Memory-based Pattern Completion in Database Semantics

Roland Hausser[*]

Friedrich-Alexander-Universität Erlangen-Nürnberg

Roland Hausser. 2005. Memory-based Pattern Completion in Database Semantics. *Language and Information 9.1*, 1–##. Pattern recognition in cognitive agents is based on (i) the uninterpreted input data (e.g. parameter values) provided by the agent's hardware devices and (ii) and interpreted patterns (e.g. templates) provided by the agent's memory. Computationally, the task consists in finding the memory data corresponding best to the input data, for any given input. Once the best fitting memory data have been found, the input is recognized by applying to it the interpretation which happens to be stored with the memorized pattern.

This paper presents a fast converging procedure which starts from a few initially recognized items and then analyzes the remainder of the input by systematically checking for items shown by memory to have been *related* to the initial items in previous encounters. In this way, known patterns are tried first, and only when they have been exhausted, an elementary exploration of the input is commenced. Efficiency is improved further by choosing the candidate to be tested next according to frequency. (**Universität Erlangen**)

Key words: Database Semantics, concepts, geons, features, proplets, case-based recognition, semantics fields, indexing, retrieval, pattern completion

The elementary features provided by the input hardware of a cognitive agent may be combined in many different ways, in different modalities, and at different levels of complexity, creating a huge search space. The software problem of pattern matching is to classify any given input constellation without having to try a potentially very large number of interpreted patterns, as in linear search. In other words, there is the task of finding an *efficient* procedure for going from raw input data to corresponding patterns stored in memory. In addition, the system must be capable of analyzing and storing brand-new constellations of raw data, such that they will be recognized when encountered a second time.

These tasks are treated here with the hand-in-glove combination of an algorithm and a data structure. The system, called Database Semantics (DBS), de-

[*] Computational Linguistics Uni Erlangen (CLUE), Bismarckstr. 6, D-91054 Erlangen, Germany. E-mail: rrh@linguistik.uni-erlangen.de. Fax: +49 9131 852 9251

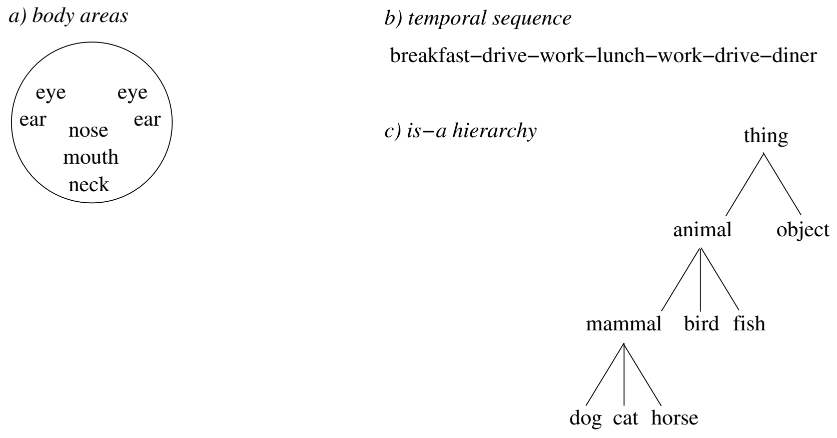
termines the storage location of any input in terms of properties provided by the input analysis, such that similar analyses – modulo some primary attribute(s) – are stored at adjacent locations, in the order of their arrival (token line, cf. 6.1). Memory-based pattern completion in DBS proceeds from the partial analysis of the input to the analysis-based retrieval of similar items. It utilizes the fact that the number of relations between items stored in memory is finite and much smaller than the infinite number of relations resulting from their abstract combinatorics.

1. The spatial organization of semantic fields

Designing a procedure for pattern recognition raises the question of choosing the right data structure. For example, should the spatial organization of cognitive content in the brain be treated as necessary for efficient retrieval, or as accidental, resulting from the history of evolution but with no functional consequence for the abstract software design?

As instances of meaningful spatial organizations of content consider the following representations of (a) a face, (b) a temporal sequence, and (c) an *is-a* hierarchy. Such neighborhoods forming coherent areas of cognitive content are called *semantic fields* in linguistics:

1.1 Simplified examples of semantic fields



In this representation the relations between, for example, the eyes, the nose, the mouth, etc., are expressed in terms of their spatial organization, and similarly for the temporal sequence and the hierarchy. The question is whether or not such a 'natural' organization at the lowest machine level does per se facilitate efficient retrieval of a particular item.[1]

As a theoretical answer, let us consider the corresponding case of a library where the books may be stored in accordance with some physical property, for example, all small books in the first shelve, all medium size books in the second shelve, and all large books in the third shelve. Other, similar storage principles

[1] At this point, we simply assume that retrieval is a necessary part of cognition. Section 6 describes cognitive procedures in which retrieval plays a clearly defined, vital role.

are arranging the books in accordance with the color of their covers, the temporal order of their purchase, or their topic.

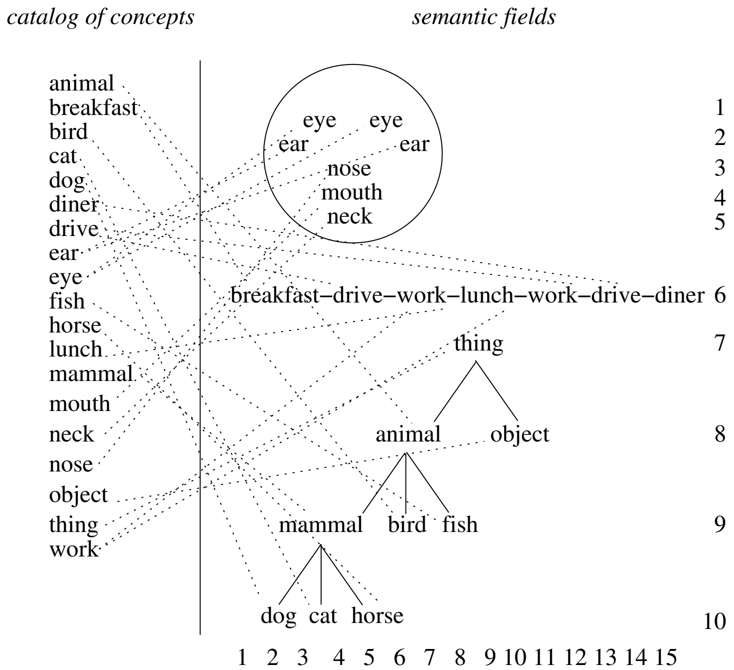
The retrieval problem posed by these different storage principles has an abstract solution which is as general as it simple. It consists in building a library catalog. Each catalog card specifies (i) a key word, for example the size or color of the book, or, more realistically, the name of the author, the title, or the topic, and (ii) the location. In this way, the catalog cards may be ordered alphabetically, based for example on the last name of the author, while the books may be physically stored in accordance with any spatial ordering principle of choice, even no principle at all.

Whenever a certain book is to be retrieved, it is simply looked up in the alphabetical catalog and the card in question will tell its exact location. The only costs of this method are (i) building the catalog (indexing) and (ii) taking care to always bring each book back to its specified location (storage). Today these costs are minimized by using computers, which allow to do storage and retrieval automatically, greatly increasing efficiency.

If location can be made irrelevant by coding the relevant positions with the help of a catalog, then the books can be retrieved efficiently no matter where they are being stored. Accordingly, any 'natural' organization of cognitive content can be treated as a product of evolutionary history rather than inherent efficiency.

