

Reconstructing Propositional Calculus in Database Semantics

Roland Hausser

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

Abstract

Propositional calculus is reconstructed as a simplified version of natural language used for communication. This requires a number of changes with respect to the traditional approach,

First, the method must be changed from the meta-language definitions of old logic to a declarative specification for computer software. Second, the ‘realist ontology’ of old logic, treating meaning as a direct relation between sentences and the ‘world’ (defined as a set-theoretic model) must be replaced by a functional system comprising an agent’s cognitive operations.¹ Third, the syntax of propositional calculus must be adapted to the time-linear nature of language. Fourth, the truth conditional semantics of propositional calculus must be integrated into the time-linear process of communication.

The latter task requires solutions to the following problems: First, whether an expression like $\mathbf{p} \ \& \ \tilde{\mathbf{p}}...$ is contradictory cannot be decided in a time-linear interpretation until the continuation, e.g., $\mathbf{p} \ \& \ \tilde{\mathbf{p}} \ \vee \ \mathbf{q}...$, is known; second, using truth conditions for checking consistency must avoid the SAT problem, which is known to be computationally intractable.

The reconstruction is presented as the declarative specification of a computer program, comprising (i) a data structure for storing formulas of propositional calculus as concatenated propositions, and (ii) LA-grammars for conceptualization, production, and interpretation. Written at a high level of abstraction, the reconstruction provides a comparison between the metalanguage-based handling of truth in traditional logic and the procedural model of communication in Database Semantics.

1 Introduction

In Database Semantics (DBS), communication is modeled as a time-linear procedure for (i) mapping content in the speaker’s database into language and (ii) mapping language into equivalent content in the hearer’s database. Technically, DBS consists of (i) a data structure called a word bank, and (ii) a time-linear algorithm called LA-grammar for navigating through the content of the data structure and for reading content into and out of the database. Navigation is also used in query-answering and inferencing (Hausser 2001b).

Compared with DBS, propositional calculus is a model of simplicity, and reconstructing it in the powerful DBS system may seem like an academic exercise. Nevertheless, the reconstructing is interesting for the following reasons:

¹This is partially in concord with Frege’s 1879 original design of propositional calculus as a ‘formula language for pure thought’ (op. cit, p. 6). Our reconstruction differs from Frege, however, in that it includes the procedures of language production and interpretation inside the cognitive agent.

First, it is easier to understand the new (i.e., DBS) after enabling a view from the old (i.e., propositional calculus). Second, old logic guards a treasure trove of important results, and reconstructing its most basic component in DBS is an effective way of importing these results into the new system. Third, the novel use of propositional calculus as a simplified language for communication makes it abundantly clear that basic assumptions of traditional propositional calculus, such as its realist ontology and meta-language-based methodology,² must be sacrificed for the success of the reconstruction, thus shedding new light on old logic.

Furthermore, the reconstruction is non-trivial because it requires (i) a time-linear treatment of propositional calculus and (ii) storage in a database. The problem is that logical formulas treat the relations between propositions ‘graphically’ by depending on the order of their signs – which is unsuitable for a database.³

DBS solves this problem by defining the semantic atoms, called proplets,⁴ as *feature structures* with attributes which specify the relations to other atoms (bidirectional pointering). By coding conceptual relations between proplets⁵ by means of attributes located inside their feature structure, atoms related to each other semantically may be stored arbitrarily far from each other in the database.

In addition, this new data structure is suitable for a time-linear model of communication: First, syntactic-semantic language analysis may be easily defined to code all inter-proplet relations in terms of the attribute-value pairs of individual proplets (hearer mode). Second, once the proplets have been read into the database, they are suitable for time-linear navigation along the proplet connections, providing a model of thought and thus conceptualization (*what to say* in language production). Third, the proplets traversed by the navigation may be copied into a sequence which may be easily mapped into suitable language surfaces (speaker mode).

When DBS is used for communication in natural language, there arise a number of descriptive tasks, such as controlling the navigation, pragmatic interpretation in the speaker mode, pragmatic interpretation in the hearer mode, etc. Propositional calculus as a simplified language of communication, however, is much too basic to require any work in these departments. All that is needed for the task of this paper is a time-linear interpretation, conceptualization, and production of this simple formal language.

Our task is enriched, however, by the main achievement of propositional calculus, namely the truth conditional semantics of the connectives &, ∨, negation, etc. At first, these truth conditions may seem of rather minor interest to DBS with its procedurally defined semantic primitives.⁶ What could be the purpose of truth conditions in a functional model of communication between cognitive agents?

The natural answer is: *for checking consistency* by integrating truth conditions into the time-linear navigation through the content of the data structure, here concatenated propositions. But what about SAT (Boolean SATisfiability)? This well-known problem is a textbook example of exponential complexity. If integrating the truth conditions of propositional calculus into a consistency-checking navigation were equivalent to Boolean Satisfiability, then the intended reconstruction would be computationally intractable – and therefore useless.

²For a more detailed discussion see Hausser 2001a.

³The problem of graphical presentation is shared by theoretical linguistics’ use of tree structures, which are likewise unsuitable for databases.

⁴In the case of propositional calculus, the proplets represent the propositional constants.

