Language Production Based on Autonomous Control
A Content-Addressable Memory for a Model of Cognition

Roland Hausser

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

Abstract This paper analyzes natural language production as the speaker mode of a cognitive agent. By integrating it functionally into the cycle of natural language communication, language production participates in all conceptual and computational constructs motivated by the other two steps of the cycle, namely the hearer mode and the think mode, as modeled in Database Semantics (DBS).
For the speaker mode, the agent's cognition must supply the content to be mapped into natural language surfaces. This content is provided by two sources: (i) the agent's recognition, either current or stored in memory, and (ii) blue-prints for action derived on-the-fly by the agent's autonomous control.
While earlier DBS analyses of the speaker mode concentrated on language production from stored content, this paper explores the foundations of an autonomous control, including the control of language production. To bring across the basic ideas, the presentation tries to be as intuitive as possible. Nevertheless, the formal illustrations of contents, patterns, and rules as well as the description of the component structure and functional flow provide the outline of a declarative specification for a straightforward transfer into efficiently running code.[1]

Introduction

A model of cognition requires a memory – which raises the question of its structure. The most basic choice is between a *coordinate*-addressable and a *content*-addressable memory (cf. Chisvin and Duckworth 1992 for an overview). Though peppered with patents, the content approach is much less widely used than the coordinate approach. The content approach is applied mostly for the super-fast retrieval of fixed content.

A coordinate-addressable memory resembles a modern public library in which books can be stored wherever there is space (random access) and retrieved using a separate index (inverted file) relating a primary key (e.g., author, title, year) to its location of storage (e.g., 1365). A content-addressable memory, in contrast, is like a private library in which books with certain properties are grouped together on certain shelves, ready to

[1] More details may be found in earlier publications. Implementation of the SLIM theory of language as a computer program began in 1983–86 at Stanford U., published in NEWCAT'86 with the Lisp source code for parsing 221 syntactic constructions of German. In 1986–89 the software was extended at CMU Pittsburgh to the 421 syntactic-semantic constructions of English underlying CoL'89. Since then, new versions have been written continuously, first in C (up to 2002) and then in Java (2002 to present).

be browsed without the help of a separate index. For example, at Oxford University the 2 500 volumes of Sir Thomas Bodley's library from the year 1598 are still organized according to the century and the country of their origin.

As an initial reaction to a content-addressable approach, main stream database scientists usually point out that it can be simulated by the coordinate-addressable approach (Fischer 2002), using well-established relational databases. The issue here, however, is whether the formal intuitions of the content-addressable approach can be refined naturally into a model of cognition or not.

Our point of departure is the software system of Database Semantics (NLC'06), which was designed to accommodate certain obvious requirements of cognition[2] and computation.[3] It was in hindsight that this system's memory, called Word Bank, was recognized to be content-addressable. The following pages explore the ramifications of this realization for modeling cognition and autonomous control at a level of abstraction which is applicable to natural and artificial agents alike.

1 Data Structure of Proplets

In Database Semantics, propositional content is coded as an order-free set of flat (non-recursive) feature structures called *proplets*, serving as the data structure (a.k.a. abstract data type). Consider the following example of coding a content:

1.1 PROPLETS CODING THE CONTENT OF Julia knows John.

$$\begin{bmatrix}\text{noun: Julia}\\ \text{cat: nm}\\ \text{fnc: know}\\ \text{prn: 625}\end{bmatrix} \quad \begin{bmatrix}\text{verb: know}\\ \text{cat: decl}\\ \text{arg: Julia John}\\ \text{prn: 625}\end{bmatrix} \quad \begin{bmatrix}\text{noun: John}\\ \text{cat: nm}\\ \text{fnc: know}\\ \text{prn: 625}\end{bmatrix}$$

In a proplet, the lexical and the compositional aspects of meaning are systematically distinguished. The lexical aspect is represented by the core value, e.g., know, of the core attribute specifying the part of speech, e.g., verb. The compositional aspect is represented by the continuation attribute(s), e.g., arg, and its continuation value(s), e.g., Julia John, which serve to code compositional semantic relations between proplets, namely functor-argument and coordination structure, intra- as well as extrapropositionally. The order-free proplets of a proposition are held together by a common prn (for proposition number) value, here 625.

2 Database Schema of a Word Bank

In their most basic form, the core values are defined as concepts which are implemented as recognition and action procedures of the cognitive agent and represented by corresponding English words. The letter sequence of these concept names is used to completely determine a proplet's location for storage and retrieval:

[2] Such as an agent with a body and external interfaces, a language and a context component, the sign kinds of symbol, index, and name, a speaker and a hearer mode, etc.

[3] Such as the definition of a data structure with a time-linear algorithm for storing, processing, and retrieving content in a suitable database schema, low complexity, easy upscaling, etc.

2.1 Storing the proplets of 1.1 in a content-addressable Word Bank

$$\begin{array}{lllcr}
 & \multicolumn{2}{r}{\textit{member proplets}} & \textit{position for new} & \textit{owner proplets} \\
 & & \ldots & \textit{member records} & \ldots \\
\ldots & \left[\begin{array}{l}\text{noun: John}\\ \text{cat: nm}\\ \text{fnc: ...}\\ \text{prn: 610}\end{array}\right] & \left[\begin{array}{l}\text{noun: John}\\ \text{cat: nm}\\ \text{fnc: know}\\ \text{prn: 625}\end{array}\right] & & \left[\text{core: John}\right] \\
 & & \ldots & & \ldots \\
\ldots & \left[\begin{array}{l}\text{noun: Julia}\\ \text{cat: nm}\\ \text{fnc: ...}\\ \text{prn: 605}\end{array}\right] & \left[\begin{array}{l}\text{noun: Julia}\\ \text{cat: nm}\\ \text{fnc: know}\\ \text{prn: 625}\end{array}\right] & & \left[\text{core: Julia}\right] \\
 & & \ldots & & \ldots \\
\ldots & \left[\begin{array}{l}\text{verb: know}\\ \text{cat: decl}\\ \text{arg: ...}\\ \text{prn: 608}\end{array}\right] & \left[\begin{array}{l}\text{verb: know}\\ \text{cat: decl}\\ \text{arg: Julia John}\\ \text{prn: 625}\end{array}\right] & & \left[\text{core: know}\right] \\
 & & \ldots & & \ldots
\end{array}$$

This conceptual schema resembles a classic network database with owner records and member records (cf. Elmasri and Navathe 1989). It is just that the records are represented equivalently as proplets.

A sequence of member proplets followed by an owner proplet is called a *token line*. The proplets in a token line must all have the same core value and are in the temporal order of their arrival (reflected by the value of a proplet's prn attribute).

In contrast to the task of designing a practical schema for arranging the books in a private library, the sorting of proplets into a Word Bank is simple and mechanical. It is content-addressable in the sense that no separate index (inverted file) is required because the letter sequence of a proplet's core value completely determines its location of storage, namely the penultimate position of the corresponding token line.

Furthermore, it is scalable (a property absent or problematic in some other content-addressable systems). This is because the cost of insertion is constant, independent of the size of the stored data, and the cost of retrieving a specified proplet grows only logarithmically with the data size (external access) or is constant (internal access). External access to a proplet requires (i) its core and (ii) its prn value, e.g., know 625.[4] Most cognitive operations, however, require internal access, as in the navigation from one proplet to the next (e.g., 4.2). Because content is fixed, internal access may be based on pointers, resulting in a major speed advantage over the coordinate-addressable approach.