As an example of applying the catalog solution to semantic fields, consider the following proposal:

1.2 Indexing semantic fields with a catalog



This example contains the same semantic fields as 1.1. In addition, (i) there is an alphabetical list of all the elements contained and (ii) there is a grid of numbers characterizing the location of each element in terms by an ordered pair. The relation between an element in the catalog, e.g. bird, and its location, e.g. (6,9), is indicated by means of a dotted line.

For purposes of retrieval, the catalog solution 1.2 is much more versatile and efficient than the purely graphical representation 1.1. For example, to find the location of the element bird in the graphical representation, we have to remember to look in the *is-a* hierarchy with the top node thing, whereas the catalog representation provides the location of bird directly.

2. Extrinsic relations

If we abandon the graphical representation and rely only on the location of the items relative to an external grid, the semantic fields illustrated in 1.2 may be shown as follows:

2.1 Representing semantic fields by extrinsic relations

animal	(6,8)	horse	(5,10)
breakfast	(1,6)	lunch	(9,6)
bird	(6,9)	mammal	(3,9)
cat	(3,10)	mouth	(4,4)
dog	(2,10)	neck	(4,5)
diner	(6,15)	nose	(4,3)
drive	(13,6)	object	(10,8)
ear	(2,2) (6,2)	thing	(8,7)
eye	(3,1) (5,1)	work	(11,6)
fish	(8,9)		

In the transition from 1.2 to the catalog 2.1 of names and pairs of numbers there is clearly something being lost. For example, the intuitively relevant relations between the eyes, the nose, and the mouth in the face are now characterized only indirectly: they must be reconstructed from the catalog by translating the numbers back into locations.

Furthermore, there are many equivalent ways to define the external grid simply because, for example, the same face may be represented in different sizes or moved in space. Thus, a definition of semantic fields in terms of locations defined relative to an external grid is largely accidental. In order to characterize the necessary properties of a semantic field we must replace locations in a geometrical space with intrinsic relations in a *conceptual space.*

3. First example of intrinsic relations: phrase structure tree

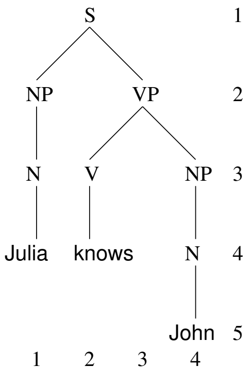
The literature on conceptual space is as huge as it is diverse[2] and here is neither the place nor the space to review it. Instead let us explain the correlation between the retrieval of content, on the one hand, and the representation of intrinsic relations

[2] Prominent examples are Lakoff (1987), Lakoff and Johnson (1980), Fauconnier (1994), Schank and Abelson (1977), Johnson-Laird (1983), Sowa (1984), Wierzbicka (1980).

between contents at various levels of abstraction, on the other.

As our first example, we use the phrase structure trees familiar from the grammatical analysis in nativist linguistics. To get an intuitive understanding of the options for defining a conceptual space and the necessary choices between them we begin with the transition from (i) a phrase structure tree to (ii) a corresponding catalog:

3.1 Translating a phrase structure tree into a catalog



John	(4,5)	
Julia	(1,4)	
knows	(2,4)	
N	(1,3)	(4,4)
NP	(1,2)	(4,3)
S	(2,1)	
V	(2,3)	
VP	(3,2)	

The tree structure on the left combines a graphical representation with a grid of numbers. The catalog on the right relies exclusively on the characterization of location in terms of catalog names and pairs of numbers provided by the grid, in analogy to 2.1.

This catalog solution is suitable to index large numbers of phrase structure trees, as in a tree bank, providing for efficient retrieval: it directly specifies the location of all S nodes, all NP nodes, etc., relative to an external grid. The intuitive intrinsic relations making the tree structure meaningful to the linguist are lost, however.

What are the intrinsic relations of a phrase structure tree? In the literature, phrase structure trees have been analyzed in terms of *immediate dominance* and *linear precedence* (Gazdar et al., 1985; Shieber, 1986) between nodes. Using the format of Database Semantics, these intrinsic relations may be coded as follows:

3.2 Coding a phrase structure tree as proplets in a word bank

John	name: John mo: N da: pr: fo: tn: 23
Julia	name: Julia mo: N da: pr: fo: knows tn: 23
knows	name: knows mo: V da: pr: Julia fo: N tn: 23

$$\begin{array}{lll}
\text{N} & \begin{bmatrix}\text{name: N}\\ \text{mo: NP}\\ \text{da: Julia}\\ \text{pr:}\\ \text{fo: V}\\ \text{tn: 23}\end{bmatrix} & \begin{bmatrix}\text{name: N}\\ \text{mo: NP}\\ \text{da: John}\\ \text{pr: knows}\\ \text{fo:}\\ \text{tn: 23}\end{bmatrix} \\
\text{NP} & \begin{bmatrix}\text{name: NP}\\ \text{mo: S}\\ \text{da: N}\\ \text{pr:}\\ \text{fo: VP}\\ \text{tn: 23}\end{bmatrix} & \begin{bmatrix}\text{name: NP}\\ \text{mo: VP}\\ \text{da: N}\\ \text{pr: V}\\ \text{fo:}\\ \text{tn: 23}\end{bmatrix} \\
\text{S} & \begin{bmatrix}\text{name: S}\\ \text{mo:}\\ \text{da: NP, VP}\\ \text{pr:}\\ \text{fo:}\\ \text{tn: 23}\end{bmatrix} & \\
\text{V} & \begin{bmatrix}\text{name: V}\\ \text{mo: VP}\\ \text{da: knows}\\ \text{pr: N}\\ \text{fo: NP}\\ \text{tn: 23}\end{bmatrix} & \\
\text{VP} & \begin{bmatrix}\text{name: VP}\\ \text{mo: S}\\ \text{da: V, NP}\\ \text{pr: NP}\\ \text{fo:}\\ \text{tn: 23}\end{bmatrix} &
\end{array}$$

The above proplets all have the same attributes, namely name of the item represented, the mother node mo, the daughter node(s) da, the preceding node pr, the following node fo, and the tree number tn (here 23), which holds the proplets belonging to the same tree together. Proplets which are similar in the sense that their first attributes, here name, have the same value are stored in the order of arrival in the same token line (see N and NP).

By recoding the necessary properties of the phrase structure tree in terms of the attributes name, mo, da, pr, and fo, the catalog solution 3.1 for retrieving nodes relative to grid locations is being replaced by an analysis specified in terms of node name, dominance, and precedence.[3] This analysis is used for (i) storing similar proplets in the same token line and for (ii) retrieving similar proplets for any given input analysis.

As an example of retrieval consider a search for an N node dominated by NP and preceded by knows, using 3.2 as the database. This search would be based on the following pattern:

3.3 Search pattern for a node in a phrase structure tree

$$\begin{bmatrix}\text{name: N}\\ \text{mo: NP}\\ \text{da: } \alpha\\ \text{pr: knows}\\ \text{fo: } \beta\\ \text{tn: } \gamma\end{bmatrix}$$

Proplet patterns differ from proplet tokens in that some of their values are variables, here α, β, and γ. Assuming that 3.2 is expanded to represent a large tree bank, a search with the pattern 3.3 would be applied to the token line of N, checking each proplet, going either from left to right or from right to left. The result are all proplets representing nodes in the tree bank fitting the pattern (in 3.2 the result would be the second proplet in the token line starting with N).

[3] If desired, locations relative to a grid may be specified as well, using the additional attribute loc in each proplet.

4. Second example of intrinsic relations: *is-a* hierarchy

Representing phrase structure trees as proplets in a word bank provides an efficient solution for retrieval. As intrinsic relations for characterizing grammatical structure, however, dominance and precedence are much too weak.[4] This may be demonstrated by comparing the proplet analysis of a phrase structure tree in the previous section with the following proplet analysis of the *is-a* hierarchy shown in the examples 1.1 and 1.2.