⁵In natural language, there are proplets for verbs, nouns, and adjectives. Furthermore, there are (i) *intrapropositional relations* between proplets belonging to the same proposition, and *extrapropositional relations* (ii) between the verbs of different propositions, related by conjunctions, and (iii) between the nouns of different propositions, related in terms of identity.

⁶For example, *red* defined as an electromagnetic measurement.

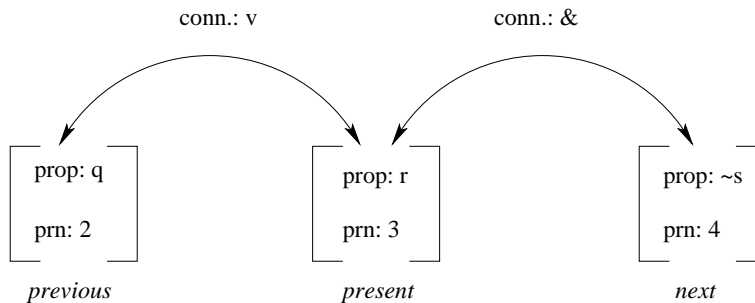
Thus, equivalence with SAT must be avoided and low (preferably linear) complexity must be assured. This is achieved by replacing the arbitrary truth value assignment underlying SAT with a more restricted value assignment based on treating propositions as *database assertions*. In this sense, our reconstruction of predicate calculus in DBS is yet another example where a resounding *no* of old logic is turned into a simple *yes* by adapting old logic's outmoded 'realist' ontology to that of cognitive agents.⁷

2 Data structure of the reconstruction

For simplicity, formulas of propositional calculus are written in conjunctive normal form (CNF).⁸ The first step of the reconstruction consists in parsing CNF formulas and translating them automatically into the data structure of a word bank. Thereby, the information represented graphically in traditional logical formulas, e.g., $\mathbf{p} \vee \tilde{\mathbf{q}} \ \& \ \mathbf{r}$ must be recoded in such a way that the elements of the formula, e.g., \mathbf{p} , $\tilde{\mathbf{q}}$, and \mathbf{r} , as well as the connectives can be stored in arbitrary places in the database.

This is achieved by analyzing propositional constants in the database not as primitives, but as feature structures designed to represent elementary propositions as well as their extrapositional relations. As an illustration of what needs to be coded into the propositional feature structures consider the following analysis of the propositional constant \mathbf{r} occurring as part of the formula $\mathbf{q} \vee \mathbf{r} \ \& \ \tilde{\mathbf{s}}$:

2.1 BASIC UNITS OF PROPOSITIONAL CONCATENATION



The feature structure representing \mathbf{r} is the item in the middle, called *present*. It is characterized by two attributes: the name of the proposition (here [name: \mathbf{r}]) and its proposition number (here [prn: 3]).

Within the formula, \mathbf{r} has two kinds of external relations, one to the previous \mathbf{q} , the other to the next $\tilde{\mathbf{s}}$. The relations to previous and next are each characterized by two attributes: The previous by the connective \vee and the propositional constant \mathbf{q} ; the next by the connective $\&$ and the propositional constant $\tilde{\mathbf{s}}$.⁹

⁷Other examples of overcoming the limitations of old logic by adopting an adequate ontology are propositional attitudes (Hausser 1999/2001, pp. 395 f.), vagueness and three-valued logic (Hausser 1999/2001, pp. 402 f.), and the Epimenides paradox (Hausser 1999/2001, pp. 383 f., 413 f.).

⁸CNF consists of constants with or without negation, connected by '&' (and) and ' \vee ' (or). The negation of, for example, \mathbf{p} is written as $\tilde{\mathbf{p}}$, called *external negation* and paraphrased as *It is not the case that p*. CNF formulas are written without parentheses whereby, for example, $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \mathbf{s} \vee \mathbf{t} \vee \mathbf{u}$ is treated as equivalent to $(\mathbf{p} \vee \mathbf{q} \vee \mathbf{r}) \ \& \ (\mathbf{s} \vee \mathbf{t} \vee \mathbf{u})$. Cf. Hopcroft & Ullman 1979, p. 325.

⁹Extrapositional relations have the form [n-1 c] for the previous and [c n+1] for the next, where n is the number of the current proposition and c is the connective.

Consider example 2.1: the proposition \mathbf{r} has the number 3 ($n = 3$); consequently, the backward connection is [2 \vee], while the forward connection is [$\&$ 4].

According to this analysis, the propositional constant **r** within the formula of 2.1 may be characterized by the following feature structure:

2.2 FEATURE STRUCTURE OF A PROPOSITIONAL CONSTANT

$\left[\begin{array}{l} \text{prop: } \mathbf{r} \\ \text{ctn: } \& \tilde{\mathbf{s}} \\ \text{ctp: } \mathbf{q} \vee \\ \text{prn: } \mathbf{3} \end{array} \right]$	<p>prop =_{def} name of propositional constant</p> <p>ctn =_{def} connective to next</p> <p>ctp =_{def} connective to previous</p> <p>prn =_{def} proposition number</p>
---	--

In a lexical *type*, most attributes of the feature structures have the value NIL (represented by space), except for the attribute naming the item, as illustrated below:

