

Turn Taking in Database Semantics

Roland Hausser

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

Abstract

One of the most basic structural phenomena of natural language communication is turn taking, i.e. the communicating agents taking turns in being speaker and hearer. Without turn taking, there is only one-sided monologue as the limiting case.

The linguistic framework called Database Semantics (AIJ'01) treats the agent's switching back and forth between the speaker and the hearer mode as a well-defined and well-motivated computational problem. Its solution is a precondition for modeling natural language communication by programming suitable machines (talking robots). All syntactic and semantic analysis of natural language must be integrated into the turn taking underlying communication.

Turn taking is modeled as the interaction of three kinds of LA-grammars, called LA-hear, LA-think, and LA-speak. LA-hear interprets natural language signs by reading their content into the hearer's database. LA-think provides the content to be verbalized by autonomously navigating through the speaker's database content (conceptualization). LA-speak maps the content traversed by LA-think into natural language surfaces.

After a general introduction to the time-linear algorithm of LA-grammar, the rule and derivation format of the three grammars modeling turn taking are explained. The examples are based on a Java implementation currently under development by Mr. Arkadius Kycia in cooperation with the author.¹

1 Basic Algorithm of LA-grammar

Today, there is a choice between two full-blown grammar formalisms complete with algebraic definition, hierarchy of language classes, complexity classes, and standard methods of computational implementations in different programming languages, namely PS-grammar (PSG) and LA-grammar (LAG). The most basic difference between these two frameworks is that the derivation order of LA-grammar is time-linear while that of PS-grammar is not.

Time-linear means linear like time and in the direction of time. LA-grammar models time-linearity in that it takes a sequence, for example the words $a b c d e \dots$, as input and always combines what has been analyzed so far, called sentence start (ss), with the current next word (nw) into a new sentence start (ss'), for example a and b into $(a b)$, $(a b)$ and c into $((a b) c)$, $((a b) c)$ and d into $((((a b) c) d))$, etc. This kind of combination is called *left-associative*.

Database Semantics (dbs) chooses LA-grammar as its algorithm because the structure of natural language is inherently time-linear. This is reflected not only in the structural order of the natural language signs, but also in the procedures of language interpretation (hearer mode) and language production (speaker mode). In addition, the complexity properties of

¹This paper benefited from comments by Prof. Meyer-Wegener, Department of Computer Science at the Universität Erlangen-Nürnberg. Naturally, he is not responsible for any weaknesses or mistakes.

LA-grammar are much more suitable for the structures found in natural language than those of PS-grammar. Also, LA-grammar is type transparent² while PS-grammar is not.

LA-grammar as exemplified by NEWCAT'86, CoL'89, and TCS'92 was developed prior to dbs (FoCL'99, AIJ'01) and is a conventional system in that it takes unanalyzed language expressions as input and renders grammatically analyzed language expressions as output. Such LA-grammars consist of (i) a lexicon, (ii) a set of start states, (iii) a set of rules, and (iv) a set of final states.

As an example, consider the following definition of an LA-grammar for the formal language $a^k b^k c^k$, consisting of expressions like *abc*, *aabbcc*, *aaabbbccc*, etc.

1.1 LA-GRAMMAR FOR $a^k b^k c^k$

$$\begin{aligned} LX &=_{def} \{[a (A)], [b (B)], [c (C)]\} \\ ST_S &=_{def} \{[(A) \{r_1, r_2\}]\} \\ r_1: (X) (A) &\Rightarrow (AX) \{r_1, r_2\} \\ r_2: (AX) (B) &\Rightarrow (XB) \{r_2, r_3\} \\ r_3: (BX) (C) &\Rightarrow (X) \{r_3\} \\ ST_F &=_{def} \{[\varepsilon rp_3]\}. \end{aligned}$$

The lexicon *LX* is defined as a list of analyzed words. The analysis of a word consists of the surface, also called lemma, and some kind of lexical characterization (e.g., the syntactic category, the meaning, etc.). The lexicon defined in 1.1 consists of three analyzed words, namely *[a (A)]*, *[b (B)]*, and *[c (C)]*. The word *[a (A)]*, for example, consists of the surface *a* and the category (A).³

The set of start states *ST_S* specifies which words (represented by their category) can start the parsing of a sentence, and which specific rules should be tried to combine the first word with the next. In our example, the set of start states has only one element, namely *[(A) {r₁, r₂}]*. Like any state in LA-grammar, the start state is defined as an ordered pair consisting of a category, here (A), and a rule package, here {r₁, r₂}.

An LA-grammar may have several start states and start computing on various kinds of input. In 1.1, however, the set named *ST_S* contains only one start state. This LA-grammar is like a little machine which will start to compute if and only if it gets a sign with the category (A) on its input tape as initial input.

The set of rules in 1.1 contains three elements, called r-1, r-2, and r-3. Their general structure may be characterized as the following schema:

1.2 SCHEMA OF LEFT-ASSOCIATIVE RULES IN LA-GRAMMAR

$$r_i: ss \ nw \Rightarrow ss' \ rp_i$$

A LAG rule consists (i) of a rule name, e.g. *r_i*, (ii) a pattern for the *ss* (sentence start, first input), (iii) a pattern for the *nw* (next word, second input), (iv) a pattern for the output called *ss'* (resulting, or new, sentence start), and (v) a rule package, e.g. *rp_i*.

²According to Berwick & Weinberg 1984, a grammar is type transparent if the declarative linguistic analysis and the derivation procedure on the computer are merely different realizations of the same algorithm. For further explanation see FoCL'99, p. 170 f.

³In LA-grammars for natural languages, with their very large number of words and wordforms, the lexicon is part of a component for automatic word form analysis. For small lexica, like the one needed for $a^k b^k c^k$, it suffices to give the explicit list.

The combination of rule names and rule packages, e.g. r_i and rp_i , in LAG constitute a finite state transition network. The combination of an ss , nw , and ss' pattern in a LAG rule constitutes the rule's categorial operation.⁴

A left-associative combination step requires a current state ($ss\ rp_i$) and a next word nw :

(i) $(ss\ rp_i) + nw$

The combination step is computed by applying the rules of the current state's rule package rp_i to the current sentence start and the current next word:

(ii) $rp_i: (ss\ nw)$

Applying a rule successfully to the input ($ss\ nw$) results in a new state consisting of a new sentence start ss' (represented formally by its category) and a new rule package rp_j . By adding a new next word to this new state provides the ingredients for the next left-associative combination step:

(i) $(ss'\ rp_j) + nw'$

Again, the system applies the rules of the rule package to the input pair:

(ii) $rp_j: (ss'\ nw')$

This process continues until all the next words in the input have been consumed. If more than one rule is successful in a combination step, the resulting branches are pursued in parallel (cf. 2.5). If none of the rules is successful in a combination step, the derivation fails.