3 Pinball Machine Model of Cognition

The books in a private library and the proplets in a Word Bank are alike in that they are each used by a cognitive agent. It is just that in a private library the cognitive agent is located inside the arrangement of books on their shelves, while the proplets in a Word Bank are located inside the cognitive agent.

[4] The token line for any core value is found by using a trie structure (Fredkin 1960). Finding a proplet within a token line may be based on binary search (*O(log(n))*, Corman et al. 2009) or

As a consequence, the role of the browsing user in a private library seems to be left vacant in a Word Bank. DBS fills the position, however, with a *focus*, which is like a point of light navigating from proplet to proplet. This raises the question of how such a navigation through the content of a Word Bank should be controlled.

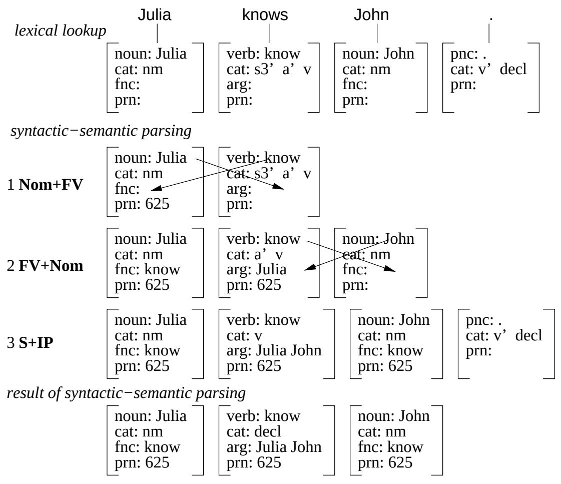
The control is not located in the focus itself. Instead, it is provided in part by the semantic relations between the proplets, which are used like a railroad system for the navigation. Thus, just as the core value of a proplet serves the double function of (i) representing the lexical semantics and (ii) determining the location for storage and retrieval, its continuation value(s) serve the double function of (i) representing the compositional semantics and (ii) establishing a railroad system for the focus navigation, providing each proplet with only a limited choice of successor proplets.

Putting it metaphorically, the navigation of the focus through the railroad system of connected proplets resembles a pinball machine, with the focus serving as the ball and the slanted playing field with its springs and levers serving as the railroad system. There is no “retrieval” in the usual sense. Instead, the navigation of the focus merely *visits* successor proplets – as an activation which may be visualized as lighting up the proplets traversed, with an afterglow of suitable duration and rate of decay (annealing).

4 Cycle of Natural Language Communication

Having outlined the specifics of the content-addressable DBS memory, let us review the cognitive operations which are based on it. We begin with the cycle of natural language communication (AIJ’01), consisting of the hearer mode, the think mode, and the speaker mode. Consider the following hearer mode derivation:

4.1 DBS HEARER-MODE DERIVATION OF Julia knows John.

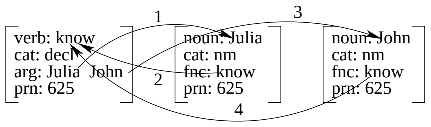

Julia knows John .
lexical lookup
noun: Julia cat: nm fnc: prn:
verb: know cat: s3’ a’ v arg: prn:
noun: John cat: nm fnc: prn:
pnc: . cat: v’ decl prn:
syntactic−semantic parsing
1 Nom+FV
noun: Julia cat: nm fnc: prn: 625
verb: know cat: s3’ a’ v arg: prn:
2 FV+Nom
noun: Julia cat: nm fnc: know prn: 625
verb: know cat: a’ v arg: Julia prn: 625
noun: John cat: nm fnc: prn:
3 S+IP
noun: Julia cat: nm fnc: know prn: 625
verb: know cat: v arg: Julia John prn: 625
noun: John cat: nm fnc: know prn: 625
pnc: . cat: v’ decl prn:
result of syntactic−semantic parsing
noun: Julia cat: nm fnc: know prn: 625
verb: know cat: decl arg: Julia John prn: 625
noun: John cat: nm fnc: know prn: 625

interpolation (*O(log(log (n)))*, Weiss 2005), where *n* is the length of the token line. The search uses the prn value of the address in relation to the strictly linear increasing prn values of the token line. Thus, there is no need for a hash function (which is unusual compared to most other content-addressable approaches).

In line with the SLIM theory of language (cf. FoCL'99, NLC'06), the analysis is Surface compositional in that each word form is analyzed as a lexical proplet (lexical lookup, cf. Handl et al. 2009). The derivation is time-Linear, as shown by the stair-like addition of one lexical proplet in each new line. Each line represents a derivation step, based on the application of the specified LA-hear grammar rule, e.g., **Nom+FV** (cf. 5.1). The rules establish semantic relations by copying values, as indicated by the diagonal arrows.

The result of the derivation is an order-free set of proplets, suitable for storage in the agent's content-addressable memory (as shown in 2.1). Based on the semantic relations between the stored proplets, the second step in the cycle of natural language communication is a navigation which activates content selectively in the think mode:

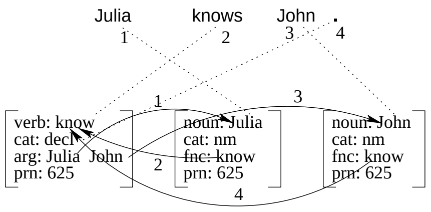
4.2 DBS THINK MODE NAVIGATION



Using the arg, fnc, and prn values, the navigation proceeds from the verb to the subject noun (1), back to the verb (2), to the object noun (3), and back to the verb (4).

Such a think mode navigation provides the *what to say* for language production from stored content, while the third step in the cycle of communication, i.e., the speaker mode, provides the *how to say it* (McKeon 1985) in the natural language of choice:

4.3 DBS SPEAKER MODE REALIZATION



The surfaces are realized from the *goal* proplet of each navigation step, using mainly the core values. In NLC'06, the DBS cycle of communication has been worked out in detail for more than 100 English constructions of intra- and extrapropositional functor-argument and coordination structures as well as coreference.[5]

5 Algorithm of LA-Grammar

Having shown how the data structure of proplets may be used for mapping surfaces into content (cf. 4.1), for activating content selectively (cf. 4.2), and for mapping content into surfaces (cf. 4.3), let us turn to the formal rules performing these procedures. Given that the three steps of the cycle of natural language communication are time-linear, i.e.,

[5] For a concise summary see Hausser 2009.

linear like time and in the direction of time, they may be handled by the same algorithm, namely time-linear LA-grammar (TCS'92).

The rules of an LA-hear grammar combine a “sentence start” (*ss*) with a “next word” (*nw*) into a new “sentence start” (*ss*′). Furthermore, the rules are formulated on the basis of pattern proplets, i.e., proplets containing variables as values, which connect to the input level by means of pattern matching.