2.3 PROPOSITIONAL CONSTANTS AND CONNECTIVES AS LEXICAL ITEMS

propositional constants

connectives

$\left[\begin{array}{l} \text{prop: } \mathbf{q} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right]$	$\left[\begin{array}{l} \text{prop: } \mathbf{r} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right]$	$\left[\begin{array}{l} \text{prop: } \tilde{\mathbf{s}} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right]$	$\left[\text{conn: } \vee \right]$	$\left[\text{conn: } \& \right]$
---	---	---	-------------------------------------	-----------------------------------

During parsing of traditional CNF formulas by a semantically interpreted LA-grammar, these lexical types are (i) assigned to the propositional constants and connectives of the input and (ii) their attributes are filled with proper values by means of copying. For example, the formula $\mathbf{q} \vee \mathbf{r} \& \tilde{\mathbf{s}}$ represented graphically in 2.1 translates equivalently into the following set of proplets (tokens):

2.4 RECODING $\mathbf{q} \vee \mathbf{r} \& \tilde{\mathbf{s}}$ AS A SET OF PROPLETS

$\left[\begin{array}{l} \text{prop: } \mathbf{q} \\ \text{ctn: } \vee \mathbf{r} \\ \text{ctp:} \\ \text{prn: } 2 \end{array} \right]$	$\left[\begin{array}{l} \text{prop: } \mathbf{r} \\ \text{ctn: } \& \tilde{\mathbf{s}} \\ \text{ctp: } \mathbf{q} \vee \\ \text{prn: } 3 \end{array} \right]$	$\left[\begin{array}{l} \text{prop: } \tilde{\mathbf{s}} \\ \text{ctn:} \\ \text{ctp: } \mathbf{r} \& \\ \text{prn: } 4 \end{array} \right]$
---	--	---

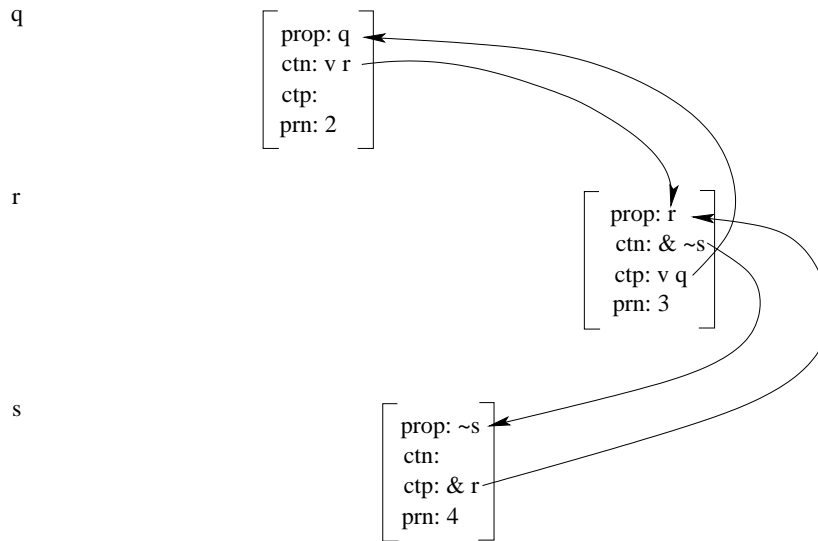
These proplet tokens are suitable for storage in the following data structure:

2.5 DATA STRUCTURE OF A WORD BANK

type _a	proplet _{a1} proplet _{a2} proplet _{a3} proplet _{a4} , etc.
type _b	proplet _{b1} proplet _{b2} proplet _{b3} proplet _{b4} , etc.
type _c	proplet _{c1} proplet _{c2} proplet _{c3} proplet _{c4} , etc.
etc.	

The types are ordered alphabetically, and proplets of the same type are sorted in a line behind their type, as in a network database. Proplets are related to each other in terms of attributes contained in their feature structures.

2.6 BIDIRECTIONAL POINTERING BETWEEN PROPLETS

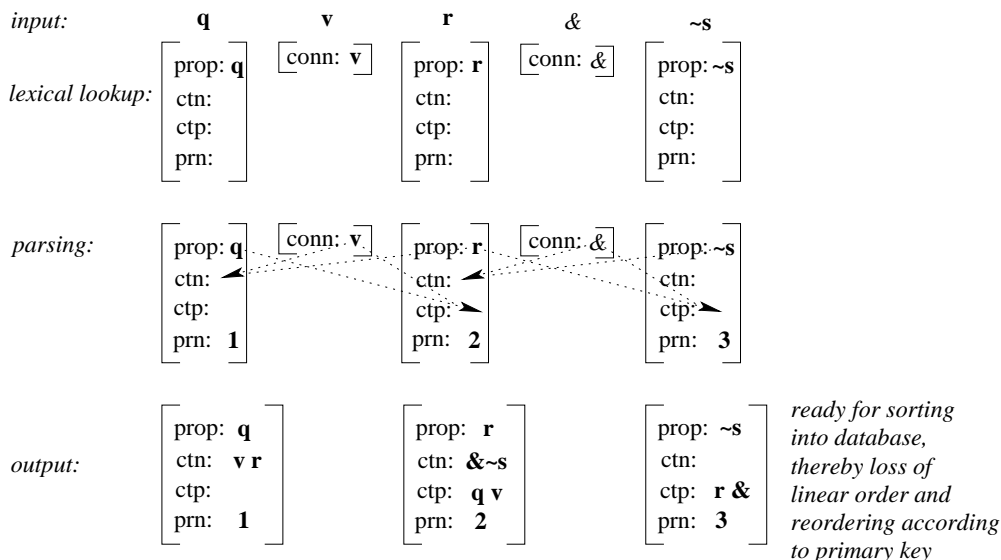


Navigation along the concatenations, depicted here by arrows, is realized technically by means of the retrieval mechanism of the database employed. Take for example proplet **r**: specification of its proposition number, the name of the previous proplet, and the type of conjunction are sufficient not only to characterize the relation between **r** and **q**, but also to retrieve the previous proplet (backward navigation) – and similarly for forward navigation from **r** to **s**.

