of the attribute *node* is the name of a node in

⁷Actually, there is a third instance, namely the precipitation of the punctuation p from the V proplet.

the tree, for example node: **S**. The value of the attribute **up** specifies the next higher node, while the value of the attribute **down** specifies the next lower nodes. The linear precedence in the tree is coded over the order of the **down** values. Furthermore, the root node **S** is formally characterized by having an empty **up** value, while the terminal nodes are formally characterized by having empty **down** values.

Consider the following example of systematically recoding the phrase structure tree 1.2 (correct) as a recursive feature structure:

6.1 RECODING A TREE AS A RECURSIVE FEATURE STRUCTURE

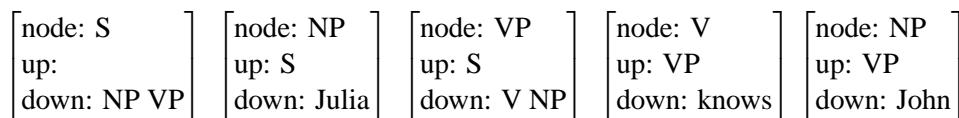


The translation of a phrase structure tree into a recursive feature structure leaves ample room for additional attributes, e.g., *phon* or *synsem*, as used by the various schools of Nativism.

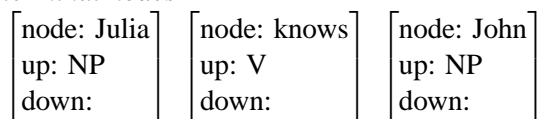
Furthermore, the recursive feature structure may be recoded as a set of non-recursive feature structures, i.e., proplets. The procedure consists in recursively replacing each value consisting of a feature structure by its elementary **node** value, as shown below:

6.2 RECODING 6.1 AS A SET OF PROPLETS

non-terminal nodes



terminal nodes



Formally, these proplets may be stored and retrieved in a word bank like the ones shown in Example 4.1.

The mapping from phrase structure trees to recursive feature structures (e.g., 6.1) to sets of proplets (e.g., 6.2) is not symmetric, however, because there are structures which can be easily coded as a set of proplets, but have no natural representation as a phrase structure tree. This applies, for instance, to a straight line, as in the following example:

6.3 GRAPHICAL REPRESENTATION OF A LINE



Such a line has no natural representation as a phrase structure tree, but it does as a set of proplets, as in the following definition:

6.4 RECODING THE LINE 6.3 AS A SET OF PROPLETS

<i>start</i>	line: H	
	prev:	
	next: I	
<i>intermediate</i>	line: I	line: J
	prev: H	prev: I
	next: J	next: K
<i>finish</i>	line: K	
	prev: J	
	next:	

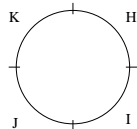
The beginning of the line is characterized by the unique proplet with an empty `prev` attribute, while the end is characterized by the unique proplet with an empty `next` attribute.⁸ Proplets of this kind are used in Database Semantics for the linguistic analysis of *coordination*.

The asymmetry between the expressive power of phrase structure trees and proplets must be seen in light of the fact that the language and complexity hierarchy of substitution-based phrase structure grammar (also called the Chomsky hierarchy) is orthogonal to the language and complexity hierarchy of time-linear LA-grammar (cf. TCS'92 and FoCL'99, Part II). For example, while the formal languages $a^k b^k$ and $a^k b^k c^k$ are in different complexity classes in phrase structure grammar, namely polynomial versus exponential, they are in the same class in LA-grammar, namely linear. Conversely, while the formal languages $a^k b^k$ and HCFL are in the same complexity class in phrase structure grammar, namely polynomial, they are in different classes in LA-grammar, namely linear versus exponential.