As an example, consider the rule application of the first derivation step in 4.1 (explanations in italics):

5.1 LA-HEAR RULE APPLICATION

i. *rule name* ii. *rule package*

Nom+FV {FV+Nom}

iii. *ss-pattern* iv. *nw-pattern* v. *resulting ss-pattern*

$$
\begin{array}{lccccc}
\textit{rule level} &
\begin{bmatrix}\text{noun: }\alpha\\ \text{cat: NP}\\ \text{fnc:}\end{bmatrix} &
\begin{bmatrix}\text{verb: }\beta\\ \text{cat: NP}' \text{ X VT}\\ \text{arg:}\end{bmatrix} &
\Rightarrow &
\begin{bmatrix}\text{noun: }\alpha\\ \text{cat: NP}\\ \text{fnc: }\beta\end{bmatrix} &
\begin{bmatrix}\text{verb: }\beta\\ \text{cat: X VT}\\ \text{arg: }\alpha\end{bmatrix}\\
\multicolumn{3}{l}{\textit{matching and binding of variables} \ \Uparrow} & & \textit{output} & \Downarrow\\
\textit{language level} &
\begin{bmatrix}\text{noun: Julia}\\ \text{cat: nm}\\ \text{fnc:}\\ \text{prn:}\end{bmatrix} &
\begin{bmatrix}\text{verb: know}\\ \text{cat: s3' a' v}\\ \text{arg:}\\ \text{prn:}\end{bmatrix} & &
\begin{bmatrix}\text{noun: Julia}\\ \text{cat: nm}\\ \text{fnc: know}\\ \text{prn: 1}\end{bmatrix} &
\begin{bmatrix}\text{verb: know}\\ \text{cat: a' v}\\ \text{arg: Julia}\\ \text{prn:1}\end{bmatrix}
\end{array}
$$

An LA-hear grammar rule consists of (i) a rule name, (ii) a rule package, (iii) a pattern for an *ss* (sentence start), (iv) a pattern for an *nw* (next word), and (v) a pattern for the resulting (next) sentence start. A pattern at the rule level matches a proplet at the language level if (a) the attributes of the pattern are a subset of the attributes of the proplet and (b) the values of the pattern are compatible with the values of the proplet.

By binding the variables of the input patterns (rule level) to the corresponding constants at the language level, the output can be derived at the language level. For example, by binding the variable α of the input pattern to the constant *Julia* (language level), the [arg: α] feature of the resulting *ss*-pattern (rule level v) provides the value Julia to the arg attribute of the output at the language level. If the current rule application is successful, the resulting sentence start is provided with the proplet of a next word (if available) by automatic word form recognition, resulting in a new input pair to which the rule(s) of the current rule package are applied.[6]

Variants of the method illustrated in 5.1 are used for the other two steps of the communication cycle, i.e., the think mode powered by LA-think and the speaker mode, realized by LA-speak. One basic task[7] of an LA-think grammar is a selective activation of content stored in the Word Bank by navigating from one proplet to the next, following the grammatical relations between them. The LA-think rules for this operation take a current proplet as input and compute a next proplet as output.

[6] The 100 English constructions analyzed in NLC'06 have been programmed for Chinese (Mei 2007), German (Mehlhaff 2007), Russian (Kalender 2009), and Tagalog (Söllch 2009), among others. See Hausser (2008) on the handling of different word orders in DBS.

[7] The other task is the on-the-fly application of inferences.

Consider, for example, the LA-think rule **VNs**, which executes navigation step 1 from the first to the second proplet in 4.2:

5.2 LA-THINK RULE APPLICATION

$$\begin{array}{llcl}
 & \text{i. } \textit{rule name} & & \text{ii. } \textit{rule package} \\
 & \textbf{VNs} & & \{\text{NVs}\} \\
 & \text{iii. } \textit{current proplet} & & \text{iv. } \textit{next proplet} \\
\textit{rule level} & \begin{bmatrix}\text{verb: } \beta \\ \text{arg: X } \alpha \text{ Y} \\ \text{prn: K}\end{bmatrix} & \Rightarrow & \begin{bmatrix}\text{noun: } \alpha \\ \text{fnc: } \beta \\ \text{prn: K}\end{bmatrix} \\
\textit{matching and binding of variables} & \Uparrow & & \Downarrow \\
\textit{Word Bank level} & \begin{bmatrix}\text{verb: know} \\ \text{cat: decl} \\ \text{arg: Julia John} \\ \text{prn: 625}\end{bmatrix} & & \begin{bmatrix}\text{noun: Julia} \\ \text{cat: nm} \\ \text{fnc: know} \\ \text{prn: 625}\end{bmatrix}
\end{array}$$

By using the same variables α, β, and K in the pattern for the current proplet and in the pattern for the next proplet, and by binding them to the values know, Julia, and 625 of the input proplet *know*, the pattern for the next proplet provides the information required for visiting the successor proplet, here *Julia*.

Finally consider the LA-speak grammar. Its rules are like LA-think rules, except that they are extended to produce appropriate word form surfaces by using the core value as well as the morphosyntactic information of the cat and sem attributes. The following example shows the LA-speak rule application underlying transition 2 in 4.3, which navigates from the noun *Julia* back to the verb *know*, mapping the core value of the goal proplet into the appropriate surface know+s.

5.3 LA-SPEAK RULE APPLICATION

$$\begin{array}{llclcl}
 & \text{i. } \textit{rule name} & & \text{ii. } \textit{rule package} & & \\
 & \textbf{NVs} & & \{\text{VNs}\} & & \\
 & \text{iii. } \textit{current proplet} & & \text{iv. } \textit{next proplet} & & \textit{output pattern} \\
\textit{rule level} & \begin{bmatrix}\text{noun: } \alpha \\ \text{cat: sn} \\ \text{fnc: } \beta \\ \text{prn: K}\end{bmatrix} & \Rightarrow & \begin{bmatrix}\text{verb: } \beta \\ \text{sem: pres} \\ \text{arg: } \alpha \text{ Y} \\ \text{prn: K}\end{bmatrix} & \Rightarrow & \beta\text{+s} \\
\textit{matching and binding of variables} & \Uparrow & & \Downarrow & & \Downarrow \\
\textit{Word Bank level} & \begin{bmatrix}\text{noun: Julia} \\ \text{cat: nm} \\ \text{fnc: know} \\ \text{prn: 625}\end{bmatrix} & & \begin{bmatrix}\text{verb: know} \\ \text{sem: pres} \\ \text{arg: Julia John} \\ \text{prn: 625}\end{bmatrix} & & \text{know+s}
\end{array}$$

As in an LA-think grammar, the output proplet (here *know*) serves as the input for the next rule application(s). The difference between an LA-think and an LA-speak rule is that the latter also produces a surface (here know+s) used as a blueprint for the agent's language synthesis component.

6 Retrieving Answers to Questions

So far, the database schema of a Word Bank, i.e., ordered token lines listing connected proplets (cf. 2.1), has been shown to be suitable for (i) storage in the hearer mode (cf. 4.1) and (ii) visiting successor proplets (cf. 4.2) in the most basic kind of the think mode, with the speaker mode riding piggyback (cf. 4.3). Next we turn to another operation enabled by this database schema, namely (iii) retrieving answers to questions. This operation is based on moving a query pattern along a token line until matching between the query pattern and a member proplet is successful.

Consider an agent thinking about girls. This means activating the corresponding token line, as in the following example:

6.1 EXAMPLE OF A TOKEN LINE

$$\begin{array}{ccccc}
 & & & \textit{member proplets} & \textit{owner proplet}\\
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: walk}\\ \text{mdr: young}\\ \text{prn: 10}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: sleep}\\ \text{mdr: blond}\\ \text{prn: 12}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: eat}\\ \text{mdr: small}\\ \text{prn: 15}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: read}\\ \text{mdr: smart}\\ \text{prn: 19}\end{bmatrix} &
\begin{bmatrix}\text{core: } \textit{girl}\end{bmatrix}
\end{array}$$

As indicated by the fnc and mdr values of the member proplets, the agent happened to observe or hear about a young girl walking, a blonde girl sleeping, a small girl eating, and a smart girl reading.