3 Syntactic-semantic interpretation

The semantically interpreted LA-grammar for parsing CNF formulas and translating them into equivalent sets of proplets is called LA-propositional_calculus_interpretation, or *LA-pci* for short. The semantic interpretation of *LA-pci* consists in copying values between feature structures, as illustrated intuitively below:

3.1 TRANSLATING CNF FORMULA INTO PROPLETS



LA-pci consists of a lexicon LX , a variable definition, a set of start states ST_S , a set of rules (here r-1 and r-2), and a set of final states ST_F .

3.2 DEFINITION OF *LA-pci*

$$LX =_{def} \left[\begin{array}{l} \text{prop: } p \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \left[\begin{array}{l} \text{prop: } \tilde{p} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \left[\begin{array}{l} \text{prop: } q \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \left[\begin{array}{l} \text{prop: } \tilde{q} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \left[\begin{array}{l} \text{prop: } r \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right], \text{ etc. ,}$$

$$[\text{conn: } \vee], [\text{conn: } \&]$$

Variable definition: $\alpha, \beta \in \{p, \tilde{p}, q, \tilde{q}, r, \tilde{r}, \dots\}$, $c \in \{\vee, \&\}$,
 $x, y, z =_{def}$ optional values (can be NIL)

$$ST_S =_{def} \left\{ \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \{r-1\} \right\}$$

$$r-1: \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn:} \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] [\text{conn: } c] \Rightarrow \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \{r-2\}$$

nw-conn \xrightarrow{e} ss-ctn
copy_{ss}

$$r-2: \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \beta \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha c \\ \text{prn: } y \end{array} \right] \{r-1\}$$

ss-prop: \xrightarrow{e} nw-ctp
ss-ctn: \xrightarrow{a} nw-ctp
nw-prop \xrightarrow{a} ss-ctn
copy_{ss}, copy_{nw}

$$ST_F =_{def} \left\{ \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn:} \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] rp-2 \right\}$$

The lexicon LX contains *types* of feature structures (cf. 2.3). They comprise the propositional constants $p, \tilde{p}, q, \tilde{q}, r, \tilde{r}$, etc., and the connectives \vee and $\&$.

The variable definition specifies variables for defining the rule patterns and the operations on these patterns by the rules of *LA-pci*. The variables α and β are restricted to propositional constants, while the variable c is restricted to connectives.

The set ST_S contains a single state. It indicates that any derivation of *LA-pci* must begin with a propositional constant and the rule r-1.

The rule r-1 attaches a connective to a proposition sequence, e.g., $p q r + \&$, by copying the value of the connective into the ctp attribute of the last proposition, e.g., r . The rule r-2

attaches a next propositional constant to a proposition sequence, e.g., $\mathbf{p} \mathbf{q} \mathbf{r} + \mathbf{s}$ by supplying proposition numbers to the ctn and ctp attributes of the last proplet of the current sequence, e.g., \mathbf{r} , and the next proplet, e.g., \mathbf{s} .¹⁰

The working of *LA-pci* is illustrated below with the parsing of $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \tilde{\mathbf{p}}$. The start state of *LA-pci* matches the name of the first proposition \mathbf{p} and activates rule r-1, which reads \vee as the next. The control structure of the parser assigns the proposition number 1 to the feature structure of \mathbf{p} .

3.3 APPLYING r-1 TO ‘ $\mathbf{p} + \vee$ ’

$$\begin{array}{l}
 \text{r-1:} \quad \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn:} \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \quad [\text{conn: } c] \quad \Rightarrow \quad \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \quad \{\text{r-2}\} \\
 \\
 \text{nw-conn } \boxed{e} \rightarrow \text{ss-ctn} \\
 \text{copy}_{ss} \\
 \\
 \left[\begin{array}{l} \text{prop: } \mathbf{p} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \quad [\text{conn: } \vee] \quad \Rightarrow \quad \left[\begin{array}{l} \text{prop: } \mathbf{p} \\ \text{ctn: } \vee \\ \text{ctp:} \\ \text{prn: } 1 \end{array} \right]
 \end{array}$$

The operations of r-1 copy the connective into the ctn slot of the last sentence start proposition, called ss (nw-conn $\boxed{e} \rightarrow$ ss-ctn). In the result, the proplet of the connective is discarded by copying only the ss (copy_{ss}). The proposition number 1 is assigned to the resulting ss proplet by the control structure of the system.

The successful application of rule r-1 activates its rule package. Its only rule, r-2, is applied to the new start and the next word \mathbf{q} , read from the input formula.

