Computational Foundations of Database Semantics

Roland Hausser
Universität Erlangen-Nürnberg (em.)
rrh@linguistik.uni-erlangen.de

May 23, 2016

In the nineteen seventies and eighties, linguists gave their phrase structure grammars of small natural language fragments to computer scientists for turning them into running software. Theoretically, context-free PS grammar runs in n^3 time, but the linguistically sophisticated fragments parsed very slowly. Moreover, many refinements broke up the dominant theory of nativism into numerous subtheories, creating an embarrassment of riches.

The response was broadening the empirical base regarding completeness of (i) data coverage and of (ii) function. Data coverage was addressed by the rise of corpus linguistics. Instead of having to choose from a bewildering variety of subtheories providing alternative analyses of traditional examples, attention turned to building large collections of "real" language. The method of choice for analyzing these data (a) automatically and (b) free from any of the available subtheories is statistics. Corpus linguistics, like the theories it set out to replace, is *sign-based*.

Completeness of function, however, presupposes an *agent-based* approach such as Database Semantics. DBS (1) integrates external *interfaces* for recognition and action, a *memory*, and an *algorithm* for mapping between them; (2) improves the data coverage by a continuous cycle of *upscaling* and automatic *verification*; and (3) combines traditional notions of grammar with a time-linear derivation order, resulting in *real time* performance. This paper describes the contact points between the linguistic aspects of DBS and its foundations in computer science.

1 String Search

An agent-based approach requires an explicit theory of how communicating with natural language works (CLaTR 4.1.2). Natural language processing (NLP), in contrast, tries to get by without natural language understanding, as shown by the direct approach in machine translation (FoCL 2.4.2). Nevertheless, some computational methods of NLP are also essential for building a talking robot, especially *string search*. Assuming a possible modality conversion between

speech and writing (CLaTR Sect. 2.3), string search is based on treating natural language surfaces as a numbered list of alphanumerical characters, e.g. letters, as in the following example:

1.1 TURNING NATURAL LANGUAGE SURFACES INTO A NUMBERED LIST

```
                1         2         3         4
m:  01234567890123456789012345678901234567890...
S:  She was the youngest of the two daughters...
```

Line S is the text to be searched (here from Jane Austen's EMMA, beginning of the 2nd paragraph.) Its letters are numbered in line m, with each number directly above the associated letter in line S. Turning a text automatically into a numbered list is computationally straightforward.

For simplicity and abstractness, the following example uses the letters ABCDE and space only. In addition to the lines m and S, there is the line W showing the abstract word searched for in S and the line i showing the numbering of the letters in W:

1.2 ABSTRACT EXAMPLE OF THE KMP ALGORITHM

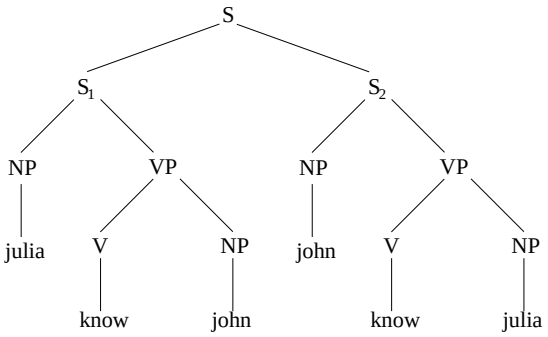
```
              1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Naively, instances of W in S may be found by moving W letter by letter along S, comparing each letter of W with the letter currently opposite in S. The worst case is an S sequence which matches W except for the last letter, such as letter 10 in S, i.e. space, and letter 6 in W, i.e. D.

The KMP algorithm by Knuth, Morris, and Pratt (1972) achieves a substantial improvement of efficiency over the naive approach by avoiding needless comparisons between W and S: instead of moving W through S letter by letter, the algorithm jumps ahead by ignoring letters *inferred* to be matching failures. For example, after trying ABCDAB unsuccessfully beginning with letter 4 in S, the algorithm restarts the next matching attempt with letter 11 (and not with 5) in S.

Search based on numbered characters may also be applied to tree structures by mapping trees automatically into equivalent alphanumerical sequences. Consider the following example of two phrase structure trees conjoined in an extrapropositional coordination:

1.3 PS TREE REPRESENTING Julia knows John. John knows Julia.



In this constituent structure (FoCL Sect. 8.4), upper case letters are used for nonterminal nodes, while lower case letters are used for terminal nodes (words).

By interpreting the format A[B C] as (i) A dominates B and C and (ii) B precedes C, the tree 1.2 may be mapped automatically into an equivalent list of numbered characters:

1.4 TRANSFORMING THE TREE 1.3 INTO A NUMBERED LIST

```
                          1                   2                   3                   4
m:  01 2 3 4 5  6  789 0 1234  5  678 9 0  1  23456 7 8 9 0  1  234 5 678  9  012 3 4  5  6789
S:  S[S1[NP[julia]][VP[[V[know]][NP[john]]]][S2[NP[john]][VP[V[know]][NP[julia]]]]
```

In this format, the KMP algorithm may be applied to millions of trees, as in a tree bank. The representation in line S is susceptible for matching with abstract patterns, allowing processing at higher levels of abstraction.

Another application of numbered lists is the automatic conversion of a text into an *inverted file* (Zobel and Moffat 2006). While the KMP algorithm is for batch mode computing a prespecified list of key words in large text files, the inverted file method allows on-the-fly incremental search. An inverted file specifies for each alphanumerical sign all its positions in the numbered list, as in the following example:

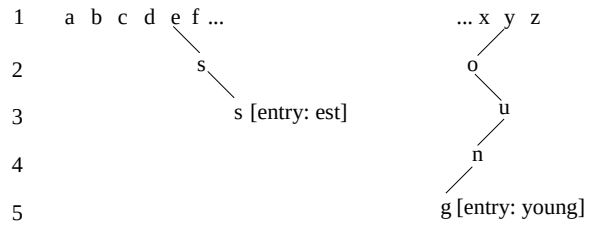
1.5 INVERTED FILE FOR THE ENGLISH SENTENCE IN 1.1

a: 5, 33
b:
c:
d: 32
e: 2, 10, 17, 26, 38
f:
g: 16, 26, 35
h: 1, 36
i:
j:
k:
l:
m:
n: 15
o: 13, 30
p:
q:
r: 39
s: 0, 6, 18, 40
t: 8, 19, 24, 28, 37
u: 14, 34
v:
w: 4, 29
x:
y: 12
z:

For example, t: 8, 19, 24, 28, 37 means that the 8th, 19th, 24th, 28th, and the 37th letter in 1.1 is a t. If the user spontaneously decides to look for a word, e.g. two, in the text of 1.1, typing a search command and the first letter of the word, i.e. t, will highlight its positions in the online text on the screen. When the second letter w is typed, the inverted file line w: 4, 29 is activated and only positions of a t followed by a w are highlighted, i.e. 28, 29. When the third letter o is typed, the inverted file line o: 13, 30 is activated and positions of a tw followed by an o, i.e. 28, 29, 30, are highlighted. Just as turning a written text into a numbered list, turning a numbered list automatically into an inverted file is computationally simple and efficient.

