ELSEVIER

Artificial Intelligence 130 (2001) 27–74

Artificial Intelligence

www.elsevier.com/locate/artint

Database semantics for natural language

Roland Hausser

Universität Erlangen-Nürnberg, Institut für Germanistik, Abteilung für Computerlinguistik, Bismarckstrasse 6 & 12, 91054 Erlangen, Germany

Received 11 October 1998; received in revised form 12 September 2000

Abstract

This paper presents a formal 'fragment' of database semantics as a declarative model of a cognitive agent. It is called a SLIM machine and functionally integrates the procedures of natural language interpretation, conceptualization, and production as well as query and inference. Each of these functions is illustrated explicitly by a corresponding LA-grammar. In addition, a control structure based on the principle of balance is presented. This principle mediates between the knowledge of the SLIM machine and its current situation by selecting a suitable action.

Keywords: Human–computer communication; Natural language; Query; Inference; LA-grammar; SLIM theory of language; Procedural semantics; Concatenated propositions; Autonomous navigation; Motor algorithm; Semantic networks

Overview

Current approaches to natural language semantics may be divided into two types: metalanguage-based and procedural. Metalanguage-based semantics are in turn of two kinds, namely truth-conditional (as in model theory) and use-conditional (as in speech act theory). Because metalanguage definitions are not designed for procedural implementation, neither truth-[1] nor use-conditional semantics are suitable for a computational model of natural language communication.[2]

E-mail address: rrh@linguistik.uni-erlangen.de (R. Hausser).

[1] Though Prolog procedurally models a subset of predicate calculus, its applications to natural language, such as definite clause grammar (Pereira and Warren [32]), are used to parse sentences by assigning tree structures. Thus, DCGs are concerned with characterizing the syntactic well-formedness of sentences, and not with assigning truth-conditional semantic interpretations to them.

[2] See Hausser [17] for a detailed argument with special attention to truth-conditional semantics. A critique of use-conditional semantics may be found in Hausser [16, pp. 83–86].

0004-3702/01/$ – see front matter
PII: S0004-3702(01)00083-2

A procedural semantics, on the other hand, uses concepts based on the recognition and action procedures of cognitive agents as its basic elements. Examples are Lakoff and Johnson 1980 [23], Sowa 1984 [40], Hausser 1984, 1989, 1999 [11,13,16], Lakoff 1987 [22], Langacker 1987/1991 [21], Fauconnier 1997 [7], Gärdenfors 2000 [10], and Talmy 2000 [42]. This method of analyzing natural—and modeling artificial—cognitive agents provides a viable alternative to metalanguage-based semantics.

The advantages of the procedural approach may be endangered, however, by a lack of generality and formal rigor. This may result in computational systems which work only for special cases (the problem of upscaling) and/or do not have a declarative specification (as in hacks), or in vague descriptions of an anecdotal nature.

The SLIM theory of language is a procedural approach aiming at a systematic general model of cognition. It is based on the methodological, empirical, ontological, and functional principles of **S**urface compositionality, time-**L**inearity, and **I**nternal **M**atching.

The acronym SLIM is also motivated as the word meaning *slender*. This is because detailed mathematical and computational investigations have proven SLIM to be efficient in morphology, syntax, semantics, and pragmatics—both relatively, in comparison to existing alternatives, and absolutely, in accordance with the formal principles of mathematical complexity theory (cf. Hausser [14]).

The cognitive foundations of natural language communication within the SLIM theory of language are called database semantics. This component of SLIM models the transfer of information from the speaker to the hearer by representing the knowledge of speaker and hearer, respectively, in the form of databases. Natural language communication is successful if a certain database content, coded by the speaker into natural language signs, is reconstructed analogously in the database of the hearer.

The database content is represented in the form of concatenated propositions. A proposition is coded as a set of coindexed proplets. These are feature structures which specify the functor-argument structure and the relations to other propositions by means of attributes. Proplets eliminate the restrictions of graphically-based representations, allowing

(i) a time-linear reading-in of propositions (storage),
(ii) an activation of propositions by means of time-linear navigation, and
(iii) an accessing of propositions using concepts as the key (retrieval).

This special data structure, called a word bank, interacts with a special time-linear motor algorithm, called LA-grammar. Metaphorically speaking, the word bank provides a railroad system and LA-grammar the locomotive for navigating through it.

Navigation activates the propositional content traversed. This serves as a simple model of thought, which is used as the basis of conceptualization (*what to say*) in language production. In addition, navigation provides important aspects of realization (*how to say it*), namely the basic serialization and a substantial part of lexicalization.

There are three kinds of propositions in database semantics: episodic, absolute, and language propositions. They are built from corresponding kinds of proplets, which differ in their feature structure, the use of concept types versus concept tokens, and the presence or absence of a language surface. Different kinds of propositions interact with each other in terms of matching their respective proplets.

The propositions of a word bank are processed by LA-grammars for (a) recognition, (b) action, and (c) inference. The LA-grammars for recognition read propositions resulting from (1) external, (2) internal, and (3) language recognition into the word bank. The LA-grammars for action realize certain propositions as (4) external, (5) internal, and (6) language actions. The LA-grammars for inference derive new propositions by means of (7) generalization, as well as (8) episodic, and (9) absolute inference.

These nine LA-grammars are fixed, stationary components—like engines mounted in a workshop. They start running if and when they receive input. In recognition, the input is provided by events in the external and internal environment. But what provides input for action and inference?

The answer is a special LA-grammar, called LA-MOTOR, which is mobile—like an engine on wheels. This locomotive moves continuously through the word bank's propositional content, choosing continuations either at random (free association) or by following a highlighting of continuation proplets provided by the control structure. The location of LA-MOTOR in the word bank constitutes the cognitive agent's current focus of thought.

Navigation by LA-MOTOR may be one-level or two-level. In one-level navigation, the contextual proplets are simply traversed. In two-level navigation the contextual proplets traversed are matched by corresponding absolute or language proplets. Depending on their content, these matching proplets are passed on to the stationary LA-grammars for action and for inference, activating them by providing them with input.

After a general introduction in Sections 1–3, the three kinds of propositions and their internal matching are described in Sections 4–6. The motor algorithm and the control structure are presented in Sections 7 and 8. Sections 9–16 define the formal fragment, providing substantial parts of a declarative specification for a SLIM machine.

1. Database metaphor of natural communication

In interactions with a standard computer, the understanding of natural language is restricted to the user. For example, when a user searches a database for a red object, (s)he understands the word red before it is put into—and after it is given out by—the computer. But inside the computer, red is manipulated as a sign which is uninterpreted with respect to the color denoted.

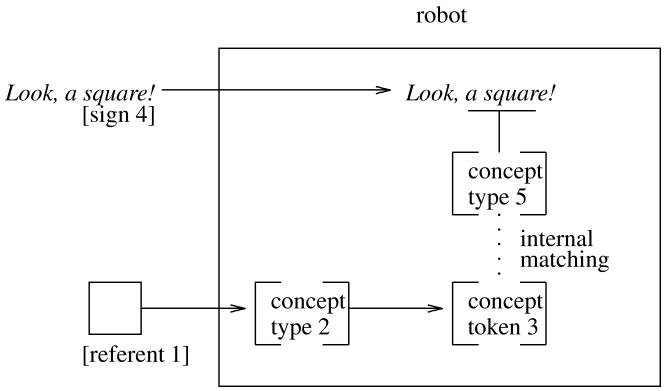
However, what is true for standard computers does not apply to human–computer communication in general. It is possible with current technology to construct an autonomous robot which may be asked to go and get a certain object of a known type, which has not been previously encountered, e.g., the red book on the desk in the other room. If the machine is able to spontaneously perform an open range of different jobs like this, it has an understanding of language which at some level may be regarded as functionally equivalent to the corresponding cognitive procedures in humans.

The robot's cognitive interaction with its task environment is based on patterns which are applied to incoming and outgoing parameter values of its perception and action

components. These patterns provide for a procedural definition of concepts which may double as the meanings of language. [3]

The robot's language interpretation must work without assuming any external relation between the sign and the referent. Instead, it is based on matching language meanings with contextual counterparts inside the robot, as illustrated below:

1.1. Immediate reference based on internal matching.



The robot recognizes the referent (1) by applying the concept type (2) to the incoming parameter values (here, a bitmap outline) and instantiating the referent internally as a concept token (3). The sign (4) is recognized in a similar way. The lexicon assigns to the sign square the same concept type as (2) as its literal meaning (5).

The relation between the levels of language and of the referent (also called the *context* level) is established by matching the lexical concept type (5) and the referential concept token (3). A literal meaning lexically defined as a type may be matched with any number of referential tokens. This models the flexibility which distinguishes the natural languages from the logical and programming languages.

The kind of reference illustrated in 1.1 is called *immediate* because sign and referent are both present in the current task environment. Reference to objects not in the current task environment, for example, to the person J.S. Bach, is called *mediated* reference. In mediated reference, the cognitive agent interacts with the task environment solely on the level of language, using objects in its memory as referents.

Internal matching is illustrated in 1.1 with an isolated content word. The principle may be generalized to full-blown natural language communication by means of the *database*

[3] For a detailed analysis of recognition and action on the basis of concept types, concept tokens, and parameter values see Hausser [16, pp. 53–61]. The concepts analyzed there are two-dimensional geometric forms like triangles, squares, rectangles, pentagons, etc., and colors like red, green and blue. See also Langacker [21], whose analyses of *above* versus *below* (pp. 219, 467), *before* (p. 222), *enter* (p. 245), *arrive* (p. 247), *rise* (p. 263), *disperse* (p. 268), *under* (pp. 280, 289), *go* (p. 283), *pole climber* (p. 311), and *hit* (p. 317) may be regarded as logical analyses in our sense.

metaphor, according to which the knowledge of the speaker and the hearer is represented in the form of databases. Communication is successful if the speaker encodes a certain part of his or her database into language, and the hearer reconstructs this part *analogously* in his or her database—in terms of (i) correct decoding and (ii) correct storage at a corresponding location.

The database metaphor assumes from the outset that the communicating databases are embedded in cognitive agents (e.g., robots) with language-based as well as non-language-based recognition and action. The database metaphor may be applied to statements, questions, and requests alike. In statements, the content encoded by the speaker must be decoded by the hearer and stored in an analogous location. In questions and requests, the hearer must locate the language-based content and the non-language-based task, respectively, to derive adequate responses.

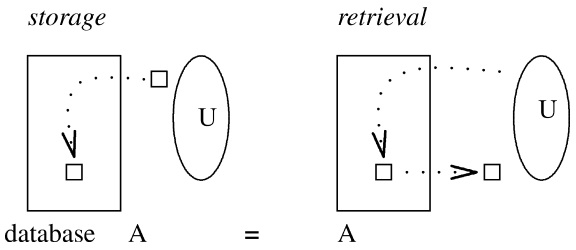
Implementing the database metaphor requires the following:

1.2. Tasks for realizing the database metaphor.

(1) Encoding database content into language (speaker mode).
(2) Decoding language into database content (hearer mode).
(3) Querying a database for a certain content (speaker questioning hearer).
(4) Inferencing over the content of the database to determine identity between nominals, temporal relations, nonliteral (e.g., metaphoric) uses, etc.
(5) Indexing the location of the speaker's content in the language sign such that the hearer can infer the analogous location for storage.

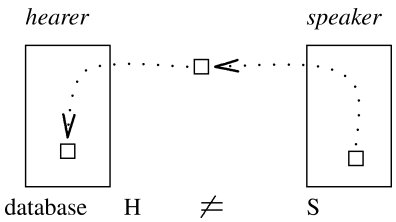
To explain the nature of the indexing problem, let us compare the interaction between a user and a conventional database, and between a speaker and a hearer.

1.3. Interaction with a conventional database.



The interaction takes place between two different entities, the user (ovals) and the database (larger boxes), whereby the small boxes represent the language signs serving as input and output. The user controls the storage and retrieval operations of the database with a programming language, the commands of which are executed as electronic procedures.

1.4. Interaction between speaker and hearer.



Here the larger boxes represent cognitive agents which may be natural or artificial. There is no user. Instead, the interaction takes place between two similar and equal cognitive agents, which control each other's flow of information by alternating in the speaker- and the hearer-mode (*turn taking*). The speaker controls language production as an autonomous agent. The hearer's interpretation is controlled by the incoming language expressions.

2. Representing content: Concatenated propositions

The first two tasks for realizing the database metaphor, i.e., encoding and decoding, require a general, language- and domain-independent method of representing content. For this, we go back to the classic notion of a proposition. According to Aristotle (384–322 B.C.), propositions are simple representations of what is. Propositions are so general and abstract that they have been regarded as both the states of real or possible worlds and the meanings of language sentences.

Propositions are built from three basic kinds of elements, called *arguments*, *functors*, and *modifiers*. An elementary proposition consists of one functor, which combines with a characteristic number of arguments. Modifiers are optional and may apply to functors as well as to arguments.

In the world, the arguments are *objects* (in the widest sense), the functors are (intrapropositional) *relations*, and the modifiers are *properties*. These constitute a simple ontology which is intended here for a general representation of cognitive states—in contradistinction to other possible ontologies for modeling aspects of the world from the viewpoint of physics (based on atoms, gravity, etc.), biology (based on metabolism, reproduction, etc.), or economics (based on markets, inflation, interest rates, etc.).

In language, the arguments are the *nominals*, the functors are the *verbs*, and the modifiers are the *adjectives* (whereby adnominals modify nominals and adverbs modify verbs). The verbs, nominals, and adjectives [4] comprise the content words of a language.

[4] Phrases like *to live in London* raise the question of whether *in London* should be treated as a(n obligatory) modifier or as an argument—touching upon the close functional relation between case-assigned noun phrases and prepositional phrases. Also, some languages like Korean raise the question whether certain one-place functors should be treated as adjectives or one-place verbs—touching upon the close functional relation between modifiers and one-place functors. How these alternatives are decided is an empirical question which any linguistic functor-argument analysis must tackle sooner or later.

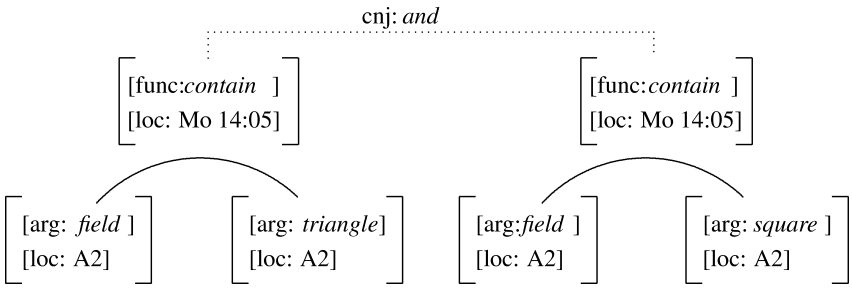
2.1. Elementary components of basic propositions.

	logic	*world*	*language*
1.	argument	object	nominal
2.	functor	relation	verb
3.	modifier	property	adjective

Elementary propositions can combine with operators such as negation or be concatenated by operators such as disjunction. In natural languages, operators are represented by function words. In the world, operators are realized in part by extrapropositional relations.

As an example of representing content by means of concatenated propositions, consider the robot in the process of automatically analyzing a situation in which it finds a triangle and a square in field A2 of its task environment. We assume that the concepts of *field*, *triangle*, *square*, and *contain* are realized in terms of programs which analyze the outlines of the bitmap representations reflecting the robot's visual input.

2.2. Example of two concatenated propositions.



The propositions in 2.2 may be paraphrased as *field contain triangle* and *field contain square*. They are connected by the extrapropositional relation *and*.