3.4 APPLYING r-2 TO ‘ $\mathbf{p} \vee + \mathbf{q}$ ’

$$\begin{array}{l}
 \text{r-2:} \quad \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \quad \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \quad \Rightarrow \quad \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \beta \\ \text{ctp: } z \\ \text{prn: } x \end{array} \right] \quad \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha c \\ \text{prn: } y \end{array} \right] \quad \{\text{r-1}\} \\
 \\
 \text{ss-prop: } \boxed{e} \rightarrow \text{nw-ctp} \\
 \text{ss-ctn: } \boxed{a} \rightarrow \text{nw-ctp} \\
 \text{nw-prop } \boxed{a} \rightarrow \text{ss-ctn} \\
 \text{copy}_{ss}, \text{copy}_{nw} \\
 \\
 \left[\begin{array}{l} \text{prop: } \mathbf{p} \\ \text{ctn: } \vee \\ \text{ctp:} \\ \text{prn: } 1 \end{array} \right] \quad \left[\begin{array}{l} \text{prop: } \mathbf{q} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{array} \right] \quad \Rightarrow \quad \left[\begin{array}{l} \text{prop: } \mathbf{p} \\ \text{ctn: } \vee \mathbf{q} \\ \text{ctp:} \\ \text{prn: } 1 \end{array} \right] \quad \left[\begin{array}{l} \text{prop: } \mathbf{q} \\ \text{ctn:} \\ \text{ctp: } \mathbf{p} \vee \\ \text{prn: } 2 \end{array} \right]
 \end{array}$$

The operations of r-2 copy the name of ss proplet into the ctp slot of the next proplet, called nw, (ss-prop $\boxed{e} \rightarrow$ nw-ctp), the value of the ctn slot of the ss into the ctp slot of the nw (ss-ctn $\boxed{a} \rightarrow$ nw-ctp), and the name of the nw into the ctn slot of the ss (nw-prop $\boxed{a} \rightarrow$

¹⁰Syntactically, this LA-grammar is finite state, and its rules do not need to represent the sentence start (cf. Hausser 1989, pp. 167 f.). The current formulation was chosen to resemble the standard format of more powerful systems, based on matching rule patterns with input/output expressions.

ss-ctn). The ss and the nw are both retained in the result (copy_{ss} copy_{nw}). The proposition number 2 is assigned to the nw by the control structure.

The successful application of rule r-2 activates its rule package. Its only rule, r-1, is applied to the new start and the next word \vee , read from the input formula.

3.5 APPLYING r-1 TO ‘ $\mathbf{p} \vee \mathbf{q} + \vee$ ’

$$\begin{array}{l}
 \text{r-1:} \quad \begin{bmatrix} \text{prop: } \alpha \\ \text{ctn:} \\ \text{ctp: } z \\ \text{prn: } x \end{bmatrix} \quad [\text{conn: } c] \quad \Rightarrow \quad \begin{bmatrix} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{bmatrix} \quad \{\text{r-2}\} \\
 \\
 \text{nw-conn } \boxed{c} \rightarrow \text{ss-ctn} \\
 \text{copy}_{ss} \\
 \\
 \begin{bmatrix} \text{prop: } \mathbf{q} \\ \text{ctn:} \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix} \quad [\text{conn: } \vee] \quad \Rightarrow \quad \begin{bmatrix} \text{prop: } \mathbf{q} \\ \text{ctn: } \vee \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix}
 \end{array}$$

Again, the feature structure of the connective is discarded in the output. First, however, the name of the connective is copied into the ctn attribute of \mathbf{q} .

The successful application of rule r-1 activates its rule package. Its only rule, r-2, is applied to the new start \mathbf{q} and the next word \mathbf{r} , read from the input formula.

3.6 APPLYING r-2 TO ‘ $\mathbf{p} \vee \mathbf{q} \vee + \mathbf{r}$ ’

$$\begin{array}{l}
 \text{r-2:} \quad \begin{bmatrix} \text{prop: } \alpha \\ \text{ctn: } c \\ \text{ctp: } z \\ \text{prn: } x \end{bmatrix} \quad \begin{bmatrix} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \text{prop: } \alpha \\ \text{ctn: } c \beta \\ \text{ctp: } z \\ \text{prn: } x \end{bmatrix} \quad \begin{bmatrix} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha c \\ \text{prn: } y \end{bmatrix} \quad \{\text{r-1}\} \\
 \\
 \text{ss-prop: } \boxed{c} \rightarrow \text{nw-ctp} \\
 \text{ss-ctn: } \boxed{a} \rightarrow \text{nw-ctp} \\
 \text{nw-prop } \boxed{a} \rightarrow \text{ss-ctn} \\
 \text{copy}_{ss}, \text{copy}_{nw} \\
 \\
 \begin{bmatrix} \text{prop: } \mathbf{q} \\ \text{ctn: } \vee \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix} \quad \begin{bmatrix} \text{prop: } \mathbf{r} \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \text{prop: } \mathbf{q} \\ \text{ctn: } \vee r \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix} \quad \begin{bmatrix} \text{prop: } \mathbf{r} \\ \text{ctn:} \\ \text{ctp: } q \vee \\ \text{prn: } 3 \end{bmatrix}
 \end{array}$$

The control structure of the parser assigns the proposition number 3 to the lexical feature structure of \mathbf{r} . The copying operations of r-2 write the name and ctn value of \mathbf{q} into the ctp slot of \mathbf{r} , and add the name of \mathbf{r} to the ctn slot of \mathbf{q} . This parsing procedure may be continued indefinitely, turning arbitrarily long CNF sequences of propositional calculus into corresponding sets of proplets.