A third kind of a string matching algorithm is building and using a *trie structure* as a lexical storage facility (Briandais 1959, Fredkin 1960, Flouri 2012). DBS uses trie structures for the automatic (i) segmentation of word forms into allomorphs[1] and the (ii) lookup of their lexical definition (FoCL Sect. 14.3) during automatic word form recognition. The following example sketches the segmentation of the word form youngest of 1.1 into its allomorphs young and est:

1.6 EXAMPLE OF A DBS TRIE STRUCTURE STORING young AND est



The top level shows the letters in their alphabetical order. At this level, storage and retrieval of a lexical entry begins with the first letter of the input surface, here y. At the 2. level below the y, the second letter of the input surface, here o, is connected to the first. At the 3. level, the third letter of the input surface, here u, is connected to the second letter, and so on. The lexical entry to be looked up is stored at the last letter of the input surface, here the g at level 5.

As young- continues into young-est, the algorithm jumps to the letter e at the top level, walks down from there to the final letter t of est, and retrieves another entry. The entries are combined by morphological rules of English (FoCL Sect. 14.4) and result in an analysis like "superlative form of the adj young" (categorization and lemmatization). By segmenting complex word forms into their allomorphs, here young and est, their entries may be found and combined into lexical analyses with any desired degree of grammatical detail.

On the one hand, the computational method of trie structures is general in that it may be used to find entries of any kind, as in astronomy, biology, chemistry, etc. On the other hand, it supports linguistic desiderata such as the on-the-fly analysis of neologisms and the rule-based treatment of irregular paradigms (FoCL Chaps. 13–15). By testing a system of automatic word form recognition on suitable data, errors may be identified and permanently corrected by adding missing entries to the lexicon and improving existing entries and rules.[2] The use of a trie structure constitutes the first of several clearly defined contacts between DBS and computer science.

[1] In linguistics, the smallest meaningful entity in a word form is called a *morpheme*. Alternative forms of a morpheme, e.g. wolf, wolv-, are called *allomorphs* (FoCL Chap. 13).

[2] This is in contradistinction to the statistical tagging of today's corpus linguistics (FoCL Sect. 15.5).

2 Data Structure

In DBS, the allomorphs and their combination into word forms share a basic format, called *proplet*.[3] Proplets are the building blocks of *content*. An elementary language content is a word form proplet. A complex content is formed by a set of proplets concatenated by address, at the phrasal and clausal levels of grammar. For the abstract reconstruction of cognition, proplets play a similar role as the cell in biology and the atom in physics and chemistry (CLaTR Sect. 1.5).

From the view point of computer science, proplets are an abstract data type. Called *data structure* in DBS,[4] a proplet is defined as a *nonrecursive* feature structure with *ordered* attributes:[5]

2.1 Abstract example of a proplet as a data structure

attribute-1: value-a1 value-a2 ...
attribute-2: value-b1 value-b2 ...
...
attribute-n: value-m1 value-m2 ...

A proplet is nonrecursive because values are flat, i.e. they may not themselves be feature structures. Instead, an attribute takes a list of n (n≥0) elementary items as values. The order in the column of attributes is fixed by definition.[6] Otherwise, proplets as a computational data structure are completely general: they may be defined for any column of attributes and for any list of elementary items as value.

This makes proplets versatile for empirical purposes, as shown by the following examples:

2.2 The six main kinds of proplets illustrated for German

N, *symbol*	N, *indexical*	N, *name*	A, *symbol*	A, *indexical*	V, *symbol*
sur: Dach	sur: ihn	sur: Waldi	sur: schwarz	sur: dort	sur: las
noun: roof	noun: pro3	noun: (dog 23)	adj: black	adj: loc2	verb: read
cat: snp	cat: obq	cat: snp	cat: adn	cat: adv	cat: #ns3′ #a′ decl
sem: def sg	sem: sg m	sem: nm m	sem: pad	sem:	sem: past
fnc: see	fnc: see	fnc: see	mdd: book	mdd: sleep	arg: John book
mdr: red	mdr:	mdr:	mdr:	mdr:	mdr:
nc:	nc:	nc:	nc:	nc:	nc: (sleep 7)
pc:	pc:	pc:	pc:	pc:	pc:
prn: 1	prn: 2	prn: 3	prn: 4	prn: 5	prn: 6

The sur(face) attributes take language-dependent word form surfaces as their value. The core attributes in second position distinguish between noun, verb, and adj proplets; their core values

[3] The term is coined in analogy to droplet, indicating the function of proplets as the basic elements of propositions.

[4] Early computer science distinguished between a *data structure* for certain hardware parts and an *abstract data type* for software constructs. Today, this distinction has been eroded by a continuous process of abstracting away from the hardware level. DBS uses the shorter and simpler term *data structure* to characterize a basic abstract format, regardless of how it may be realized at the hard ware level.

[5] A proplet is a highly restricted case of a feature structure, generally defined (Carpenter 1992) as (i) a(n unordered) set of features and features defined as attribute value pairs (avp) which may (ii) take feature structures as values. Mathematically, a proplet seems to be an instance of a distributive lattice, which would connect DBS to the important field of order theory.

[6] Unordered items are as inefficient for computers to process as they are cumbersome for humans to read.