For retrieval, the member proplets of a token line may be checked systematically by using a pattern proplet as the query. The following example shows the application of a pattern proplet representing the query Which girl walked? to the token line 6.1:

6.2 APPLYING A QUERY PATTERN

$$\begin{array}{ccccc}
 & & \textit{query pattern} & \begin{bmatrix}\text{noun:}\textit{girl}\\ \text{fnc: walk}\\ \text{mdr: } \sigma\\ \text{prn: K}\end{bmatrix} & \\
 & & & \textit{matching?} & \\
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: walk}\\ \text{mdr: young}\\ \text{prn: 10}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: sleep}\\ \text{mdr: blonde}\\ \text{prn: 12}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: eat}\\ \text{mdr: small}\\ \text{prn: 15}\end{bmatrix} &
\begin{bmatrix}\text{noun: girl}\\ \text{fnc: read}\\ \text{mdr: smart}\\ \text{prn: 19}\end{bmatrix} &
\begin{bmatrix}\text{core: } \textit{girl}\end{bmatrix}
\end{array}$$

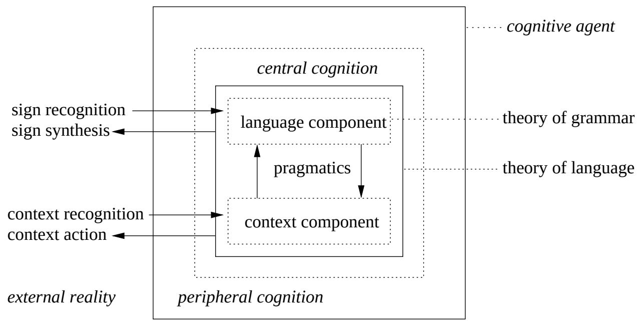
The indicated attempt at matching fails because the fnc values of the pattern proplet (i.e., walk) and of the member proplet (i.e., read) are incompatible. The same holds after moving the pattern proplet one member proplet to the left. Only after reaching the leftmost member proplet is the matching successful. Now the variable σ is bound to the value young and the variable K to the value 10. Accordingly, the answer provided to the question Which girl walked? is The young girl (walked).[8]

[8] For a more detailed presentation including yes/no questions see NLC'06, Sect. 5.1.

7 Reference as a Purely Cognitive Procedure

After discussing the derivation, storage, and retrieval of natural language content in a database let us turn to reference. In linguistics and philosophy reference is commonly defined in a metalanguage as a relation between language and "the world" (cf. FoCL'99, Chapt. 19). DBS re-interprets reference as a cognitive relation between a language and a context level inside the agent. This results in a distinction between *immediate* reference (to items in the agent's current environment) and *mediated* reference (to items in the agent's memory). Both kinds of reference are treated uniformly as computational procedures, independent of any metalanguage (cf. NLC'06, Chapt.2), as shown by the following diagram (for a more inclusive component structure see Sect. 10):

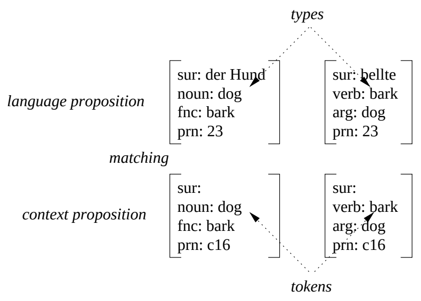
7.1 INTERFACES AND COMPONENTS OF A COGNITIVE AGENT WITH LANGUAGE



Immediate reference relies on the external interfaces (recognition and action) at the language and at the context level, while mediated reference uses the external interfaces at the language level only. The crucial relation of both kinds of reference is between the language and the context level inside the cognitive agent.

In DBS, the language and the context component code content in the same way, namely as concatenated proplets in a Word Bank. As a consequence, the interaction between the two components may be based on yet another application of matching:

7.2 LANGUAGE-CONTEXT MATCHING



Language and context proplets differ in that the sur (for surface) attribute of language proplets has a non-NIL value, whereas sur value of context proplets is NIL. Another difference is that in language proplets representing the sign kind of symbols (cf. FoCL'99, Chapt. 6) the core value is a concept *type*, while the core value of corresponding context proplets is a concept *token*. Thus, while the matching between rule patterns and proplets (cf. 5.1, 5.2, 5.3) is based on the use of restricted variables in the patterns, the matching between (symbol) language proplets and context proplets is based on the type-token relation (Peirce, CP, Vol.4, p. 537) between corresponding core values.

8 Coreference-by-Address

In computer science, the use of the term “reference” is quite different from that in philosophy and linguistics. A computational reference is a pointer to a physical storage location. Like a symbolic address, a pointer enables a program to access a particular data item. However, a pointer is faster than a symbolic address, providing instant access in constant time, and especially suited for data which are written once and never changed – as in the content-addressable memory of DBS.

Because pointers to a physical storage location do not lend themselves to a declarative specification, our representations use symbolic addresses instead. For example, the assignment of continuation values during language interpretation (cf. 4.1) is described as a copying operation resulting in symbolic addresses, but is implemented as pointers. While the symbolic copying operations are from core values to continuation attributes, the resulting pointers are in the opposite direction, as needed for navigation (cf. 4.2).

Another use of pointers is for connecting new content to old content by means of coreference. Consider, for example, a cognitive agent observing at moment t_i that *Julia is asleep* and at t_j that *Julia is awake*, referring to the same person. Instead of representing this change by revising the first proposition into the second,[9] the second proposition is added as new content, like sediment, leaving the first proposition unaltered:

8.1 COREFERENTIAL COORDINATION IN A WORD BANK

$$\begin{array}{lllll}
\ldots & \begin{bmatrix}\text{noun: Julia}\\ \text{fnc: sleep}\\ \text{prn: 675}\end{bmatrix} & \ldots & \begin{bmatrix}\text{noun: (Julia 675)}\\ \text{fnc: wake}\\ \text{prn: 702}\end{bmatrix} & \ldots \begin{bmatrix}\text{core: Julia}\end{bmatrix}\\
\ldots & & & & \\
\ldots & & \ldots & \begin{bmatrix}\text{verb: wake}\\ \text{arg: (Julia 675)}\\ \text{prn: 702}\end{bmatrix} & \ldots \begin{bmatrix}\text{core: wake}\end{bmatrix}\\
\ldots & & & & \\
\ldots & \begin{bmatrix}\text{verb: sleep}\\ \text{arg: Julia}\\ \text{prn: 675}\end{bmatrix} & \ldots & & \ldots \begin{bmatrix}\text{core: sleep}\end{bmatrix}
\end{array}$$

The occurrence of Julia in the second proposition is represented by a proplet with a core attribute containing a symbolic address as value, i.e., [noun: (Julia 675)], instead of a regular core value, e.g., [noun: Julia]. In the implementation, the symbolic address is realized by a pointer to the original *Julia* proplet with the prn value 675. A second, bidirectional pointer (not shown) connects the current proplet to its immediate coreferent predecessor, resulting in an incremental chain to the original.

[9] A more application-oriented example would be *fuel level high* at t_i and *fuel level low* at t_j.

This method, called coreference-by-address, enables a given item to code as many relations to other proplets as needed. For example, the proplets in the token line of *Julia* in 8.1 have the fnc value sleep in proposition 675, but wake in proposition 702. The most recent (and thus most up-to-date) content relating to the original proplet is found by searching the relevant token line from right to left, i.e., in the anti-temporal direction.