4 Storage and basic navigation in the database

Parsing formulas of propositional calculus with *LA-pci* results in *sets* of proplets. The parsing is strictly time-linear, analyzing the input from left to right, but the output of the parser is independent of any graphical constraints:

4.1 SEMANTIC REPRESENTATION OF ‘ $p \vee q \vee r \ \& \ \tilde{p} \vee s \vee q$ ’

$\begin{bmatrix} \text{prop: } p \\ \text{ctn: } \vee q \\ \text{ctp:} \\ \text{prn: } 1 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } q \\ \text{ctn: } \vee r \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } r \\ \text{ctn: } \& \tilde{p} \\ \text{ctp: } q \vee \\ \text{prn: } 3 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } \tilde{p} \\ \text{ctn: } \vee s \\ \text{ctp: } r \& \\ \text{prn: } 4 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } s \\ \text{ctn: } \vee q \\ \text{ctp: } \tilde{p} \vee \\ \text{prn: } 5 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } q \\ \text{ctn:} \\ \text{ctp: } s \vee \\ \text{prn: } 6 \end{bmatrix}$
---	---	---	---	---	---

The storage of such a set in the data structure of a word bank consists in adding the completed proplets at the end of their respective token lines (reordering).

4.2 STORING ‘ $p \vee q \vee r \ \& \ \tilde{p} \vee s \vee q$ ’ IN A WORD BANK

<i>types</i>	<i>proplets</i>	
$\begin{bmatrix} \text{prop: } p \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix}$	$\begin{bmatrix} \text{prop: } p \\ \text{ctn: } \vee q \\ \text{ctp:} \\ \text{prn: } 1 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } \tilde{p} \\ \text{ctn: } \vee s \\ \text{ctp: } r \& \\ \text{prn: } 4 \end{bmatrix}$
$\begin{bmatrix} \text{prop: } q \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix}$	$\begin{bmatrix} \text{prop: } q \\ \text{ctn: } \vee r \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{bmatrix}$	$\begin{bmatrix} \text{prop: } q \\ \text{ctn:} \\ \text{ctp: } s \vee \\ \text{prn: } 6 \end{bmatrix}$
$\begin{bmatrix} \text{prop: } r \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix}$	$\begin{bmatrix} \text{prop: } r \\ \text{ctn: } \& \tilde{p} \\ \text{ctp: } q \vee \\ \text{prn: } 3 \end{bmatrix}$	
$\begin{bmatrix} \text{prop: } s \\ \text{ctn:} \\ \text{ctp:} \\ \text{prn:} \end{bmatrix}$	$\begin{bmatrix} \text{prop: } s \\ \text{ctn: } \vee q \\ \text{ctp: } \tilde{p} \vee \\ \text{prn: } 5 \end{bmatrix}$	

The purpose of storing proplets in this manner is (i) easy retrieval based on (ii) easy storage plus (iii) easy navigation.

Storage of a proplet is based on the name of the proplet and the temporal order of its arrival. Retrieval is based on the name of the proplet searched for and its proposition number. Navigation from the current proplet to the next is based on the retrieval of the next as specified in the current one.

Simple navigation (e.g., without consistency checking) serves to *activate* the content traversed. In our reconstruction, it is powered by an LA-grammar called LA-propositional_calculus_navigation or *LA-pcn* for short, and defined as follows:

4.3 DEFINITION OF *LA-pcn*

$LX =_{def}$ proplets in a word bank

Variable definition: $\alpha, \beta \in \{p, \tilde{p}, q, \tilde{q}, r, \tilde{r}\dots\}$; $c \in \{\vee, \&\}$; $n, n' \in \mathbb{N}$;

$w, x, y, z =_{def}$ optional values.

$$ST_S =_{def} \left\{ \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \beta \\ \text{ctp:} \\ \text{prn: } n \end{array} \right] \{r-1\}, \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \{r-2\} \right\}$$

The operations of *LA-pcn* do not modify the input to the rules, – in contradistinction to the copying operations of *LA-pci* as defined in 3.2.

5 Propositional calculus consistency navigation

The next step of the reconstruction consists in integrating the truth conditions of propositional calculus into the navigation algorithm of *LA-pcn*. For this, the different ontologies of predicate calculus and its reconstruction must be taken into account: propositional calculus is a metalanguage-based theory relating propositions and the world, while the reconstruction is a procedural theory treating propositions as database assertions.

This difference is crucial for avoiding equivalence between consistency navigation and the problem of SAT mentioned at the end of the Introduction. The SAT problem is to determine for arbitrary CNF formulas whether there exists a value assignment which makes it true. SAT is based on the following assumptions:

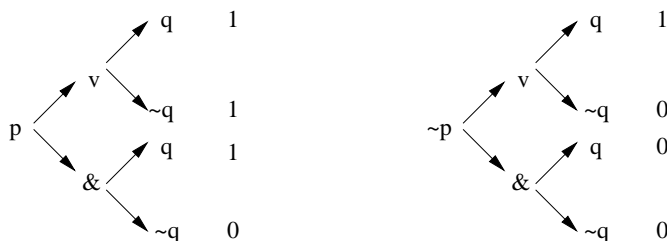
1. *Arbitrary values*: for checking consistency, propositional constants may be assigned arbitrary values, e.g., **p** may be 1 or 0.
2. *Coreference*: in a formula, e.g., ...**p**...**p**..., different occurrences of **p** must have the same truth value.