The set of final states S_F specifies which expressions are *complete* final expressions, like $aaabbbccc$ of $a^k b^k c^k$, in contradistinction to incomplete but nevertheless legal intermediate expressions like $aaabbbcc$. In our example, the set of final states contains only one element, namely $[\epsilon\ rp_3]$. This means that a parsed expression is a legal complete expression of the formal language $a^k b^k c^k$ if its final category is ϵ ,⁵ and its final rule package has been activated by r-3.

Because of the absolute type transparency of LA-grammar, the declarative linguistic analysis and the derivation procedure on the computer are merely different realizations of the same left-associative algorithm. For example, the LAG parser NEWCAT implemented in Lisp takes an LA-grammar and a string as input and generates an analysis like the following. It is based on the LA-grammar defined in 1.1 and the input string $aaabbbccc$:

1.3 NEWCAT PARSING $aaabbbccc$ USING LIST-BASED NOTATION

```
*START-0
1
  (A) a
  (A) a
*RULE-1
2
  (A A) a a
  (A) a
*RULE-1
3
  (A A A) a a a
  (B) b
*RULE-2
4
  (A A B) a a a b
  (B) b
*RULE-2
5
```

⁴The LA-grammars considered here belong to the subclass of C-LAGs. In C-LAGs the pattern of the sentence start is restricted to patterns like (aX) , (Xa) , or (aXb) , while patterns like (XaY) are excluded – whereby a , b are constants and X , Y are variables. Because of this restriction, a rule application may be taken as the primitive operation of C-LAGs for complexity analysis. For further discussion see FoCL'99, p. 207 f.

⁵Also called 'empty category', corresponding to 'empty stack' in automata theory.

```

      (A B B) a a a b b
      (B) b
*RULE-2
6
      (B B B) a a a b b b
      (C) c
*RULE-3
7
      (B B) a a a b b b c
      (C) c
*RULE-3
8
      (B) a a a b b b c c
      (C) c
*RULE-3
9
      (NIL) a a a b b b c c c

```

This LA-grammar analysis is generated directly as the formatted protocol (trace) of the LA-parser NEWCAT.⁶ Note that LA-grammar parses context-sensitive $a^k b^k c^k$ in linear time.⁷

NEWCAT'89 presents a Lisp implementation complete with source code for LA-parsing 221 sentence structures of German and 114 sentence structures of English. The syntactic analysis provided by the LA-parser is designed to handle agreement, word order, and valency, and is derived in a strictly time-linear fashion.

2 Feature Structure Notation of JLAG Implementation

In the long run, the list-based format of early LA-grammar as described in the previous Section proved to be too cumbersome for natural language analysis. This motivated a transition from the original list-based notation to an equivalent feature structure notation. The following example illustrates the two notations, using rule r-1 of the LA-grammar defined in 1.1.

2.1 COMPARING LIST NOTATION WITH FEATURE STRUCTURE NOTATION

original list-based notation:

$r_1: (X) (A) \Rightarrow (AX) \{r_1, r_2\}$

current feature structure notation:

$r_1: [\text{cat}: X] \quad [\text{cat}: A] \quad \begin{array}{l} \text{leftcopy } A \text{ ss-cat} \\ \text{copy_ss} \end{array} \quad \{r_1, r_2\}$

While the list-based notation represents the ss-patterns as the list (X), the feature structure notation represents it as the attribute/value-pair [cat: X], and similarly for the nw-pattern.⁸ Furthermore, the categorial operation coded in the list-based notation by means of the result pattern (aX) is coded in the feature structure notation by specifying the operations explicitly. For example, the operation `leftcopy A ss-cat` in the 2.1 means that the constant A is to be added to the left of the current value of the cat-attribute in the ss.⁹

⁶To obtain a better comparison of the input categories, surface and category are printed in inverse order, e.g., (A) a instead of a (A).

⁷For a complexity-theoretical analysis of LA-grammar see TCS'92 and FoCL'99, Chapters 11 and 12.

⁸The attributes of a feature structure are also called features. We use the term 'attribute' in order to avoid confusion with the term 'feature' as used in psychology, for example in the comparison of features and templates.

⁹The operations `copy_ss` or `copy_nw` derive from the fact that the feature structure version of LA-grammars for natural language treat the sentence start as a set of proplets and allow to specify whether or not the next word is to be added to this set.

Feature structures consisting of attributes with constants or variables as values are descriptively more powerful than lists or list patterns. It is therefore easy to translate from the old notation into the new, as illustrated by the following LA-grammar definition of $a^k b^k c^k$:

2.2 LA-GRAMMAR FOR $a^k b^k c^k$ IN FEATURE STRUCTURE NOTATION

$$LX =_{def} \left\{ \begin{bmatrix} \text{sur: } a \\ \text{cat: } A \end{bmatrix}, \begin{bmatrix} \text{sur: } b \\ \text{cat: } B \end{bmatrix}, \begin{bmatrix} \text{sur: } c \\ \text{cat: } C \end{bmatrix} \right\}$$

$$ST_S =_{def} \{ [[\text{cat: } A] \{ r_1, r_2 \}] \}$$

$$r_1: \begin{bmatrix} \text{cat: } X \\ \text{cat: } A \end{bmatrix} \quad \begin{array}{l} \text{leftcopy } A \text{ ss-cat} \\ \text{copy_ss} \end{array} \quad \{ r_1, r_2 \}$$

$$r_2: \begin{bmatrix} \text{cat: } AX \\ \text{cat: } B \end{bmatrix} \quad \begin{array}{l} \text{leftremove } A \text{ ss-cat} \\ \text{rightcopy } B \text{ ss-cat} \\ \text{copy_ss} \end{array} \quad \{ r_2, r_3 \}$$

$$r_3: \begin{bmatrix} \text{cat: } BX \\ \text{cat: } B \end{bmatrix} \quad \begin{array}{l} \text{leftremove } B \text{ ss-cat} \\ \text{copy_ss} \end{array} \quad \{ r_3 \}$$

$$ST_F =_{def} \{ [[\text{cat: } \epsilon] rp_3] \}.$$

This LA-grammar is equivalent – in the sense of a notational variant – to that defined in 1.1.