4.1 Coding an *is-a* hierarchy as proplets in a word bank

```
            ┌name: animal            ┐
animal      │is-a: thing             │
            │inst: mammal bird fish  │
            └sh: 25                  ┘
            ┌name: bird   ┐
bird        │is-a: animal │
            │inst:        │
            └sh: 25       ┘
            ┌name: cat    ┐
cat         │is-a: mammal │
            │inst:        │
            └sh: 25       ┘
            ┌name: dog    ┐
dog         │is-a: mammal │
            │inst:        │
            └sh: 25       ┘
            ┌name: fish   ┐
fish        │is-a: animal │
            │inst:        │
            └sh: 25       ┘
            ┌name: horse  ┐
horse       │is-a: mammal │
            │inst:        │
            └sh: 25       ┘
            ┌name: mammal         ┐
mammal      │is-a: animal         │
            │inst: dog cat horse  │
            └sh: 25               ┘
            ┌name: object┐
object      │is-a: thing │
            │inst:       │
            └sh: 25      ┘
            ┌name: thing          ┐
thing       │is-a:                │
            │inst: animal object  │
            └sh: 25               ┘
```

As a straightforward tree structure, the *is-a* hierarchy could have been coded in terms of dominance and precedence, like 3.2. Instead, however, we use two domain-specific intrinsic relations, namely is-a and inst, for 'instantiates'. These are semantically much more meaningful than dominance and precedence because they characterize *subset relations.* For example, animal is characterized as a thing and instantiated by mammal, bird, and fish.[5]

The elements of the *is-a* hierarchy are held together with the attribute sh, for 'semantic hierarchy', which takes a common number as value (here 25). A corresponding analysis for the *has-a* hierarchy would be based on the intrinsic relations has-a and is-part-of.

[4] This has not gone unnoticed in nativist linguistics, where phrase structure tree analyses have been propped up with the auxiliary principle of constituent structure, the additional procedure of unification, and the import of truth-conditional semantics.

[5] Note that the word bank representing the *is-a* hierarchy contains no proplets representing non-terminal nodes like S, N, NP, etc., in contradistinction to the word bank 3.2 representing a phrase structure tree.

Example 4.1 shows the importance of choosing the right internal relations for a meaningful empirical analysis. This holds also for the grammatical analysis of natural language. Instead of dominance and precedence we choose the traditional intrinsic relations of *functor, argument*, and *modifier*. Consider the following reanalysis of example 3.1 as a set of proplets:

4.2 Representing Julia knows John as a set of proplets

$$\begin{bmatrix}\text{noun: Julia}\\ \text{fnc: know}\\ \text{mdr:}\\ \text{prn: 22}\end{bmatrix} \begin{bmatrix}\text{verb: know}\\ \text{arg: Julia John}\\ \text{mdr:}\\ \text{prn: 22}\end{bmatrix} \begin{bmatrix}\text{noun: John}\\ \text{fnc: know}\\ \text{mdr:}\\ \text{prn: 22}\end{bmatrix}$$

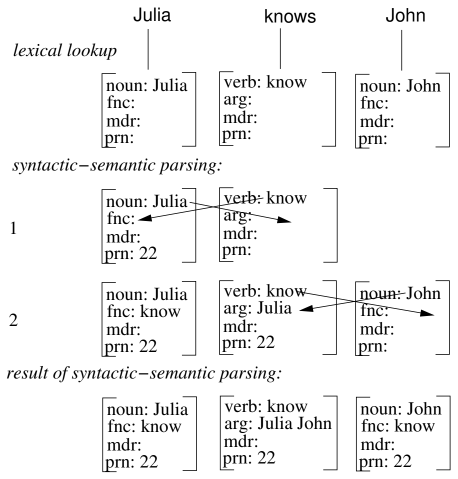
In contradistinction to 3.1 and 3.2, this representation is semantically complete. The intrinsic relations of functor, argument, and modifier are represented by the attributes fnc, arg, and mdr, respectively. The noun proplets Julia and John specify the verb proplet know as the value of their fnc attribute, and the verb proplet know specifies the noun proplets Julia and John as the values of its arg attribute. The three proplets are characterized as belonging to the same proposition by having a common prn value (here 22).

5. Rule-based recognition: the hearer mode

The retrieval required by memory-based recognition must be preceded by storage. After all, before the data provided by memory can be used for recognition they must have been stored there in the first place. Storage includes the analysis and interpretation of the raw input data.

As an example consider the hearer-mode in natural language communication. It requires that unanalyzed surfaces, such as the sequence Julia knows John, are lexically analyzed, syntactic-semantically interpreted, and then stored. This procedure is illustrated by the following derivation:

5.1 Syntactic-semantic interpretation in DBS


Julia
knows
John
lexical lookup
noun: Julia fnc: mdr: prn:
verb: know arg: mdr: prn:
noun: John fnc: mdr: prn:
syntactic−semantic parsing:
1
noun: Julia fnc: mdr: prn: 22
verb: know arg: mdr: prn:
2
noun: Julia fnc: know mdr: prn: 22
verb: know arg: Julia mdr: prn: 22
noun: John fnc: mdr: prn:
result of syntactic−semantic parsing:
noun: Julia fnc: know mdr: prn: 22
verb: know arg: Julia John mdr: prn: 22
noun: John fnc: know mdr: prn: 22

Lexical lookup consists in matching an unanalyzed surface, for example Julia, with a corresponding value of the first attribute of a lexical proplet. Lexical proplets are stored in the agent's memory as the result of language learning. They are isolated proplets because most of their attributes have no values and are therefore not yet connected to other proplets.

The task of syntactic-semantic parsing is to turn isolated proplets into connected proplets by means of *value copying.* In line 1, the noun value Julia of the first lexical proplet is copied into the arg value of the second, and the verb value know of the second proplet is copied into the fnc attribute of the first. The result is shown in line 2.

Next the verb value know of the second proplet is copied into the fnc attribute of the third proplet, and the noun value John of the third proplet is copied into the arg attribute of the second. In the course of this procedure, the control structure provides the proplets with a common prn value, here 22. The three proplets derived are ready to be stored at the end of the alphabetically ordered token lines John, Julia, and know of the hearer's word bank.

The syntactic-semantic interpretation shown in 5.1 is based on the rules of a time-linear LA-grammar, called **LA-hear**. Consider the following example, which illustrates the first combination step of 5.1 (explanations in italics):

5.2 Example of an LA-hear rule application

	rule name	*ss pattern*	*nw pattern*	*operations*	*rule package*
rule level	NOM+FV:	$\begin{bmatrix}\text{noun: } \alpha \\ \text{fnc:}\end{bmatrix}$	$\begin{bmatrix}\text{verb: } \beta \\ \text{arg:}\end{bmatrix}$	copy α nw-arg copy β ss-fnc	{FV+OBJ, ...}
proplet level		$\begin{bmatrix}\text{noun: Julia} \\ \text{fnc:} \\ \text{mdr:} \\ \text{prn: 22}\end{bmatrix}$	$\begin{bmatrix}\text{verb: know} \\ \text{arg:} \\ \text{mdr:} \\ \text{prn:}\end{bmatrix}$		

An LA-grammar rule consists of (i) a rule name, (ii) a pattern for the sentence start (*ss*), (iii) a pattern for the next word (*nw*), (iv) a set of operations, and (v) a rule package. The *ss* and *nw* patterns are specified as feature structures with variables and constants, which are matched with the input, here language proplets. A rule pattern matches a proplet iff (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 corresponding values of the proplet.