7 Possibilities of Constructing Equivalences

Regarding the use of feature structures, the most obvious difference between Nativism and Database Semantics are recursive feature structures in Nativism (cf. 1.5) and flat feature structures in Database Semantics (cf. results in 3.4 and 3.7). The recursive feature structures of Nativism are motivated by the constituent structure of the associated phrase structure trees, while the flat feature structures (proplets) of Database Semantics are motivated by the task of providing (i) a well-defined matching procedure between the language and the context level (cf. 3.3) and (ii) a time-linear storage of content in the hearer mode, a time-linear navigation in the think mode, and a time-linear production in the speaker mode (cf. 4.1).

⁸Another structure unsuitable for representation as a phrase structure is a circle:



There is no natural beginning and no natural end, as shown by the following definition as a set of proplets:

arc: H	arc: I	arc: J	arc: K
prev: K	prev: H	prev: I	prev: J
next: I	next: J	next: K	next: H

In this set, none of the proplets has an empty `prev` or `next` attribute, thus aptly characterizing the essential nature of a circle as compared to a line (cf. Example 6.4).

These differences do not in themselves preclude the possibility of equivalences between the two systems, however. Given our purpose to discover common ground, we found that phrase structure trees and the associated recursive feature structures (cf. 6.1) can be systematically translated into equivalent sets of proplets (cf. 6.2), thus providing Nativism with a data structure originally developed for matching, indexing, storage, and retrieval in Database Semantics. Furthermore, we have seen that something like constituent structure is present in Database Semantics, namely the correlation of semantically related surfaces to the proplet from which they are realized in the speaker mode (cf. 5.1).

How then should we approach the possible construction of equivalences between the two systems? From a structural point of view, there are two basic possibilities: to either look for an equivalence between corresponding components of the two systems (small solution), or to make the two candidates more equal by adding or subtracting components (large solution).

Regarding a possible equivalence of corresponding components (small solution), a comparison is difficult. Relative to which parameters should the equivalence be defined: Complexity? Functionality? Grammatical insight? Data coverage? Language acquisition? Typology? Neurology? Ethology? Robotics? Some of these might be rather difficult to decide, requiring lengthy arguments which would exceed the limits of this paper.

So let us see if there are some parts in one system which are missing in the other. This would provide us with the opportunity to add the component in question to the other system, thus moving inadvertently to a large solution for constructing an equivalence.

Beginning with Nativism, we find the components of a universal base generated by the rules of a context-free phrase structure grammar, constrained by constituent structure, and mapped by transformations or similar mechanisms into the grammatical surfaces of the natural language in question. These components have taken different forms and are propagated by different linguistic schools. Their absence in Database Semantics raises the question of how to take care of what the components of Nativism have been designed to do.

Thereby, two aspects must be distinguished: (i) the characterization of wellformedness and (ii) the characterization of innateness. For Chomsky, these are inseparable because without a characterization of innateness there are too many ways to characterize wellformedness.⁹ For Database Semantics, in contrast, the job of characterizing syntactical and semantical wellformedness is treated as a side-effect which results naturally from a well-functioning mechanism of interpreting and producing natural language during communication.

8 Can Nativism be Turned into an Agent-oriented Approach?

Next let us turn to components which are absent in Nativism.¹⁰ Their presence in DBS follows from the purpose of building a talking robot. The components, distinctions, and procedures in question are the external interfaces for recognition and action (cf. 3.1), a data structure with an associated algorithm modeling the hearer mode and the speaker mode (cf. 4.1), a systematic distinction between the language and the context level as well as their correlation in terms of matching (cf. 3.3), inferences at the context level (cf. NLC'06, Chapter 5), turn-taking, etc., all of which are necessary in addition to the grammatical component proper.

Extending Nativism by adding these components raises two challenges: (i) the technical problem of connecting the historically grown phrase structure system with the new compo-

⁹This problem is reminiscent of selecting the “right” phrase structure tree from a large number of possible trees (cf. 1.2), using the principle of constituent structure.

¹⁰They are also absent in truth-conditional semantics relative to a set-theoretical model defined in a metalanguage, which has been adopted as Nativism’s favorite method of semantic interpretation.

nents and (ii) finding a meaningful functional interaction between the original system and the new components.