The new notation of the data and the LA-grammar rules was developed as part of an implementation of Database Semantics in Java, currently under development. There also resulted a new display format for the syntactic derivation:

2.3 COMPARING LIST-BASED WITH FEATURE STRUCTURE DISPLAY

list-based NEWCAT display (see also 1.3)

```

1
  (A) a
  (A) a
*RULE-1
2
  (A A) a a

```

feature structure JLAG display (see also 2.4)

```

2.1
r-1: [cat: X] [cat: A] leftcopy      rp:{r-1, r-2} #rule level
      copy_ss
      [sur: a] [sur: a] #input level
      [cat: A] [cat: A]

```

While the NEWCAT format displays only the input, the rule name, and the output, the JLAG format shows the complete rule in conjunction with the input, whereby the patterns of the rule level are aligned to be directly above the input. The output (not shown in 2.3) is displayed as the *ss* of the next combination step. Consider the following derivation:

2.4 JLAG PARSING aabbcc USING FEATURE STRUCTURE NOTATION

```

rrh@venus:~/java/JLAG/bin> java anbncn.ANBNCN

# Enter a sentence or '-help'.

anbncn.ANBNCN-HEAR> a a b b c c
# Hearer mode #

1.
Start state, NW:      [cat: A]                rp:{r-1, r-2}
                     [sur: a]
                     [cat: A]

2.1
r-1: [cat: X] [cat: A] leftcopy      rp:{r-1, r-2}
      copy_ss
      [sur: a] [sur: a]
      [cat: A] [cat: A]

3.12
r-2: [cat: AX] [cat: B] leftremove   rp:{r-2, r-3}
      rightcopy
      copy_ss
      [sur: aa] [sur: b]
      [cat: AA] [cat: B]

4.121
r-2: [cat: AX ] [cat: B] leftremove  rp:{r-2, r-3}
      rightcopy
      copy_ss
      [sur: aab] [sur: b]
      [cat: AB ] [cat: B]

5.1212
r-3: [cat: X  ] [cat: C] leftremove  rp:{r-3}
      copy_ss
      [sur: aabb] [sur: c]
      [cat: BB  ] [cat: C]

6.12121
r-3: [cat: X   ] [cat: C] leftremove  rp:{r-3}
      copy_ss
      [sur: aabbc] [sur: c]
      [cat: B    ] [cat: C]

##### Final State: [Branch: 12121]

[sur: aabbcc]
[cat:      ]
-----
### Parsing finished successfully.

```

Each combination step has a number, for example 5.1212. The digit preceding the full stop refers to the word number in the input, here the fifth word (for which reason the start state has the number 1 and the final state has no number). The digits after the full stop specify which rule in the rule package of a combination was successful (whereby a 1 represents the first rule name in the package, a 2 the second rule, etc.). For example, the combination number 3.12 indicates the addition of the third word in the input, based on a successful application of the first rule of the rule package in the first combination and the second rule of the rule package of the second combination.

If more than one rule in a rule package is successful, the derivation is ambiguous. Ambiguities are handled by LA-parsers as parallel paths. Consider for example the following

derivation of the context-sensitive language WW:¹⁰

2.5 JLAG FOR WW PARSING abab

```
ww.WW-HEAR> a b a b
# Hearer mode #

1.
Start state, NW:      [cat: A]                rp:{r-2, r-1}
                     [sur: a]
                     [cat: A]

2.2
r-1: [cat: X] [cat: seg]  rightcopy      rp:{r-2, r-1}
      copy_ss
      [sur: a] [sur: b ]
      [cat: A] [cat: B ]

3.21
r-2: [cat: seg X] [cat: seg]  leftremove   rp:{r-2}
      copy_ss
      [sur: ab ] [sur: a ]
      [cat: AB ] [cat: A ]

3.22
r-1: [cat: X ] [cat: seg]  rightcopy      rp:{r-2, r-1}
      copy_ss
      [sur: ab] [sur: a ]
      [cat: AB] [cat: A ]

4.211
r-2: [cat: seg X] [cat: seg]  leftremove   rp:{r-2}
      copy_ss
      [sur: aba ] [sur: b ]
      [cat: B  ] [cat: B ]

4.222
r-1: [cat: X ] [cat: seg]  rightcopy      rp:{r-2, r-1}
      copy_ss
      [sur: aba] [sur: b ]
      [cat: ABA] [cat: B ]

##### Final State: [Branch: 211]

[sur: abab]
[cat:      ]
-----
### Parsing finished successfully.
```

The time-linear ambiguity arises with the addition of word 3: in 3.21, rule r-2 treats word 3 as the first word of the second W, while in 3.22, rule r-1 treats word 3 as an addition to the first W. The ambiguity is continued in the addition of word 4. However, as shown by the branch number 211 of the Final State, only the hypothesis represented by the first of the two parallel path results in a correct complete expression of WW.

The new notation based on feature structures and specified operations does not change the mathematical properties of LA-grammar as compared to the old one. This is because (i) the rules are restricted to left-associative compositions, and the set of operations of each rule is (ii) finite in number and (iii) non-recursive. LA-grammar parses context-sensitive WW in low polynomial time, namely n^2 .

¹⁰See FoCL, example 11.5.6 on p. 218, for the declarative specification of WW in the list-based format.

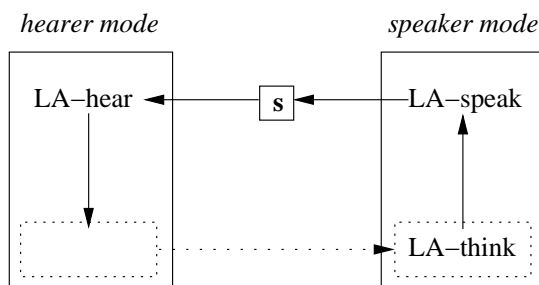
3 Turn Taking as a Software Engineering Task

The feature structures, illustrated above with formal languages, are called *proplets* in applications of LA-grammar to natural language. Proplets represent the functor-argument structure of a proposition (intrapositional relations) and the connections between propositions (extrapositional relations) in terms of attribute values. Proplets are well-suited for the storage and retrieval required for modeling natural language communication.

Based on proplets, the time-linear algorithm of LA-grammar is used to realize the following procedures: (i) natural language interpretation during input, handled by **LA-hear**, (ii) inferring and the sorting of content during thought, handled by **LA-think**, and (iii) language production during output, handled by **LA-speak**.¹¹

In communication, these three LA-grammars cooperate as follows:

3.1 THE CYCLE OF NATURAL LANGUAGE COMMUNICATION



This schema¹² represents the process of *turn taking*, i.e., the communicating agents taking turns in being speaker and hearer. The small box containing *s* represents the sign being transmitted. Turn taking has the following aspects:¹³

3.2 TWO ASPECTS OF TURN TAKING

1. *From the outside:*

Two communicating agents are observed as they are taking turns. This is represented by 3.1 when the two boxes are taken to be two different agents, one in the hearer and the other in the speaker mode.