The feature structure of arguments (objects, nominals) specifies in its loc-feature the place at which the object was encountered. The feature structure of functors (relations, verbs) specifies in its loc-feature the time at which the relation was determined. The loc-features of modifiers (properties, adjectives) have values which agree with those of the structures modified.

3. Data structure of a word bank

The format of 2.2 represents the content's functor-argument structure by graphical means. These are well-suited for expressing certain intuitions, as shown by the widespread use of tree structures in linguistics. For implementing the database metaphor of communication, however, graphical representations are suboptimal. This is because databases code relations in an abstract manner, with related items in arbitrarily distant places (see for example Elmasri and Navathe [6]).

In order to support a simple and efficient procedure to embed propositional content in and extract it out of a database, the graphical format of 2.2 is recoded as a table of feature structures, whereby the functor-argument structure of elementary propositions (intrapropositional relations) and the concatenation between elementary propositions (extrapropositional relations) are expressed by means of attributes.[5] This new data structure, first presented in Hausser [15], is called a *word bank*.

As an example of a word bank consider 3.1.

3.1. Propositions 2.2 as a word bank.

TYPES	PROPLETS	
func: *contain* ARG: cnj: prn:	func: *contain* ARG: field triangle cnj: 23 and 24 prn: 23	func: *contain* ARG: field square cnj: 23 and 24 prn: 24
arg: *field* FUNC: id: prn:	arg: *field* FUNC: contain id: 7 prn: 23	arg: *field* FUNC: contain id: 7 prn: 24
arg: *square* FUNC: id: prn:	arg: *square* FUNC: contain id: 9 prn: 24	
arg: *triangle* FUNC: id: prn:	arg: *triangle* FUNC: contain id: 8 prn: 23	

The data are arranged as a column of types, such that each type is followed by an open number of tokens. A type is a feature structure in which the first attribute specifies the grammatical role and takes a concept as value while all its other attributes have the value NIL. The tokens, called *proplets*,[6] consist of completed types and serve as the elements of individual propositions. A row, consisting of a type followed by an open number of proplets, is called a *token line*.

Proplets belonging to the same proposition are held together by a common proposition number prn. The attributes specifying associated proplets in terms of intra- and extrapropositional relations are called *continuation predicates*.

[5] According to Mel'čuk [28, pp. 18, 19] the elements of the 'deep semantic representation' are unordered. This linguistic position opens the way to arranging the propositional elements (proplets) for another purpose, namely indexing and retrieval in a database.

[6] The term *proplet* is coined by analogy to *droplet* and refers to a basic part of an elementary proposition. Superficially, proplets may seem to resemble the feature structures of HPSG (see Pollard and Sag [33]). The latter, however, are intended to be part of a phrase structure tree rather than a database, do not code the functor-argument structure in terms of *bidirectional* pointers, do not concatenate propositions in terms of extrapropositional relations, and thus do not provide a basis suitable for time-linear navigation.

In intrapropositional relations, an argument (object, noun) specifies the associated functor, a functor (relation, verb) specifies the associated arguments, and a modifier (property, adjective) specifies the associated modified item, and vice versa. In extrapropositional relations, an argument may specify an identity [7] or non-identity relation with another argument, and a functor may specify a conjunctional relation (e.g., *and*, *then*, *because*) with another functor.

The token lines of a word bank correspond to the structure of a classic *network database*. A network database defines a 1:n relation between two kinds of records, the *owner* and the *member* records. In a word bank, the types function as the owner records and the associated proplets as the member records.

4. Three kinds of propositions

Database semantics distinguishes episodic, absolute, and language propositions. These are built from proplets with different feature structures, as illustrated below:

4.1. Three kinds of proplets.

EPISODIC PROPLET

arg:	*square*
FUNC:	contain
MODR:	small
id:	9
prn:	24

ABSOLUTE PROPLET

arg:	*square*
FUNC:	have
MODR:	
id:	y
prn:	abs-3
arg:	*square*

LANGUAGE PROPLET

sur:	Quadrat	
syn:	noun	
sem:	FUNC:	contain
	MODR:	small
	id:	9
	prn:	24
	arg:	*square*

Episodic proplets use concept tokens (cf. 1.1), while absolute and language proplets use concept types. Because the three kinds of proplets have different feature structures, the distinction between concept types and concept tokens may be left implicit in the attributes containing them, e.g., arg: *square*.

Language proplets have language-specific surfaces (here, German Quadrat). The different locations of the concepts in first, first and last, and last position in episodic, absolute, and language proplets, respectively, are motivated by their different roles in internal matching. These are illustrated below (cf. 4.5, 4.6, 4.7, 4.8, and 4.9).

Combining proplets into propositions is based on completion. Completion takes proplet types and turns them into tokens by assigning suitable values. The result is a(n unordered) set of proplets which are related to each other solely in terms of attributes. This is in contrast to the usual method of constructing propositions, based on substitution algorithms with corresponding graphical representations, such as trees in linguistics, symbol sequences in logic, or conceptual graphs in CG theory.

[7] Natural language uses the category noun for a wide variety of different 'objects', such as apples, events, and moments of time, but also speed, efficiency, and hope. For the latter, the identity relation may have to be weakened to an equivalence relation.

Assume, for example, that the robot's recognition has produced the concept tokens *contain, field, small,* and *square*. These are inserted into suitable proplet schemata, resulting in the following episodic proplet types:

4.2. Insertion of concepts into proplet schemata.

func: *contain*	arg: *field*	arg: *square*	modr: *small*
ARG:	FUNC:	FUNC:	MODD:
MODR:	MODR:	MODR:	id:
cnj:	id:	id:	prn:
prn:	prn:	prn:	

These episodic proplet types are turned into tokens and combined into a episodic proposition by assigning a common proposition number prn and reconstructing the functor-argument structure by means of copying the argument names into the ARG slot of the functor, the functor name into the FUNC slot of the arguments, etc. This establishes the intrapropositional relations. Extrapropositional relations are established by providing the cnj, and id slots with suitable values. Thus, 4.2 may be completed as follows:

4.3. Episodic proposition *Field contains small square*.

func: *contain*	arg: *field*	arg: *square*	modr: *small*
ARG: field square	FUNC: contain	FUNC: contain	MODD: square
MODR:	MODR:	MODR: small	id: 9
cnj: 23 and 24	id: 7	id: 9	prn: 24
prn: 24	prn: 24	prn: 24	

This proposition is activated by means of navigating from one proplet to the next based on their intrapropositional relations:

4.4. Activating an episodic proposition by navigation.

[epi: contain] ⇒ [epi: field] ⇒ [epi: square] ⇒ [epi: small] [8]

Such a navigation is driven by the LA-grammar defined in 7.3 (LA-MOTOR) and illustrated graphically in 5.1, 7.4, and 7.5.

Absolute propositions are like episodic propositions except that their proplets use different feature structures (cf. 4.1). In database semantics, absolute propositions are not limited to mathematical truth, but represent any kind of knowledge not bound to a particular event. For example, if the robot recognizes three or four times in sequence *Postman comes at 11 a.m.*, then these episodic propositions may be replaced by a corresponding absolute proposition:

[8] For simplicity, the proplets are represented as [epi: contain], [epi: field], etc., leaving the continuation predicates implicit.

4.5. Generalization from episodic propositions.

absolute proposition:	Postman	comes_at	11 a.m.
	⇑	⇑	⇑
episodic propositions:	4. Postman ⇒	comes_at ⇒	11 a.m.
	3. Postman ⇒	comes_at ⇒	11 a.m.
	2. Postman ⇒	comes_at ⇒	11 a.m.
	1. Postman ⇒	comes_at ⇒	11 a.m.

The generalization is based on matching absolute proplets with episodic ones. [9] This kind of matching is called complete because a whole episodic proposition is matched by a corresponding absolute one. Thereby, not only must the concept types of the absolute and the concept tokens of the episodic proplets be compatible, but also the feature structures of the proplets involved.

Absolute propositions function in episodic inferencing, i.e., inferencing from one episodic proposition to another, as in the following example:

4.6. Episodic inference.

2. [abs: salad] ⇒ [abs: is] ⇒ [abs: food]
⇑ ⇓
1. [epi: Peter] ⇒ [epi: eat] ⇒ [epi: salad] 3. [epi: Peter] ⇒ [epi: eat] ⇒ [epi: food]

The episodic proposition 1 and the absolute proposition 2 are the premises from which the episodic proposition 3 is derived as the conclusion. The inference is based on matching the concept token [epi: *salad*] of proposition 1 with the corresponding concept type [abs: *salad*] of proposition 2. The rule of inference, formalized in 16.5, constructs the conclusion systematically from the content of the two premises.

Absolute propositions function also in absolute inferencing, i.e., inferencing from one absolute proposition to another, as in the following example:

4.7. Absolute inference.

2. [abs: drink] ⇒ [abs: is] ⇒ [abs: liquid]
⇑ ⇓
1. [abs: milk] ⇒ [abs: is] ⇒ [abs: drink] ⇓
⇓
3. [abs: milk] ⇒ [abs: is] ⇒ [abs: liquid]

Here, the absolute propositions 1 and 2 are the premises, from which the absolute proposition 3 is derived as the conclusion. The inference rule, formalized in 16.9, is

[9] The power of the matching is strengthened by the fact that the compatibility between concept types and concept tokens is not limited to concepts of exactly the same kind. In this way, database semantics mirrors the widely held view, expressed for example by Lakoff and Johnson [23], that metaphor pervades cognition in general. Cf. Hausser [16, p. 92, Example 5.2.1].

based on matching the concept type [abs: *drink*] of the absolute proposition 1 with the corresponding concept type [abs: *drink*] of the absolute proposition 2.

The matching in episodic as well as absolute inferencing is called partial. Thereby the compatibility between the proplets involved is limited to their respective concepts.

Language propositions are like episodic and absolute ones in that they are defined as an unordered set of proplets, held together by a common proposition number and related by intra- and extrapropositional continuation predicates. Language propositions differ from episodic and absolute ones only in terms of their feature structures (cf. 4.1).

Natural language production (speaker mode) is activated by navigating through episodic or absolute propositions. Thereby, the proplets traversed are matched with the sem-features of corresponding language proplets.

4.8. Schema of language production.

language proposition:	blue	truck	belongs_to	milkman.
	⇑	⇑	⇑	⇑
episodic or absolute proposition:	blue ⇒	truck ⇒	belongs_to ⇒	milkman.

For communicating the propositional content, only the resulting sequence of surfaces is used. It must be adapted to the word order properties of the natural language in question, equipped with proper inflection for tense, number, and agreement, provided with function words, etc. (cf. LA-OUTPUT in Section 14).

Natural language interpretation (hearer mode) is based on supplying the incoming surfaces with language proplet types via lexical look-up. Syntactic composition is interpreted semantically by filling the attributes of the proplet types with appropriate values analogous to the transition from 4.2 to 4.3 (cf. LA-INPUT in Section 12).

4.9. Schema of language interpretation.

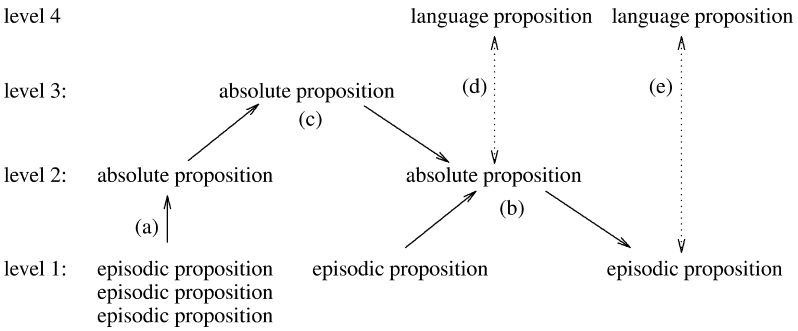
language proposition:	blue ⇒	truck ⇒	belongs_to ⇒	milkman.
	⇓	⇓	⇓	⇓
episodic or absolute proposition:	blue	truck	belongs_to	milkman.

For communication, only the sem-features of the resulting language proplet tokens are used. They are stored in the word bank using the concepts as the keys.

The procedures of generalization (cf. 4.5), episodic inference (cf. 4.6), absolute inference (cf. 4.7), language production (cf. 4.8), and language interpretation (cf. 4.9), as well as external and internal recognition and action are all based on a combination of navigation and matching. In recognition, the input drives the navigation via matching. In action, the navigation drives the output via matching. In inference, the combination of navigation and matching is more complicated.

Proplet matching is based on an upper and a lower level. The different kinds of matching may be combined, however, into the following four-level structure:

4.10. Matching relations in database semantics.

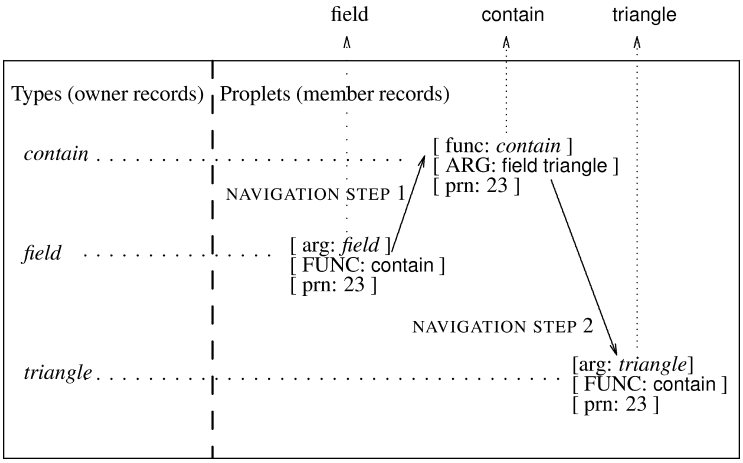


Episodic propositions appear at level 1, absolute propositions resulting from (a) generalization and used for (b) episodic inference at level 2, absolute propositions used for (c) absolute inference at level 3, and language propositions used for coding or decoding (d) absolute or (e) episodic propositions at level 4.

5. Communication between the speaker and the hearer

A word bank goes beyond a classical network database because it defines possible continuations within its record-based structure. These form the basis for a kind of operation which conventional databases do not provide, namely an autonomous time-linear navigation through the concatenated propositions of the database. A navigation through the word bank 3.1 may be represented graphically as follows:

5.1. Production of field contains triangle.

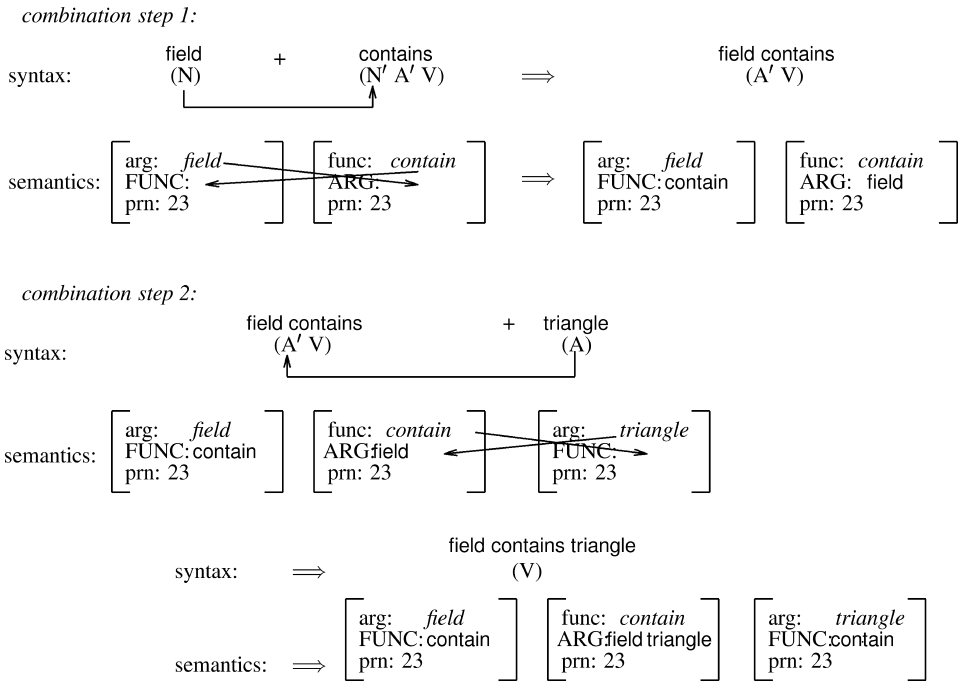


The navigation happens to begin at the proplet *field* with the proposition number (prn) 23. Based on the proplet's continuation predicate, i.e. [FUNC: contain], the navigation algorithm looks for the token line of contain and proceeds from the owner record to the member record with the prn 23. The 'next' proplet *contain* has the continuation predicate [ARG: field triangle]. The first value, field, confirms the previous navigation. The second value triangle is the new 'next' proplet. It is found by going to the token line of *triangle* and again proceeding from the owner record to the member record with the prn 23.