may be of the sign kinds symbol,[7] indexical, or name. In accordance with the Seventh Principle of Pragmatics (PoP-7, FoCL 6.1.7), nouns may take symbols, indexicals, and names, adjs may take symbols and indexicals, and verbs may take only symbols as their core values.

The features cat and sem specify grammatical properties such as number, gender, or tense. The attributes fnc (functor), arg(ument), and mdd (modified) provide the slots for obligatory continuation values, while the attributes mdr (modifier), nc (next conjunct), and pc (previous conjunct) provide the slots for optional continuation values. The values are supplied by cross-copying rules (5.1) during syntactic-semantic parsing in recognition, e.g. the hear mode, which turn lexical proplets into the concatenated proplets of a complex content. The prn value specifies the number of the elementary proposition which a content proplet belongs to.

Additional systematic distinctions result from the dichotomies between *lexical* and *content* proplets and between *language* and *context* proplets. They may be illustrated as follows:

2.3 FOUR PROPLET VARIANTS RESULTING FROM TWO DICHOTOMIES

(i) *lexical language*	(ii) *lexical context*	(iii) *content language*	(iv) *content context*
sur: Dach	sur:	sur: Dach	sur:
noun: roof	noun: roof	noun: roof	noun: roof
cat: sn	cat: sn	cat: snp	cat: snp
sem: sg	sem: sg	sem: def sg	sem: def sg
fnc:	fnc:	fnc: see	fnc: see
mdr:	mdr:	mdr: red	mdr: red
nc:	nc:	nc:	nc:
pc:	pc:	pc:	pc:
prn:	prn:	prn: 23	prn: 23

Proplets (i) and (ii) are lexical (no continuation and prn values), but differ in that (i) is a language proplet (sur value) while (ii) is a context proplet (no sur value). Proplets (iii) and (iv) are content (nonlexical) proplets (nonempty continuation and prn slots), but differ in that (iii) is a language proplet (sur value) while (iv) is a context proplet (no sur value). Proplets (i) and (iii are language proplets (nonempty sur slots), but differ in that (i) is lexical (no concatenation) while (iii) is a non-elementary content (concatenated). Variants (ii) and (iv) are nonlanguage (context) proplets, but differ in that (ii) is lexical while (iv) is concatenated. The use of proplets as a data structure constitutes the second of several clearly defined contacts between DBS and computer science.

3 Content

Agent-based DBS divides the mechanism of natural language communication into the hear, the think, and the speak mode. The hear mode maps unanalyzed external surfaces into content, the think mode selectively activates content for recall, reasoning, and action, and the speak mode maps activated content into the unanalyzed surfaces of a language of choice. While the external surfaces are concretely given as sound patterns or dots on paper which may be measured by the natural sciences, the associated encoding and processing of content must be reconstructed via *functional equivalence* between the artificial agent and the human prototype (CLaTR 1.1.1).

[7] We are following the terminology of Peirce (CP 2.228, 2.229, 5.473).

As an elementary content, a proplet combines two orthogonal aspects: (i) the semantic core and (ii) the combinatorics. The semantic core is coded by the *core value*, while the combinatorics are coded by the remainder of the proplet, called the *proplet shell* (CLaTR Sect. 6.6). The task of a proplets shell is to code human language intuitions regarding such grammatical properties as number, gender, tense, mood, as well as valency, agreement, functor-argument, and coordination. The following example shows a proplet shell taking different core values:

3.1 SINGLE PROPLET SHELL TAKING DIFFERENT CORE VALUES

proplet shell *context proplets*

sur:	sur:	sur:	sur:	sur:	sur:
noun: α	noun: square	noun: dog	noun: book	noun: child	noun: apple
cat: pn	cat: pn	cat: pn	cat: pn	cat: pn	cat: pn
sem: pl	sem: pl	sem: pl	sem: pl	sem: pl	sem: pl
fnc:	fnc:	fnc:	fnc:	fnc:	fnc:
mdr:	mdr:	mdr:	mdr:	mdr:	mdr:
nc:	nc:	nc:	nc:	nc:	nc:
pc:	pc:	pc:	pc:	pc:	pc:
prn:	prn:	prn:	prn:	prn:	prn:

$\Rightarrow$ $<$

The proplet shell is a pattern with a variable, here α, as the value of the core attribute, here noun. Content proplets are derived from the proplet shell by replacing the variable with different constants (see <). The proplets happen to be lexical because of their empty continuation attributes fnc, mdr, nc, and pc as well as their empty book-keeping attribute prn. The proplets happen to be context proplets because their sur attribute has no value.

As an example of a complex content consider the representation of The big dog likes the small bird. The small bird likes the big dog. as a(n unordered) set of proplets:

3.2 COMPLEX CONTENT AS A SET OF PROPLETS

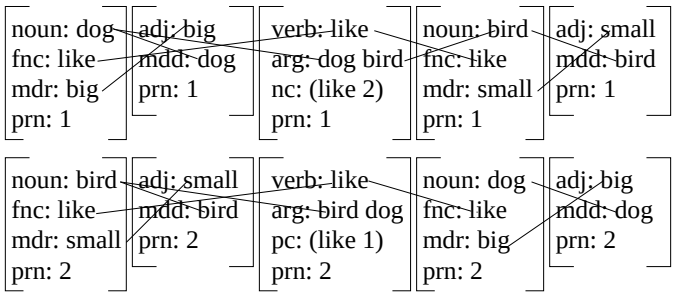
sur:	sur:	sur:	sur:	sur:
noun: dog	adj: big	verb: like	noun: bird	adj: small
cat: snp	cat: advn	cat: #s3 #a decl	cat: snp	cat: advn
sem: def sg	sem: pad	sem: pres	sem: indef sg	sem: pad
fnc: like	mdd: dog	arg: dog bird	fnc: like	mdd: bird
mdr: big	mdr:	mdr:	mdr: small	mdr:
nc:	nc:	nc: (like 2)	nc:	nc:
pc:	pc:	pc:	pc	pc:
prn: 1	prn: 1	prn: 1	prn: 1	prn: 1

sur:	sur:	sur:	sur:	sur:
noun: bird	adj: small	verb: like	noun: dog	adj: big
cat: snp	cat: advn	cat: #s3 #a decl	cat: snp	cat: advn
sem: def sg	sem: pad	sem: pres	sem: def sg	sem: pad
fnc: like	mdd: bird	arg: bird dog	fnc: like	mdd: dog
mdr: small	mdr:	mdr:	mdr: big	mdr:
nc:	nc:	nc:	nc:	nc:
pc:	pc:	pc: (like 1)	pc	pc:
prn: 2	prn: 2	prn: 2	prn: 2	prn: 2

The content consists of two propositions with the prn values 1 and 2, connected by the nc values (like 2) of the first and (like 1) of the second verb (extrapropositional coordination).