If the pattern matching of a rule is successful, the variables of the patterns are bound to the corresponding proplet values, and the rule operations are executed at the proplet level. The result is a new sentence start (*ss'*). After retrieval of a new next word, the rules of the current rule package are applied to the new input. If more than one rule in the current rule package is successful on the current input, more than one derivation branch is continued in parallel (ambiguity). If the patterns of a rule do not match the input proplets, the rule fails, and the derivation branch in question is discarded. The process of time-linear combination continues as long as (i) at least one derivation branch is successful and (ii) a next word is available.

In 5.2, the first input proplet is provided by the start state and the second by lexical lookup. The pattern matching is successful, and the variables α and β are bound to the values Julia and know, respectively. The operation *copy* α *nw-arg* copies Julia into the arg slot of the verb; the operation *copy* β *ss-fnc* copies know into the fnc slot of the noun. The output is as follows:

5.3 Result of the LA-hear rule application

$$\begin{bmatrix}\text{noun: Julia}\\ \text{fnc: know}\\ \text{mdr:}\\ \text{prn: 22}\end{bmatrix} \begin{bmatrix}\text{verb: know}\\ \text{arg: Julia}\\ \text{mdr:}\\ \text{prn: 22}\end{bmatrix}$$

In the next combination, the current result serves as the sentence start while lexical lookup provides the isolated proplet John as the next word.

6. Retrieval in different cognitive operations of DBS

Retrieval in Database Semantics is based on a search pattern like 3.3, whereby the degree of specificity depends on the use of variables vs. constants. A search begins by entering the token line corresponding to the value of the first attribute of the proplet pattern. From there, the proplet pattern is applied sequentially to the items in the token line.

Executing one search operation after another is called a *navigation* through the word bank. One kind of navigation is based on the repeated use of the same search pattern, as in question answering. As an example, consider a word bank with the following token line:

6.1 Example of token line with a search pattern

$$\text{girl} \begin{bmatrix}\text{noun: girl}\\ \text{fnc: walk}\\ \text{mdr: blonde}\\ \text{prn: 10}\end{bmatrix} \begin{bmatrix}\text{noun: girl}\\ \text{fnc: sleep}\\ \text{mdr: young}\\ \text{prn: 12}\end{bmatrix} \begin{bmatrix}\text{noun: girl}\\ \text{fnc: eat}\\ \text{mdr: big}\\ \text{prn: 15}\end{bmatrix} \begin{bmatrix}\text{noun: girl}\\ \text{fnc: read}\\ \text{mdr: smart}\\ \text{prn: 19}\end{bmatrix} \qquad \begin{bmatrix}\text{noun: girl}\\ \text{fnc: eat}\\ \text{mdr: } \alpha\\ \text{prn: } \beta\end{bmatrix}$$

This token line contains all the girl proplets, specifying for each in its fnc attribute what the girl in question did: the blonde girl walked, the young girl slept, the big girl ate, etc. The temporal order of arrival is indicated by (i) the sequence of storage and (ii) the prn values.

The search pattern at the end of the token line, containing the variables α and β, represents the wh-question Which girl ate [the cookie]? Applying the search pattern to the right-most token fails because the constants eat and read do not agree. Applying the search pattern to next token to the left, however, is successful, binding the variable α to the value big and the variable β to the value 15. Thus the big girl is a potential answer to the question.

To complete the answer, a second kind of navigation is used. Other than the first kind, it goes *across* different token lines. This is based on deriving a new search pattern from the current proplet (here girl with the prn value 15). The new pattern is constructed from the fnc and prn values of the current proplet. When this verb proplet is retrieved from the token line of eat, its arg attribute will specify what the girl in question consumed. If the value is cookie, the wh-question is successfully answered: it is the big girl with the prn value 15. If the value is not cookie, the girl token line is searched for another proplet with the fnc value eat, and the procedure is repeated.

The second kind of navigation, based on deriving new search patterns from current proplets, is also used for *activating* propositional content, for example as the basis of language production. This kind of navigation is handled by a kind of LA-grammar which is called **LA-think**. For example, assuming that the proplets of 4.2 have been sorted into the token lines John, Julia and know of the agent's word bank, and the current focus point is at the know proplet, then the following **LA-think** rule will move the navigation to the Julia proplet:

6.2 Example of an LA-think rule application

	rule name	*ss pattern*	*nw pattern*	*operations*	*rule package*
rule level	V_N_V:	$\begin{bmatrix}\text{verb: } \beta \\ \text{arg: X } \alpha \text{ Y} \\ \text{prn: k}\end{bmatrix}$	$\begin{bmatrix}\text{noun: } \alpha \\ \text{fnc: } \beta \\ \text{prn: k}\end{bmatrix}$	output position ss mark α ss	{V_N_V, ...}
proplet level		$\begin{bmatrix}\text{verb: know} \\ \text{arg: Julia John} \\ \text{mdr:} \\ \text{prn: 22}\end{bmatrix}$			

The *ss* pattern of the rule matches an activated proplet in the word bank, binding the variable β to the value know, α to Julia or John, and k to 22. Assuming that α has been bound to Julia, the rule retrieves (activates) the Julia proplet using the prn value 22, and returns to the verb proplet (operation output position ss). In order to prevent repeated traversal of the same proplet (relapse, see *tracking principles*, Hausser (1999, p. 454)), the arg value currently retrieved is marked:

6.3 Result of the LA-think rule application

$$\begin{bmatrix}\text{verb: know} \\ \text{arg: \#Julia John} \\ \text{mdr:} \\ \text{prn: 22}\end{bmatrix} \quad \begin{bmatrix}\text{noun: Julia} \\ \text{fnc: know} \\ \text{mdr:} \\ \text{prn: 22}\end{bmatrix}$$

Next, the rule V_N_V can apply again (see rule package), this time activating the proplet John.[6] There are several other kinds of navigation in Database Semantics, the most important being inference (cf. Hausser (2001)).

[6] Given that any proplet in the word bank usually provides more than one possible successor, **LA-think** must make choices. The most basic solutions are either random choices or fixed choices. For rational behavior, including meaningful natural language communication, however, the **LA-think** grammar must be refined into a control structure which chooses between continuation alternatives based on the evaluation of external and internal stimuli, the frequency of previous traversals, learned procedures, theme/rheme structure, etc. See Hausser (2002).

7. Pattern completion

The data structure of a word bank containing proplets in combination with the time-linear algorithm of LA-grammar is suitable not only

1. for reading content coded in natural language into the database,
2. for navigating through the database to activate content,
3. for reading activated content out of the database using natural language, and
4. for inferencing,

but also

5. for pattern completion.

The reason for this is the use of *functor-argument structure* and *coordination* as the two basic internal relations of natural language, coded directly as a bidirectional pointering between proplets.

As an example illustrating point 5, consider the following technical procedure in the hearer-mode: Whenever a given sentence start is to be combined with a next word, the system searches in the word bank for the *most frequent continuation* to this sentence start, defined (i) in terms of morphosyntactic properties of the next word, (ii) the kind of its syntactic-semantic attachment, and (iii) its concept. When this most frequent continuation of previous interpretations has been determined for the current sentence start, the system *actively checks* whether or not it is matched by the current next word (as an unanalyzed raw surface).

More specifically, in example 5.2 the sentence start consists of the proplet Julia. To determine the most frequent continuation, the system searches the token line of Julia for the most frequent fnc value, e.g. know, and checks whether the next word is indeed know. If the result is negative, the system checks for the next frequent fnc value, etc.