In this way, proposition 23 is traversed completely. Thereby, its content may be read out automatically by matching each episodic proplet traversed with a corresponding language proplet and realizing its surface value—as indicated in the top line of 5.1. Next there may be an extrapropositional navigation to another proposition, based either on an identity relation between arguments (objects, nominals) or a conjunctional relation between functors (relations, verbs).

The hearer receives the proplet surfaces field, contain, and triangle resulting from 5.1, and assigns suitable language proplets to them via lexical look-up. As shown below, the syntactic analysis consists in a time-linear canceling of valency positions by valency fillers. In the first combination step, the sentence start field cancels the nominative valency N′ of contain. In the second composition, the next word triangle cancels the accusative valency A′ in the sentence start field contains.

5.2. Interpretation of field contains triangle.

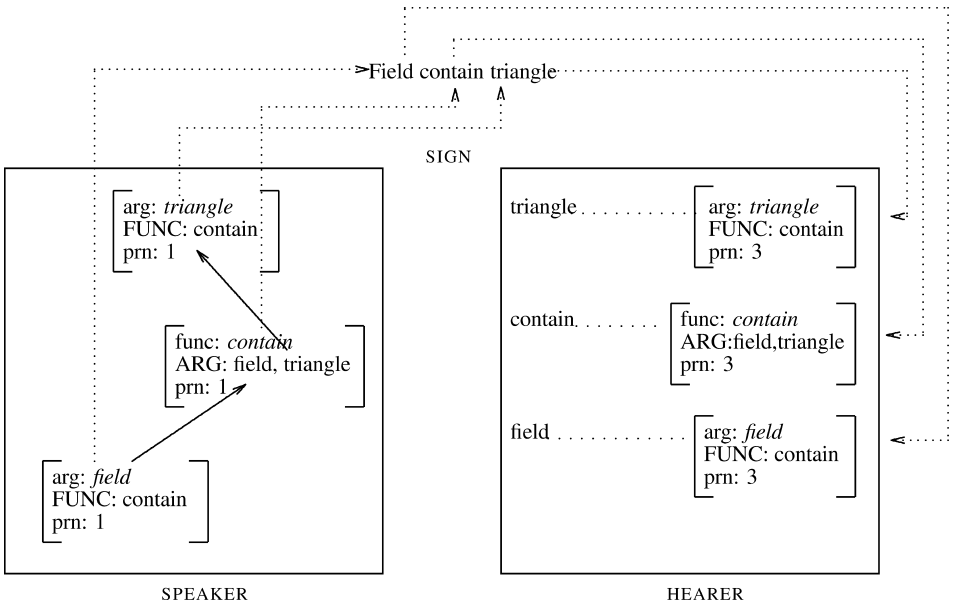


The semantic interpretation consists in copying concept names into the continuation features of other proplets (completion, cf. 4.2 and 4.3). [10] The interpretation of the first syntactic composition copies the concept name field into the ARG feature of *contain*, and the concept name contain into the FUNC feature of *field*. The interpretation of the second composition copies the concept name contain into the FUNC feature of *triangle* and the concept name triangle into the ARG feature of *contain*.

The result is a(n unordered) set of the three co-indexed proplets *field*, *contain*, and *triangle*. These autonomous entities jointly specify the functor-argument structure of their proposition in terms of their respective continuation predicates (bidirectional pointering). The indexing problem is solved automatically by storing each proplet at the end of its appropriate token line in the word bank.

The transfer of information from speaker to hearer may be illustrated as follows:

5.3. Transfer of information from the speaker to the hearer.



The speaker's navigation serves as the conceptualization and basic serialization of language production. The database content is coded into language by matching the contextual proplets traversed with language proplets and uttering their surface (cf. SIGN).

The hearer's interpretation consists in deriving a corresponding set of proplets. This procedure is based on

(i) assigning lexical proplet types to the sign's surfaces and

[10] For simplicity, the semantic parts of the language proplets in 5.2 have the feature structure of episodic proplets rather than the sem-feature of language proplets (cf. 4.1).

(ii) completing the types into tokens by copying proplet names into appropriate slots of other proplets during the syntactic-semantic analysis (as shown in 5.2).

In this way, the word meanings, the functor-argument structure, and the extrapropositional relations of the speaker's content are reconstructed by the hearer. Furthermore, the time-linear order of the sign induced by the speaker's navigation is eliminated, allowing storage of the proplets in accordance with the principles of the data structure in question. Apart from the hearer's prn, the result of the interpretation is exactly analogous to the original structure in the speaker's database.

6. Why is database semantics different?

There are at least three reasons why the coding of propositional content in terms of proplets is not equivalent to formalisms based on graphical representations. First, graphical representations like one-dimensional logical formulas and two-dimensional tree structures or conceptual graphs have restrictions which do not hold for the non-graphical representations of database semantics. Second, graphical representations are motivated by intuitions which database semantics has no cause to replicate. Third, the bidirectional pointering between the distributed proplets of database semantics offers descriptive possibilities not open to graphical means.

For example, most linguistic analyses are based on trees which are supposed to capture the intuitions of constituent structure. In the well-known case of discontinuous elements, however, these intuitions cannot be expressed by two-dimensional trees, leading to the constituent structure paradox.[11] The coding of concatenated propositions in terms of proplets is based on the related, but more basic intuitions concerning

(i) the functor-argument structure of elementary propositions and
(ii) the nature of their concatenation.

Thus, the constituent structure paradox is avoided in database semantics.

Similarly, analyses of propositions in predicate calculus use quantifiers binding variables to establish intrapropositional identity relations. Thereby different orders of quantifiers express different meanings. This commitment to one-dimensional order produces problems of

(i) creating artificial scope ambiguities and
(ii) failing to express readings intuitively required.

As an example of the second kind, consider the sentences

Every man who loves a woman is happy.
Every man who loves a woman loses her.

and the following attempt to represent their dominant reading in predicate calculus:

$\forall x\ [[\text{man}(x)\ \&\ \exists y\ [\text{woman}(y)\ \&\ \text{love}\ (x, y)]] \rightarrow [\text{happy}\ (x)]]$
$\forall x\ [[\text{man}(x)\ \&\ \exists y\ [\text{woman}(y)\ \&\ \text{love}\ (x, y)]] \rightarrow [\text{lose}\ (x, y)]]$

[11] Hausser [13, p. 24f.], [16, p.157f.].

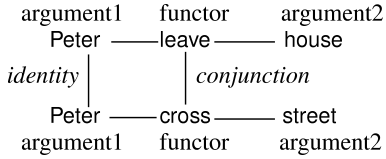
These parallel formulas are intended to reproduce the subordinate structure of the relative clause. The second formula, however, has the problem that the scope of the quantifier ∃y does not reach the y in lose (x, y). An alternative formula with the quantifier ∃y at the top level would solve the scope problem, but cannot be derived compositionally by translating the words of the sentence into suitable lambda expressions—as required in Montague grammar. This problem does not arise within database semantics, which asserts the identity between proplets both intra- and extrapropositionally by means of attributes rather than by means of quantifiers which bind variables.

The new descriptive possibilities offered by database semantics are based on a strict separation between the representation of content, on the one hand, and its activation, on the other. The representation of content is based on coding all intra- and extrapropositional relations between proplets in a distributed, bidirectional manner by means of attributes. This allows storage and retrieval of proplets in accordance with the principles of a suitable database, using the concepts as the primary key.

The activation of content is based on the principle of navigation. As illustrated in 5.1, the content's representation provides a railroad system for navigating through a proposition and from one proposition to the next. [12] This procedure is based on the retrieval mechanism of the database and does not depend in any way on where and how the proplets are stored.

As another example consider the propositions *leave Peter house* and *cross Peter street*, concatenated via the conjunction *then* and the identity between the respective names. The corresponding proplet connections may be represented as follows:

6.1. 'Railroad system' provided by two propositions.



The content expressed by this railroad system may be activated by a multitude of different navigation kinds. The most basic distinctions [13] are between intra- and extrapropositional forward and backward navigation, and between paratactic (coordinating) and hypotactic (subordinating) navigation, yielding realizations such as the following: [14]

[12] At the level of context, neither the representation of content nor its activation constitute a language—in contrast to Fodor's Mentalese, e.g., [8]. Instead, the contextual representation is like the wheels of a mechanical calculator, while the activation is like a calculation based on those wheels.

[13] For a more detailed analysis see Hausser [16, pp. 483–491].

[14] Sowa [40,41] may seem to come close when he presents a 'linear form' (LF) in addition to a graphical 'display form' (DF). However, his linear form is just a convenient ASCII recoding of the graphical form, and there is no distinction between content representation and activation, no navigation, and no motor algorithm.

6.2. Different realizations based on different navigations.

extrapropositional forward navigation:
Peter leaves the house. Then Peter crosses the street.

extrapropositional backward navigation:
Peter crosses the street. Before that Peter leaves the house.

intrapropositional backward navigation:
The street is crossed by Peter.

subordinating identity-based navigation:
Peter, who leaves the house, crosses the street.

subordinating conjunction-based forward navigation:
Peter, after leaving the house, crosses the street.

subordinating conjunction-based backward navigation:
Peter, before crossing the street, leaves the house.

These realizations in English reflect different navigations through the propositional content 6.1. The descriptive task of database semantics regarding 6.1 and 6.2 is twofold. One is the coding and decoding task: in the hearer mode, the different surfaces of 6.2 must be mapped onto the representation 6.1, and in the speaker mode, the representation 6.1 must be mapped onto one of the surfaces of 6.2. The other is the control task (cf. Section 8): the speaker must choose the navigation most appropriate to the rhetorical task at hand, and the hearer must interpret this choice by analyzing the resulting surface.

7. Motor algorithm for powering the navigation

The cognitive agent's word bank interacts with the internal and external environment by means of LA-grammars. An LA-grammar[15] is specified in terms of

(i) a lexicon LX,
(ii) a set of initial states ST_S,
(iii) a sequence of rules r_i of the form r_i: $cat_1\ cat_2 \Rightarrow cat_3\ rp_i$,
(iv) a set of final states ST_F.

A rule consists of the name r_i, the category patterns cat_1, cat_2, and cat_3, and the rule package rp_i. The category patterns define a categorial operation which maps a sentence start *ss* (matched by cat_1) and a next word *nw* (matched by cat_2) into a new sentence start *ss*′ (characterized by cat_3). The output of a successful application of rule r_i is a *state* defined as an ordered pair (*ss*′ rp_i).

[15] For an algebraic definition of the time-linear algorithm of LA-grammar as well as its language and complexity classes see Hausser [14].

As an example, consider the LA-grammar for the context-sensitive formal language

$a^{2^i} =_{def} \{a^i \mid i \text{ is a positive power of } 2\}$.

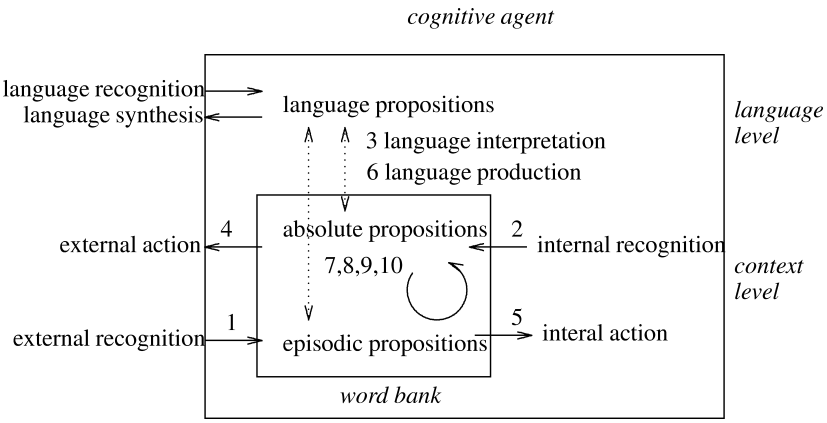
7.1. LA-grammar for context-sensitive a^{2^i}.

$LX =_{def} \{[a\ (a)]\}$
$ST_S =_{def} \{[(a)\ \{r_1\}]\}$
r_1: (a) (a) $\Rightarrow$ (aa) $\{r_2\}$
r_2: (aX) (a) $\Rightarrow$ (Xbb) $\{r_2, r_3\}$
r_3: (bX) (a) $\Rightarrow$ (Xaa) $\{r_2, r_3\}$
$ST_F =_{def} \{[(aa)\ rp_1], [(bXb)\ rp_2], [(aXa)\ rp_3]\}$.

This LA-grammar belongs to the lowest complexity class of the LA-grammar hierarchy and parses in linear time. [16]

The LA-grammars used by the cognitive agent are of three basic kinds:
(a) recognition,
(b) action, and
(c) inference.
They interact with the word bank as follows:

7.2. LA-grammars for recognition, action, and inference.



These LA-grammars are just like the one for a^{2^i} illustrated above, except that the patterns for cat_1, cat_2, and cat_3 are specified in terms of feature structures rather than lists.

The LA-grammars for recognition read (1) external, (2) internal, and (3) language propositions into the word bank. The LA-grammars for action realize certain propositions

[16] A comparison with corresponding PS-grammars for a^{2^i} illustrates the formal and conceptual simplicity of LA-grammar. Hopcroft and Ullman [19] present the canonical context-sensitive PS-grammar of a^{2^i} (p. 224) and a version as unrestricted PS-grammar (p. 220).

as (4) external, (5) internal, and (6) language actions. The inference LA-grammars are for (7) generalization as well as (8) episodic and (9) absolute inference.

These nine LA-grammars are fixed, stationary components—like engines mounted in a workshop. They start running if and when they receive input. In recognition, the input is provided by events in the external and internal environment. But what provides input for action and inference?

The answer is a tenth LA-grammar, called LA-MOTOR (cf. 10 in 7.2). It is special in that it is mobile—like an engine on wheels. This locomotive moves continuously through the word bank's propositional content, choosing continuations either at random (free association) or by following a highlighting of continuation proplets provided by the control structure described in Section 8. The location of LA-MOTOR in the word bank constitutes the cognitive agent's current focus of thought. [17]

Navigation by LA-MOTOR may be one-level or two-level. In one-level navigation, the proplets are simply traversed (cf. 4.4). In two-level navigation the episodic proplets traversed are matched by corresponding absolute (cf. 4.5) or language proplets (cf. 4.8 and 4.9). Depending on their content, these matching proplets are passed on to the stationary LA-grammars (4)–(6) for action and (7)–(9) for inference, activating them by providing them with input.