Regarding (i), there is the familiar problem of the missing external interfaces: how should a phrase structure system with transformations or the like be integrated into a computational model of the hearer mode and the speaker mode? Regarding (ii), it must be noted that Chomsky and others have emphasized again and again that Nativism is *not intended* to model the use of language in communication.

Nevertheless, an extension of Nativism to an agent-oriented system would have great theoretical and practical advantages. For the theory, it would substantially broaden the empirical base,¹¹ and for the practical applications, it would provide a wide range of much needed new functionalities such as procedures modeling the speaker mode and the hearer mode.

Let us therefore return to the possibility of translating phrase structure trees systematically into proplets (cf. 6.1 and 6.2). Is this formal possibility enough to turn Nativism into an agent-oriented system? The answer is simple: while the translation in question is a necessary condition for providing Nativism with an effective method for matching, indexing, storage, and retrieval, it is not a sufficient condition.

What is needed in addition is that the connections between the proplets (i) characterize the basic semantic relations of functor-argument structure and coordination as simply and directly as possible and (ii) support the navigation along these semantic relations in a manner which is as language-independent as possible. For these requirements, constituent structure presents two insuperable obstacles, namely (a) the proplets representing non-terminal nodes and (b) the proplets representing function words.

Regarding (a), consider the set of proplets shown in 6.2 and the attempt to navigate from the terminal node *Julia* to the terminal node *knows*. Because there is no direct relation between these two proplets in 6.2, such a navigation would have to go from the terminal proplet *Julia* to the non-terminal proplet *NP* to the non-terminal proplet *S* to the non-terminal proplet *VP* to the non-terminal proplet *V* and finally to the terminal proplet *knows*. Yet eliminating these non-terminal nodes¹² would destroy the essence of constituent structure as defined in 1.1 and thus the intuitive basis of Nativism.

The other crucial ingredient of constituent structure, besides the non-terminal nodes, are the function words. They are important insofar as the words belonging together semantically are in large part the determiners with their nouns, the auxiliaries with their non-finite verbs, the prepositions with their noun phrases, and the conjunctions with their associated clauses. Regarding problem (b) raised by proplets representing function words, let us return to the example *Suzu looked the word up*, analyzed above in 1.3, 1.4, 2.2, 3.7, and 4.7.

¹¹As empirical proof for the existence of a universal grammar, Nativism offers language structures claimed to be learned *error-free*. They are explained as belonging to that part of the universal grammar which is independent from language-dependent parameter setting. Structures claimed to involve error-free learning include

1. structural dependency
2. C-command
3. subjacency
4. negative polarity items
5. that-trace deletion
6. nominal compound formation
7. control
8. auxiliary phrase ordering
9. empty category principle

After careful examination of each, MacWhinney 2004 has shown that there is either not enough evidence to support the claim of error-freeness, or that the evidence shows that the claim is false, or that there are other, better explanations.

¹²In order to provide for a more direct navigation, as in Example 2.1 (result).

Given that this sentence does not have a well-formed constituent structure in accordance with Definition 1.1, let us look for a way to represent it without non-terminal nodes, but with proplets for the function words **the** and **up**. Consider the following tentative proposal, which represents each terminal symbol (word) as a proplet and concatenates the proplets using the attributes **previous** and **next**, in analogy to 6.4:

8.1 TENTATIVE REPRESENTATION WITH FUNCTION WORD PROPLETS

noun: Suzy	verb: look	det: the	noun: word	prep: up
prev:	prev: Suzy	prev: look	prev: the	prev: word
next: look	next: the	next: word	next: up	next:
prn: 2	prn: 2	prn: 2	prn: 2	prn: 2

For the purposes of indexing, this analysis allows the storage of the proplets in – and the retrieval from – locations in a database which are not subject to any of the graphical constraints induced by phrase structure trees, and provides for a time-linear navigation, forward and backward, from one proplet to the next.¹³

For a linguistic analysis within Nativism or Database Semantics, however, the analysis 8.1 is equally unsatisfactory. What is missing for Nativism is a specification of what belongs together semantically. What is missing for Database Semantics is a specification of the functor-argument structure. For constructing an equivalence between Nativism and Database Semantics we would need to modify the attributes and their values in 8.1 as to

1. retain the proplets for the function words,
2. characterize what belongs semantically together in the surface, and
3. specify the functor-argument structure.