2. *From the inside:*

One communicating agent is observed as it switches between being the speaker and the hearer. This is represented by 3.1 when the two boxes are taken to be the hearer and the speaker mode of the same agent (dotted right hand arrow).

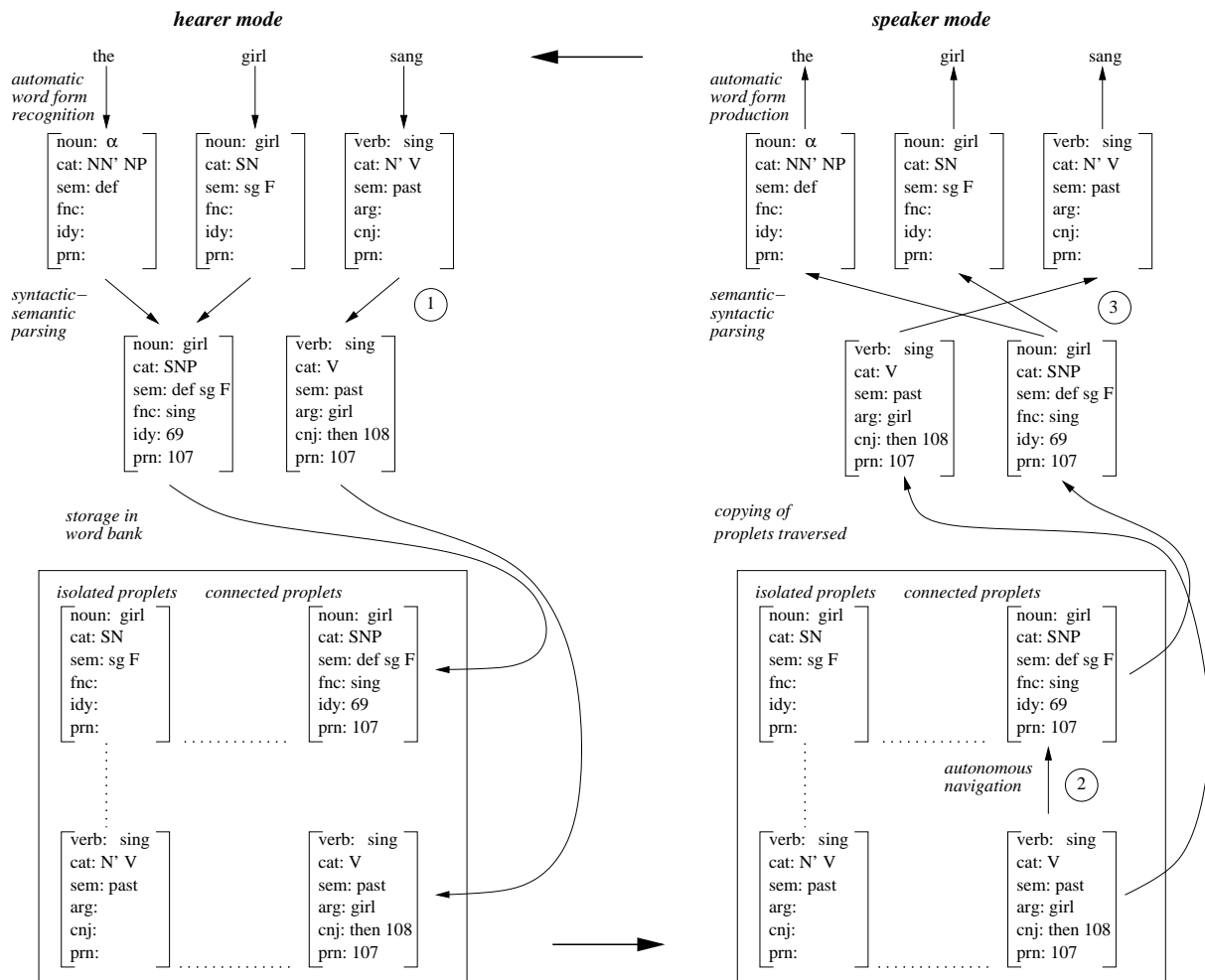
Before taking a closer look at the rules and derivations of LA-hear, LA-think, and LA-speak in Sections 4, 5, and 6, respectively, let us illustrate our conceptual solution to turn taking with a simple example. It shows the processing of the sentence *the girl sang* in the hearer and the speaker mode.

¹¹The names of these LA-grammars are motivated by the most basic variants of the hearer and the speaker mode. They are not intended to exclude, for example, written signs as input to LA-hear and as output of LA-speak. Instead, the system of Database Semantics presented here is based on modality-independent sign representations. The part of peripheral cognition is omitted for simplicity.

¹²In schema 3.1 contextual recognition and action are omitted for simplicity.

¹³For a study of turn taking with a much wider scope than the present paper see Thórisson 2002.

3.3 A FORMAL ILLUSTRATION OF TURN TAKING



Starting with the hearer mode on the left hand side, we see the input sequence in the top line: *the girl sang*. Next below is automatic word form recognition. It provides each of the three input surfaces with a corresponding isolated proplet retrieved from the lexicon.

These lexically analyzed word forms are the input to LA-hear, represented as ①. This interpretation parser produces two connected proplets as output, whereby the noun *girl* has been absorbed into the determiner *the*. The connected proplets *girl* and *sing* are stored at the end of their respective token lines in the database,

Then the agent switches into the speaker mode, as represented by the bottom right arrow. Inside the database on the right, language production starts with an autonomous navigation serving as conceptualization. This navigation is powered by LA-think, represented as ②.

The proplets traversed are copied and used as input to LA-speak, represented as ③. This production parser is for handling (i) the language-specific word order and (ii) function word precipitation (which is the counterpart to absorption during interpretation). In other words, the language-independent proplet sequence *sing girl* is mapped by LA-speak into the language-dependent proplet sequence *DET girl sing*.

Using the values of certain attributes, for example, *sing* and *past*, these proplets are mapped by automatic word form production into appropriate language-dependent surfaces, here *sang*. When these surfaces have been produced as an external sign, as indicated by the top left arrow, the system switches again into the hearer mode.¹⁴ In this way, the same content may be

¹⁴We may assume that the agent utters the sentence *The girl sang*. (speaker mode), thereby hearing this utterance (hearer mode). After interpretation and storage of the content, the hearer turns into a speaker, remembers

sent back and forth ad infinitum.