7.3. Definition of LA-MOTOR.

LX: episodic proplets in the word bank

ST_S: {([func: α] {1 F + A = F, 2 F + A = A})}

F + A = F:
$$\begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x\ \beta\ y \\ \text{prn: } m \end{bmatrix} \begin{bmatrix} \text{arg: } \beta \\ \text{FUNC: } \alpha \\ \text{prn: } m \end{bmatrix} \Rightarrow [\text{func: } \alpha]\ \{3\ \text{F} + \text{A} = \text{F},\ 4\ \text{F} + \text{A} = \text{A},\ 5\ \text{F} + \text{cnj} = \text{F}\}$$

F + A = A:
$$\begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x\ \beta\ y \\ \text{prn: } m \end{bmatrix} \begin{bmatrix} \text{arg: } \beta \\ \text{FUNC: } \alpha \\ \text{prn: } m \end{bmatrix} \Rightarrow [\text{arg: } \beta]\ \{6\ \text{A} + \text{id} = \text{F}\}$$

A + id = F:
$$\begin{bmatrix} \text{arg: } \alpha \\ \text{FUNC: } \beta \\ \text{id: } m \\ \text{prn: } k \end{bmatrix} \begin{bmatrix} \text{arg: } \alpha \\ \text{FUNC: } \gamma \\ \text{id: } m \\ \text{prn: } l \end{bmatrix} \Rightarrow \begin{bmatrix} \text{func: } \gamma \\ \text{ARG: } x\ a\ y \\ \text{prn: } l \end{bmatrix} \{7\ \text{F} + \text{A} = \text{F},\ 8\ \text{F} + \text{A} = \text{A}\}$$

F + cnj = F:
$$\begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x \\ \text{cnj: } m\ C\ n \\ \text{prn: } m \end{bmatrix} \begin{bmatrix} \text{func: } \beta \\ \text{ARG: } y \\ \text{cnj: } m\ C\ n \\ \text{prn: } n \end{bmatrix} \Rightarrow [\text{func: } \beta]\ \{9\ \text{F} + \text{A} = \text{F},\ 10\ \text{F} + \text{A} = \text{A}\}$$

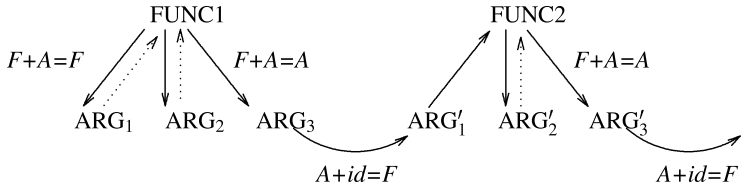
ST_F: {([func: x] $rp_{F+A=F}$)}

[17] The computational implementation treats LA-MOTOR as stationary as well. The navigation is simulated by feeding continuation proplets to LA-MOTOR, based on the retrieval mechanism at hand.

The rules of LA-MOTOR [18] specify patterns which are applied to contextual proplets. Thereby, the sentence start proplet provides the continuation predicate for moving to the next proplet.

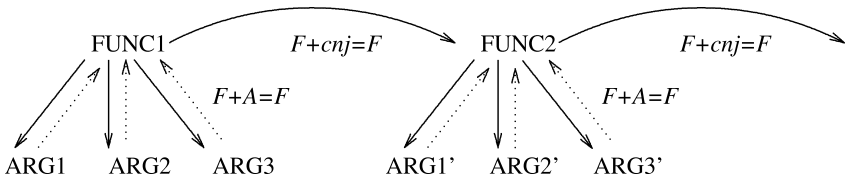
The rule F + A = F is for intrapropositional navigation; it moves from a functor proplet to an argument proplet and back to the functor, using the intrapropositional continuation predicate of the functor to find the argument. F + A = A initiates an extrapropositional navigation based on identity. A + id = F continues this extrapropositional navigation, as indicated by the following continuation structure:

7.4. Extrapropositional id**-navigation.**



The fourth rule F+cnj=F continues an intrapropositional navigation by moving to the next proposition based on the conjunction of the functors, as indicated below:

7.5. Extrapropositional cnj**-navigation.**



Detailed formal examples illustrating the navigation through a word bank by means of LA-MOTOR are presented in 13.1–13.3 below.

8. Autonomous control structure

As an algorithm, LA-MOTOR executes legal transitions from one episodic proplet to the next. However, given that each proplet usually provides several possible continuations, there is the question of how to guide the navigation.

The first obvious step is to connect LA-MOTOR to the cognitive agent's recognition procedures. For example, when an episodic proposition is read into the word bank by means of LA-grammars for external, internal, or language recognition, the new proplets are highlighted, causing LA-MOTOR to jump to the first proplet and to follow the incoming

[18] For simplicity, the treatment of modifiers is omitted in LA-MOTOR.

propositions from there. LA-MOTOR's tracing of incoming propositions is important because it positions the focus point in preparation for subsequent action.

In order to guide LA-MOTOR's navigation from recognition to action, the concatenated propositions in a word bank are individuated into recognition-action-recognition (rac) sequences. [19] These store past experiences relating to, for example, feeling hungry. When internal recognition reads *CA feels hungry* into the cognitive agent's (CA) word bank once more, this proposition is used to retrieve and highlight all earlier rac sequences beginning with it:

8.1. Recognition highlighting associated rac sequences.

CA feels hungry
|
CA feels hungry—CA searches at place X—CA finds no food
|
CA feels hungry—CA searches at place Y—CA finds a little food
|
CA feels hungry—CA searches at place Z—CA finds lots of food

The retrieval of rac sequences beginning with *CA feels hungry* is based on the word bank's data structure, using the concepts as the key. LA-MOTOR navigates through these rac sequences, jumping throughout the word bank from proplet to proplet by following their continuation predicates, ignoring those which are not highlighted. [20]

The question raised by this simplified example is: within which rac sequence's action proposition should LA-MOTOR switch from one-level to two-level navigation, thus activating a suitable LA-grammar for action? The solution requires that the actions initiated by LA-MOTOR have a purpose for the cognitive agent.

For the implementation of purpose it is useful to distinguish between long and short term purposes. It seems that short term purpose is guided by a need for balance in the sense of keeping various parameters of the cognitive agent within normal range. In contrast, long term purpose like accumulating love, money, offspring, etc., seems to guided by a drive to maximize—at least at a primitive level.

In database semantics, short term purposes are activated by need parameters. Examples are energy supply and temperature. The current value of the former may leave normal range because of internal consumption, of the latter because of external changes. Other short term need parameters are rest, sleep, fear, but also shelter, social acceptance, intellectual stimulation, etc.

According to the balance principle, CA strives to maintain the values of the need parameters within normal range. For this, rac sequences originating as a real action are

[19] A related approach based on transition functions within a possible world semantics is presented in Son and Baral [39]. Our notion of a rac sequence is reminiscent of Skinner's [38] 'stimulus–response–consequence', though he would object to our representation of content in terms of concatenated propositions.

[20] In memory, the propositions of a rac sequence are naturally connected by subsequent propositions numbers, e.g., 54, 55, 56, reflecting the temporal order of their occurrence (cf. Hausser [18]). This allows their traversal by means of LA-MOTOR based on the cjn-values of the respective functor proplets.

evaluated as to whether they result in raising or lowering any of the need parameters, or leaving them unchanged. The evaluation is expressed by means of need vectors, pointing up or down at various degrees, whereby → expresses 'no change'.

When a stored rac sequence is activated later by an incoming recognition, its need vector is related to the current parameter values. If one of the parameters is out of normal range, the rac sequence with a need vector most likely to regain balance for that parameter is highlighted most. In this way, LA-MOTOR is enticed to

(i) choose the rac sequence most appropriate to regain balance, and
(ii) to switch to two-level navigation when traversing the action proposition of that rac sequence.

For example, if CA's need parameter for energy supply has dropped a little out of range, the proposition *CA feels hungry* is read into the word bank. Because the state of the energy parameter corresponds best to the second rac sequence in 8.1, it would be highlighted most, causing CA to search at place Y. The first and the third rac sequence in 8.1 would come into play only if CA's search at place Y is unsuccessful.

Similarly, if CA's need parameter for temperature has risen out of range a lot, *CA feels hot* is read into the word bank. This results in highlighting rac sequences beginning with that proposition, e.g.,

(1) *CA feels hot—CA moves into the shade—CA cools down a little*,
(2) *CA feels hot—CA moves into the basement—CA cools down a lot*, and
(3) *CA feels hot—CA moves into the sun—CA feels hotter*.

Here the need vector of the second rac sequence is best suited to regain balance, resulting in the corresponding action.

But what about situations in which there are no rac sequences in the word bank to be activated by the current recognition? In this case, predefined ('innate') open rac sequences provide options for action. Consider the following example:

8.2. Open rac sequences for unknown situations.

CA meets unknown animal
|
CA meets unknown animal—CA approaches—?
|
CA meets unknown animal—CA waits—?
|
CA meets unknown animal—CA flees—?

These rac sequences are open because due to lack of experience the result proposition is represented as unknown ('?'). However, they have predefined need vectors relating to the fear parameter, the first being low, the second medium, and the third high.

Appearance of the unknown animal as small and cuddly will lower the fear parameter, resulting in agreement with the first rac sequence. Appearance as big and ferocious, on the other hand, will increase the fear parameter, resulting in agreement with the third rac sequence. If the animal's appearance is uncertain, CA chooses the second rac sequence. Provided that CA survives the encounter, a new complete rac sequence

evaluating CA's action towards this particular animal will be stored in CA's word bank.

An indirect way to mediate between known (cf. 8.1) and unknown (cf. 8.2) situations is inference. When there is no ready-made rac sequence to respond to a given situation, the control structure entices LA-MOTOR to switch to two-level navigation and pass the resulting proplet sequences to the LA-grammars for generalization (cf. 4.5), episodic inference (cf. 4.6), and absolute inference (cf. 4.7). In this way, a rac sequence relevant for the situation at hand may be deduced, even though the rac sequence's first proposition does not literally match the proposition representing the situation. Furthermore, success or failure of the inference will be stored for future reference as a rac sequence.

Finally, there is language production (cf. 4.8), which is also activated by LA-MOTOR switching to two-level navigation, though the proplets used to match the path traversed are language rather than absolute proplets. As in non-verbal action, the control of language production is based on rac sequences and need parameters. The main difference from the nonverbal examples described above is that the need parameters for language interaction are of a social rather than a physiological nature.

As an example, consider the social rac sequences

(1) *CA meets colleague in the morning—CA greets colleague—Colleague greets CA* and

(2) *CA meets colleague in the morning—CA looks the other way—Colleague doesn't greet CA*.

The need vectors of these rac sequences relate to the need parameter of social acceptance, whereby the first need vector increases the current value of that parameter, while the second one does the opposite.

9. Contextual cognition as the basis of coherence

The schematic presentations of database semantics in 1.1 and 7.2 distinguish between the levels of language and context (see also 10.1). In the course of evolution, contextual cognition came first and comprised non-language-based recognition and action, both external and internal. The level of language evolved later as a specialization of contextual recognition and action.

For computational modeling, however, the level of context seems to be much harder than the level of language. This is because today's standard computers are basically designed for the language channel, handling input and output in terms of the keyboard and the screen, respectively. An adequate treatment of the context level goes beyond the capabilities of standard computers, requiring the hardware basis of robots instead. Unfortunately, modeling cognitive agents as robots is still in its infancy.

To avoid the difficulties of using robots, one might consider simulating contextual cognition. For this, the methods of cognitive modeling are available. Cognitive modeling creates animated agents in virtual environments on standard computers. The user can control movement through an artificial world (automated cinematography, Funge et al. [9])

and communicate with artificial agents which interact with their virtual task environment (Rickel and Johnson [36], Loyall and Bates [25]).

Upon closer inspection one realizes, however, that cognitive modeling and database semantics pursue opposite goals: cognitive modeling designs artificial worlds containing artificial agents for the interaction with natural agents (i.e., the users), while database semantics designs artificial agents for the interaction with the natural world containing natural agents. Therefore, a message dispatcher or blackboard architecture is likely to be a better control structure for cognitive modeling than word bank navigation. Conversely, reliance on virtual reality is problematic for database semantics in the following respects:

First, it is a misdirection of effort: rather than grounding coherence in real interaction with the external world, the programming work is directed mostly toward simulating convincing 3-D task environments with animated agents. Second, an artificial model of the world is not credible as a procedural foundation of word semantics. Third, the simulation of language-based reference is of little help for arriving at real reference.

There are both practical and theoretical reasons why the cognitive agent's interaction with the real world in terms of artificial recognition and action should be neither simulated nor avoided. Regarding practical applications, an absence of such real interaction makes it impossible for the resulting system to translate natural language commands into actions or to report observations in natural language. Regarding theoretical development, such an absence deprives the system of two essential foundations: One is a procedural definition of elementary concepts, needed for the autonomous build-up of contextual content and as the literal meanings of language (see 1.1). The other is the grounding of cognitive coherence in contextual content.

A propositional content such as *field contain triangle* may be read into the word bank either directly via immediate perception, or indirectly via media such as language, film, etc. When content is read in directly by a well-functioning system, then the content's coherence follows from the coherence of the external world, i.e., the temporal and spatial sequencing of events, the part-whole relations, etc.

Content stored on media, in contrast, allows for the possibility that an author has reordered and reconnected elements familiar from immediate recognition and action. This is why media-based content may be incoherent.

For example, a swimmer standing at the pool side, diving into the water, and disappearing with a splash is coherent. In contrast, a pair of feet appearing in the foaming water with a swimmer flying feet first into the air and landing at the pool side would be incoherent—unless it is specified in addition that the content is represented by means of a medium, e.g., a backward running movie.

Correspondingly, for people to be conversing in English is coherent. In contrast, a deer conversing with a skunk in English would be incoherent—unless it is specified that the content is represented by means of a medium, e.g., a piece of fictional language.

Adequate behavior of an artificial cognitive agent, including coherent language production, presupposes the autonomous build-up of coherent knowledge. For this the distinction between immediate and mediated contexts is crucial.

Founding coherence on direct recognition and action requires the hardware basis of robots rather than standard computers. Because the latter are inherently limited to mediated contexts, it is the user who is responsible for the content's coherence—and not the system. This must be kept clearly in mind when the hardware basis of standard computers is chosen for practical reasons—as in the following fragment.

10. Procedurally-oriented variant of a logical 'fragment'

Cognitive modeling and database semantics have in common that they are computational. Model-theoretic semantics, in contrast, is based on metalanguage definitions mostly unsuitable for procedural implementation. [21]

Model-theoretic semantics and cognitive modeling have in common that they operate with artificial models of the world. Database semantics, in contrast, presupposes the natural world and relies on procedural interactions with it.

Database semantics and model-theoretic semantics have in common that they provide explicit formal definitions for an overall system. In model theory, this approach was pioneered by Montague [30], who defined 'fragments' of natural language.

A fragment in this sense is a natural language interpretation system which is complete in terms of its function or functions, but can handle only a subset of the natural language in question. Once its functioning has been explicitly described in principle, a fragment is expected to be gradually expanded [22] until it covers the whole language.

Montague's method, called Montague grammar, repairs a defect of earlier (as well as later) logical semantics for natural language: instead of presenting isolated examples like All ravens are black and relating them intuitively to logical formulas, e.g.,

$$\forall x\,[\mathrm{raven}(x) \rightarrow \mathrm{black}(x)],$$

Montague defined the syntactic analysis and semantic interpretation in complete formal detail. This has the methodological advantage of stating hypotheses explicitly.