The functor argument relations of subject/predicate, object\predicate, and modifier|modified are also coded by the same value in different slots of different proplets, as shown by the diagonal lines in the following example, which indicate the intrapropositional relations of 3.2:

3.3 GRAPHICAL REPRESENTATION OF A COMPLEX CONTENT



Modeling the combinatorial aspects of a content as a set of proplets is correct if, and only if, the relations in the set correspond to the language intuition of the native speakers. For example, if the human content of the first proposition were the small dog likes the big bird, 3.2 and 3.3 would be wrong.

Next let us turn to the other aspect of an elementary content, namely the semantic core. The following example shows the single core value square (FoCL 3.3.1) serving in three proplet shells differing in their core attributes noun, verb, and adj (see <):

3.4 SINGLE CORE VALUE TAKING DIFFERENT PROPLET SHELLS

square $\Longrightarrow$				<
	sur: square	sur: squared	sur: square	
	noun: square	verb: square	adj: square	
	cat: sn	cat: n′ a′ v	cat: adn	
	sem: sg	sem: past	sem: pad	
	fnc:	arg:	mdd:	
	mdr:	mdr:	mdr:	
	nc:	nc:	nc:	
	pc:	pc:	pc:	
	prn:	prn:	prn:	

The first proplet may serve in a content corresponding to Mary picked the square (noun)., the second in a content corresponding to Mary squared (verb) her account., and the third in a content corresponding to Mary bought a square (adj) table (CLaTR 6.6.4–6.6.7).

While the combinatorial properties of a proplet shell must correspond to native speaker intuitions, the correctness of a core value as an elementary concept may be based on corresponding behavior in recognition and action. For example, an artificial agent's concept of *shoe* may be considered adequate if it picks out the same object(s) from a collection of different things as a human would (CLaTR Sects. 8.5, 15.6; NLC Sect. 4.3). Similarly for action: to be successful an artificial agent must realize elementary contents in the same way as a human would.

The procedural grounding of an agent's semantics requires (i) the hardware of external interfaces and a memory, and (ii) the software of concept types and concept tokens. The types must combine a declarative definition and a procedural implementation. They may be attached by convention to natural language surfaces to serve as literal meanings (CLaTR 1.4.1, 2.1.1).

In recognition, external interfaces provide raw data which are classified by concept types provided by memory (NLC Sect. 4.3). In action, blue prints for action provide concept tokens, which external interfaces realize as raw data (NLC Sect. 4.4). The processing of concepts, as a kind of elementary content besides pointers (indexicals) and markers (names), constitutes the third of several clearly defined contacts between DBS and computer science.

4 Database Schema

In DBS, the memory of a talking robot is realized as a database. In computer science, the structure supporting storage and retrieval is called the *database schema*. The database used in DBS, called word bank, is *content-addressable*[8] (Bachman 1973) in that it does not need an inverted file (1.5) as a separate index. The schema of a word bank may be illustrated as follows:

4.1 ABSTRACT DATABASE SCHEMA OF A WORD BANK

	member proplets		*now front*	*owner values*
	att-p1: con-a1 att-p2: con-d2	att-p1: con-a1 att-p2:		att-p1: con-a1
...	att-p3: con-f4 att-p4:	att-p3: con-b2 att-p4: con-x4		
	prn: prn-con-i	prn: prn-con-j		
	att-q1: con-b1 att-q2:	att-q1: con-b1 att-q2: con-a5		att-q1: con-b1
...	att-q3: con-g2 att-q4: con-h4	att-q3: att-q4: con-c4		
	prn: prn-con-m	prn: prn-con-n		
...			...	
	att-r1: con-z1 att-r2: con-k2	att-r1: con-z1 att-r2: con-c2		att-r1: con-z1
...	att-r3: att-rl: con-o4	att-r3: att-r4: con-b4		
	prn: prn-con-x	prn: prn-con-y		

This schema resembles the two-dimensional structure of a classic[9] network database with a column of owners and an associated list of members. It differs from a network database, however, in that the owners are values and the members are items conforming to the data structure of proplets (2.1) instead of records.

Compared to a network database, a word bank is highly restricted in that the list of member proplets preceding an owner value must (i) all have the owner value as their core value (no multiple owners) and (ii) be in the order of arrival, indicated by their position in the token

[8]The widely used relational databases (RDMS), in contrast, are coordinate-addressable (CLaTR Sect. 4.1).

[9]Elmasri and Navathe (2010) call a database *classic* if it is based on the data structure of records.

line and their prn value. Also, the owner value and the preceding list of member proplets are separated by a free slot, called the *now front*, which serves as the place of data processing.

Incoming proplets provided by recognition are stored at the current now front, thus fixing the moment of arrival (horizontal order). The correct token line is determined by the alphanumerical properties of a lexical proplet's core value (vertical order). The moment of arrival and the core value completely determine the location of any proplet in a word bank. The method does not interfere with the semantic relations of structure between proplets because the relations are coded as proplet-internal addresses, making the concatenated proplets of a content *order-free*.

The database schema of the abstract word bank 4.1 may be concretely instantiated as follows:

4.2 WORD BANK STORING THE CONTENT 3.2

member	*proplets*	*now front*	*owner values*
adj: big cat: adnv sem: pad mdd: dog ... prn: 1	adj: big cat: adnv sem: pad mdd: dog ... prn: 2		big
noun: bird cat: snp sem: indef sg fnc: like mdr: small ... prn: 1	noun: bird cat: snp sem: def sg fnc: like mdr: small ... prn: 2		bird
noun: dog cat: snp sem: def sg fnc: like mdr: big ... prn: 1	noun: dog cat: snp sem: def sg fnc: like mdr: big ... prn: 2		dog
verb: like cat: #s3 #a decl sem: pres arg: dog bird mdr: nc: (like 2) pc: prn: 1	verb: like cat: #s3 #a decl sem: pres arg: bird dog mdr: nc: pc: (like 1) prn: 2		like
adj: small cat: adnv sem: pad mdd: bird ... prn: 1	adj: small cat: adnv sem: pad mdd: bird ... prn: 2		small

The word bank contains the five *token lines* for the owner values big, bird, dog, like and small. Within each token line, the arrival order is indicated by position and the prn value.