The use of addresses as core values complements the strictly time-linear storage of proplets with relations which may refer back to coreferent proplets in the same token line. As a result, coreference-by-address provides for a third kind of LA-think navigation – in addition to moving along the semantic relations between proplets (cf. 4.2), and moving a query pattern along a token line (cf. 6.2). Consider the following example:

8.2 COREFERENTIAL NAVIGATION

$$\begin{bmatrix}\text{verb: sleep}\\ \text{arg: Julia}\\ \text{prn: 675}\end{bmatrix} \overset{1}{\leftrightarrow} \begin{bmatrix}\text{noun: Julia}\\ \text{fnc: sleep}\\ \text{prn: 675}\end{bmatrix} \overset{2}{\leftrightarrow} \begin{bmatrix}\text{noun: (Julia 675)}\\ \text{fnc: wake}\\ \text{prn: 702}\end{bmatrix} \overset{3}{\leftrightarrow} \begin{bmatrix}\text{verb: wake}\\ \text{arg: (Julia 675)}\\ \text{prn: 702}\end{bmatrix}$$

The connections 1 and 3 are intrapropositional and based on the functor-argument relations between *Julia* and *sleep*, and *Julia* and *wake*, respectively. Connection 2 is extrapropositional and based on the coreference between the pointer proplet of proposition 702 and the original *Julia* proplet of proposition 675.[10] The content of 8.2 may be realized in English as Julia was asleep. Now she is awake.

9 Inference for Creating Summaries

Coreference-by-address allows not only (i) to revise the fixed information in a content-addressable memory by extending it, as in 8.1, but also (ii) to derive new content from stored content by means of inferencing. One kind of DBS inference is condensing content into a meaningful summary. As an example, consider a short text, derived in detail in Chapts. 13 (hearer mode) and 14 (speaker mode) of NLC'06:

> The heavy old car hit a beautiful tree. The car had been speeding. A farmer gave the driver a lift.

A reasonable summary of this content would be *car accident*. This summary may be represented in the agent's Word Bank as follows:

9.1 RELATING SUMMARY TO TEXT

$$\begin{array}{lcc} & \textit{member proplets} & \textit{owner proplets}\\ \ldots & & \\ \ldots & \begin{bmatrix}\text{noun: accident}\\ \text{mdr: (car 1)}\\ \text{prn: 67}\end{bmatrix} & \begin{bmatrix}\text{core: accident}\end{bmatrix}\end{array}$$

[10] As usual, the proplets in 8.2 are order-free. During language production, an order is re-introduced by navigating from one proplet to the next, following the semantic relations (pointers) between them.

$$\begin{array}{llllll}
\ldots & & & & & \\
\ldots & \begin{bmatrix}\text{noun: car}\\ \text{fnc: hit}\\ \text{prn: 1}\end{bmatrix} & \begin{bmatrix}\text{noun: (car 1)}\\ \text{fnc: speed}\\ \text{prn: 2}\end{bmatrix} \ldots & \begin{bmatrix}\text{noun: (car 1)}\\ \text{mdd: accident}\\ \text{prn: 67}\end{bmatrix} & & \ldots \begin{bmatrix}\text{core: car}\end{bmatrix} \\
\ldots & & & & & \\
\ldots & \begin{bmatrix}\text{verb: hit}\\ \text{arg: car tree}\\ \text{nc: 2 speed}\\ \text{pc:}\\ \text{prn: 1}\end{bmatrix} \ldots & & & & \ldots \begin{bmatrix}\text{core: hit}\end{bmatrix} \\
\ldots & & \begin{bmatrix}\text{verb: speed}\\ \text{arg: (car 1)}\\ \text{pc: 1 hit}\\ \text{nc: 3 give}\\ \text{prn: 2}\end{bmatrix} & & & \ldots \begin{bmatrix}\text{core: speed}\end{bmatrix}
\end{array}$$

Propositions 1 and 2 are connected (i) by adjacency-based coordination coded in the nc (next conjunct) and pc (previous conjunct) attribute values of their verb proplets *hit* and *speed*, and (ii) by coreferential coordination based on the original *car* proplet in proposition 1 and the corresponding pointer proplet (pronoun) in proposition 2.

The summary consists of another *car* pointer proplet and the *accident* proplet, each with the same prn value (here 67) and related to each other by the modifier-modified relation. The connection between the summary and the original text is based on the address value (car 1), which serves as the core value of the rightmost *car* proplet as well as the mdr (modifier) value of the *accident* proplet.

By summarizing content into shorter and shorter versions, there emerges a hierarchy which provides retrieval relations for upward or downward traversal. An upward traversal supplies more and more general notions, which may be used by the agent to access inferences defined at the higher levels. A downward traversal supplies the agent with more and more concrete instantiations.

The summary-creating inference deriving the new content with the prn value 67 is formally defined as the following rule, shown with the sample input and output of 9.1 at the content level:

9.2 Summary-creating inference

$$\begin{array}{llllllll}
 & \textit{antecedent} & & & & \textit{consequent} & \\
\begin{array}{l}\textit{rule}\\ \textit{level}\end{array} & \begin{bmatrix}\text{noun: }\alpha\\ \text{fnc: hit}\\ \text{prn: K}\end{bmatrix} & \begin{bmatrix}\text{verb: hit}\\ \text{arg: }\alpha\ \beta\\ \text{prn: K}\end{bmatrix} & \begin{bmatrix}\text{noun: }\beta\\ \text{fnc: hit}\\ \text{prn: K}\end{bmatrix} & \Rightarrow & \begin{bmatrix}\text{noun: (}\alpha\text{ K)}\\ \text{mdd: accident}\\ \text{prn: K+M}\end{bmatrix} & \begin{bmatrix}\text{noun: accident}\\ \text{mdr: (}\alpha\text{ K)}\\ \text{prn: K+M}\end{bmatrix} \\
 & \multicolumn{6}{l}{\text{where } \alpha\ \varepsilon\ \{\text{car, truck, boat, ship, plane, ...}\} \text{ and } \beta\ \varepsilon\ \{\text{tree, rock, wall, mountain, ...}\} \cup \alpha} \\
 & \textit{matching and binding}\ \Uparrow & & & & \quad\Downarrow & \\
\begin{array}{l}\textit{content}\\ \textit{level}\end{array} & \begin{bmatrix}\text{noun: car}\\ \text{fnc: hit}\\ \text{prn: 1}\end{bmatrix} & \begin{bmatrix}\text{verb: hit}\\ \text{arg: car tree}\\ \text{nc: 2 speed}\\ \text{pc:}\\ \text{prn: 1}\end{bmatrix} & \begin{bmatrix}\text{noun: tree}\\ \text{fnc: hit}\\ \text{prn: 1}\end{bmatrix} & & \begin{bmatrix}\text{noun: (car 1)}\\ \text{mdd: accident}\\ \text{prn: 67}\end{bmatrix} & \begin{bmatrix}\text{noun: accident}\\ \text{mdr: (car 1)}\\ \text{prn: 67}\end{bmatrix} \\
 & \textit{input} & & & & \textit{output} &
\end{array}$$

The rule consists of two sets of pattern proplets, connected by "⇒." The set of patterns preceding the arrow is called the *antecedent*, while the set of patterns following the arrow is called the *consequent*. The patterns are defined using the restricted variables α for the subject, β for the object, and K, M for the prn values.