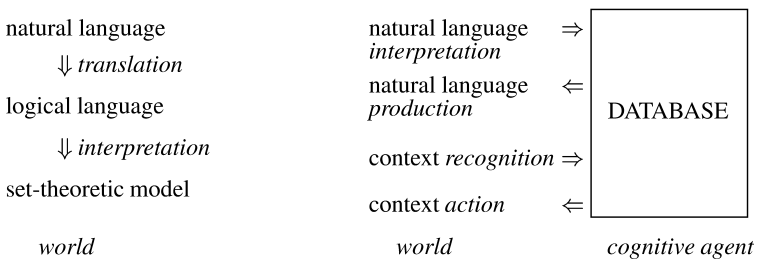
The defect of logical semantics repaired by Montague has a counterpart in procedural semantics, namely systems which are not implemented and/or implemented systems which have no declarative specification. As a procedurally-oriented equivalent to Montague's metalanguage-based 'fragments', the remainder of this paper presents explicit LA-grammars for recognition, action, and inference, which—together with a lexicon and LA-MOTOR—constitute a declarative description of a SLIM machine.

The SLIM machine's declarative specification differs from Montague's fragments not only because logical semantics is metalanguage-based while database semantics is procedural. In addition, there is a deep-seated ontological difference between the two approaches, which may be characterized schematically as follows:

[21] Examples are infinite sets of possible worlds and extensional definitions of basic concepts like *red*.

[22] For an overview of such—more or less rigorous—extensions see Lappin [24].

10.1 Montague grammar versus database semantics.



In Montague grammar, there is no speaker producing language in order to communicate cognitive content and no hearer interpreting language in order to store such content. Instead, natural language expressions and the set-theoretic model are both treated as part of the 'world'—which comprises past, present, and future actual and possible states. Database semantics, in contrast, establishes the relation between language and the real world by means of cognitive procedures inside the speaker-hearer (cf. 1.1).

The fragment presented in Chapter 8 of Montague [30], also known as PTQ, consists of the following components.

10.2. Components of Montague's PTQ fragment.

(1) A recursive definition of syntactic categories, and their intuitive interpretation for a finite subset.
(2) A lexicon, called a set of basic expressions, using these categories.
(3) A categorial grammar for the syntactic analysis of the sentences of the fragment.
(4) An intensional logic with a model-theoretic interpretation.
(5) A set of translation rules which map syntactically analyzed sentences to logical formulas with corresponding truth conditions.
(6) A set of meaning postulates.

In comparison, a SLIM machine consists of the following components:

10.3. Components of a SLIM machine.

(A) DBL-LEX: a lexicon for episodic, absolute, and language proplet types.
(B) WORD-BANK: a record-based network database containing proplet tokens.
(1) LA-RCN-E: an LA-grammar for external recognition.*
(2) LA-RCN-I: an LA-grammar for internal recognition.*
(3) LA-RCN-L: an LA-grammar for language recognition (interpretation).
(4) LA-ACN-E: an LA-grammar for external action.*
(5) LA-ACN-I: an LA-grammar for internal action.*
(6) LA-ACN-L: an LA-grammar for language action (production).

(7) LA-INF-G: an LA-grammar for generalization.*
(8) LA-INF-E: an LA-grammar for episodic inference.
(9) LA-INF-A: an LA-grammar for absolute inference.*
(10) LA-MOTOR: an LA-grammar for navigating through the concatenated propositions of the word bank.

The numbers of the LA-grammars in 10.3 correspond to those in 7.2. Based on recognition, the LA-RCN grammars read propositional content into the word bank. Based on two-level navigation, the LA-ACN grammars read propositional content out of the word bank, realizing it as action. The LA-INF grammars are for inferences, deriving new propositions from existing ones.

In addition, a complete fragment requires preconceptual processing for the LA-RCN grammars (e.g., speech recognition) and postconceptual processing for the LA-ACN grammars (e.g., speech synthesis). Also, LA-MOTOR requires a control structure correlating need parameters with the need vectors of rac-sequences (cf. Section 8), thus guiding the navigation.

For practical reasons (cf. Section 9), the following system is designed for standard computers. As a consequence, the LA-RCN and LA-ACN grammars are each represented here by only one kind, namely LA-RCN-L and LA-ACN-L. Furthermore, the LA-INF grammars are represented by an instantiation of LA-INF-E.

The design of this subsystem, called DBL-fragment, is analogous to designing, for example, a fragment of Montague grammar. Just as Montague's fragments are developed to model certain truth-conditional aspects of natural language meaning, DBL-fragment is designed to model the hearer's automatic interpretation and the speaker's automatic production of natural language. Just as Montague's fragments are interpreted with respect to a set-theoretically defined model, DBL-fragment is interpreted with respect to a word bank containing bidirectionally pointered proplets.

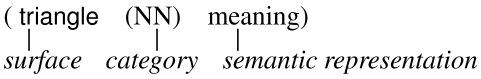
Like a model of logical semantics, a word bank is language independent. However, while a logical model is a set-theoretic representation of the world, a word bank represents the cognitive content of an individual agent. Furthermore, while logical semantics defines elementary meanings like *red*, *triangle*, and *contain* in the metalanguage by assigning extensions, database semantics constructs these meanings as elementary concepts based on the recognition and action procedures of the cognitive agent.

11. DBL-LEX: episodic, absolute, and language proplets

In DBL-fragment, episodic, absolute, and language proplets are provided by a lexicon, called DBL-LEX. The language proplets of DBL-LEX represent word forms which are conceptually analyzed as ordered triples, consisting of

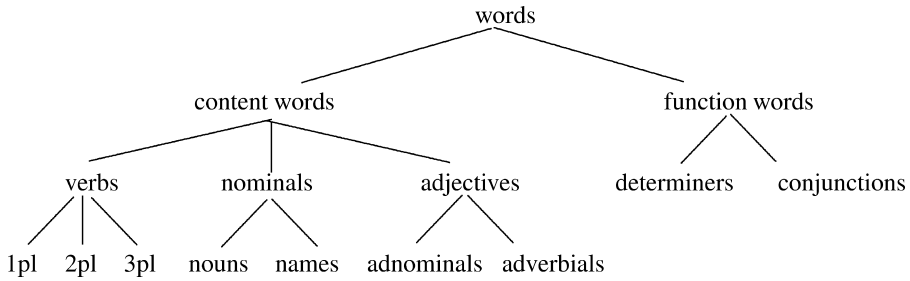
(i) the surface,
(ii) the category, and
(iii) the semantic representation.

11.1 Basic structure of a word form.



The words of DBL-LEX are divided into the following classes:

11.2. Hierarchy of word classes (parts of speech).



DBL-LEX represents the ordered triple analysis of word forms in terms of the following language proplet schemata for the different parts of speech (see also 4.1):

11.3. Lexical frames for the parts of speech.

nominals: *nouns*

$$\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{NN}) \\ \text{sem: } \begin{bmatrix} \text{FUNC:} \\ \text{modr:} \\ \text{id:} \\ \text{prn:} \\ \text{arg: } \alpha \end{bmatrix} \end{bmatrix}$$

names

$$\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{NM}) \\ \text{sem: } \begin{bmatrix} \text{FUNC:} \\ \text{modr:} \\ \text{id: } +1 \\ \text{prn: } = \\ \text{arg: } \alpha \end{bmatrix} \end{bmatrix}$$

verbs: *1 place*

$$\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{N}'\text{V}) \\ \text{sem: } \begin{bmatrix} \text{ARG:} \\ \text{modr:} \\ \text{cnj:} \\ \text{prn: } = \\ \text{func: } \alpha \end{bmatrix} \end{bmatrix}$$

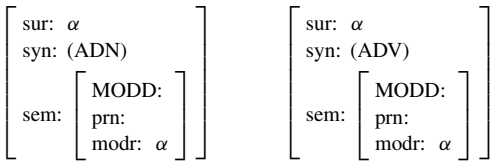
2 place

$$\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{N}'\text{A}'\text{V}) \\ \text{sem: } \begin{bmatrix} \text{ARG:} \\ \text{modr:} \\ \text{cnj:} \\ \text{prn: } = \\ \text{func: } \alpha \end{bmatrix} \end{bmatrix}$$

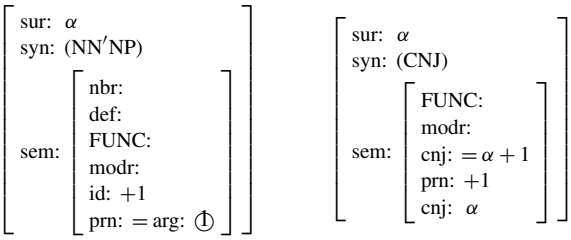
3 place

$$\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{N}'\text{D}'\text{A}'\text{V}) \\ \text{sem: } \begin{bmatrix} \text{ARG:} \\ \text{modr:} \\ \text{cnj:} \\ \text{prn: } = \\ \text{func: } \alpha \end{bmatrix} \end{bmatrix}$$

adjectives: *adnominals* *adverbials*



function words: *determiners* *conjunctions*



These feature structures are turned into lexical items by replacing the variable α by suitable concept names (cf. LX[23] in 12.4). We assume implicitly that the α in the sur-feature represents a language-specific surface, whereas the α in the sem-feature represents a corresponding universal concept.[24]

In addition, DBL-LEX provides schemata for episodic and absolute proplets. As in 11.3, these are turned into proplet types by replacing their concept variable by suitable concepts (see 4.2 for examples).

12. LA-INPUT: interpreting English

In DBL-fragment, incoming language signs are analyzed by an LA-RCN-L grammar. It is called LA-INPUT, defined in 12.4, and designed to handle the following sentences:

12.1. Sample sequence of propositions for DBL-fragment.

1. Peter leaves the house. 2. Peter crosses the street. 3. Peter enters a restaurant. 4. Peter orders a salad. 5. Peter eats the salad. 6. Peter pays for the salad. 7. Peter leaves the restaurant. 8. Peter crosses the street. 9. Peter enters the house.

The rules of LA-INPUT have a syntactic and a semantic part. The syntactic part is for canceling valency positions with suitable fillers (cf. 5.2). The semantic part specifies which

[23] For a full treatment of the lexical basis of agreement in English see Hausser [16, p. 331f.].

[24] This is required for the matching of lexical items and contextual proplets during language production (cf. 14.1). Relating language-specific words to universal concepts raises the problem of 'structural divergence,' familiar from machine translation. Solutions as proposed in Nasr et al. [31], Dorr [4], and Mel'čuk and Polguère [29] are applicable in database semantics, but go beyond the scope of this simple fragment.

attribute value of one proplet is to be copied into which attribute of another (cf. 4.2, 4.3, and 5.2).

12.2. Schema of a semantically interpreted LA-rule.

rule:
syn: ⟨ss-pattern⟩ ⟨nw-pattern⟩ ⇒ ⟨ss′-pattern⟩
sem: semantic copying operations

input: output:

$$\begin{bmatrix} \text{sur:} \\ \text{syn: } \langle a \rangle \\ \text{sem: } b \end{bmatrix}_1 \cdots \begin{bmatrix} \text{sur: } m \\ \text{syn: } \langle c \rangle \\ \text{sem: } d \end{bmatrix}_i + \begin{bmatrix} \text{sur: } n \\ \text{syn: } \langle e \rangle \\ \text{sem: } f \end{bmatrix}_{i+1} \Rightarrow \begin{bmatrix} \text{sur:} \\ \text{syn: } \langle a \rangle \\ \text{sem: } b \end{bmatrix}_1 \cdots \begin{bmatrix} \text{sur: } m+n \\ \text{syn: } \langle g \rangle \\ \text{sem: } h \end{bmatrix}_{i+1}$$

While an uninterpreted syntax (e.g., 7.1) operates with a sentence start represented by a single expression consisting of a surface and a category, the semantic interpretation operates with a *set* of feature structures with attributes for surface, category, and proplet. In this set, there is exactly one feature structure with a non-NIL surface value. This feature structure (here with the attribute [sur: *m*]) is called *syntactically active*. Its category represents the sentence start syntactically, while all other feature structures in the set are ignored by the syntactic part of a rule.

In the beginning of an LA-grammatical derivation, the sentence start consists of a one-element set containing the first word. This item is the result of lexical look-up, for which reason it has a non-NIL sur value. Subsequently, the syntactically active item of the sentence start is determined by the syntactic operations of the rules.

The semantic interpretation of an LA-grammar is based on the following operations, which are specified as clauses added to the syntactic rules:

12.3 Copying operations of semantic interpretation.

(1) copy_{ss}: include the items of the *sentence start* in the result.
(2) copy_{nw}: include the *next word* item in the result.
(3) $n_1.x \xrightarrow{a} n_2.y$: copy the values of the source feature x in n_1 *additively* into the goal feature y in n_2.
(4) $n_1.x \xrightarrow{e} n_2.y$: copy the values of the source feature x in n_1 *exclusively* into the goal feature y in n_2, whereby the original value of y must be NIL.
(5) $n_1.x \xrightarrow{r} n_2.①$: *replace* all occurrences of the variable ① in n_2 simultaneously with the value of the source feature x in n_1.

In exclusive and additive copying, values of the feature structure sem are restricted to target attributes of items which have the same proposition number as the source item(s). Copying values into target attributes of items with another proposition number is allowed only in the specification of extrapropositional relations. The variable ① originates lexically in determiners (cf. 11.3).

12.4. Definition of LA-INPUT.

LX =$_{def}$ {[Peter (NM) *],
[house (NN) *], [street (NN) *], [restaurant (NN) *], [salad (NN) *],

[the (NN′ NP) *], [a (NN′ NP) *],
[leave (N′ A′ V) *], [cross (N′ A′ V) *], [enter (N′ A′ V) *],
[order (N′ A′ V) *], [eat (N′ A′ V) *], [pay (N′ A′ V) *]

Variable definition:
np' ∈ {N′, D′, A′} (nominal valency positions)
np ∈ {NP, NM} (nominal valency fillers)
n' ∈ {NN′} (noun valency positions)
n ∈ {NN} (noun valency fillers)
x, y = .?.?.?.? (arbitrary sequence up to length 4)

$ST_S =_{def}$ { [(x) {DET + N, NOM + FV}] }

DET + N:	(n' x) (n)	⇒	(x) {NOM + FV, FV + MAIN} nw.arg —[r]→ ss.①, $copy_{ss}$
NOM + FV:	(np) (np' x V)	⇒	(x V) {FV + MAIN, AUX + NFV} nw.func —[e]→ ss.FUNC ss.arg —[a]→ nw.ARG $copy_{ss}$ $copy_{nw}$
FV + MAIN:	(np' x V) (y np)	⇒	(y x V) {DET + N, FV + MAIN} nw.arg —[a]→ ss.ARG ss.func —[e]→ nw.FUNC $copy_{ss}$ $copy_{nw}$

$ST_F =_{def}$ { [(V) rp_{nom+fv}], [(V) $rp_{fv+main}$], [(V) rp_{det+n}] }

For simplicity, the lexical entries are presented in the ordered triple notation. It is equivalent to the language proplets defined in 11.3, whereby the '*' should be replaced by suitable sem-features, filling the α by appropriate concept types.

Each rule contains the rule name, e.g., DET + N, the input pattern of the sentence start, e.g. (n' x), the input pattern of the next word, e.g. (n), the output pattern, e.g. (x), and the rule package, e.g. {NOM + FV, FV + MAIN}, followed by the semantic operations of the rule. Consider the operations of LA-INPUT in the analysis of the first sentence of the sample sequence 12.1.