Of these three desiderata, the third one is the most important: without functor-argument structure the semantic characterization of content in Database Semantics would cease to function and the extension of Nativism to an agent-oriented approach would fail.

For specifying functor-argument structure, the proplets for function words are an insuperable obstacle insofar as they introduce the artificial problem of choosing whether the connection between a functor and an argument should be based on the function words (modifiers) or on the content words (heads). For example, should the connection between **looked** and **the word** be defined between **looked** and **the**, or between **looked** and **word**? Then there follows the question of how the connection between **word** and **the** should be defined, and how the navigation should proceed.

These questions are obviated in Database Semantics by defining the grammatical relations directly between the content words. Consider the following semantic representation of **Suzy looked the word up**, repeating the result line of 3.7, though with the additional attribute **sem** to indicate the contribution of the determiner **the** after function word absorption:

8.2 SEMANTIC REPRESENTATION WITH FUNCTION WORD ABSORPTION

noun: Suzy	verb: look up	noun: word
sem: nm	sem: pres	sem: def sg
fnc: look up	arg: Suzy word	fnc: look up
prn: 2	prn: 2	prn:2

¹³The navigation would be powered by rules like that illustrated in 4.2, modified to apply to the attributes of 8.1. For a complete DBS-system handling Example 8.1, consisting of an LA-hear grammar and an LA-think/speak-grammar, see NLC'06, Section 3.6.

Compared to the five proplets of Example 8.1, this analysis consists of only three. The attributes `prev` and `next` have been replaced by the attributes `sem` (for semantics), `fnc` (for the functor of a noun), and `arg` (for the argument(s) of a verb). The functor-argument structure of the sentence is coded by the value `look up` of the `fnc` slot of the nouns `Suzy` and `word`, and the values `Suzy word` of the `arg` slot of the verb *look up* (bidirectional pointering).

During the time-linear LA-hear analysis, shown in 3.7, the function words are treated as full-fledged lexical items (proplets). The resulting semantic representation 8.2 provides grammatical relations which support forward as well as backward navigation. These navigations, in turn, are the basis of the production of different language surfaces. For example, while forward navigation would be realized in English as *Suzy looked the word up*, the corresponding backward navigation would be realized as *The word was looked up by Suzy*.¹⁴

In 8.2, the contribution of the absorbed function word *the* is the value `def` of the `cat` attribute of the proplet *word*, while the contribution of the absorbed function word *up* is the corresponding value of the `verb` attribute of the proplet *look up*.¹⁵ Defining the grammatical relations solely between content words is motivated not only by the need to establish semantic relations suitable for different kinds of navigation, but also by the fact that function words are highly language-dependent, like morphological variation and word order.

9 Conclusion

While Nativism and Database Semantics developed originally without feature structures, they were added later for a more detailed grammatical analysis. This paper describes the different functions of feature structures in Nativism and Database Semantics, and investigates the possible establishment of equivalences between the two systems.

Establishing equivalences means overcoming apparent differences. The most basic difference between Nativism and Database Semantics is that Nativism is sign-oriented while Database Semantics is agent-oriented. Ultimately, this difference may be traced to the respective algorithms of the two systems: the rewrite rules of PS-grammar (Nativism) do not have an external interface, while the time-linear rules of LA-grammar (Database Semantics) do. It is for this reason that Nativism cannot be extended into an agent-oriented approach, thus blocking the most promising possibility for constructing an equivalence with Database Semantics. This result complements the formal non-equivalence between the complexity hierarchies of PS-grammar and LA-grammar proven in TCS'92.