The extrapropositional coordination between the two propositions of the content 3.2 is coded by the features [nc: (like 2)] in the first verb proplet and [pc: (like 1)] in the second. Extrapropositional address values consist of (i) a core value, e.g. like, and a prn value, e.g. 1, surrounded by parentheses, e.g. (like 1). Called *long address*, they serve as the primary key of a word bank.

An intrapropositional address value, in contrast, is written as the core value only and is called a *short address*. For example, the adj proplets *small* in 3.2 and 4.2 have the continuation feature [mdd: bird] with the attribute mdd (modified) and the short address value bird, just as the noun proplets *bird* have the continuation feature [mdr: small] with the attribute mdr (modifier) and the short address value small. Omitting the prn value of an intrapropositional address is possible because it is the same as that of the proplet containing the address; therefore specifying the prn value once more in an intrapropositional address would be redundant.

The input to recognition operations is restricted to the proplets at the current now front. A recognition operation applies whenever the two proplet patterns of its antecedent (5.3) find matching input (self-organization, Kohonen 1988). In DBS, concatenation is limited to the semantic relations of structure in natural language, i.e. functor-argument and coordination, intra- and extrapropositionally (NLC Chaps. 6–9).[10]

Proplets which have ceased to be candidates for further concatenation are regularly cleared from the now front by moving the owner values of the affected token lines one step to the right, leaving the current non-candidates behind as member proplets (loom-like clearance).[11] Once a content has been left behind as a set of concatenated member proplets, it may not be modified. However, it may be accessed by pattern or address, read, and copied to the now front for participating in current processing whenever needed. Accordingly, data correction is limited to diary-like comments at the now front, referring back to stored content never to be touched.

In the think mode, content is activated selectively by navigating along the semantic relations of structure between member proplets, using a continuation address of the current proplet as the primary key for finding a successor (5.5). For example, the navigation from the first *dog* proplet to *big* in 4.2 uses (i) the feature [mdr: big] to find the token line of big and (ii) the feature [prn: 1] to find the item in question within the token line. The database schema of a word bank constitutes the fourth of several clearly defined contacts between DBS and computer science.

5 Algorithm

The algorithm of DBS is defined in terms of operations which consist of an antecedent pattern and a consequent pattern (TCS). Their application is content-driven in that the currently available proplets activate all operations which match the pattern(s) of their antecedent. Binding constants of the input proplet(s) to corresponding variables in the antecedent pattern(s), enables

[10] The semantic relations of structure differ from the semantic relations of meaning, such as hypernymy and antonymy, and of content, such as cause-and-effect (CLaTR Sect. 5.3), which are handled by inferences instead of concatenation.

[11] Equivalently, the now front may be cleared by moving the member proplets one step to the left, as in a push down automaton. If proplets with the same core value are repeated in a proposition, as in slept and slept and slept, the current now front may store more than one proplet in a given token line (CLaTR 13.5.3).

the consequent containing the same variables, but in different slots, to derive an output.[12] Consider the application of an abstract operation to an abstract input content, deriving an abstract output content by means of its consequent:

5.1 ABSTRACT FORMAT OF AN INTRAPROPOSITIONAL RECOGNITION OPERATION

Name of operation

pattern level	[att-a1: α att-a2: con-a2 att-a3: prn: K]	[att-b1: β att-b2: con-b2 att-a3: prn:]	<	⇒	[att-a1: α att-a2: con-a2 att-a3: β prn: K]	[att-b1: β att-b2: con-b2 att-b3: α prn: K]	≪
	variable restrictions	⇑			⇓		
content level	[att-a1: con-a1 att-a2: con-a2 att-a3: att-a4: con-a4 prn: prn-con]	[att-b1: con-b1 att-b2: con-b2 att-b3: att-b4: con-b4 prn:]	<		[att-a1: con-a1 att-a2: con-a2 att-a3: con-b1 att-a4: cona-4 prn: prn-con]	[att-b1: con-b1 att-b2: con-b2 att-b3: con-a1 att-b4: con-b4 prn: prn-con]	≪
	antecedent				*consequent*		

By binding the abstract constants con-a1 and con-b1 to the corresponding operation variables α and β, respectively (see <), and by showing these same variables in the slots att-a3 and att-b3 of the operation consequent, the associated constants are copied into the corresponding slots of the content level (see ≪),[13] thus establishing a binary semantic relation of structure between the two output proplets.

For an operation to be successful on an input, the following conditions must be fulfilled:

5.2 CONDITIONS ON PATTERN MATCHING BETWEEN OPERATION AND INPUT

1. The attributes of an operation pattern must be a sublist or equal to the attributes of the matching input and output. For example, in 5.1 the attributes att-a1–att-3 in the first antecedent pattern are a sublist of att-a1–att-4 in the first input proplet at the content level.

2. Constant values of an antecedent pattern must have identical counterparts in the matching content. For example, the constant value con-a2 in the att-a2 slot of the first antecedent pattern has a counterpart in the corresponding slot of the associated input proplet.

As a concrete instantiation of the abstract operation 5.1 consider the following hear mode operation NOM+FV as it establishes the subject/predicate concatenation in Julia knows John. (NLC 11.6.1):

[12] The content-driven algorithm of DBS is in contradistinction to the substitution-based algorithms in the paradigm of phrase structure, e.g. the production rules of context-free BNF.

[13] The < and ≪ markers are used solely for guiding the human readers' attention.

5.3 HEAR MODE OPERATION ADDING PREDICATE TO SUBJECT

NOM+FV

pattern level

noun: α	verb: β	<	noun: α	verb: β	≪
cat: NP	cat: NP′ X VT	⇒	cat: NP	cat: #NP′ X VT	
fnc:	arg:		fnc: β	arg: α	
prn: K	prn:		prn: K	prn: K	

NP ε {snp, pnp , s3, p3}; NP′ ε {n′,ns3′, n-s3′}.