12.5. Applying NOM + FV to *Peter* + *leaves*.

syn: ⟨np⟩ ⟨np′ x V⟩ ⟹ ⟨x V⟩
sem:
nw.func —[e]→ ss.FUNC
ss.arg —[a]→ nw.ARG
$copy_{ss}$ $copy_{nw}$

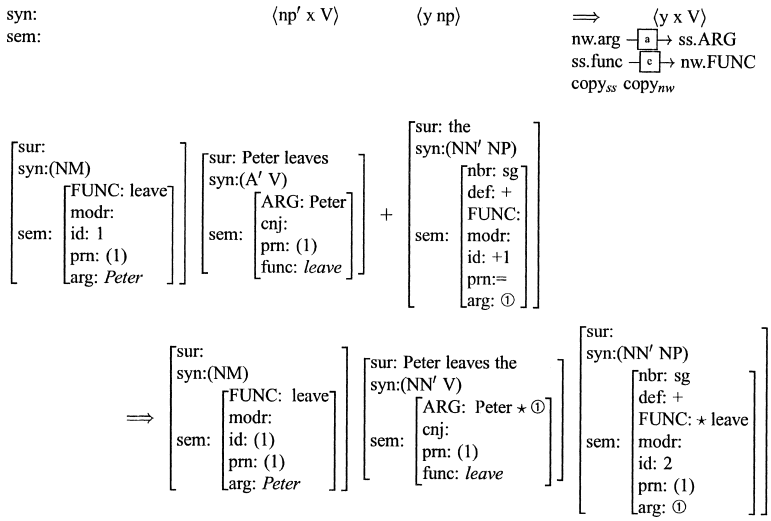
sur: Peter syn:(NM) sem: FUNC: modr: id: 1 prn: (1) arg: *Peter*	+	sur: leaves syn:(N′ A′ V) sem: ARG: cnj: prn:= func: *leave*	⟹	sur: syn:(NM) sem: FUNC: ⋆leave modr: id: 1 prn: (1) arg: *Peter*		sur: Peter leaves syn:(A′ V) sem: ARG: ⋆Peter cnj: prn: (1) func: *leave*	

The syntactic category patterns of the rule, i.e., (np) and (np′ x V), are matched onto the corresponding attributes of the input, i.e., (NM) and (N′ A′ V), respectively. The semantic operations complete both input proplets and include the result in the output,

with ⋆ indicating where the input proplets have been modified. The initial id and prn numbers are assigned by a simple grammatical control structure presumed here.

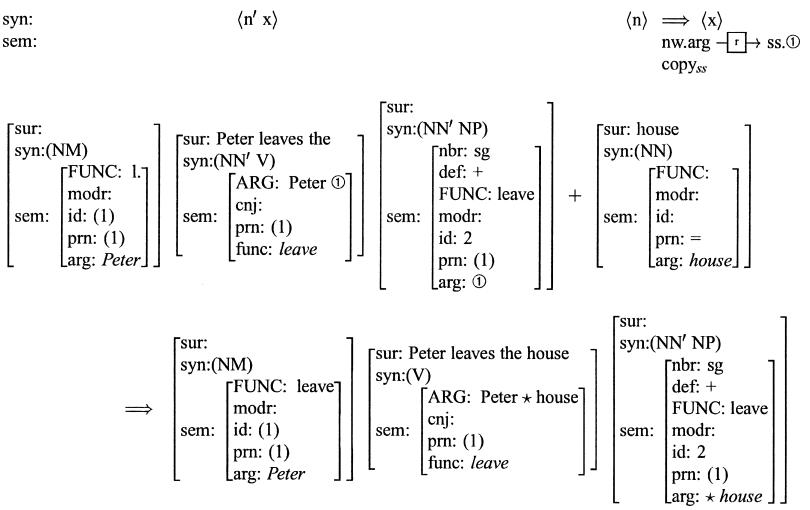
The output of 12.5 and the lexical analysis of the next word the serve as input to the rule FV + MAIN:

12.6. Applying FV + MAIN to *Peter leaves + the*.



The lexical analysis of the determiner assigns the variable ① as the value of the arg-attribute. This value is copied as an additional valency filler into the ARG-attribute of *leave*. The variable will be replaced by a noun in the next composition.

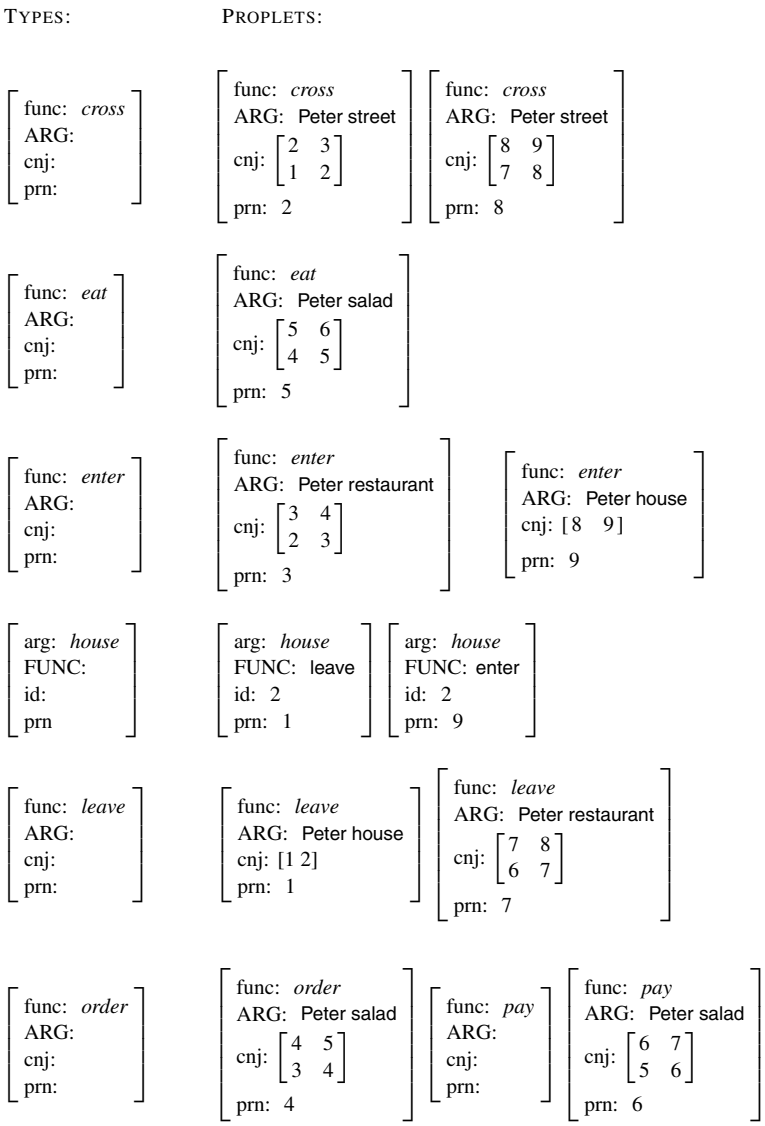
12.7. Applying DET + N to *Peter leaves the + house*.

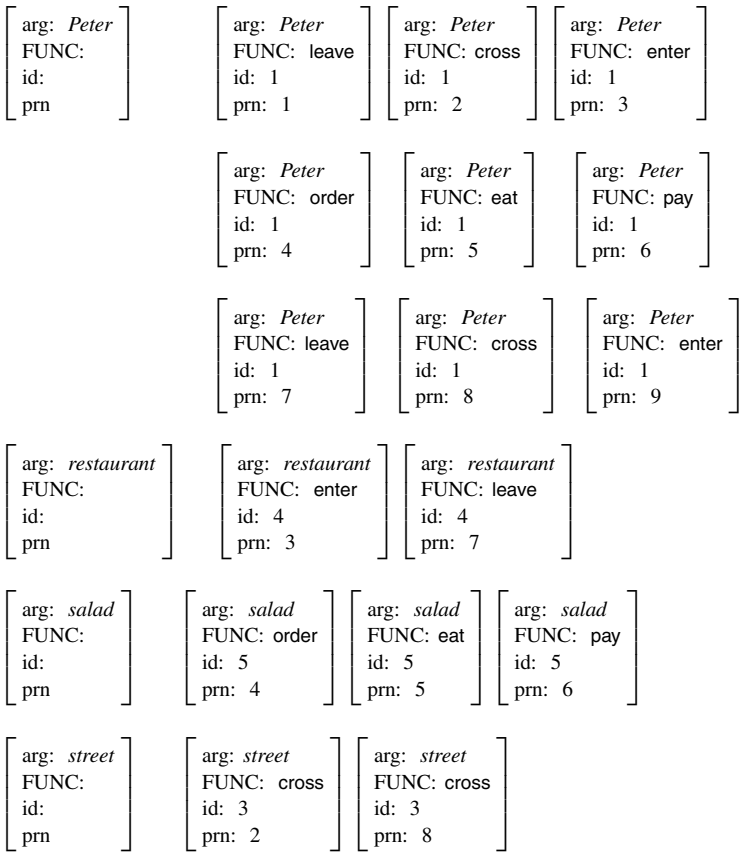


The transition to the next sentence is handled in DBL-fragment by the grammatical control structure. It increases the proposition number from 1 to 2, starts the analysis of the next sentence, and inserts the value [1 2] into the cnj-attribute of the verbal items. For a complex derivation involving a subordinate clause see Hausser [16, pp. 454–466].

When the sentences in the sample sequence 12.1 are analyzed syntactically and semantically in the manner indicated above, the completed sem-features are matched by episodic proplets (cf. 4.9) which are sorted into the word bank:

12.8. Sorting proplets of example 12.1 into word bank.





This representation of propositional content exhibits two kinds of extrapropositional relations:

One is the sequencing relation coded in the cnj attribute of functor items (verbs). The beginning and the end of a sequence are indicated by single values, e.g., [1 2] in the first leave and [8 9] in the second enter proplet of 12.8. The remaining functor items all have two values in the cnj attribute, one for the preceding and one for the following proposition (bidirectional pointering).

The other is the identity relation between arguments (nominals). If a name or a definite noun phrase is repeated, or an indefinite noun phrase is followed by a definite one using the same noun, the grammatical control structure assigns the same id number as a default. Thus coreference is asserted between related arguments, both within an elementary proposition and from one elementary proposition to the next. [25]

[25] In contrast to quantifiers in logic, which are an intrapropositional device restricted to quantified noun phrases, the treatments of identity and cardinality are separated in database semantics. Identity is treated here in terms of identity numbers and applies to quantified noun phrases, proper names, and pronouns alike, across long sequences of propositions. Cardinality is treated as a lexical property of determiners, names, and pronouns. In more advanced systems, the default treatment of identity is complemented by inferencing.

13. LA-MOTOR: navigating through the word bank

Of the five tasks of implementing the database metaphor listed in 1.2, DBL-fragment as defined so far handles task 2 (decoding language into database content) and the hearer part of task 5 (retrieving the location of the speaker's content to achieve analogous storage). We turn now to task 1 (encoding content into language) and the speaker part of task 5 (specifying the location of the content in language).

The task at hand is automatic language production, which has to solve the following problems:

(1) conceptualization (*what to say*),
(2) coherence (*how to say it*, macro structure),
(3) cohesion (*how to say it*, micro structure).

Important aspects of cohesion are:

(a) serialization (ordering the content),
(b) lexicalization (which words to choose).

Implemented systems of natural language generation (NLG) come in three kinds of architecture, pipeline (McDonald [26], McKeown [27], Reiter et al. [34]), interactive (Hovy [20], Reithinger [35]), and integrated (Appelt [1]). Pipeline systems move in only one direction, from a complete selection of content via planning to realization. Interactive systems provide interfaces for a bidirectional decision procedure regarding the choice of topic, sentence content, sentence structure, etc. Integrated systems do not use the distinction between 'what to say' and 'how to say it', but treat generation instead in terms of a planning procedure based on one hierarchy involving both language knowledge and contextual knowledge (cf. Busemann [3]).

The database semantic approach to language production differs from these in that it does not use any planning hierarchy. Instead, conceptualization is based on a general notion of thought, implemented in terms of navigating through propositional content. This provides also essential aspects of cohesion, namely the basic serialization of the language expressions and partially lexicalization.[26] In this way, the traditional distinction between content (*what to say*) and form (*how to say it*) is minimized. Furthermore, coherence is achieved by feeding the word bank with coherent contextual information (cf. Section 9).

To illustrate how LA-MOTOR (cf. 7.3) powers the navigation, let us select an arbitrary position in the word bank 12.8, for example the proplet *eat* with the prn-value 5. It is matched by the start state ST_S of LA-MOTOR, which provides the possible rules F + A = F and F + A = A.

The proplet in question provides two possible continuations, *Peter* and *salad*. Assuming that the rule F + A = F and the continuation *Peter* are chosen, we obtain a navigation from *eat* to *Peter* and back to *eat* (cf. 7.4 or 7.5), which may be realized in English as Peter eats (the salad).

[26] For example, choosing *then* in forward and *before that* in backward navigation, cf. 6.2.

13.1. First Application of F + A = F in the word bank 12.8.

$$\text{F+A=F:}\quad \begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } x\,\beta\,y\\ \text{prn: } k\end{bmatrix} \qquad \begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } \alpha\\ \text{prn: } k\end{bmatrix} \Rightarrow [\text{func: } \alpha] \left\{\begin{array}{l}3\ \text{F+A=F},\\ 4\ \text{F+A=A},\\ 5\ \text{F+cnj=F}\end{array}\right\}$$

$$\begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix} + \begin{bmatrix}\text{arg: } \textit{Peter}\\ \text{FUNC: eat}\\ \text{id: } 1\\ \text{prn: } 5\end{bmatrix} \Rightarrow \begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix}$$

The rule package of F + A = F provides a choice between F + A = F, F + A = A, and F + cnj = F. Furthermore, the continuation attribute of the sentence start gives a choice between *Peter* and *salad*. However, because *Peter* has just been traversed in the previous traversal, it has a negative highlighting, for which reason *salad* is chosen. For simplicity, the rule F + A = F is applied again. [27]

13.2. Second application of F + A = F in the word bank 12.8.

$$\text{F+A=F:}\begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } x\,\beta\,y\\ \text{prn: } k\end{bmatrix} \qquad \begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } \alpha\\ \text{prn: } k\end{bmatrix} \Rightarrow [\text{func: } \alpha] \left\{\begin{array}{l}3\ \text{F+A=F},\\ 4\ \text{F+A=A},\\ 5\ \text{F+cnj=F}\end{array}\right\}$$

$$\begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix} + \begin{bmatrix}\text{arg: } \textit{salad}\\ \text{FUNC: eat}\\ \text{id: } 5\\ \text{prn: } 5\end{bmatrix} \Rightarrow \begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix}$$

At this point, all arguments of proposition 5 have been traversed. Due to their reduced highlighting, the rule chosen for the next transition is F + cnj = F.

13.3. Application of F + cnj = F in the word bank 12.8.

$$\text{F+cnj=F:}\begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } x\\ \text{cnj: } m\,C\,n\\ \text{prn: } m\end{bmatrix} \qquad \begin{bmatrix}\text{func: } \beta\\ \text{ARG: } y\\ \text{cnj: } m\,C\,n\\ \text{prn: } n\end{bmatrix} \Rightarrow [\text{func: } \beta]\ \{7\ \text{F+A=F},\ 8\ \text{F+A=A}\}$$

$$\begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix} + \begin{bmatrix}\text{func: } \textit{pay}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}6 & 7\\ 5 & 6\end{bmatrix}\\ \text{prn: } 6\end{bmatrix} \Rightarrow \begin{bmatrix}\text{func: } \textit{pay}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}6 & 7\\ 5 & 6\end{bmatrix}\\ \text{prn: } 6\end{bmatrix}$$

[27] For a detailed analysis of more complex navigations, including those underlying relative clauses and adverbial clauses see Hausser [16, pp. 486–491].