For a model of natural language communication, this artificial, absolutely minimal example of turn taking must be extended systematically to more realistic exchanges. First, instead of one agent blindly processing the same sentence by switching between the hearer and the speaker mode, there should be at least two agents exchanging different sentences. Second, the expressions used in the dialog should include all syntactic moods, i.e., not only declarative, but also interrogative and imperative. Third, the interpretation of a sentence in the hearer mode should initiate the production of a *meaningful* response in the speaker mode.

To get an idea of what is required, consider the following examples:

3.4 EXAMPLES OF TURN TAKING USING DIFFERENT SYNTACTIC MOODS

1. Two declarative sentences:

A1S: The girl sang. :A2H
A2S: That is good. :A1H

2. Yes/no-interrogative and yes/no-answer:

A1S: Did the girl sing? :A2H
A2S: Yes. :A1H

3. Wh-interrogative and wh-answer:

A1S: Who sang? :A2H
A2S: The girl. :A1H

4. Imperative and responding action:

A1S: Sing! :A2H
A2S: performs action requested.

A1S stands for agent 1 in the speaker mode, A1H for agent 1 in the hearer mode, and correspondingly for A2S and A2H. The sign passed from the speaker, e.g. A2S, to the hearer, e.g. A1H, is placed between the two agents involved, for example A2S: *That is good.* :A1H. In the case of the imperative, the most appropriate response by the hearer, e.g. robot, is not the production of another sentence, but rather the performance of the action requested.

In order to extend the minimal system illustrated in 3.3 to the examples of 3.4, the syntactic-semantic parsing of LA-hear and the semantic-syntactic parsing of LA-speak must be up-scaled to handle the constructions in question. Equally importantly, however, LA-think must be extended to control the navigation underlying language production in a meaningful way.

These tasks are of an empirical nature. In other words, once the functioning of natural language communication has been solved in principle, the addition of new words to the lexicon, of new constructions to LA-hear and LA-speak, and of new inferences and behavior patterns to LA-think¹⁵ is just a matter of doing the descriptive work, both at the level of the declarative specification and the software implementation.

the content, and produces the utterance again, etc.

¹⁵In order to ensure long term upscalability of the system, the contextual database housing LA-think should be complemented with contextual recognition and action as soon as technically possible.

4 LA-Hear

We begin the process of incremental upscaling by describing the rule and derivation format of LA-think. Consider the rule DET+NN (determiner plus noun), defined as part of LA-hear.2 in Hausser 2002.

4.1 LA-HEAR RULE DET+NN

$$\text{DET+NN} \left[\begin{array}{l} \text{noun: } \textcircled{1} \\ \text{cat: } n' x \\ \text{sem: } y \end{array} \right] \left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: } n \\ \text{sem: } z \end{array} \right] \begin{array}{l} \text{delete } n' \text{ ss-cat} \\ \text{nw-sem } \boxed{a} \rightarrow \text{ss-sem} \\ \text{nw-noun } \boxed{r} \rightarrow \textcircled{1} \\ \text{copy}_{ss} \end{array} \quad \{\text{NOM+FV, FV+NP, S+IP}\}$$

Like the rules for the formal language $a^k b^k c^k$ defined in 2.2, this LA-hear rule consists of (i) the rule name DET+NN, (ii) patterns for the sentence start and the next word, defined as feature structures with variables and constants as values, (iii) a set of operations, and (iv) a rule package.

Conceptually, the difference between the rules for formal languages and for natural languages is most visible in their respective operations. In formal languages, the operations are mostly for adding, e.g. `leftcopy`, or subtracting e.g. `leftremove`, category segments in the `cat`-attribute of the sentence start. In natural language, in contrast, the operations are for canceling valency positions, e.g. `delete n' ss-cat`, and for copying values between proplets, e.g. `nw-sem \boxed{a} \rightarrow ss-sem`.

As an example of a complete LA-hear derivation consider the syntactic-semantic parsing of the girl sang.

4.2 LA-HEAR DERIVATION OF The girl sang.

```
lal.LA1-HEAR> the girl sang.
# Hearer mode #
1.
Start state, NW:      [cat: NN' NP]          rp:{DET+NN, DET+ADN, NOM+FV}
                      [sur: the   ]
                      [noun: _n1  ]
                      [cat: NN' NP]
                      [sem: def   ]
                      [mdr:       ]
                      [fnc:       ]
                      [idy: +1    ]
                      [prn:       ]
2.1
DET+NN: [cat: NN' NP] [cat: NN   ] delete_n'_ss-cat  rp:{NOM+FV,
[noun: 1   ] [noun: alpha] nw_noun-|r|->_n1  FV+NP, S+IP}
[sem: y   ] [sem: z   ] nw_sem-|a|->ss_sem
                      copy_ss
                      [sur: the   ] [sur: girl ]
                      [noun: _n1  ] [noun: girl ]
                      [cat: NN' NP] [cat: SN   ]
                      [sem: def   ] [sem: sg F  ]
                      [mdr:       ] [mdr:       ]
                      [fnc:       ] [fnc:       ]
                      [idy: +1    ] [idy: +1    ]
                      [prn:       ] [prn:       ]
3.11
NOM+FV: [noun: alpha ] [verb: beta ] nw_verb-|e|->ss_fnc  rp:{AUX+NFV,
[cat: NP   ] [cat: NS3' V] delete_np'_nw-cat  S+IP, FV+NP}
[fnc:     ] [arg:     ] ss_noun-|a|->nw_arg
                      [ctn:     ] ps_verb-|a|->nw_ctp
                      ps_prn-|a|->nw_ctp
                      nw_verb-|a|->ps_ctn
```

copy_nw

```
[sur:      ] [sur: sang ]
[noun: girl ] [verb: sing ]
[cat: SNP   ] [cat: N' V  ]
[sem: def sg F] [sem: past  ]
[mdr:      ] [mdr:      ]
[fnc:      ] [arg:      ]
[idy: 3     ] [ctn:      ]
[prn: 4     ] [ctp:      ]
[wrdn: 1    ] [prn:      ]

4.112
S+IP: [cat: V      ] [cat: V' DECL] delete_vt_ss-cat      rp:{IP+START}
      [verb: alpha] delete_vt'_nw-cat
                        nw_cat-|e|->ss_cat
                        copy_ss

[sur:      ] [sur: .      ]
[verb: sing ] [cat: V' DECL]
[cat: V      ]
[sem: past  ]
[mdr:      ]
[arg: girl  ]
[ctn:      ]
[ctp:      ]
[prn: 4     ]
[wrdn: 3    ]
```

Final State: [Branch: 112]

```
[sur:      ]
[verb: sing]
[cat: DECL ]
[sem: past ]
[mdr:      ]
[arg: girl ]
[ctn:      ]
[ctp:      ]
[prn: 4     ]
[wrdn: 3    ]
```

Parsing finished successfully.
Proplets Added to Wordbank.

```
[sur:      ] [sur:      ]
[noun: girl ] [verb: sing]
[cat: SNP   ] [cat: DECL ]
[sem: def sg F] [sem: past  ]
[mdr:      ] [mdr:      ]
[fnc: sing   ] [arg: girl ]
[idy: 3     ] [ctn:      ]
[prn: 4     ] [ctp:      ]
[wrdn: 1    ] [prn: 4     ]
                        [wrdn: 3    ]
```

The derivation begins with the start state pattern [cat: NN' NP] matching the lexical lookup of the first word, *the*. In composition 2.1, the first rule in the rule package of the start state, namely the familiar DET+NN, is applied to the lexical proplets *the* and *girl*, resulting in a more complete *girl*-proplet.