If NP ε {s3, snp} then NP′ ε {ns3′, n′}; otherwise, NP′ ε {n-s3′, n′}.

content level

sur: Julia	sur: knows	<	sur:	sur:	≪
noun: Julia	verb: know		noun: Julia	verb: know	
cat: snp	cat: ns3′ a′ v		cat: snp	cat: #ns3′ a′ v	
sem: nm f	sem: pres		sem: nm f	sem: pres	
fnc:	arg:		fnc: know	arg: Julia	
…	…		…	…	
prn: 1	prn:		prn: 1	prn: 1	

In the antecedent, the constants Julia and know are bound to the variables α and β, respectively (see <). In the consequent, the slots of arg and fnc, respectively, are filled with these values (see ≪). Because the operation is run via the core values of the content proplets, it may be applied to language proplets (non-empty sur slot) and context proplets (empty sur slot) alike. The restrictions on the variables NP (nominal valency filler) and NP′ (nominal valency position), positioned below the operation at the matching frontier, however, are language-dependent and handle the agreement between the subject and the predicate in English.

The following four kinds of cross-copying may be distinguished in DBS (CLaTR 16.6.7):

5.4 FOUR KINDS OF CROSS-COPYING

(i) slant duplex	*(ii) flat duplex*	*(iii) slant simplex*	*(iv) flat simplex*
core: α ~~core: β~~	sur: ~~sur: β~~	core: α ~~core: β~~	core: α core: β
cont.: cont.:	core: α ~~core:~~	cont.: cont.:	cat: ~~cat: X~~
prn: prn:	prn: prn:	prn: prn:	prn: prn:

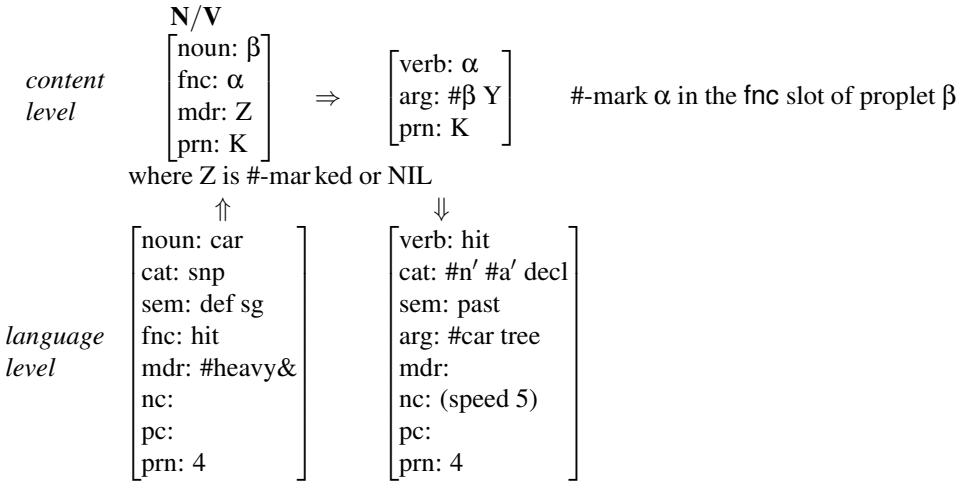
The cross-copying in 5.3 is of the kind ‘slant duplex.’ It is slant, because core values bound in the antecedent are copied into continuation slots of the consequent. It is duplex, because the copying is from the first proplet to the second and from the second proplet to the first.

The matching and binding used in DBS operations differs from the computational pattern matching in text processing. First, the two approaches are based on different data structures: the algorithms discussed in Sect. 1 use numbered elementary items such as letters in a text, whereas DBS uses non-recursive feature structures with ordered attributes (proplets). Second, DBS distinguishes between pattern proplets and content proplets; a pattern proplet must have at least one variable as a value, while a content proplet must not have any variable value at all. No such distinction arises in the string search mechanisms presented in Sect. 1.

A variant of the abstract operation format shown in 5.1 is used for navigating along the semantic relations of structure between stored proplets. As shown by the following example of a subject/predicate traversal, think mode operations consist of a single antecedent and a single

consequent pattern (NLC 12.3.3, 14.2.11, 14.3.4, 14.4.4).

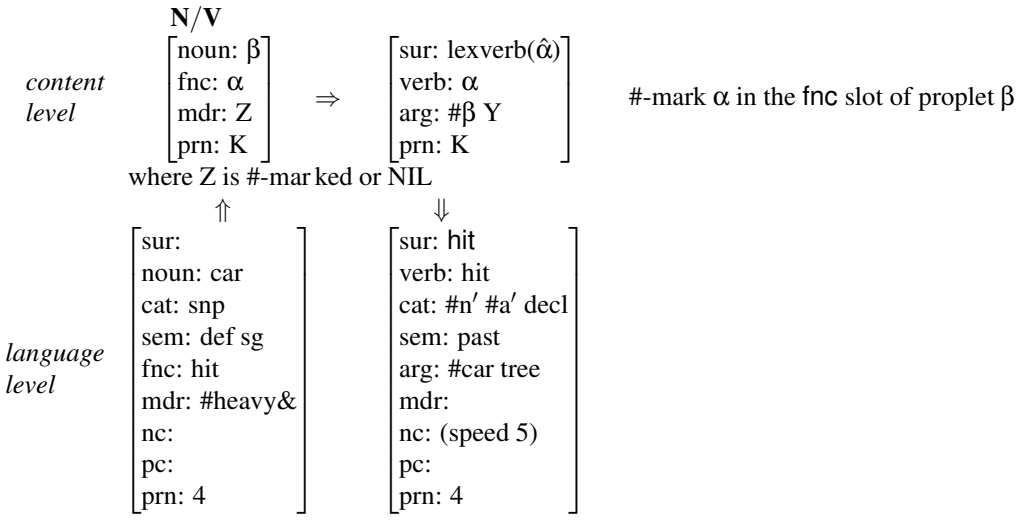
5.5 THINK MODE NAVIGATION FROM SUBJECT TO PREDICATE



The antecedent uses the continuation feature [fnc: hit] and the prn feature [prn: 4] of the input proplet *car* to navigate intrapropositionally to and selectively activate the next proplet *hit*. The condition where Z is #-marked or NIL prevents the operation from applying if a determiner is followed by an untraversed modifier. The instruction #-mark α in the fnc slot of proplet β cancels a continuation value and prevents reappliation of the operation, as in a loop.