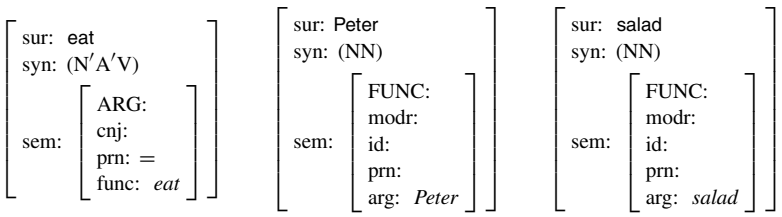
From here, the navigation continues as in 13.1, rendering the overall proplet sequence *Eat Peter salad. Pay Peter salad.*

14. LA-OUTPUT: producing English

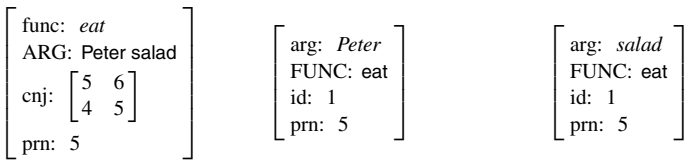
The navigation powered by LA-MOTOR is of a language-independent, universal nature. This navigation activates language production only if LA-MOTOR switches from one-level to two-level navigation. In two-level navigation the contextual proplets traversed are matched by corresponding language proplets of the natural language of choice, as in the following example:

14.1 Lexically matching a proplet sequence.

Language level:



Context level:



During matching, the language proplet types are turned into tokens by copying the corresponding values from the proplets at the context level. The resulting sequence of language proplet tokens is input to an LA-ACN-L grammar (cf. 10.3). In DBL-fragment it is called LA-OUTPUT and has tasks like the following:

14.2. Tasks of LA-OUTPUT.

(1) Readjusting the lexical proplet sequence to the word order of the language.
(2) Deriving determiners based on the features of argument proplets.
(3) Deriving inflectional endings to take care of agreement relations.
(4) Properly reflecting specific kinds of navigation in the language.

As a counterpart to the semantic operations of LA-INPUT (cf. 12.3), LA-OUTPUT uses a *realization function* R. R is a two-place function, taking a set of features and a surface as its arguments, and rendering a modified surface as its value.

14.3. Definition of LA-OUTPUT.

Definition of R: R({[syn: (NM)]}, sur: α) $= \alpha$
R({[syn: (NN)], [def: +]}, sur: α) $=$ the α
R({[syn: (NN)], [def: −]}, sur: α) $=$ a(n) a
R({[syn: (x V)]}, sur: α) $= \alpha$s

ST_S: $\{(\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix} \{\mathrm{V1} + \mathrm{NP}\})\}$

V1 + NP: $\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix} \begin{bmatrix} \text{sur: } \beta \\ \text{syn: } np \\ \text{sem:} \end{bmatrix} \Rightarrow [\text{sur: } \mathrm{R}(\beta)] \begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix} \{\mathrm{V2} + \mathrm{NP}\}$

V2 + NP: $\begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix} \begin{bmatrix} \text{sur: } \beta \\ \text{syn: } np \\ \text{sem:} \end{bmatrix} \Rightarrow [\text{sur: } \mathrm{R}(\alpha)\,\mathrm{R}(\beta)]\,\{\}$

ST_F: $\{([\text{sur: }]\,\mathrm{rp}_{\mathrm{V2+NP}})\}$

The rules of LA-OUTPUT disregard realized items in the new sentence start. [28]

LA-OUTPUT is illustrated below with an application to the lexical proplet sequence [eat] [Peter] [salad] (cf. 14.1). The rule V1 + NP inverts the VERB + NP order by realizing the NP first. Because the NP in our example happens to be of the category (NM), i.e., a proper name, R does not change the surface.

14.4. Applying V1 + NP to [eat] [Peter].

$$\mathrm{V1} + \mathrm{NP}: \begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix} \quad \begin{bmatrix} \text{sur: } \beta \\ \text{syn: (NP)} \\ \text{sem:} \end{bmatrix} \quad \Rightarrow [\text{sur: } \mathrm{R}(\beta)] \begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (x\,V) \\ \text{sem:} \end{bmatrix}$$

$$\begin{bmatrix} \text{sur: eat} \\ \text{syn: } (\mathrm{N'A'V}) \\ \text{sem: } \begin{bmatrix} \text{func: eat} \\ \text{ARG:} \\ \text{cnj:} \\ \text{prn: } = \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \text{sur: Peter} \\ \text{syn: (NM)} \\ \text{sem: } \begin{bmatrix} \text{arg: Peter} \\ \text{FUNC:} \\ \text{id: } +1 \\ \text{prn: } = \end{bmatrix} \end{bmatrix} \Rightarrow [\text{sur: Peter}] \begin{bmatrix} \text{sur: eat} \\ \text{syn: } (\mathrm{A'V}) \\ \text{sem: } \begin{bmatrix} \text{func: eat} \\ \text{ARG: Peter} \\ \text{cnj:} \\ \text{prn: } = \end{bmatrix} \end{bmatrix}$$

Next V2 + NP applies, which takes the (still unrealized) functor item of the sentence start and the next word as input.

[28] Just as the rules of LA-INPUT treat only items with a non-empty surface as syntactically active.

14.5. Applying V2 + NP to [eat] [salad].

$$\text{V2+NP:}\quad \begin{bmatrix} \text{sur: } \alpha \\ \text{syn: } (\text{xV}) \\ \text{sem:} \end{bmatrix} \quad \begin{bmatrix} \text{sur: } \beta \\ \text{syn: } (\text{NP}) \\ \text{sem:} \end{bmatrix} \quad \Rightarrow [\text{sur: } \text{R}(\alpha)\ \text{R}(\beta)]$$

$$\begin{bmatrix} \text{sur: eat} \\ \text{syn: } (\text{N}'\text{A}'\text{V}) \\ \text{sem:} \begin{bmatrix} \text{func: eat} \\ \text{ARG:} \\ \text{cnj:} \\ \text{prn: } = \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \text{sur: salad} \\ \text{syn: (NN)} \\ \text{sem:} \begin{bmatrix} \text{arg: salad} \\ \text{FUNC:} \\ \text{def: } + \\ \text{id: } +1 \\ \text{prn: } = \end{bmatrix} \end{bmatrix} \Rightarrow [\text{sur: eats the salad}]$$

In the output, both the verb and the noun are realized. In the verb, R changes the surface by adding '*s*'. In the noun, R changes the surface by adding the determiner.

15. LA-QUERY: retrieving information

Next we turn to task 3 of 1.2 (querying the database content using natural language). Natural language has two basic kinds of query, called (i) Wh-questions and (ii) yes/no-questions, illustrated below with examples relating to the word bank example 12.8.

15.1. Basic kinds of questions in natural language.

Wh-question	**Yes/no-question**
Who entered the restaurant?	Did Peter enter the restaurant?

Here, the formal syntactic-semantic interpretation is handled by an extension of LA-INPUT (cf. 12.4). The interpretation of a Wh-question results in a functor proplet where the value occupied by the Wh-word is represented by the variable σ_1 and the value of prn is represented by the variable σ_2. The interpretation of a yes/no-question results in a functor proplet where only the prn-attribute contains a variable as its value.

15.2. Search proplets of the two basic kinds of questions.

Wh-question	**Yes/no-question**
func: *enter* ARG: σ_1 restaurant prn: σ_2	func: *enter* ARG: Peter restaurant prn: σ_2

A question is answered by attempting to match the search proplet with the last (most recent) proplet of the corresponding token line (here, the line of *enter* in 12.8). If the continuation value(s) of this last proplet match the search proplet, the answer is found and there is no need for further search. Otherwise, the token line is systematically searched, proceeding backwards from the last proplet. In the case of Wh-questions,

this search is based on the following LA-grammar, which is part of an extended LA-MOTOR:

15.3. LA-Q1 (Wh-questions).

ST_S: $\{([\alpha]\{1\ r_1,\ 2\ r_2\})\}$

r_1: $\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2\end{bmatrix}\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2-1\end{bmatrix}\Rightarrow\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2-1\end{bmatrix}\{3\ r_1\ 4\ r_2\}$

r_2: $\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2\end{bmatrix}\begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2-1\end{bmatrix}\Rightarrow\begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } y\sigma_1 z\\ \text{prn: } \sigma_2-1\end{bmatrix}\{5\ r_3\}$

r_3: $\begin{bmatrix}\text{func: } \alpha\\ \text{ARG: } y\sigma_1 z\\ \text{prn: } n\end{bmatrix}\begin{bmatrix}\text{arg: } \sigma_1\\ \text{FUNC: } \alpha\\ \text{prn: } n\end{bmatrix}\Rightarrow\begin{bmatrix}\text{arg: } \sigma_1\\ \text{FUNC: } \alpha\\ \text{prn: } n\end{bmatrix}\{\}$

ST_F: $\{([\text{arg: } \sigma_1]\ rp_3)\}$

The prn-patterns follow the convention that $\sigma_2 - 1$ (σ_2 minus one) stands for the proplet immediately preceding the current proplet in the token line.

The first two LA-Q1 rules apply if their START-pattern does *not* match the continuation predicates of the input proplet. If the preceding proplet likewise does not match, then the rule r_1 fires and the search is continued with this preceding proplet as the new START. If the preceding proplet does match, rule r_2 fires and the variable σ_1 is bound to the value searched for. Finally, rule r_3 navigates to the proplet of the queried continuation value and returns this proplet as the desired answer.

For example, applying the pattern of the Wh-question 15.2 to the last proplet of the token line *enter* in the word bank 12.8 will result in failure because the continuation values [Peter, house] do not match the search pattern [σ_1, restaurant]. Thus, the START-conditions of the rules r_1 and r_2 of LA-Q1 are satisfied.

When applying r_1 and r_2 to the next preceding proplet (here with prn 3), r_2 happens to be successful. Thereby the variable σ_1 is bound to the answer of the query (i.e., Peter). Rule r_3 navigates to this value (i.e., the proplet *Peter* of proposition 3) and returns it as the answer. At this point, a suitable control system could entice LA-MOTOR to navigate through the subcontextual surroundings of the answer proplet, thus initiating the basis for a more detailed answer.

A query based on a yes/no-question is handled in a similar manner, using the following LA-grammar, also part of an extended LA-MOTOR:

15.4. LA-Q2 (yes/no-questions).

ST_S: $\{([\alpha]\{1\ r_1,\ 2\ r_2\})\}$

r_1: $\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } x\\ \text{prn: } \sigma_2\end{bmatrix}\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } x\\ \text{prn: } \sigma_2-1\end{bmatrix}\Rightarrow\begin{bmatrix}\text{func: } \alpha\\ \neg\text{ARG: } x\\ \text{prn: } \sigma_2-1\end{bmatrix}\{3\ r_1\ 4\ r_2\}$

$$r_2\text{:}\quad \begin{bmatrix} \text{func: } \alpha \\ \neg\text{ARG: } x \\ \text{prn: } \sigma_2 \end{bmatrix} \begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x \\ \text{prn: } \sigma_2 - 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x \\ \text{prn: } \sigma_2 - 1 \end{bmatrix} \{\,\}$$

ST_F: $\{([\text{func: } \alpha]\ \text{rp}_1)([\text{func: } \alpha]\ \text{rp}_2)\}$

In LA-Q2, the answers are based on the final states ST_F, the first of which represents the answer *no* and the second the answer *yes*. For example, in the attempt to answer the yes/no-question Did Peter cross the restaurant? relative to the word bank 12.8, the token line *cross* will be searched in vain. When there are no preceding proplets left in the token line, LA-Q2 terminates in the final state ([func: α] rp_1), which is realized linguistically as the answer no. The yes/no-question Did Peter enter the restaurant?, on the other hand, results in the final state ([func: α] rp_2), which is realized as the answer yes.

16. LA-INFERENCE: a simple example of reasoning

Of the tasks listed in 1.2, the last one remaining is number 4, i.e., inferencing over the content of a word bank. For DBL-fragment we present formalized examples of episodic and absolute inference. They form a partially specified LA-grammar, called LA-INFERENCE, which is an instance of LA-INF-E (cf. 10.3).

In DBL-production and -interpretation, inferences are needed for many tasks. For example, the coreference between argument proplets must be coded into the language surfaces by the speaker using pronouns or definite descriptions, which in turn must be decoded by the hearer. Also, in the answering of questions a hierarchy of hyper- and hyponyms must be used to infer instantiations which are not contained directly in the word bank, etc.

Inferences of this kind have been studied in detail in the logical systems from antiquity to contemporary AI. [29] This raises the question of whether and how the inferences of the classical as well as the modern systems should be recreated within database semantics. As a simple example, consider some classic inferences of propositional calculus.

16.1. Inference schemata of propositional calculus.

1. $\dfrac{A, B}{\vdash A\&B}$ 2. $\dfrac{A \vee B, \neg A}{\vdash B}$ 3. $\dfrac{A \rightarrow B, A}{\vdash B}$ 4. $\dfrac{A \rightarrow B, \neg B}{\vdash \neg A}$

5. $\dfrac{A\&B}{\vdash A}$ 6. $\dfrac{A}{\vdash A \vee B}$ 7. $\dfrac{\neg A}{\vdash A \rightarrow B}$ 8. $\dfrac{\neg\neg A}{\vdash A}$

The first inference is called conjunction and says that the truth of two arbitrary propositions A and B implies the truth of the complex proposition $A\&B$. A transfer of this inference into database semantics amounts to an operation which establishes new extrapropositional relations based on the conjunction *and*. This operation may be realized by the following LA-grammar rule:

[29] Cf. Bibel [2].

16.2. Hypothetical inference rule inf_{conj} for conjunction.

$$\mathsf{inf}_{conj}\text{:}\ \begin{bmatrix}\text{func: } \alpha\\ \text{prn: } m\end{bmatrix}\begin{bmatrix}\text{func: } \beta\\ \text{prn: } n\end{bmatrix} \Rightarrow \begin{bmatrix}\text{func: } \alpha\\ \text{prn: } m\\ \text{cnj: } m \textit{ and } n\end{bmatrix}\begin{bmatrix}\text{func: } \beta\\ \text{prn: } n\\ \text{cnj: } m \textit{ and } n\end{bmatrix}$$

This inference rule produces the new extrapropositional relation [cnj: *m and n*] between two arbitrary propositions *m* and *n*, enabling navigation from any proposition to any other proposition asserted.

In its logical interpretation, the inference in question expresses a conjunction of truth and is as such intuitively obvious. Under these circumstances it is not relevant *which* propositions are conjoined – as long as they are true.

In its database semantic interpretation, however, this same inference raises the question of *why* two—up to now unconnected—propositions should be concatenated with *and*. Even though such a concatenation would not result in asserting a falsehood, an uncontrolled application of inf_{conj} would destroy coherence. This shows that the classical inference schemata of logic change their character substantially when transferred to database semantics, for which reason they are not suitable to be adopted directly. [30]

A logical inference more suitable for reconstruction in database semantics is 3 in 16.1, known as modus ponens [31] and illustrated below with the formal analysis of the inference described intuitively in 4.6. Its first premise is formalized as follows:

16.3. Proplets of an absolute proposition *Peter eat salad*.

$$\begin{bmatrix}\text{func: } \textit{eat}\\ \text{ARG: Peter salad}\\ \text{cnj: } \begin{bmatrix}5 & 6\\ 4 & 5\end{bmatrix}\\ \text{prn: } 5\end{bmatrix}\begin{bmatrix}\text{arg: } \textit{Peter}\\ \text{FUNC: } \textit{eat}\\ \text{id: } 1\\ \text{prn: } 5\end{bmatrix}\begin{bmatrix}\text{arg: } \textit{salad}\\ \text{FUNC: } \textit{eat}\\ \text{id: } 5\\ \text{prn: } 5\end{bmatrix}$$

The second premise of 4.6 is the absolute proposition salad is food:

16.4. Proplets of the absolute proposition *salad is food*.

$$\begin{bmatrix}\text{func: } \textit{be}\\ \text{ARG: salad food}\\ \text{prn: abs-327}\\ \text{func: } \textit{be}\end{bmatrix}\begin{bmatrix}\text{arg: } \textit{salad}\\ \text{VERB: be}\\ \text{prn: abs-327}\\ \text{arg: } \textit{salad}\end{bmatrix}\begin{bmatrix}\text{arg: } \textit{food}\\ \text{VERB: be}\\ \text{prn: abs-327}\\ \text{arg: } \textit{food}\end{bmatrix}$$

The episodic proposition 16.3 and the absolute proposition 16.4 serve jointly as the premises of the following episodic inference rule:

[30] The problems of logical inference systems have been clearly identified even in textbooks. See for example Russell & Norvig 1995.