This proplet is shown as the sentence start of composition 3.11, in which the rule NOM+FV adds the third word of the sentence, *sang* as the *nw*, copying the value *girl* into the arg-attribute of *sing* and the value *sing* into the fnc-attribute of *girl*. Finally, in composition 4.112 the full stop is added by the rule S+IP (sentence plus interpunctuation), changing the value of the cat-attribute of *sing* to DECL (declarative).

The natural language derivation 4.2 resembles those for the formal languages $a^k b^k c^k$ in 2.4 and WW in 2.5 in that the explicit rule patterns are aligned above the input proplets. The result of the derivation is a set of two proplets, *girl* and *sing*, which are ready to be sorted into the word bank.

This LA-hear derivation may seem like a conventional linguistic analysis (and associated parser) insofar as both take language expressions as input and render some analysis as output. They differ, however, in that the object of a conventional linguistic analysis is the language sign, independent of the process of natural language communication, while the object of LA-hear is a computational model of the hearer mode.

From this important difference in their goals there follow several technical differences between the two approaches:

4.3 COMPARING LA-HEAR WITH A CONVENTIONAL ANALYSIS

1. *Lexical semantics of content words*

Conventional linguistics offers various different methods to characterize word meaning, ranging from semantic primitives, oppositions, metalanguage definitions, truth functions, etc., none of which are integrated into a computationally viable theory of cognition.

Database Semantics defines lexical semantics in terms of (i) isolated proplets (feature structures) and (ii) the concepts serving as the values of their part-of-speech attributes. The concepts are procedurally defined and originate in the recognition and action of peripheral cognition.

2. *Compositional semantics*

Conventional linguistics characterizes ‘what belongs together semantically’ in terms of constituent structures represented as trees and motivated by movement and substitution tests. In categorial grammar, constituent structures double as functor-argument structures. The semantics analysis does not go beyond the limits of the sentence.

Database Semantics characterizes the functor-argument structure in terms of attribute values rather than trees. Compositional semantics comprises all the values provided in addition to lexical semantics – resulting from the copying operations of the LA-hear rules. The semantic analysis includes the concatenation of sentences in terms of conjunction and identity.

3. *Relation between syntax and semantics*

Conventional linguistics treats syntax and semantics as two separate levels, whereby the semantics is usually defined for a preexisting autonomous syntax.

Database Semantics treats syntactic and semantic parsing simultaneously in terms of connecting proplets via value copying, without a separate or autonomous level of syntax. Semantically uninterpreted languages simply have fewer and different operations than interpreted ones (compare 2.5 and 4.2).

4. *Derivation order and derivation results*

Conventional linguistics uses a derivation order based on the principle of possible substitutions, resulting in hierarchies represented as trees.

Database Semantics uses a time-linear derivation order based on the principle of possible continuations, resulting in unordered sets of proplets.

Other differences are surface compositionality, type transparency, and the functionality of reference based on internal matching. All these properties of Database Semantics in general and LA-hear in particular are necessary for achieving a computationally viable, functional model of the hearer mode.

5 LA-Think

The second kind of LA-grammar used in Database Semantics is LA-think. While the task of LA-hear is filling the database with content by interpreting natural language, the task of LA-think is navigating autonomously through the content of the database. This navigation models the process of thought and is used for (i) inferencing and (ii) the speaker’s conceptualization.

For simplicity, let us consider a word bank into which LA-hear has read the content of the sentence *The girl sang.*, as described in the previous section:

5.1 PROPLETS REPRESENTING *The girl sang.* IN THE WORD BANK

```
Owner Proplets |----- token line ----->
[noun: girl] [sur:      ]
               [noun: girl ]
               [cat: SNP   ]
               [sem: def sg F]
               [mdr:      ]
               [fnc: sing  ]
               [idy: 3     ]
               [prn: 4     ]
               [wrdn: 1    ]
               [trc: 2 6   ]
```

```
Owner Proplets |----- token line ----->
[verb: sing] [sur:      ]
               [verb: sing ]
               [cat: DECL  ]
               [sem: past  ]
               [mdr:      ]
               [arg: girl  ]
               [ctn:      ]
               [ctp:      ]
               [prn: 4     ]
               [wrdn: 3    ]
               [trc:      ]
```

This display of the database content shows two isolated proplets, [noun: girl] and [verb: sing], and two corresponding connected proplets (underneath the terms ‘token line’). The isolated (lexical) proplets are simplified to show only the part of speech attributes and their values (concepts). The word bank 5.1 differs from the result of derivation 4.2 in that the proplets in 5.1 are ordered, using the concepts as the key.

Next consider the definition of an LA-think rule for navigating through this word bank, namely **V_N_V** of LA-think.2 defined in Hausser 2002.

5.2 LA-THINK RULE **V_N_V**

$$\mathbf{V_N_V} \left[\begin{array}{l} \text{verb: } \beta \\ \text{arg: } \#x \alpha y \\ \text{prn: } m \end{array} \right] \left[\begin{array}{l} \text{noun: } \alpha \\ \text{func: } \beta \\ \text{mdr: NIL} \\ \text{prn: } m \end{array} \right] \begin{array}{l} \text{mark } \alpha \text{ in ss-arg} \\ \text{output position ss} \\ \text{activate LA-speak.2} \end{array} \quad \{ 3 \mathbf{V_N_V}, 4 \mathbf{V_N_N}, 5 \mathbf{V_V_V} \}$$

Like the LA-hear rule 4.1, this LA-think rule consists of (i) the rule name **V_N_V** (for verb-noun-verb), (ii) two patterns, one for the current proplet (serving as the ‘sentence start’) and

one for the next proplet (serving as the ‘next word’), (iii) a set of operations, and (iv) a rule package.