As a consequence, the SAT algorithm¹¹ must keep track of previous truth value assignments in a formula, which is why determining satisfiability for arbitrary formulas is exponential in the worst case and *NP*-complete.

For a database, however, the above assumptions are not realistic because one would not randomly switch truth values to see whether there exists a value assignment satisfying a sequence of concatenated propositions. Instead, some propositions in the database are asserted to be 1 (true), e.g., **p**, while others are asserted to be 0 (false), e.g., $\tilde{\mathbf{q}}$.

As a result there is no need to remember which value was assigned to earlier occurrences of, for example, **p** in a formula, and consistency checking may be described in terms of the following continuations:

5.1 BRANCHING STRUCTURE OF ‘&’ AND ‘∨’ WITH TRUTH VALUES



On the left-hand side, the navigation starts with **p**. The first continuation alternative is whether the connective is \vee or $\&$. For each connective, the second continuation alternative is whether the next proposition is unnegated (**q**) or negated ($\tilde{\mathbf{q}}$).

On the right-hand side, the navigation starts with $\tilde{\mathbf{p}}$. The subsequent continuation alternatives are the same as those on the left hand side. Because of the different starts, however, the truth values resulting on the left differ from those resulting on the right-hand side.

¹¹An *LA*-grammar for SAT is defined in Hausser 1992, footnote 19. See also Hausser 1989, pp. 157 f., where SAT is treated as a lexically ambiguous *LA*-grammar, and Hausser 1999/2001, p. 218.

The continuation patterns 5.1 may be interpreted in the traditional way by focusing on the connectives: the upper half of 5.1 may be reassembled into the traditional truth table for \vee , the lower half into that for $\&$. But what about tautologies and contradictions?

Consider the following comparison of the traditional value assignments in predicate calculus and its reconstruction in Database Semantics:

5.2 TAUTOLOGIES, CONTRADICTIONS, AND CONTINGENCIES IN DBS

tautology contradiction contingent complex propositions

	$\mathbf{p} \vee \tilde{\mathbf{p}}$	$\mathbf{p} \& \tilde{\mathbf{p}}$	$\mathbf{p} \vee \tilde{\mathbf{q}}$	$\mathbf{p} \& \tilde{\mathbf{q}}$	$\mathbf{p} \vee \mathbf{q}$	$\mathbf{p} \& \mathbf{q}$
<i>traditional logic:</i>	1 0	1 0	1 1	1 1	1 1	1 1
	0 1	0 1	1 0	1 0	1 0	1 0
			0 1	0 1	0 1	0 1
			0 0	0 0	0 0	0 0
DBS reconstruction:	1 0	1 0	1 0	1 0	1 1	1 1

In traditional logic, a tautology like $\mathbf{p} \vee \tilde{\mathbf{p}}$ and a contradiction like $\mathbf{p} \& \tilde{\mathbf{p}}$ each has two value assignments, while a contingent proposition like $\mathbf{p} \vee \tilde{\mathbf{q}}$ has four.¹²

From the purpose of checking the consistency of a DBS navigation, these multiple value assignments are redundant. Furthermore, all possible truth conditional constellations of propositional calculus may be expressed equivalently by assigning truth values in accordance with whether or not a propositional constant carries external negation. For example, instead of choosing between the four possible values (1 0), (1 1), (0 1), and (0 0) as assignments to $(\mathbf{p} \vee \tilde{\mathbf{q}})$, the same truth conditional constellations may be expressed by choosing between the DBS assertions $(\mathbf{p} \vee \tilde{\mathbf{q}})$, $(\mathbf{p} \vee \mathbf{q})$, $(\tilde{\mathbf{p}} \vee \mathbf{q})$, and $(\tilde{\mathbf{p}} \vee \tilde{\mathbf{q}})$.

The only aspect of traditional logic which is somewhat reduced in the DBS reconstruction are the truth conditions of tautologies and contradictions as compared to contingent propositions. For example, $\mathbf{p} \vee \tilde{\mathbf{p}}$ (tautology) and $\mathbf{p} \vee \tilde{\mathbf{q}}$ (contingent) are evaluated the same, i.e., 1, because DBS does not assign 0 to \mathbf{p} . Similarly, $\mathbf{p} \& \tilde{\mathbf{p}}$ (contradiction) and $\mathbf{p} \& \tilde{\mathbf{q}}$ (contingent) are evaluated the same, i.e., 0, because DBS does not assign 1 to $\tilde{\mathbf{q}}$.¹³

With the complexity issue raised by SAT out of the way, let us consider how to integrate the truth conditions of propositional calculus into a consistency navigation. When traversing concatenated propositions like $\mathbf{p} \vee \tilde{\mathbf{q}} \vee \mathbf{r} \& \mathbf{s} \vee \mathbf{t}$, etc., the navigation continues as long as (i) the concatenated assertions are consistent and (ii) possible continuations are available. When the end of the concatenation is reached, the navigation terminates in a legal final state. It terminates in an error, however, as soon as the concatenated propositions traversed turn out to be inconsistent.