[31] The school-book example illustrating modus ponens intuitively is *Socrates is a human* & *All humans are mortal* ⇒ *Socrates is mortal*. In database semantics, the universal quantifier in the second premise is expressed alternatively by the use of concept types—which are suitable also for mass nouns.

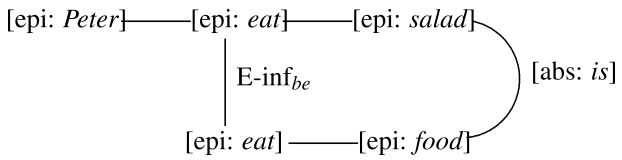
16.5. Episodic inference rule E-inf$_{be}$.

$$\text{E-inf}_{be}\colon \begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x\ \beta\ y \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{func: } be \\ \text{ARG: } \beta\ \gamma \\ \text{prn: } abs\text{-}m \\ \text{func: } be \end{bmatrix} \Rightarrow \begin{bmatrix} \text{func: } \alpha \\ \text{ARG: } x\ \gamma\ y \\ \text{prn: } n' \end{bmatrix} \begin{bmatrix} \text{arg: } \gamma \\ \text{FUNC: } \alpha \\ \text{prn: } n' \end{bmatrix}$$

The START-pattern of rule E-inf$_{be}$ is matched onto the episodic functor proplet of 16.3, whereby the variable β is assigned the value salad. This enables the rule E-inf$_{be}$ to navigate to the functor proplet of the absolute proposition 16.4 as the NEXT, whereby the variable γ is bound to *food*. The result proplets represent a new episodic proposition which has the prn n' and expresses *eat Peter food*.

Intuitively, the inference may be described in terms of the following network:

16.6. Content extension based E-inf$_{be}$.



In this schema, the representation of the original episodic proposition 16.3 has the simplified form [epi: *Peter*], [epi: *eat*], [epi: *salad*]. The inference creates a temporary extension of the original network, represented as [epi: *Peter*], [epi: *eat*], [epi: *food*]. The rule E-inf$_{be}$ is applicable to functor proplets of all absolute propositions with the verb *be*, i.e., to all the elements of the is-a-hierarchy.

The episodic proposition *Peter eat salad* is part of the word bank 12.8. Without further provision, the question Did Peter eat some food? would be answered with no relative to the word bank 12.8. However, as soon as the absolute proposition *salad is food* is added to the word bank, the question Did Peter eat some food? is answered correctly with yes, based on the inference E-inf$_{be}$.

Next consider the absolute inference outlined intuitively in 22.4.7. Its formal analysis is based on the two propositions and the inference rule A-inf$_{be}$ defined below:

16.7. Proplets of the absolute proposition *milk is drink*.

$$\begin{bmatrix} \text{func: } be \\ \text{ARG: milk drink} \\ \text{prn: abs-328} \\ \text{func: } be \end{bmatrix} \begin{bmatrix} \text{arg: } milk \\ \text{VERB: be} \\ \text{prn: abs-328} \\ \text{arg: } milk \end{bmatrix} \begin{bmatrix} \text{arg: } drink \\ \text{VERB: be} \\ \text{prn: abs-328} \\ \text{arg: } drink \end{bmatrix}$$

16.8. Proplets of the absolute proposition *drink is liquid*.

$$\begin{bmatrix} \text{func: } be \\ \text{ARG: drink liquid} \\ \text{prn: abs-329} \\ \text{func: } be \end{bmatrix} \begin{bmatrix} \text{arg: } drink \\ \text{VERB: be} \\ \text{prn: abs-329} \\ \text{arg: } drink \end{bmatrix} \begin{bmatrix} \text{arg: } liquid \\ \text{VERB: be} \\ \text{prn: abs-329} \\ \text{arg: } liquid \end{bmatrix}$$

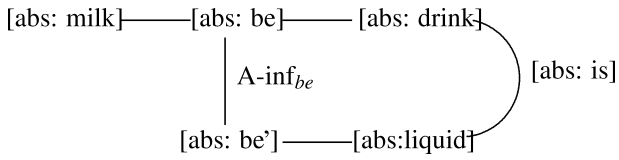
16.9. Absolute inference rule A-inf$_{be}$.

$$\text{A-inf}_{be}\text{:}\quad \begin{bmatrix}\text{func: } be\\ \text{ARG: } \alpha\ \beta\\ \text{prn: } abs\text{-}n\\ \text{func: } be\end{bmatrix}\begin{bmatrix}\text{func: } be\\ \text{ARG: } \beta\ \gamma\\ \text{prn: } abs\text{-}m\\ \text{func: } be\end{bmatrix} \Rightarrow \begin{bmatrix}\text{func: } be\\ \text{ARG: } \alpha\ \gamma\\ \text{prn: } abs\text{-}n'\\ \text{prn: } be\end{bmatrix}\begin{bmatrix}\text{arg: } \gamma\\ \text{FUNC: } be\\ \text{prn: } abs\text{-}n'\\ \text{prn: } \gamma\end{bmatrix}$$

This rule takes an absolute proplet as its START, in contradistinction to E-inf$_{be}$.

Like E-inf$_{be}$, the inference rule A-inf$_{be}$ results in a temporary extension of the content which may be represented intuitively as follows:

16.10. Content extension based on A-inf$_{be}$.



The general knowledge that milk is a liquid may be used, for example, when the question of transporting a particular drink of milk arises. In such a situation, all the knowledge of transporting liquids stored in the token line of *liquid* may be brought into play.

In addition to the classical inferences of propositional and predicate calculus, there are numerous implications connected to certain word meanings. An example is the distinction between intensional and extensional contexts, which may be illustrated intuitively by the following implications:

If α finds β, then β exists.

If α seeks β, then possible β does not exist.

These may be formalized in database semantics as the following rules:

16.11. Implication rule imp$_{find}$.

$$\text{imp}_{find}\text{:}\quad \begin{bmatrix}\text{func: } find\\ \text{ARG: } \alpha\ \beta\\ \text{prn: } n\end{bmatrix}\begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } find\\ \text{id: } \gamma\\ \text{prn: } n\end{bmatrix} \Rightarrow \begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } exist\\ \text{id: } \gamma\\ \text{prn: } n'\end{bmatrix}\begin{bmatrix}\text{func: } exist\\ \quad\begin{bmatrix}\text{polarity: } +\\ \text{certainty: } 1\end{bmatrix}\\ \text{ARG: } \beta\\ \text{prn: } n'\end{bmatrix}$$

16.12. Implication rule imp$_{seek}$.

$$\text{imp}_{seek}\text{:}\quad \begin{bmatrix}\text{func: } seek\\ \text{ARG: } \alpha\ \beta\\ \text{prn: } n\end{bmatrix}\begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } seek\\ \text{id: } \gamma\\ \text{prn: } n\end{bmatrix} \Rightarrow \begin{bmatrix}\text{arg: } \beta\\ \text{FUNC: } exist\\ \text{id: } \gamma\\ \text{prn: } n'\end{bmatrix}\begin{bmatrix}\text{func: } exist\\ \quad\begin{bmatrix}\text{polarity: } -\\ \text{certainty: } 0.5\end{bmatrix}\\ \text{ARG: } \beta\\ \text{prn: } n'\end{bmatrix}$$

In each case, the implication [32] matches an episodic proplet of *find* or *seek*, respectively, with its first proplet pattern and proceeds from there to the associated argument proplet. In the result, the first implication asserts of the argument proplet that its referent exists and the second that it does possibly not exist. [33]

17. Conclusion

When the time-linear algorithm of LA-grammar was presented in 1986 [12], complete with the Lisp source code for sizeable fragments of German and English, it was used for an efficient syntactic parsing of natural language. In 1989 [13] there followed an algebraic definition of LA-grammar, which was shown in 1992 [14] to be the first—and so far the only—formal grammar algorithm with a complexity hierarchy orthogonal to that of phrase structure grammar (known as the Chomsky hierarchy).

In this paper, LA-grammar is extended from syntactic parsing to a model of natural language communication. To illustrate the complex functional interaction between interpretation, conceptualization, production as well as query and inference within the overall system, we adapt Montague's method of defining a formal 'fragment'.

Our fragment, called a SLIM machine, uses LA-grammar as a motor algorithm in combination with a new data structure, called a word bank. A word bank is a network database which defines possible continuations within its record-based structure. These form the basis for a kind of operation which conventional databases do not provide, namely an autonomous time-linear navigation through the content of the database.

The result is a declarative specification of a cognitive agent which shows in formal detail how communicating freely in natural language works in principle. While the empirical scope of the fragment is purposely limited, the specification of its different functions and their interaction is of complete generality.

One gap remains, however: though non-language-based recognition and action, both external and internal, play an essential role in the overall theory of database semantics, they are not defined explicitly as parts of the SLIM machine presented here. To fill this gap, robotics and procedural semantics will need to work together much more closely than is currently the case.

References

[1] D. Appelt, Planning English Sentences, Cambridge University Press, Cambridge, 1985.
[2] W. Bibel, Wissensrepräsentation und Inferenz, Vieweg, Braunschweig, 1993.
[3] S. Busemann, Generierung natürlichsprachlicher Texte, in: G. Görz, C.-R. Rollinger, J. Schneeberger (Eds.), Handbuch der Künstlichen Intelligenz, 3rd edn., Oldenbourg, München, 2000.

[32] Similar implication rules may be defined for other words, for example inchoative verbs like fall asleep or awake, which imply that the subject was not sleeping or awake before, respectively. Their formulation is based on the temporal order of propositions (cf. Hausser [17]). For a detailed analysis of such implications within Montague grammar see Dowty [5].

[33] Thereby, negation is expressed in terms of a negative polarity ([polarity: –]) and possibility by a certainty of 0.5 ([certainty: 0.5]).

[4] B. Dorr, Machine Translation: A View from the Lexicon, MIT Press, Cambridge, MA, 1993.
[5] D. Dowty, Word Meaning and Montague Grammar, D. Reidel, Dordrecht, 1979.
[6] R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, Benjamin-Cummings, Redwood City, CA, 1989.
[7] G. Fauconnier, Mappings in Thought and Language, Cambridge University Press, Cambridge, 1997.
[8] J. Fodor, In Critical Condition, Polemical Essays on Cognitive Science and the Philosophy of Mind, MIT Press/Bradford Books, Cambridge, MA, 2000.
[9] J. Funge, X. Tu, D. Terzopoulos, Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters, SIGGRAPH 99, Los Angeles, CA, 1999.
[10] P. Gärdenfors, Conceptual Spaces, MIT Press, Cambridge, MA, 2000.
[11] R. Hausser, Surface Compositional Grammar, Wilhelm Fink Verlag, München, 1984.
[12] R. Hausser, NEWCAT: Parsing Natural Language Using Left-Associative Grammar, Springer, Berlin, 1986.
[13] R. Hausser, Computation of Language, An Essay on Syntax, Semantics and Pragmatics in Natural Man-Machine Communication, Symbolic Computation: Artificial Intelligence, Springer, Berlin, 1989.
[14] R. Hausser, Complexity in left-associative grammar, Theoret. Comput. Sci. 106 (2) (1992) 283–308.
[15] R. Hausser, A database interpretation of natural language, Korean J. Linguistics 21 (1–2) (1996) 29–55.
[16] R. Hausser, Foundations of Computational Linguistics, Springer, Berlin, 1999.
[17] R. Hausser, The four basic ontologies of semantic interpretation, in: H. Kangassalo et al. (Eds.), Information Modeling and Knowledge Bases XII, IOS Press, Amsterdam/Ohmsha, Tokyo, 2001.
[18] R. Hausser, Spatio-temporal indexing in database semantics, in: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing, Lecture Notes in Computer Science, Vol. 2004, Springer, Berlin, 2001.
[19] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[20] E. Hovy, Generating Natural Language under Pragmatic Constraints, Lawrence Erlbaum, Hillsdale, NJ, 1988.
[21] R. Langacker, Foundations of Cognitive Semantics, Vols. 1–2, Stanford University Press, Stanford, CA, 1987/1991.
[22] G. Lakoff, Women, Fire, and Dangerous Things, University of Chicago Press, Chicago, IL, 1987.
[23] G. Lakoff, M. Johnson, Metaphors We Live By, University of Chicago Press, Chicago, IL, 1980.
[24] S. Lappin (Ed.), The Handbook of Contemporary Semantic Theory, Blackwell Publishers, Oxford, 1996.
[25] A.B. Loyall, J. Bates, Real-time control of animated broad agents, in: Proc. 15th Annual Conference of the Cognitive Science Society, Boulder, CO, 1993.
[26] D. McDonald, Natural language generation as a computational problem: An introduction, in: M. Brady, R.C. Berwick (Eds.), Computational Models of Discourse, MIT Press, Cambridge, MA, 1983.
[27] K. McKeown, Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text, Cambridge University Press, Cambridge, 1985.
[28] I.A. Mel'čuk, Dependency Syntax: Theory and Practice, State University of New York Press, New York, 1988.
[29] I.A. Mel'čuk, A. Polguère, A formal lexicon in the meaning-text theory, Computational Linguistics 13 (3–4) (1987) 13–54.
[30] R. Montague, Formal Philosophy, Yale University Press, New Haven, CT, 1974 (edited by R. Thomason).
[31] A. Nasr, O. Rambow, M. Palmer, J. Rosenzweig, Enriching lexical transfer with cross-linguistic semantic features, in: Proceedings of the Interlingua Workshop at the MT Summit, San Diego, CA, 1997.
[32] F. Pereira, D. Warren, Definite clause grammars for language analysis—A survey of the formalism and a comparison to augmented transition networks, Artificial Intelligence 13 (1980) 231–278.
[33] C. Pollard, I. Sag, Head-Driven Phrase Structure Grammar, CSLI Stanford and The University of Chicago Press, Chicago, IL, 1994.
[34] E. Reiter, A. Cawsey, L. Osman, Y. Roff, Knowledge acquisition for content selection, in: Proc. 6th European Workshop on Natural Language Generation (ENGLGWS-97), Duisburg, Germany, 1997, pp. 117–126.
[35] N. Reithinger, Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge, Infix, St. Augustin, 1992.

[36] J. Rickel, W.L. Johnson, Animated agents for procedural training in virtual reality, perception, cognition, and motor control, Appl. Artificial Intelligence 13 (1999) 343–382.
[37] S.J. Russell, P. Norvig, Artificial Intelligence, a Modern Approach, Prentice Hall, Englewood Cliffs, 1995.
[38] B.F. Skinner, About Behaviorism, Alfred A. Knopf, 1974.
[39] T.C. Son, C. Baral, Formalizing sensing actions—A transition function based approach, Artificial Intelligence 125 (2001) 19–91.
[40] J.F. Sowa, Conceptual Structures, Addison-Wesley, Reading, MA, 1984.
[41] J.F. Sowa, Conceptual Graph Standard, revised version of December 6, 2000, http://www.bestweb.net/~sowa/cg/cgstand.htm.
[42] L. Talmy, Toward a Cognitive Semantics, Vols. I & II, MIT Press, Cambridge, MA, 2000.