As an example of an LA-think navigation consider traversal of the proposition the girl sang, represented by the proplets *sing girl*.

5.3 LA-THINK NAVIGATION TRAVERSING *sing girl*

```

la1.LA1-THINK> sing
# Think mode #

1.
Start state, NW:      [verb: sing]                rp:{V_N_V, V_N_N}

                        [sur:      ]
                        [verb: sing]
                        [cat: DECL ]
                        [sem: past ]
                        [mdr:      ]
                        [arg: girl ]
                        [ctn:      ]
                        [ctp:      ]
                        [prn: 5    ]
                        [wrdn: 3   ]

2.
V_N_V: [verb: alpha] [noun: beta ] mark alpha in ss-arg  rp:{V_N_V,
      [arg: beta  ] [fnc: alpha ] output pos ss      V_N_N, V_V_V}
      [prn: m    ] [prn: m    ] activate LA-sp
      [mdr:      ] [mdr:      ]

      [sur:      ] [sur:      ]
      [verb: sing ] [noun: girl ]
      [cat: DECL ] [cat: SNP   ]
      [sem: past ] [sem: def sg F]
      [mdr:      ] [mdr:      ]
      [arg: girl ] [fnc: sing  ]
      [ctn:      ] [idy: 4    ]
      [ctp:      ] [prn: 5    ]
      [prn: 5    ] [wrdn: 1   ]
      [wrdn: 3   ]

-----
##### End of navigation

Activated sequence of proplets:

[sur:      ] [sur:      ]
[verb: sing] [noun: girl ]
[cat: DECL ] [cat: SNP   ]
[sem: past ] [sem: def sg F]
[mdr:      ] [mdr:      ]
[arg: #girl] [fnc: sing  ]
[ctn:      ] [idy: 4    ]
[ctp:      ] [prn: 5    ]
[prn: 5    ] [wrdn: 1   ]
[wrdn: 3   ] [trc: 10   ]
[trc: 9    ]

```

LA-think is activated by the input of a first concept which starts the navigation, here *sing*. This concept is usually provided automatically by autonomous recognition. For example, recognizing a feeling of hunger may start a navigation through recent hunger satisfaction routines, recognizing the Empire States Building may start the agent to reminisce about a recent trip to New York, recognizing dirty dishes may start dish washing activity, etc., depending on the agent’s current set of priorities. In our very simple example, the concept starting the LA-think navigation is provided by the user typing *sing* after the top line prompt *la1.LA1-THINK>*.

If the word bank contained several connected *sing* proplets, they would be presented to the user with a request to choose one. In our example, however, this step is omitted because the

word bank contains only one such proplet. In derivation step 1, this proplet is matched with the start state of LA-think.

Next applies the rule `V_N_V` contained in the rule package of the start state. By matching the first pattern with the proplet *sing*, the variables `alpha`, `beta`, and `m` are bound to the values *sing*, *girl*, and 5, respectively. These bound variables reoccur in the second pattern, thus providing all the information necessary to retrieve a legitimate successor proplet, *girl*. The navigation ends after the application of `V_N_V` because the word bank in our example contains only two proplets.

To prevent redundant traversal of the same verb noun sequence, the argument currently traversed is marked with # (operation mark `alpha in ss-arg`). This is necessary because in propositions with more than one argument, e.g. *give Mary man book*, the rule `V_N_V` calls itself in its rule package, activating the following proplet sequence in the word bank:

| V_N_V operations | activated sequence |
|-----------------------|-----------------------------------------|
| <i>give_Mary_give</i> | <i>give Mary</i> |
| <i>give_man_give</i> | <i>give Mary man</i> |
| <i>give_book_give</i> | <i>give Mary man book</i> ¹⁶ |

In such a derivation, the `arg` attribute of the V proplet will start out with the values *Mary man book*. After traversing the first N proplet, the mark operation of `V_N_V` changes these values to *#Mary man book*, after the second N proplet to *#Mary #man book*, and after the third to *#Mary #man #book*.

In order to allow repeated but non-redundant application of the rule `V_N_V`, the variable `alpha` in the `ss` pattern is bound to *Mary* in the value sequence *Mary man book*, to *man* in the value sequence *#Mary man book*, and to *book* in the *#Mary #man book*. Relative to the value sequence *#Mary #man #book*, the value assignment to the variable `alpha` fails, aborting this latest attempt at applying the rule.

The # marking of values by LA-think rules (only in LA-think.2 and higher) is of a temporary nature and does not change the content of the database. After completion of an intrapropositional navigation, the markers are automatically removed.

The second rule operation in 5.2, i.e., `output position ss`, specifies that the navigation returns to the `ss` proplet. This ensures that the current `ss` proplet, i.e. the verb, will be the new `ss` in the next rule application.

The third rule operation in 5.2, i.e., `activate LA-speak.2`, switches the system from LA-think.2 to LA-speak.2 for the realization of the surfaces derived from the proplet traversed. The rules of LA-speak.2 have complementary operations which switch the system back to LA-think.2 for continuation of the navigation.

6 LA-Speak

The third type of LA-grammar used in Database Semantics is called LA-speak. LA-speak is driven by an LA-think navigation.¹⁷ As an example of an LA-speak rule consider **-DET**.

¹⁶Next, the navigation may continue extrapropositionally from *give* (see left column) to the verb of another proposition, provided the current verb's `ctp` (connective to previous) or `ctn` (connective to next) attribute has a value.

¹⁷For simplicity this process has been described in 3.3 as copying the proplets traversed by the LA-think navigation and using the copies as input to LA-speak. In a more parsimonious solution, LA-speak merely matches the proplets activated by the LA-think navigation and derives the desired output without any copying of database proplets.

This rule is for precipitating the determiner of a noun phrase, and is defined in Hausser 2002 as part of LA-speak.2.

6.1 LA-speak rule -DET

$$\text{-DET} \left[\begin{array}{l} \text{verb: } \beta \\ \text{arg: } \#x \# \alpha y \\ \text{prn: } m \end{array} \right] \left[\begin{array}{l} \text{noun: } \alpha \\ \text{mdr: } z \\ \text{func: } \beta \\ \text{prn: } m \end{array} \right] \text{lex-d } \left[\begin{array}{l} \text{noun: } \alpha \\ \text{if } z \neq \text{NIL activate LA-think.2} \end{array} \right] \{-\text{ADN}, -\text{NOUN}\}$$

Like the LA-hear rule 4.1 and the LA-think rule 5.2, this LA-speak rule consists of (i) the rule name -DET, (ii) two patterns, (iii) a set of operations, and (iv) a rule package.