In this way, the hearer-mode would not rely solely on its ability to accurately recognize isolated next words and compute their combination with the current sentence start, but rather attempt to reduce the search space by suggesting likely properties of the next word candidates, based on previous continuations stored in memory. This procedure would not replace rule-based lexical lookup and syntactic-semantic parsing, but rather be used to improve its average efficiency when faced with a large number of combinatorial possibilities.

8. Case-based recognition

At the level of syntactic-semantic parsing in the hearer-mode, memory-based hypotheses about the next word are merely an option for improving average efficiency. This is because the native speakers' linguistic intuitions are sufficiently general and precise to be modeled almost completely by the lexical lookup and the rules of an

LA-grammar system. Variations of pronunciation and syntactic-semantic structure are comparatively limited.

At the lower levels of pattern recognition, however, the variety of possible interpretations may be (i) very large and (ii) of a non-general nature in the sense that they depend more on the individual's prior experience than the conventions of the language community. Thus we have two extremes, namely the rule-based and the case-based method.[7]

Rule-based recognition has already been illustrated with the example of natural language interpretation in the hearer-mode (Section 6). As an example of case-based recognition consider the following definition:

8.1 Generic LA-grammar for one-dimensional input

$$\text{ST\_}S = \{[\alpha], \text{r-1}\}$$

$$\text{r-1:} \begin{bmatrix}\text{feature: } \alpha \\ \text{previous:} \\ \text{next:}\end{bmatrix} \begin{bmatrix}\text{feature: } \beta \\ \text{previous:} \\ \text{next:}\end{bmatrix} \quad \begin{array}{l}\text{copy ss-feature nw-previous} \\ \text{copy nw-feature ss-next}\end{array} \qquad \{\text{r-1}\}$$

$$\text{ST\_}F = \{([\text{feature: } \beta], \text{rp}_{r-1})\ \}$$

This generic LA-grammar uses the generic intrinsic relations previous and next, and is based on generic lexical lookup, defined as follows:

8.2 Generic lexical lookup for one-dimensional items

long attributes *abbreviated attributes*

$$\begin{bmatrix}\text{feature: } \alpha \\ \text{previous:} \\ \text{next:} \\ \text{word number:} \\ \text{proposition number:}\end{bmatrix} \qquad \begin{bmatrix}\text{f: } \alpha \\ \text{p:} \\ \text{n:} \\ \text{wn:} \\ \text{prn:}\end{bmatrix}$$

This lexical lookup can be applied to any item in the input sequence: the variable α used as the value of the feature attribute is simply replaced by (an image of) the input item in question. The result of lexical lookup is a proplet with attributes for previous and next, word number and proposition number. While the values for the attributes p: and n: result from the copying operations of the generic LA-grammar 8.1, the values for the attributes wn: and prn: are provided by the control structure of the parser.

The result of parsing an arbitrary input sequence, for example ababaaabb, has the following form:

[7] The term 'case' is not used here in the sense of grammatical case, but rather in the sense of incident.

8.3 ababaabb parsed by generic LA-grammar

$$\begin{bmatrix}\text{f: a}\\ \text{p:}\\ \text{n: b}\\ \text{wn: 1}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: b}\\ \text{p: a}\\ \text{n: a}\\ \text{wn: 2}\\ \text{prn:1}\end{bmatrix}
\begin{bmatrix}\text{f: a}\\ \text{p: a}\\ \text{n: b}\\ \text{wn: 3}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: b}\\ \text{p: a}\\ \text{n: a}\\ \text{wn: 4}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: a}\\ \text{p: a}\\ \text{n: a}\\ \text{wn: 5}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: a}\\ \text{p: a}\\ \text{n: a}\\ \text{wn: 6}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: a}\\ \text{p: a}\\ \text{n: a}\\ \text{wn: 7}\\ \text{prn: 1}\end{bmatrix}
\begin{bmatrix}\text{f: b}\\ \text{p: a}\\ \text{n: b}\\ \text{wn: 8}\\ \text{prn: 1}\end{bmatrix}$$

$$\begin{bmatrix}\text{f: b}\\ \text{p: b}\\ \text{n:}\\ \text{wn: 9}\\ \text{prn: 1}\end{bmatrix}$$

In a word bank, these proplets would be sorted into two token lines, a and b. Let us assume that the sequence ababaaabb has been previously encountered and stored many times. After a pause, an initial a is presented to the system. Memory-based recognition determines the most frequent continuation for an a proplet with the wn value 1, and correspondingly for the rest of the sequence. In this way, next items do not have to be recognized from scratch, but are first checked for equivalence with promising candidates suggested by memory.

Rule-based and case-based recognition complement each other. Rule-based recognition is *prescriptive* in the sense that it accepts only certain combinations. The result is increased efficiency in recognition because of many possible combinations only the legal ones are tested. The drawback of rule-based recognition, however, is that it cannot handle any input which is does not conform to the rules.

Case-based recognition, in contrast, is *receptive.* It records any constellation encountered, and reflects the regularity and coherence of the agent's environment. The drawback of case-based recognition is that its analysis is comparatively crude. Also, it is not efficient when the actual input item corresponds only to the third or fourth candidate suggested by memory.

If an input structure is *brand-new*, i.e. has never been encountered before, recognition has to be done from scratch. As soon, however, as the structure in question is encountered for the second time, case-based recognition applies. If a structure is encountered very often, the generic LA-grammar and lexicon of the case-based method may evolve into a rule-based variant, like 5.2, with domain-specific attributes for intrinsic relations which characterize the content better than generic spatial co-occurrence.[8]

[8] The transition (i) from receptive case-based recognition to prescriptive rule-based recognition and (ii) from one- to multi-dimensional LA-grammars are large separate topics, which cannot be handled here. An interesting topic would be the design of a two-dimensional generic LA-grammar which is then scaled-up to a rule-based LA-grammar which incorporates the Waltz-algorithm (Waltz, 1975).

It is known that the Waltz-algorithm is generally $\mathcal{NP}$-complete, though in practice its complexity is usually much lower. One useful result of reconstructing the Waltz-algorithm in LA-grammar would be to determine its LAG complexity (Hausser, 1992). The rethinking required for the overall task described is substantial, comparable to the reconstruction of propositional calculus in DBS (Hausser, 2003).

9. Concepts

In order for memory-based recognition based on a generic LA-grammar to work, the system must have available some hardwired, innate distinctions. Otherwise, all elementary items read as proplets into the word bank would each have their own token line. What we need instead is a storage of *similar* items in the *same* token line, which requires the ability to recognize similarity in the first place.

Recognition of similarity and dissimilarity is based on patterns which are usually called *concepts*. Different analyses of concepts are the schema (Piaget, 1962; Stein and Trabasso, 1982), the template (Neisser, 1964), the feature (Hubel and Wiesel, 1962; Gibson, 1969), the prototype (Bransford and Franks, 1971; Rosch, 1975), and the geon (Biederman, 1987) approach. The definition of concepts raises the following questions:

1. what are the basic parts of a concept?
2. what are the basic parts made of?
3. what are the rules for combining the parts?

There is a fourth question hidden behind these three, namely: at what level of detail should the basic parts be defined?

From an abstract theoretical point of view, we are looking for that level of abstraction, where the necessary properties of the overall system cleanly separate from the accidental properties. From a computational point of view, we are looking for that particular degree of abstraction which happens to result in optimally efficient pattern recognition, indexing, retrieval, etc.

The importance of the fourth question is illustrated by the historical now somewhat toned down battle between two competing theories in cognitive psychology, namely features versus templates or prototypes, waged on the basis of recognition experiments. Roughly speaking, features take an atomistic position (all concepts are composed from a few elementary parts used many times), while templates take a holistic position (each concept has its own template, composition is best not used at all).