A second kind of think mode operation besides selective activation are the *inferences*, defined to derive new content from activated content (NLC Sect. 5.3; CLaTR Sects. 5.2, 6.5, 10.3, 13.5). Both kinds may be turned into speak mode operations by embedding a language-dependent lexicalization rule (NLC 12.4.3, 12.5.2, 12.6.1) into the sur slot of the consequent, as shown by the following variant of 5.5:

5.6 SPEAK MODE NAVIGATION REALIZING A PREDICATE



The definition of operations for concatenation in the hear mode, for selective activation and inferencing in the think mode, and for the realization of language-dependent surfaces in the speak mode, on the one hand, and the abstract operation schema 5.1, on the other, constitute the fifth of several clearly defined contacts between DBS and computer science.

6 Functional Flow

For testing and debugging, the derivation steps of the DBS software components may be run *separately*. For example, a list of unanalyzed word form surfaces, ordered according to frequency, alphabetically, as they occur in a text, or in linguistic examples, may be used as input to (i) automatic word form recognition, resulting in a list of lexical proplets as output.

In a next separate step, any such list of lexical proplets may be used as input to (ii) syntactic-semantic parsing, resulting in a content or a list of contents. Then the order-free, concatenated proplets of a content may be (iii) sorted into a database, and the stored proplets may be processed (iv) for reasoning and (v) for the derivation of blue prints for action.

As part of the DBS communication cycle, in contrast, these steps must be integrated to work *incrementally*. In the agent's hear mode, the (i) input of a single next word form surface is followed by (ii) lexical lookup, (iii) storage of the resulting proplet in the appropriate token line at the current now front, and (iv) concatenation with other proplets currently available at the now front. This part of the cycle is repeated as long as a next input surface is provided. The only way to interrupt the procedure before reaching the end of the input chain are (a) an unrecognized word form, (b) ungrammatical input, or (c) an event distracting the agent's attention.

Thus, the proplets of a complex content are not sorted into the word bank in a separate phase, for example, after reaching the end of a sentence. Instead, automatic word form recognition stores each lexical proplet directly in what will be its final storage position when it (i) has been concatenated and (ii) is left behind as a member proplet (4.2) by a now front clearance. Concatenation is integrated into recognition in that a newly arrived proplet activates all operations which match it with their second antecedent pattern (5.3). An activated operation applies if it finds a proplet at the now front matching its first antecedent pattern. By binding variables in the antecedent to corresponding constants in the input, the consequent derives the output.

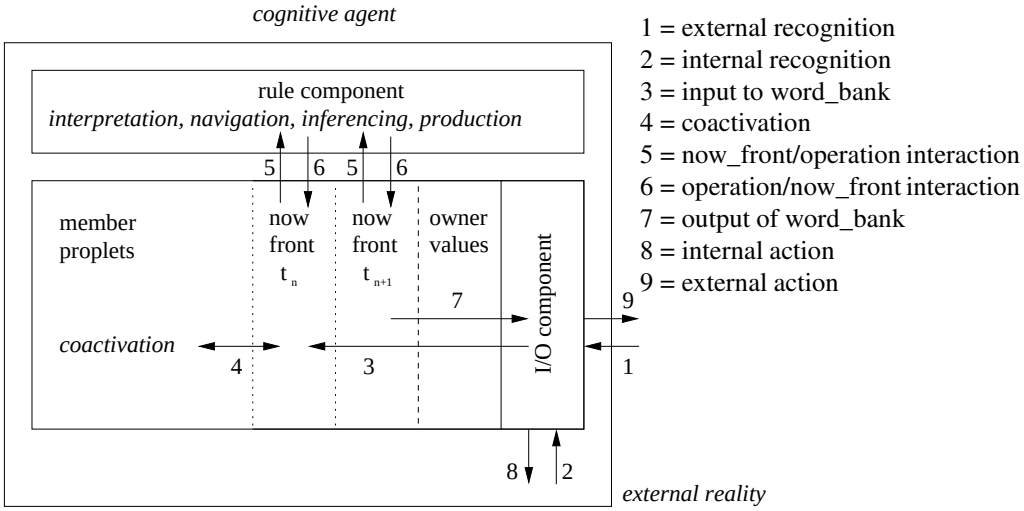
Similarly in the speak mode part of the DBS communication cycle. It is based on language-dependent lexicalization rules which are inserted into the sur slot of the goal pattern of a think mode operation. As the think mode (i) navigates along the semantic relations between proplets (selective activation) or (ii) derives new content by means of inferences, the lexicalization rules take the single goal proplet of each step as input and derive zero, one, or more unanalyzed language-dependent surface(s) as output (NLC Sects. 12.4–12.6, Chap. 14). In short, the lexicalization rules of automatic word form production do not apply to a sequence of concatenated proplets, but are integrated into each step of the think mode operations (5.6).

The combination of (i) restricting syntactic-semantic concatenation to the proplets currently available at the now front and (ii) reopening filled now front slots in regular intervals by moving the affected owner values one step to the right (loom-like clearance) provides a simple, effective form of self-organization. The now front proplets left behind as member proplets are those which have ceased to be candidates for further concatenation. Member proplets may never be

changed, though they may be activated, read, and referred to by address (CLaTR Sect. 13.3; NLC Sect. 11.2).

Graphically, the functional flow of the communication cycle through the agent's cognitive components may be shown as follows (CLaTR 14.3.2):

6.1 COMPONENTS AND FUNCTIONAL FLOW OF A DBS SYSTEM



The general I/O component for external and internal recognition and action shows unified input-output channels for the language and the context level (NLC 2.4.1): it includes automatic word form recognition and realization, as well as their nonlanguage counterparts (CLaTR Sect. 8). Input to the I/O component is the raw data from external (1) and internal (2) recognition, while the internal output of the I/O component are unconnected (lexical) proplets which are written to the current now front (3). The first such proplet is used as the sentence start and the second as a next word. If they match the input pattern of an operation (5), they are concatenated (6) by the output pattern (5.3), resulting in a new sentence start. After receiving a new next word from the I/O component (3) this incremental procedure is continued.