The argument in this paper has been based on only two language examples, namely *Julia knows John* and *Suzy looked the word up*. For wider empirical coverage see NLC'06. There, functor-argument structure (including subordinate clauses), coordination (including gapping constructions), and coreference (including 'donkey' and 'Bach-Peters' sentences) are analyzed in the hearer and the speaker mode, based on more than 100 examples.

Acknowledgments

This paper benefited from comments by Airi Salminen, University of Toronto; Kiyong Lee, Korea University; Haitao Liu, Communication University of China; and Emmanuel Giguët, Université de Caen. All remaining mistakes are those of the author.

¹⁴For a more detailed analysis see NLC'06, Section 6.5.

¹⁵In analogy to 2.2, the value `up` could also be stored as a third valency filler in the `arg` slot of the verb.

References

- Bar-Hillel, Y. (1964) *Language and Information. Selected Essays on Their Theory and Application*. Reading, MA: Addison-Wesley
- Bloomfield, L. (1933) *Language*, New York: Holt, Rinehart, and Winston
- Bresnan, J. (ed.) (1982) *The Mental Representation of Grammatical Relation*. Cambridge, MA: MIT Press
- Chomsky, N. (1965) *Aspects of a Theory of Syntax*, The Hague: Mouton
- Chomsky, N. (1981) *Lectures on Government and Binding*, Dordrecht: Foris
- Clark, H. H. (1996) *Using Language*. Cambridge: Cambridge Univ. Press
- Elmasri, R. & S.B. Navathe (1989) *Fundamentals of Database Systems*, Redwood City, CA: Benjamin-Cummings
- Gaifman, C. (1961) *Dependency Systems and Phrase Structure Systems*, P-2315, Santa Monica, CA: Rand Corporation
- Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985) *Generalized Phrase Structure Grammar*. Cambridge, MA: Harvard Univ. Press
- Harris, Z. (1951) *Methods in Structural Linguistics*, Chicago: Univ. of Chicago Press
- Hausser, R. (1986) *NEWCAT: Parsing Natural Language Using Left-Associative Grammar*, LNCS 231, Berlin Heidelberg New York: Springer (NEWCAT'86)
- Hausser, R. (1992) "Complexity in Left-Associative Grammar," *Theoretical Computer Science*, Vol. 106.2:283-308, Amsterdam: Elsevier (TCS'92)
- Hausser, R. (1999) *Foundations of Computational Linguistics, 2nd ed. 2001*, Berlin Heidelberg New York: Springer (FoCL'99)
- Hausser, R. (2001) "Database Semantics for natural language," *Artificial Intelligence*, Vol. 130.1:27-74, Amsterdam: Elsevier (AIJ'01)
- Hausser, R. (2006) *A Computational Model of Natural Language Communication*, Berlin Heidelberg New York: Springer (NLC'06)
- Kay, M. (1992) "Unification," in M. Rosner and R. Johnson (eds) *Computational Linguistics and Formal Semantics*, p. 1-30, Cambridge: Cambridge Univ. Press
- Kay, P. (2002) "An informal sketch of a formal architecture for construction grammar," *Grammars*, Vol. 5:1-19, Dordrecht: Kluwer
- MacWhinney, B. (2004) "A multiple process solution to the logical problem of language acquisition," *Journal of Child Language*, Vol. 31:883-914, Cambridge: CUP
- Pereira, F., and D. Warren (1980) "Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks," *Artificial Intelligence*, Vol. 13:231-278, Amsterdam: Elsevier
- Pollard, C., and I. Sag (1987) *Information-based Syntax and Semantics, Vol. I: Fundamentals*, Stanford: CSLI
- Pollard, C., and I. Sag (1994) *Head-Driven Phrase Structure Grammar*, Stanford: CSLI
- Saussure, F. de (1913/1972) *Cours de linguistique générale*, Édition critique préparée par Tullio de Mauro, Paris: Éditions Payot
- Shankar, V., and A. Joshi (1988) "Feature-structure based tree adjoining grammar," in *Proceedings of 12th International Conference on Computational Linguistics (Coling'88)*