Computationally, these two extremes have inverse, complementary merits. For example, if there are only 50 different kinds of objects to be recognized, a holistic approach using templates would be reasonable: the cost of running through 50 templates in the worst case is compensated by the efficiency of the matching.

However, if there are a thousand different kinds of objects to each be matched as a whole, the cost of checking all templates in the worst case would outweigh any advantage in matching (if there is any left, given the increased complexity[9] of the templates). A compositional approach, in contrast, would do its pattern matching with comparatively few basic patterns, e.g., features, which are used over and over again to form complex concepts from basic parts.

[9] This increase of complexity follows from the necessity to differentiate the large number of individual templates from each other.

10. Geons

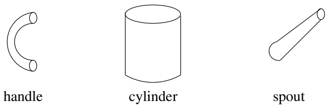
Why choose between features and templates if we can have the best of both? In cognitive psychology, there is a fairly recent proposal, called RBC (Recognition-by-Components) or geon theory, which makes the most of the inverse merits of the holistic and the atomistic approach by taking the middle of the road. Visual recognition is analyzed in terms of neither features nor templates, but rather in terms of some intermediate structures, called geons – from which all the parts for complex objects are built.

RBC or geon theory is described by Kirkpatrick (2001) as follows:

> The major contribution of RBC is the proposal that the visual system extracts geons (or geometric ions) and uses them to identify objects. Geons are simple volumes such as cubes, spheres, cylinders, and wedges. RBC proposes that representations of objects are stored in the brain as structural descriptions. A structural description contains a specification of the object's geons and their interrelations (e.g., the cube is above the cylinder).
> ...
> The RBC view of object recognition is analogous to speech perception. A small set of phonemes are combined using organizational rules to produce millions of different words. In RBC, the geons serve as phonemes and the spatial interrelations serve as organizational rules. Biedermann 1987 estimated that as few as 36 geons could produce millions of unique objects.

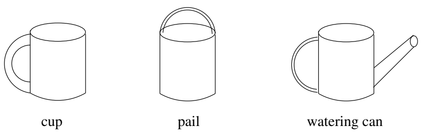
Consider the following examples of geons:

10.1 A small set of geons



These geons may be assembled into different complex concepts for objects such as the following:

10.2 Combining the geons into more complex objects

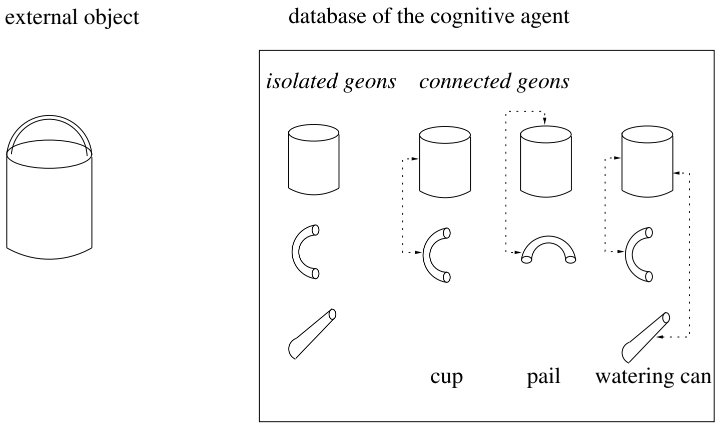


These complex objects raise the following question for database semantics: How should the combinations of geons into concepts for complex objects be stored? For example, should the handle and the cylinder of a pail be represented adjacent, as in the graphical representations 1.1 and 10.2, or should we specify their connection in a more abstract manner – thus opening the way to use geons as the basic key for storage and retrieval in a database?

This question bears on an important application of geons in visual recognition, namely *pattern completion.* If visual recognition is incremental, such that we see some part first and then rapidly reconstruct the rest of the object, how can we get the database to provide relevant content very fast and succinct in order to quickly narrow down the search?

It turns out that the data structure and the associated retrieval algorithm of Database Semantics provide a highly efficient procedure of pattern completion. As an example, consider recognition of a pail based on a word bank for geons:

10.3 Pattern completion during recognition



In this word bank, the isolated geons are the owner records and the connected geons the member records. For intuitive appeal (and as a substitute for an explicit formal definition of the internal relations in question), the connections between the geons are indicated by arrows. The complex objects specified in each column of connected geons are provided with names, i.e. cup, pail, and watering can.

Given that the external object is a pail, the agent might either first recognize the handle or the cylinder – depending on the conditions of lighting or the orientation of the object. If the cylinder is recognized first, the agent's database will indicate that cylinders are known to be connected in certain ways to handles and/or spouts. This information is used to actively analyze the cylinder's relations to the rest of the external object (pattern completion), checking for the presence or absence of the items suggested by the data base.

Similarly, if the handle is recognized first, the agent's database indicates that handles are known to be connected in certain ways to cylinders to form cups, pails, or watering cans. If the handle-cylinder connection is recognized first, there are two possibilities: pail or watering can. In our example, the system determines that there is no spout and recognizes the object as a pail.

This simple algorithm may be described somewhat more generally as follows: Assume that the agent is faced with a complex object consisting of an open number of geons and that one of these, e.g. B, has been recognized. Then the database will retrieve all B items and list all their connections to other geons. The geons on this list are all tried actively on the external object by checking (i) whether they are present or absent in the external object and, if present, (ii) the nature of their connection to B.

As soon as a second fitting geon, lets call it C, has been found, the algorithm will retrieve all BC connections of the given type from the database, as well as all their connections to other geons. This results in a substantially smaller list. By repeating the process, the algorithm converges very quickly.

For specifying the connections between geons more precisely let us replace the intuitive model illustrated in 10.3 by a full-fledged word bank format based on subproplets.[10] The following example expresses the same content as 10.3, but represents geons by names and codes the connections between geons (internal relations) by means of the attributes orientation (o:) and attach (a:). Geon subproplets belonging to the same concept are held together by a common onm value, whereby onm stands for object name and corresponds to prn.

10.4 Storing complex objects as geons in a word bank

isolated geons *connected geons*

$$\begin{bmatrix}\text{geon: cylinder}\\ \text{orientation:}\\ \text{attach:}\\ \text{onm:}\end{bmatrix}\quad \begin{bmatrix}\text{geon: cylinder}\\ \text{o: vertical}\\ \text{a: back handle}\\ \text{onm: cup}\end{bmatrix}\quad \begin{bmatrix}\text{geon: cylinder}\\ \text{o: vertical}\\ \text{a: top handle}\\ \text{onm: pail}\end{bmatrix}\quad \begin{bmatrix}\text{geon: cylinder}\\ \text{o: vertical}\\ \text{a: back handle}\\ \quad\text{front spout}\\ \text{onm: watering can}\end{bmatrix}$$

$$\begin{bmatrix}\text{geon: handle}\\ \text{orientation:}\\ \text{attach:}\\ \text{onm:}\end{bmatrix}\quad \begin{bmatrix}\text{geon: handle}\\ \text{o: vertical}\\ \text{a: back cylinder}\\ \text{onm: cup}\end{bmatrix}\quad \begin{bmatrix}\text{geon: handle}\\ \text{o: horizontal}\\ \text{a: top cylinder}\\ \text{onm: pail}\end{bmatrix}\quad \begin{bmatrix}\text{geon: handle}\\ \text{o: vertical}\\ \text{a: back cylinder}\\ \text{onm: watering can}\end{bmatrix}$$

$$\begin{bmatrix}\text{geon: spout}\\ \text{orientation:}\\ \text{attach:}\\ \text{onm:}\end{bmatrix}\qquad\qquad \begin{bmatrix}\text{geon: spout}\\ \text{o: diagonal}\\ \text{a: front cylinder}\\ \text{onm: watering can}\end{bmatrix}$$

The names of elementary geons serve here as the values of certain attributes and are represented by words of English, e.g., cylinder, handle, or spout. The orientation of geons, represented graphically in 10.3, is represented in 10.4 by the attribute values horizontal, vertical, or diagonal.