The interpretation of new data is complemented by deriving blueprints for action based on inferencing (NLC Sects. 5.3, 5.4; CLaTR Sects. 5.2, 5.3, 13.5, 16.6). Inferences resemble the other operations of DBS in that they are pattern-based and consist of an antecedent and a consequent. Stored in the rule component, inferences take (i) current content or (ii) corresponding older content (4) coactivated by current content (CLaTR Sect. 5.4) as input. The output of inferencing is written to the now front (6). The blueprint for action most likely to maintain or regain the agent's state of balance (CLaTR Sect. 5.1) may be passed to the I/O component (7) for internal (8) or external (9) realization. An example of internal action is continued reasoning.

In 6.1, the now front at moment t_n shows the interpretation of an input, while the now front at moment t_{n+1} shows the application of an inference deriving a blue print for action. The right hand side of a current now front is delineated by the column of owner values; the left hand side fades into the permanent sediment of member proplets.

The continuous coactivation of stored content by current content at the now front requires

massive, multiple search in real time. The quality of individual search operations depends on (i) the speed of the retrieval mechanism and (ii) sufficient expressive power enabling queries to retrieve content at the appropriate level of detail. In DBS, speed is provided by the database schema of a content-addressable word bank and the use of pointers. Expressive power is provided by query patterns which use the same semantic relations (CLaTR 3.2.3, 3.2.6) and some of the same constant values (CLaTR 4.2.2) as the content proplets searched for.

In summary, the now front is stationary for the agent, but the data of recognition and action move continuously through it in time. When a now front slot is filled, it is reopened by moving the owner value of the affected token line one step to the right. In this way, the now front is cleared for new data and the deprecated data are left behind as member proplets.

Even though the components listed in 6.1 originated in the design of a talking robot, they are completely general computationally in that the choice of attributes and values in the data structure of proplets (Sect. 2) is free and may be chosen as needed for any desired application taking time-linear input and producing time-linear output. Regarding the database schema, the order of token lines (column of owner values) is determined by the alphanumerical properties of the core values, regardless of the application. The temporal order of arrival (horizontal token lines) is provided by the interaction of the system with its environment. This abstract theoretical nature of the components and the functional flow from input to output constitute a sixth of several clearly defined interfaces between linguistics and computer science in DBS.

7 Conclusion

Database semantics (DBS) approaches agent-based computational linguistics as the project of designing and building a talking robot. This ensures basic functional completeness from the outset, while completeness of data coverage is approximated by incremental upscaling. Automatic testing on relevant data provides a method of (i) verification, which is essential for (ii) permanent correction and (iii) systematic extension. For practical use, the software may serve a multitude of applications in human-machine communication.

The construction of DBS began with the NEWCAT parser, written in LISP and developed on Xerox Dandelion and Dandetiger workstations.[14] Next, an *algebraic definition* was distilled from the software.[15] Restricted to the constructs of set theory, an algebraic definition satisfies the mathematical requirements. For computation, however, the abstract description of the overall system also requires a *declarative specification* which defines the external interfaces, the data structure, the algorithm, the database schema, and the functional flow from input to output.

In addition to these computational requirements, the declarative specification of a talking robot must cover the cognitive aspect. From a linguistic point of view, the latter includes the definition of content; the semantic relations of functor-argument structure and coordination; the elementary, phrasal, and clausal levels of grammatical complexity; many intrapropositional constructions such as infinitival subjects and objects; subject, predicate, and object gapping; phrasal noun

[14] Thanks to Stanley Peters and the CSLI Stanford (1984–1986) for making programming of the first left-associative parser possible (CLaTR Chap. 12).

[15] Thanks to Dana Scott and Stuart Shieber, who at different times and places helped in formulating the algebraic definition for LA grammar, published in CoL (1989) and TCS (1992).

and verb constructions; unbounded dependencies; sentential and verbal moods, etc. These have been are left aside here.[16] Instead, this paper confines itself to the computational aspects of the declarative specification, which serve as the foundation of the linguistic constructs and empirical analyses in DBS.

References

de la Briandais, R. (1959) "File Searching Using Variable Length Keys," Proc. Western Joint Computer Conf. 15, 295–298

Bachman, C.W. (1973) "The programmer as navigator," 1973 ACM Turing Award Lecture, *Comm. ACM,* Vol. 16.11:653–658

Carpenter, B. (1992) *The Logic of Typed Feature Structures*, Cambridge: CUP

CLaTR = Hausser, R. (2011) *Computational Linguistics and Talking Robots – Processing Content in Database Semantics*, Springer (preprint 2nd ed. available online at `lagrammar.net`)

CoL = Hausser, R. (1989) *Computation of Language*, soft cover reprint 2013, Springer

Elmasri, R., and S.B. Navathe (2010) *Fundamentals of Database Systems, 6th ed.*. Redwood City, CA: Benjamin-Cummings

Flouri, T. (2012) *Pattern matching in tree structures*, Dept of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Ph.D. dissertation

FoCL = Hausser, R. (1999) *Foundations of Computational Linguistics, Human-Computer Communication in Natural Language, 3rd ed. 2013*, Springer

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

Kohonen, T. (1988) *Self-Organization and Associative Memory, 2nd ed.*, Springer

Knuth, D. E., J. H. Morris, and V. R. Pratt (1977) "Fast Pattern Matching in Strings," SIAM Journal of Computing Vol. 6.2:323–350

NEWCAT = Hausser, R. (1986) *NEWCAT: Parsing Natural Language Using Left-Associative Grammar*, LNCS 231, Springer

NLC = Hausser, R. (2006) *A Computational Model of Natural Language Communication – Interpretation, Inference, and Production in Database Semantics*, Springer (preprint 2nd ed. available online at `lagrammar.net`)

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

TCS = Hausser, R. (1992) "Complexity in left-associative grammar," *Theoretical Computer Science*, Vol. 106.2:283–308

Zobel, J., and A. Moffat (2006) "Inverted Files for Text Search Engines," ACM Computing Surveys, Vol. 38.2:1–56

[16] For detailed linguistic analyses mostly of English see NLC and CLaTR.