Below the rule is an old content called the *input*, matching the antecedent, and a new content called the *output*, derived by the consequent. The variables of the antecedent are bound to corresponding values at the content level, here α to car, β to tree, and K to 1. By using the variables α and K also in the consequent, the output car accident is derived as the new content.

In the rule, the possible values which α and β may be bound to during matching are restricted by the co-domains of these variables: the restricted variable α generalizes the summary-creating inference to different kinds of accidents, e.g., *car accident, truck accident*, etc., while the restricted variable β limits the objects to be hit to trees, rocks, etc., as well as cars, trucks, etc. Any content represented by the proplet *hit* with a subject and an object proplet satisfying the variable restrictions of α and β, respectively, will be automatically (i) summarized as an accident of a certain kind whereby (ii) the summary is related to the summarized by means of an address value, here (car 1), thus fulfilling the condition that the data in a content-addressable memory may not be modified.

10 Component Structure and Functional Flow

The relation between a DBS rule and a corresponding content is based on matching between a small, order-free set of pattern proplets in the rule and a corresponding order-free set of content proplets in the database. The matching between an individual pattern proplet and a content proplet is in turn based on their non-recursive feature structures. This method has been used here for the following cognitive operations:

a. *natural language interpretation*:
matching between LA-hear grammar rules and language proplets (cf. 5.1)

b. *navigation*:
matching between LA-think grammar rules and content proplets (cf. 5.2)

c. *querying*:
matching between query patterns and content proplets (cf. 6.2)

d. *reference*:
matching between language and context proplets in the case of symbols (cf. 7.2)

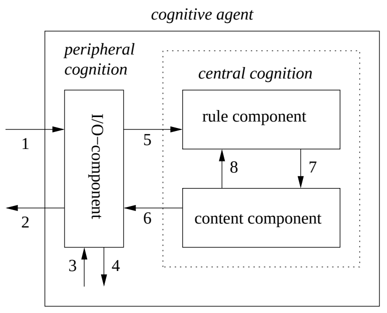
e. *inferencing*:
matching between inference rules and content proplets (cf. 9.2)

How should these different cognitive operations based on pattern matching be integrated into the component structure and functional flow of a cognitive agent?

Consider the component structure of diagram 7.1. It models (d) *reference* as viewed in analytic philosophy, i.e., as a vertical relation between horizontal language expressions and a horizontal world. At the same time, 7.1 departs from the standard assumptions of analytic philosophy, including truth-conditional semantics, because it treats reference as an agent-internal, cognitive procedure – and not as an external relation defined in a meta-language, as postulated by mathematical realism.[11]

While diagram 7.1 is essential for explaining the Seven Principles of Pragmatics in the SLIM theory of language,[12] there is a functionally more inclusive alternative:

10.1 REFINED COMPONENT STRUCTURE OF A COGNITIVE AGENT



1 = external recognition
2 = external action
3 = internal recognition
4 = internal action
5 = input to rule component
6 = output of content component
7 = rule-content interaction
8 = content-rule interaction

Here, the leading assumption is that cognition, including communication, is based on abstract patterns matching with more concrete structures of content. Therefore, the two matching components (analogous to the language and the context component in diagram 7.1) are the rule component and the content component.

The separation of patterns and contents in diagram 10.1 provides a uniform structural basis for the rule (pattern) level to govern the processing of content (7) – with data-driven feedback from the content level (8). This interaction based on pattern matching is used for the cognitive operations of (a) interpretation, (b) navigation, (c) querying, and (e) inferencing.

The matching of (d) reference, however, is provided with a different treatment *inside* the content component of 10.1. Thereby, the reference mechanisms of the different sign kinds, namely symbol (type-token relation), index (pointer), and name (marker)[13] – focused on in diagram 7.1 – reappear as horizontal relations between (i) language proplets and (ii) context or language proplets, stored in token lines.

Technically, diagram 7.1 is integrated into diagram 10.1 by changing to a different view: instead of viewing content proplets as sets with a common prn value (propositions), and separated into a language and a context level (cf. 7.2), the same proplets are viewed as items to be sorted into token lines according to their core value (cf. 2.1).

The rule and the content component of 10.1 are each connected unidirectionally to a general I/O component. All recognition output of this I/O component is input to the rule component (5), where it is processed and passed on to the content component (7). All action input to the I/O component comes from the content component (6), derived in frequent (8, 7) interaction with the rule component. The different interfaces for the language and the context component in diagram 7.1 may be recreated in 10.1 by dividing 7 and 8 into 7a, 8a for the context level, and into 7b, 8b for the language level.

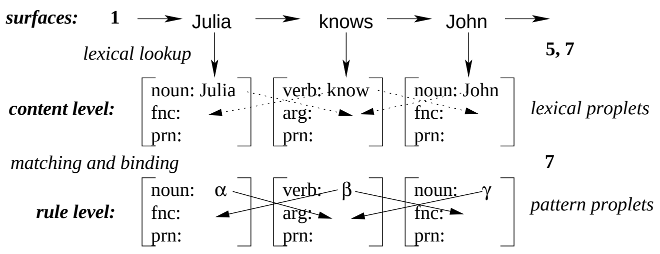
[11] For a more detailed discussion see FoCL’99, Chapts. 19–21.
[12] See NLC’06, Sect. 2.6, for a summary.
[13] Cf. FoCL’99, Sects. 6.1 and 6.2.

11 Cycle of Communication in the Refined Component Structure

The general component structure and functional flow 10.1 raises the question of how to integrate the cycle of natural language communication as modeled in DBS. In other words, how should an autonomous control based on the interaction between the general components for I/O, rules, and content accommodate the hearer mode as characterized in 4.1 and 5.1, the think mode as characterized in 4.2 and 5.2, and the speaker mode as characterized in 4.3 and 5.3? Furthermore, what are the impulses initiating the hearer, think, and speaker mode procedures, and where do these impulses come from?

Within the component structure and functional flow of diagram 10.1, the hearer mode derivation 4.1 may be shown as follows (using the same numbering as 10.1 to indicate the different mappings between components):

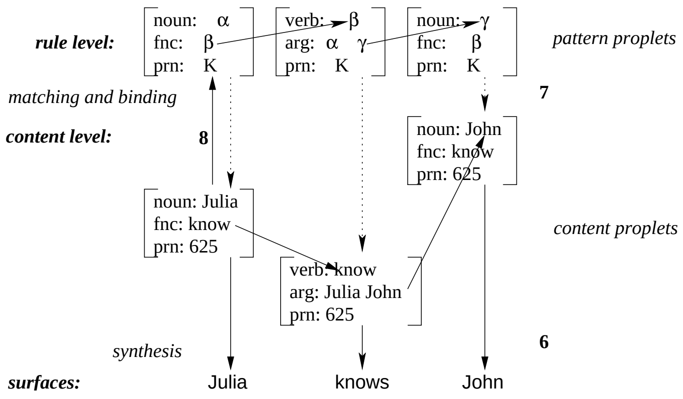
11.1 MAPPING INCOMING SURFACES INTO CONTENT (HEARER MODE)



The impulse activating the hearer mode operations are the surfaces (1) which the I/O component provides to the rule component (5) which triggers lexical lookup in the content component (7). As a result, lexical proplets are added one by one at the end of the corresponding token lines of the content component. At the same time, the LA-hear grammar rules (cf. 5.1) of the rule component connect these lexical proplets incrementally via copying (7) in a strictly time-linear derivation order.