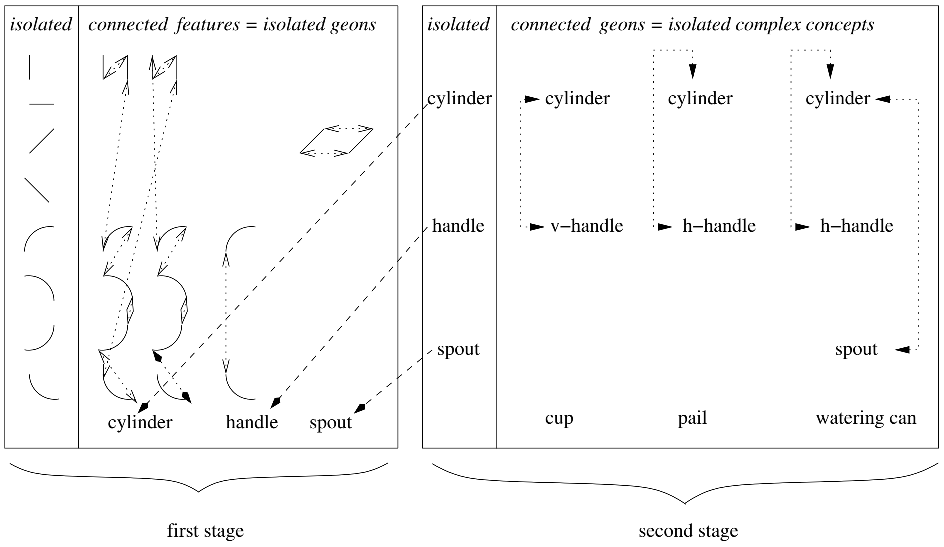
If the token line of, e.g., *cylinder* is activated in 10.4, the three connected geon subproplets with the onm values *cup, pail*, and *watering can* contained in it specify which geons to look for next in the object to be recognized. They are *back handle*, *top handle*, and *front spout*.

[10] The term proplet is used for a certain kind of data structure, whereby the value of the primary attribute is a *concept*. Subproplets refer to the same kind of data structure, but with primary attributes taking lower kinds of values, like geons or features.

11. Features

So far we have constructed relations (i) between isolated and connected geons resulting in complex concepts (Section 10), and (ii) between isolated and connected proplets (containing these concepts) resulting in propositions (Section 5). The analogous relation may be constructed also (iii) between isolated and connected features, resulting in geons:

11.1 Connecting features into geons and geons into concepts



The isolated features on the left of the first stage are for recognizing lines and arcs of different orientation, and serve as owner records. The associated member records are connected into constellations for a cylinder (first two columns),[11] a handle (third column),[12] and a spout (fourth column).[13]

At the second stage, a constellation of connected features (detected from the raw input) is classified by means of isolated geons, here *cylinder, handle*, and *spout*, which are provided by the database. At the third stage (not shown), the constellation of the isolated geons detected is analyzed and classified by means of isolated concepts provided by the database.[14]

The isolated concepts are embedded as attribute values into feature structures called proplets. In this way, the semantic core, represented by the concept, is separated from the combinatorics, represented by the other attributes of the proplet and their values. Combining proplets rather than concepts into connected

[11] The first column constitutes the left hand side and the bottom circle of the cylinder, while the second column constitutes the right hand side and the top circle of the cylinder.

[12] This column of connected features connects two ninety degree arcs into a half circle, representing a simplified, vertically oriented handle.

[13] The spout is indicated in a simplified manner as two parallel, diagonal lines.

[14] In other words, connected features = isolated geons, and connected geons = isolated concepts. What is defined at the lower stage is *named* at the next higher stage.

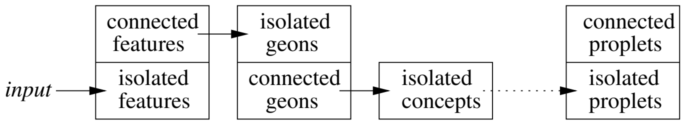
propositions has the advantage that the building up and processing of content can function independently of the nature of the concepts contained in the proplets.[15] Using proplet-like structures to separate the semantic core and the combinatorics may be done also with geons, as shown in 10.4, and with features (but is omitted here for reasons of space).

For realizing this system as an artificial cognitive agent (robot), the features (in the sense of cognitive psychology and neurology) must be implemented as *hardware* for analyzing input and actuating output. Everything else, i.e., the geons, concepts, and proplets, and the functions of cognition built on top of them, is pure *software.*

This is in concord with the fact that humans are provided with vision, hearing, etc., by nature but have to *learn* the structures of watering cans, mathematical formulas, table manners, etc., during language acquisition, development of reasoning, social behavior, etc. This kind of learning in natural agents has as its counterpart the programming of these structures in artificial agents.[16]

The step by step buildup from isolated features via geons and concepts to the connected proplets of concatenated propositions may be shown graphically as follows:

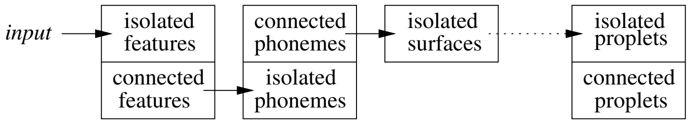
11.2 Features, geons, concepts, and context propositions



This construction of propositions based on a set of proplets containing concepts, which in turn are based on a smaller set of geons, which in turn are based on an even smaller set of features, may be generalized from the visual modality to the other modalities. For example, the concept of a cup may be composed not only of visual geons for a cylinder and a handle, but also of 'geons' in an extended sense for sound, taste, or touch.

The progression from features to geons to concepts to propositions at the context level is duplicated at the language level as a progression from features to phonemes (graphemes, etc.) to surfaces to propositions. This is in accordance with the original analogy between geons and phonemes (cf. Kirkpatrick (2001) cited above).

11.3 Features, phonemes, surfaces, and language propositions



[15] This move is similar to that taken by propositional calculus or predicate calculus, where sentences or predicates are represented by abstract variables or constants.

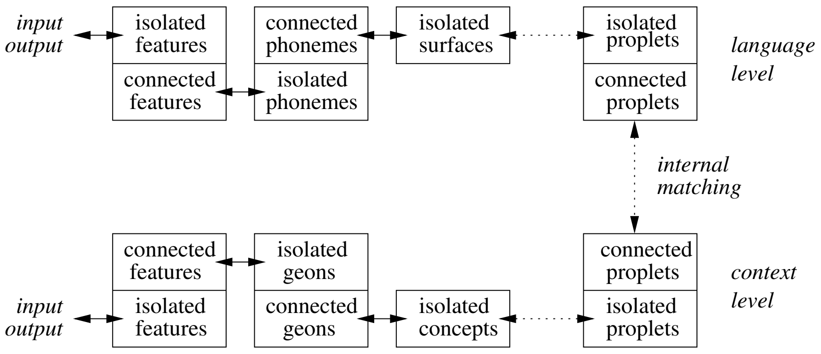
[16] Hopefully, machine-learning may soon help to obviate many of these programming chores.

During speech recognition, the input is analyzed by means of isolated features provided by the hardware and connected to the database. Analysis of the relations between the isolated features found in the input results in connected features. These are classified by isolated phonemes (or graphemes) provided by the database. Then the relations between the phonemes found in the input are analyzed, resulting in connected phonemes. These are classified by isolated surfaces provided by the database. The surfaces are matched with isolated proplets provided by the lexicon. Analysis of the relations between the isolated proplets found in the input, for example by means of parsing, results in connected proplets ready to be matched with the context.