The precipitation of the determiner from the noun proplet is handled by the -DET operation lex-d, which is defined as follows:

6.2 The function lex-d for realizing determiners

If one of the following patterns matches an activated proplet N, then lex-d produces the associated surface:

| pattern | surface | pattern | surface |
|---------------------------------------------------------------------------------------------------------------|---------|-------------------------------------------------------------------------------------------------------------|---------|
| $\left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: SNP} \\ \text{sem: indef sg} \end{array} \right]$ | a(n) | $\left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: SNP} \\ \text{sem: pl exh} \end{array} \right]$ | every |
| $\left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: NP} \\ \text{sem: sel x} \end{array} \right]$ | some | $\left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: PNP} \\ \text{sem: pl exh} \end{array} \right]$ | all |
| $\left[\begin{array}{l} \text{noun: } \alpha \\ \text{cat: NP} \\ \text{sem: def x} \end{array} \right]$ | the | | |

Other lex-operations of LA-speak are lex-n for realizing names or pronouns, lex-nn for nouns, lex-fv for finite verb forms, lex-ax for auxiliaries, lex-nv for non-finite verb forms, lex-adn for adnominal forms, and lex-p for interpunctuation signs. The method illustrated in 6.2 is the production counterpart to automatic word form recognition.

As an example of an LA-speak derivation consider production of the surface *the girl sang*. from the activated database proplets *sing girl*.

6.3 LA-speak producing *The girl sang*.

```

##### LA-SPEAK
Start state, NW:      [verb: alpha]                rp:{-DET, -NoP}

                    [sur:          ]
                    [verb: sing ]
                    [cat: DECL  ]
                    [sem: past  ]
                    [mdr:       ]
                    [arg: #girl ]
                    [ctn:       ]
                    [ctp:       ]
                    [prn: 4     ]
                    [wrdn: 3    ]
                    [trc: 9     ]

```


The LA-speak rules apply in sequence, using their lex-functions to extract the surfaces in question from the two proplets activated by the LA-think navigation.

In propositions consisting of two proplets, like the above example, the LA-think navigation traverses the proposition in question completely, then switches to LA-speak, produces the surface completely, and finally switches back to LA-think (for example, to initiate traversal of the next proposition). In propositions consisting of more than two proplets, however, the actions of LA-think and LA-speak interact incrementally.

For example, the sentence *The heavy old car hit the beautiful tree.* is based on the proplet sequence *hit car heavy old tree beautiful*, represented abstractly as VNAANA. After LA-think has traversed the first two proplets *hit car*, the system switches to LA-speak to produce the surface *the*. Then it switches to LA-think to traverse *heavy*, switches to LA-speak to produce *heavy*, switches to LA-think to traverse *old*, switches to LA-speak to produce *old*, *car*, and *hit*, switches to LA-think to traverse *tree*, switches to LA-speak to produce *the*, switches to LA-think to traverse *beautiful*, and finally switches to LA-speak to produce *beautiful*, *tree* and the full-stop.

This incremental interaction between LA-think and LA-speak is motivated not only by psychological considerations, but also by computational efficiency. The reason is that realizing the surface of proplets as soon as possible (before the navigation has reached the end of the proposition) results in a more restricted set of candidates for matching by the LA-speak rules than the activated sequence for a complete proposition.¹⁸

7 Conclusion

Our computational model of turn taking is based on two basic innovations: (i) the algorithm of LA-grammar (TCS'92) and (ii) the data structure of a word bank (AIJ'01). The algorithm of LA-grammar is based on a strictly time-linear derivation order. The data structure of a word bank represents content in the form of concatenated propositions, defined as (unordered) sets of feature structures called proplets.

Proplets code the intrapropositional functor-argument structure and the extrapropositional relations of identity and conjunction as attribute values, independent of any graphical restrictions. For this reason, proplets are optimal for the time-linear storage and retrieval as required for the theoretical reconstruction of natural language communication, including turn taking.

Communication is modeled by three LA-grammars applying in sequence. LA-hear interprets natural language as a set of proplets which are sorted into the word bank. LA-think activates proplets in the word bank by navigating from one proplet to the next, serving as conceptualization. LA-speak maps the proplets traversed by the autonomous navigation into corresponding natural language surfaces.

These LA-grammars are explicitly defined for a substantial fragment of English and serve as the declarative specification of an initial version of the overall system. They are supported by an implementation in Java, which will be demoed as part of the talk.

¹⁸This holds especially at the beginning of a derivation. Later, as more proplets are added to the activated sequence, the earlier proplets have been systematically marked in past LA-think rule applications – which reduces the set of matching candidates from the other end. As a consequence, a maximally incremental system can guide the intended matching with much simpler and more efficient pattern definitions than one which activates future proplets earlier than necessary.

Bibliography

- Berwick, R.C. & A.S. Weinberg (1984) *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition*, Cambridge, Mass.: MIT Press.
- Hausser, R. (2002) *Human-Computer Communication in Natural Language*, unpublished manuscript, p. 140. Available at: <http://hakata.mt.cs.cmu.edu/11-792/hausser3.pdf>.
- Kycia, A. (2004) *Implementierung der Datenbanksemantik in Java*, MA-thesis in progress, Universität Erlangen-Nürnberg.
- Thórisson, K. (2002) "Natural Turn-Taking needs no Manual: Computational Theory and Model, from Perception to Action," in Granström, B., D. House, & Karlsson (eds.) *Modality in Language and Speech System*, p. 173–207, Dordrecht: Kluwer.

Abbreviations:

- NEWCAT'86 = Hausser, R. (1986) *NEWCAT: Natural Language Parsing Using Left-associative Grammar*. (Lecture Notes in Computer Science 231), pp. 540, Berlin, New York: Springer-Verlag.
- CoL'89 = Hausser, R. (1989) *Computation of Language, An Essay on Syntax, Semantics and Pragmatics in Natural Man-Machine Communication*, Symbolic Computation: Artificial Intelligence, pp. 425, Berlin, New York: Springer-Verlag.
- TCS'92 = Hausser, R. (1992) "Complexity in Left-Associative Grammar," *Theoretical Computer Science*, Vol. 106.2:283-308, Dordrecht: Elsevier.
- FoCL'99 = Hausser, R. (1999/2001) *Foundations of Computational Linguistics, Human-Computer Communication in Natural Language*. 2nd Edition 2001, pp. 578, Berlin, New York: Springer-Verlag.
- AIJ'01 = Hausser, R. (2001) "Database Semantics for Natural Language." *Artificial Intelligence*, Vol. 130.1:27–74, Dordrecht: Elsevier.