Next consider the language production from stored content 4.3 within the component structure of diagram 10.1:

11.2 MAPPING STORED CONTENT INTO OUTGOING SURFACES (SPEAKER MODE)

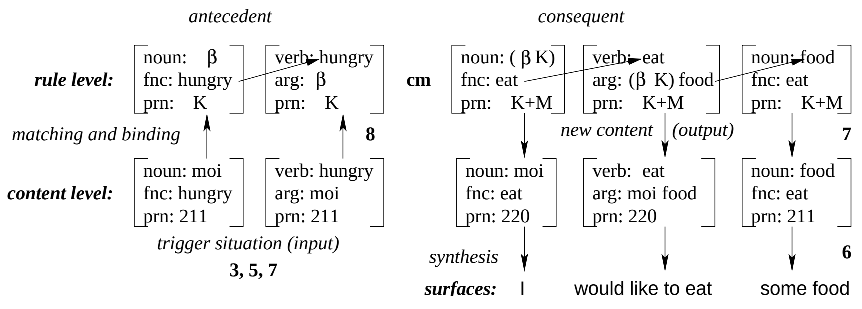


The impulse may be provided by another agent's question (cf. Sect. 6) or by a request to recount a certain event. The navigation through the content is driven by the rule level and transmitted to the content level by means of pattern matching (7).[14] The LA-think grammar rules (cf. 5.2) are initially activated by a content proplet matching the start state (8) and then by the rule packages in the derivation sequence. The derivation of blueprints for well-formed language expressions passed to the I/O component (6) is based on a frequent (7, 8) cycle between the rule and the content component.

Finally consider language production based on the on-the-fly inferencing of the agent's autonomous control. For reasons of space, the description of control has been limited here to a partial[15] intuitive outline (Sect. 3), the content-addressable memory of a Word Bank (Sect. 8), and the example of a summarizing inference (Sect. 9). Nevertheless, for language production from inferences this is sufficient insofar as the output of their consequent is a sequences of proplets which (i) have core values and (ii) code the semantic relations of functor-argument structure, coordination, and coreference.

In other words, the content produced by on-the-fly inferencing has the same format as content resulting from recognition – and can therefore be processed by the same language-dependent rules for surface synthesis. As an example, consider the inference β *hungry* **cm** β *eat food*, whereby the connective **cm** stands for countermeasure:

11.3 MAPPING NEWLY DERIVED CONTENT INTO OUTGOING SURFACES



The impulse activating this inference is a sensation of hunger provided by the agent's I/O component (antecedent 3, 5, 7, 8). One possibility to use the content derived by the consequent (7) would be as a blue-print for non-language action. Here, however, the newly derived content is mapped into language, utilizing the core values for the synthesis of surfaces (6). This kind of language production is used especially in dialog (cf. Schegloff 2007), which requires, and triggers, the agent's reasoning for real-time reactions to the actions of the partner in communication.

[14] For simplicity, the navigation is more direct than in 4.3.

[15] An analysis of autonomous control is provided in Hausser (2010), *Inferencing in Database Semantics*. Following Bernard (1865) and Wiener (1948), this paper describes autonomous control as driven by the task of maintaining the agent in a state of balance (equilibrium, homeostasis) vis-à-vis a constantly changing external and internal environment. Important ingredients are automatic schema derivation, the subactivation and evaluation of content, adaptation and learning, and the definition and chaining of reactor, deductor, and effector inferences for deriving action blueprints. An autonomous control maintaining a balance by relating recognition to the evaluated outcome of possible reactions is decentralized, in line with Brooks (1985).

12 Managing the Data Stream

The use of a content-addressable memory for cognition raises the question of how to manage the data stream which continuously enters the cognitive agent's large but finite memory space. Given that a content-addressable memory cannot be changed, incoming data should be selected and cleaned up prior to storage. Following the natural prototype, this may be done by providing a short-term and a long-term memory, and by doing selection and clean-up in short-term memory prior to long-term storage.

The other method to stave off data overflow, also following the natural prototype, is forgetting. Though a limiting case of data change (and therefore not really permitted in a content-addressable memory), forgetting is functionally acceptable if the items deleted do not result in any (or only a few) dead ends in the associative data network. As a solution, Anderson (1983) proposed a frequency-based approach, namely to bleach those areas of content from memory which are activated never or very rarely by the cognitive operations of the agent.[16] The procedure resembles garbage collection, familiar from programming languages like Lisp, Java, or Oberon.

In a Word Bank, the many dispersed little spaces created by bleaching may be made reusable by pushing the proplets in a token line to the left, like pearls on a string (defragmentation). In this way, the intrinsic ordering of the proplets is maintained and the contiguous spaces reclaimed on the right hand end of the token lines are suitable for new storage. The periods in which garbage collection and defragmentation are performed in artificial agents may be likened to the periods of sleep in natural agents.

The handling of the data stream as inspired by empirical research in cognitive psychology may be complemented by purely computational methods. Just as an aircraft may be designed to maximize payload, range, speed, and profit with little or no resemblance to the natural prototypes (e.g., birds), there may be a software routine which is applied whenever the amount of memory in the Word Bank reaches a certain limit; the procedure compresses the currently oldest content segment (containing, for example, all proplets with prn values between 30 000 and 32 000) and moves it into secondary storage without any deletion (and thus without any need to choose).

The only downside to the data in secondary storage would be a slight slowdown in initial activation. Depending on the application, an artificial agent's overall memory may either be sized from the beginning to accommodate the amount of data expected for an average life time (for example, in a robot on a space mission), or it may be expanded incrementally as needed.

Conclusion

Modeling the cycle of natural language communication in the form of a talking cognitive agent is a goal of computational linguistics which is as obvious as it is legitimate. It requires a linguistic theory which allows to reconstruct the hearer mode, the think mode, and the speaker mode as a software system which can be easily upscaled and debugged. Methodologically, the software model provides for an objective, automatic

[16] In addition, the intensity of a memory in its historical context must be taken into account in order to handle examples like Proust's madeleine episode.

testing of the theory in terms of functional completeness and data coverage. It also has a wide range of practical applications in the area of human-machine communication.

That this approach requires more than the usual grammatical analysis of isolated language signs is especially clear in the speaker mode, because the cognitive agent must choose *what to say* in the situation at hand. From a software engineering point of view, this choice raises the question of how to structure the content in the agent's memory so that the agent's cognition can select or derive in real time what seems appropriate action (including non-action) for the current situation. The better the model provided by memory fits the agent's current task, the less the agent must rely on trial and error (cf. Hausser 2002).

Our solution is a navigational database which is structured like a classic network database. However, while the navigational databases of the past (Bachman 1973) and the present (Xpath) are intended for external human users, the system presented here serves as the shell of an autonomous control, located inside an artificial cognitive agent. The database, called Word Bank, receives input from the agent's external and internal interfaces for recognition and is used by the autonomous control to produce output as blue prints for the agent's action components. The overall task is to maintain the agent in a state of balance by connecting the interfaces for recognition with those for action.

As a network database, a Word Bank stores flat feature structures called proplets instead of records. Also, compared to a classic network database, a Word Bank is highly constrained. First, the member proplets belonging to an owner proplet are listed in a token line in the temporal order of their arrival. Second, all members in a token line must share the owner's core value (no multiple owners). Third, the only connections between proplets across token lines are the grammatical relations of functor-argument structure, coordination, and coreference. Fourth, like the relations between owners and members, the grammatical relations are 1:n relations: one functor–several possible arguments, one modified–several possible modifiers, one first conjunct–several possible successors, one original–several possible coreferent pronouns.