During speech synthesis, the inverse procedure takes place. That is, the content intended to be uttered by the speaker is initially represented by concatenated propositions consisting of connected language proplets containing surfaces. These surfaces are reduced step by step to the kind of features which serve to synthesize the surfaces.

The transitions between features and propositions at the language level (cf. 11.3) and the context level (cf. 11.2) combine as shown in the following schema:

11.4 Combining the context and the language level



Note that the structure at the language level is exactly the same as the structure at the context level. It is just that the language level processes surfaces while the context level processes content. Otherwise the progression from features to geons/phonemes to concepts to proplets during recognition and from proplets to concepts to geons/phonemes to features during action are the same. The clue is that the meeting point between the two levels in 11.4 consists of (i) a connected language proplet and (ii) a connected context proplet ready to be tested for internal matching (providing the relation of *reference*).

12. Virtual Pattern Completion

The approach described is suitable also for handling another kind of pattern completion, namely the automatic reconstruction of the unseen side of a known object. For example, when we see a car from one side we have a pretty good idea of what it probably looks like from the other. This phenomenon has been explained on the basis of *frames* (Barsalu, 1999).

When recognizing an object from a certain side, the associated frame is activated, filling in the unseen side as a hypothesis. Like any complex object, frames may be coded as connected proplets (cf. 10.3, 10.4). In the terms of DBS, the unseen side is reconstructed by the agent using the most frequent continuations (stored in memory) of the seen side,

For example, if the agent has walked around different cars and noted their symmetrical nature, then the unseen side of a new car will be reconstructed hypothetically in an analogous way. The reconstruction is hypothetical because a small dent, for example, on the unseen side of the car cannot be predicted.

Conclusion

In DBS, pattern recognition comprises the analysis and the storage of the input, whereby the storage location is uniquely determined (i) by the result of the input analysis and (ii) the order of arrival. Pattern completion improves the average efficiency of a recognition analysis by suggesting *candidates to look for* in the input, based on prior continuations stored in memory. Memory-based pattern completion is an optional part of recognition, however, because it is not applicable if the input in question has not been encountered before.

The DBS procedure of recognition is based on four levels of increasing complexity, called (1) *features*, (2) *geons*, (3) *concepts*, and (4) *proplets*. At each level, three different methods of analysis may be applied, called (a) *rule-based*, (b) *case-based*, and (c) *brand-new*:

	features	geons	concepts	proplets
rule–based	a1	a2	a3	a4
case–based	b1	b2	b3	b4
brand–new	c1	c2	c3	c4

The analysis always proceeds from features via geons via concepts to proplets. Because a b1-analysis (case-based at the feature level) cannot be continued with an a2-analysis (rule-based at the geon level), for example, the derivation of any proplet will require at most six steps.

When faced with a *brand-new* item, the only method applicable are generic LA-grammars (cf. 8.1) with generic lexical lookup (cf. 8.2). At the initial, lowest level, features are provided directly by the input hardware. Thus, when a brand-new item is encountered again, its second analysis can be structurally correlated to the first, at least at the level of features. At the higher levels of geons and concepts, the distinctions of the next lower level are used in as much as they are available.

When faced with a *low frequency* item, the best method available is case-based recognition. Like brand-new recognition, case-base recognition employs generic LA-grammars. Unlike brand-new recognition, however, case-based analysis is supported by memory suggesting promising candidates to be recognized next.

When faced with a *high frequency item*, the method of choice are non-generic LA-grammars with specific lexical lookup, familiar from natural language analysis (cf. 5.2). These LA-grammars are highly restrictive and only accept input conforming to explicitly defined standards of well-formedness.

At each level, rule-based recognition is tried first because it succeeds the fasted if applicable and fails the quickest if not. Also, successful rule-based recognition provides the qualitatively best analysis because it uses domain-specific rather than generic internal relations. If the rule-based method fails at a given level, the case-based method is tried next. If the case-based method fails, the 'brand-new' method will succeed as the last resort.

<References>

Barsalu, Lawrence W. 1999. *Behavioral and Brain Sciences*, chapter Perceptual symbol systems. Cambridge University Press.

Biederman, Irving. 1987. Recognition-by-components: A theory of human image understanding. *Psychological Review* 94.2, 115–147. APA.

Bransford, J.D. and J.J. Franks. 1971. The Abstraction of Linguistic Ideas. *Cognitive Psychology* 2, 331–350.

Bresnan, Joan, editor. 1982. *The Mental Representation of Grammatical Relation.* MIT Press, Cambridge, Mass.

Fauconnier, Gilles. 1994. *Mental Spaces.* Cambridge University Press, New York.

Gazdar, Gerald, Ewan Klein, G. K. Pullum, and Ivan Sag. 1985. *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, Mass., and Blackwell, Oxford, England.

Gibson, Eleanor J. 1969. *Principles of Perceptual Learning and Development.* Prentice Hall.

Hausser, Roland. 1992. Complexity in Left-Associative Grammar. *Theoretical Computer Science* 106.2, 283–308. Dordrecht: Elsevier.

Hausser, Roland. 1999. *Foundations of Computational Linguistics, Human-Computer Communication in Natural Language.* Springer Verlag, Berlin, New York. pp. 578. 2nd edition in 2001.

Hausser, Roland. 2001. Database Semantics for Natural Language. *Artificial Intelligence* 130.1, 27–74.

Hausser, Roland. 2002. Autonomous Control Structure for Artificial Cognitive Agents. In H. Kangassalo et al. (eds.), *Information Modeling and Knowledge Bases XIII*, XIII, Amsterdam. IOS Press Ohmsha.

Hausser, Roland. 2003. Reconstructing Propositional Calculus in Database Semantics. In H. Kangassalo et al. (eds.), *Information Modeling and Knowledge Bases*, XIV, Amsterdam. IOS Press Ohmsha.

Hubel, David H. and Torston N. Wiesel. 1962. Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex. *Journal of Physiology* 160, 106–154.

Johnson-Laird, Philip N. 1983. *Mental Models.* Harvard U. Press, Cambridge, Mass.

Kirkpatrick, Kimberly, 2001. *Avian Visual Cognition*, chapter Object Recognition. cyberbook in cooperation with Comparative Cognitive Press.

Lakoff, George. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* University of Chicago, Chicago.

Lakoff, George and Mark Johnson. 1980. *Metaphors We Live by.* University of Chicago, Chicago.

Neisser, Ulric. 1964. Visual Search. *Scientific American* 210.6, 94–102.

Piaget, Jean. 1962. *The Language And Thought of the Child.* Routledge.

Rosch, Eleanor. 1975. Cognitive Representations of Semantic Categories. *Journal of Experimental Psychology* General 104, 192–253.

Schank, Roger C. and Robert P. Abelson. 1977. *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum, Hillsdale, New Jersey.

Shieber, S. M. 1986. A Simple Reconstruction of GPSG. In *Proc. of the 11th COLING*, pp. 211–215, Bonn, Germany.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA.

Stein, N. L. and T. Trabasso, 1982. *Advances in the Psychology of Instruction*, 2, chapter What's in a Story? An Approach to Comprehension, pp. 213–268. Lawrence Erlbaum Associates, Hillsdale, N. J.

Waltz, D. L. 1975. *The Psychology of Computer Vision*, chapter Understanding Line Drawings of Scenes with Shadows, pp. 19–92. McGraw-Hill.

Wierzbicka, Anna. 1980. *Lingua Mentalis. The Semantics of Natural Language.* Academic Press, Sidney.