Furthermore, a Word Bank is content-addressable because no separate index (inverted file) is required for storage and retrieval. By adding content like sediment, content is fixed in the sense that it is written once and never changed. Computationally, this is ideal for implementing the grammatical relations between proplets by means of addresses which are both procedural and declarative. The procedural implementation is by means of pointers to physical storage locations because they provide instant access in constant time, while symbolic addresses are used for the equivalent declarative specification.

The DBS database and control system is of complete generality: it works with any sets of attributes and values, any set of triggers, any lexical items, and any grammatical constructions – presupposing, of course, that they all fit together and there is a well-defined interaction with the agent's external and internal interfaces for recognition and action. Unlike Stanfill and Waltz (1986), whose memory-based reasoning requires a special hardware (connection machine), the DBS system runs on van Neumann machines.

A cognitive agent with a memory and interfaces to the external and internal environment is in a principled contrast to systems of language production for weather reports or query answering for ship locations, train schedules, and the like. These are *agentless* applications; they are popular in the research literature because they allow to fudge the lack of an autonomous control. Their disadvantage, however, is that they cannot be extended to agent-based applications such as free dialog (Schegloff 2007), whereas the inverse direction from agent-based to agentless is comparatively easy.

Acknowledgments

This paper benefitted from comments by Kiyong Lee and Jae-Woon Choe, Korea University, Seoul; Brian MacWhinney, Carnegie Mellon University, Pittsburgh; Haitao Liu, Communication University of China, Beijing; and Johannes Handl, Besim Kabashi, Thomas Proisl, and Carsten Weber, Computational Linguistics Uni Erlangen (CLUE).

References

AIJ'01 = Hausser, R. (2001) "Database Semantics for natural language," *Artificial Intelligence*, 130.1:27–74. Available online at http://www.linguistik.uni-erlangen.de/clue/de/publikationen.-html

Anderson, J. R. (1983) "A spreading activation theory of memory," *Journal of Verbal Learning and Verbal Behavior*, 22:261-295

Bachman, C. W. (1973) "The Programmer as Navigator,", 1973 ACM Turing Award Lecture, *Comm. ACM*, Vol. 16:11.653–658

Bernard. C. (1865) *Introduction à l'étude de la médecine expérimentale*, first English translation by Henry Copley Greene, published by Macmillan, 1927; reprinted in 1949

Brooks, R. (September 1985) "A Robust Layered Control System for a Mobile Robot," Cambridge, MA: *MIT AI Lab Memo 864*, 227–270.

Chisvin, L., and R. J. Duckworth (1992) "Content-Addressable and Associative Memory," in M.C. Yovits (ed.), *Advances in Computer Science, 2nd ed.*, pp. 159–235, Academic Press

CoL'89 = Hausser, R. (1989) *Computation of Language*, Symbolic Computation: Artificial Intelligence, pp. 425, Berlin Heidelberg New York: Springer

Cormen, T.H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009) *Introduction to Algorithms, 3rd ed.*, Cambridge, MA: MIT Press

Elmasri, R., and S.B. Navathe (1989) *Fundamentals of Database Systems*. Redwood City, CA: Benjamin-Cummings

Fischer, W. (2002) *Implementing Database Semantics as an RDMS* (in German), Studienarbeit am Institut für Informatik der Universität Erlangen Nürnberg (Prof. Meyer-Wegener), published as CLUE-Arbeitsbericht 7 (2004), available at
http://www.linguistik.uni-erlangen.de/clue/de/arbeiten/arbeitsberichte.html

FoCL'99 = Hausser, R. (1999) *Foundations of Computational Linguistics, Human–Computer Communication in Natural Language, 2nd ed.*, pp. 578, Berlin Heidelberg New York: Springer

Fredkin, E. (1960) "Trie Memory," *Commun. ACM*, 3.9:490-499

Handl, J., B. Kabashi, T. Proisl, and C. Weber (2009) "JSLIM - Computational morphology in the framework of the SLIM theory of language," in C. Mahlow and M. Piotrowski (eds.) *State of the Art in Computational Morphology*, Berlin Heidelberg New York: Springer

Hausser, R. (2002): “Autonomous Control Structure for Artificial Cognitive Agents,” in H. Kangassalo et al. (eds.) *Information Modeling and Knowledge Bases XIII*, Amsterdam: IOS Press Ohmsha. Available online at http://www.linguistik.uni-erlangen.de/clue/de/publikationen.html

Hausser, R. (2008) “Center Fragments for Upscaling and Verification in Database Semantics,” presented at the EJC XX, Tsukuba, Japan, June 2–6, 2008. Available online at http://www.linguistik.uni-erlangen.de/clue/de/publikationen.html

Hausser, R. (2009) “Modeling Natural Language Communication in Database Semantics,” in M. Kirchberg and S. Link (eds.), Proceedings of the APCCM 2009, Australian Computer Science Inc., CIPRIT, Vol. 96. Available online at http://www.linguistik.uni-erlangen.de/clue/de/publikationen.html

Hausser, R. (2010) “Inferencing in Database Semantics,” under review

Kalender, K. (2009) *Implementing a Center Fragment of Russian in Database Semantics*, MA-thesis, CLUE, Universität Erlangen-Nürnberg [in German]

McKeown, K. (1985) *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*, Cambridge: CUP

Mehlhaff, J. (2007) *Implementing a Center Fragment of German in Database Semantics*, MA-thesis, CLUE, Universität Erlangen-Nürnberg [in German]

Mei, Hua (2007) *Implementing a Center Fragment of Chinese in Database Semantics*, MA-thesis, CLUE, Universität Erlangen-Nürnberg [in German]

NEWCAT’86 = Hausser, R. (1986) *NEWCAT: Parsing Natural Language Using Left-Associative Grammar*, Lecture Notes in Computer Science 231, pp. 540, Berlin Heidelberg New York: Springer

NLC’06 = Hausser, R. (2006) *A Computational Model of Natural Language Communication*, pp. 365, Berlin Heidelberg New York: Springer

Peirce, C.S. (1931–1935) *Collected Papers*. C. Hartshorne and P. Weiss (eds.), Cambridge, MA: Harvard Univ. Press

Proust, M. (1913) *Du côté de chez Swann*, ed. by Jean-Yves Tadie et al., Bibliotheque de la Pleiade, Paris: Gallimard,1987-89

Schegloff, E. (2007) *Sequence Organization in Interaction*, New York: Cambridge Univ. Press

Söllch, G. (2008) *Implementing a Center Fragment of Tagalog in Database Semantics*, MA-thesis, CLUE, Universität Erlangen-Nürnberg [in German]

Stanfill, C., and D. Waltz (1986) “Toward Memory-Based Reasoning,” *Commun. ACM*, 29.12:1213–1226

TCS’92 = Hausser, R. (1992) “Complexity in Left-Associative Grammar,” *Theoretical Computer Science*, 106.2:283-308. Available online at http://www.linguistik.uni-erlangen.de/clue/de/publikationen.html

Weiss, M. A. (2005) *Data Structures and Problem Solving Using Java. 3rd ed.*, Upper Saddle River, NJ: Pearson Addison Wesley

Wiener, N. (1948) *Cybernetics: Or the Control and Communication in the Animal and the Machine,* Cambridge, MA: MIT Press