The tricky question is how to handle transitions in which the truth value is undetermined (#) because the remainder of the formula has not yet been read. Consider, for example, $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \& \tilde{\mathbf{p}} \vee \tilde{\mathbf{q}} \vee \mathbf{r}$. The navigation traverses the concatenated propositions from left to right. The initial propositions $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r}$ are all 1. Then the navigation reaches $\& \tilde{\mathbf{p}}$. At this point, no bivalent truth value can be assigned, for which reason the navigation goes into the state #

¹²The truth values of the complex propositions, derived via the standard truth tables from the values assigned to the elementary propositions, are omitted here to avoid cluttering of the presentation.

¹³In DBS semantics, tautologies and contradictions do not deserve special treatment because they are neither particularly interesting nor desirable as content in a database. Old logic's quasi-chemist dream of deriving all philosophical truth from the tautologies is not considered viable in DBS.

(undetermined) and continues until the following propositions decide whether the navigation returns from state # to state 1, or terminates in state 0.

The first possibility is illustrated by the following example:

5.3 CONSISTENCY NAVIGATION IN UNDEFINED TERRITORY (1)

1 $p \vee q \vee r \ \& \ \tilde{p} \vee \tilde{q} \vee r \dots$

2	1
3	1
4	#
5	#
6	1

The algorithm can only decide at the very end (line 6) whether the value # (line 4 and 5) will revert to 1, or turn to 0, bringing the navigation to a halt. If the last constant is unnegated, as the r shown above, the resulting state is 1. If r were to be replaced by \tilde{r} , the resulting state would be 0. Consider the following example:

5.4 CONSISTENCY NAVIGATION IN UNDEFINED TERRITORY (0)

1 $p \ \& \ \tilde{q} \vee \tilde{r} \vee \tilde{s} \ \& \ \tilde{t} \dots$

2	#
3	#
4	#
5	#
6	0

Here, the truth value of the whole sequence is decided by the last shown connective and propositional constant: if we changed the last connective from $\&$ to \vee and the last proposition from \tilde{t} to t , the state # resulting in line 6 would be 1 rather than 0.

6 Navigation algorithm of the reconstruction

A CNF formula may begin in eight different ways, corresponding to the start states of LA-propositional_calculus_consistency_navigation, or *LA-pccn* for short, to be defined in 6.3 below. These eight states may be characterized intuitively as follows:

6.1 INTUITIVE CHARACTERIZATION OF THE START STATES OF *LA-pccn*

	<i>start</i>	<i>value</i>		<i>start</i>	<i>value</i>
1.	$\mathbf{p} \vee \mathbf{q}$	1	5.	$\tilde{\mathbf{p}} \vee \tilde{\mathbf{q}}$	#
2.	$\mathbf{p} \vee \tilde{\mathbf{q}}$	1	6.	$\mathbf{p} \& \tilde{\mathbf{q}}$	#
3.	$\tilde{\mathbf{p}} \vee \mathbf{q}$	1	7.	$\tilde{\mathbf{p}} \& \mathbf{q}$	0
4.	$\mathbf{p} \& \mathbf{q}$	1	8.	$\tilde{\mathbf{p}} \& \tilde{\mathbf{q}}$	0

Start states 1–6 result in a continuing navigation, while states 7 and 8 terminate the navigation before it even starts.

From the start states 1–6, the navigation may be continued by means of five LA-grammar rules, which may be characterized intuitively as follows:

6.2 INTUITIVE CHARACTERIZATION OF *LA-pccn*'S CONTINUATION RULES

<i>rule name</i>	<i>connective</i>	<i>next</i>	<i>rule package</i>
r1-(11)	\vee \vee $\&$	\mathbf{q} $\tilde{\mathbf{q}}$ \mathbf{q}	{r1-(11), r2-(1#)}
r2-(1#)	$\&$	$\tilde{\mathbf{q}}$	{r3-(#1), r4-(##), r5-(#0)}
r3-(#1)	\vee	\mathbf{q}	{r1-(11), r2-(1#)}
r4-(##)	\vee	$\tilde{\mathbf{q}}$	{r3-(#1), r4-(##), r5-(#0)}
r5-(#0)	$\&$ $\&$	\mathbf{q} $\tilde{\mathbf{q}}$	{ }

Rule r1-(11) combines three continuations which maintain state 1. Rule r2-(1#) changes from state 1 into #. Rule r3-(#1) changes from state # into 1. Rule r4-(##) maintains state #. Rule r5-(#0) combines two continuations which change from state # into termination with an error (empty rule package).

To express the patterns of the states and rules, the variable definition of *LA-pccn* specifies the following restricted variables: α for unnegated start proplets, β for unnegated next proplets, γ for negated start proplets, and δ for negated next proplets. Also, for a parsimonious formulation of the rules, the variables κ and μ are defined for proplets which may be unnegated or negated.

6.3 DEFINITION OF *LA-pccn* (PART 1, FORWARD NAVIGATION)

$LX =_{def}$ proplets in a word bank.

Variable definition: $\alpha, \beta \in \{p, q, r, \dots\}$; $\gamma, \delta \in \{\tilde{p}, \tilde{q}, \tilde{r}, \dots\}$; $\kappa, \mu \in \{p, \tilde{p}, q, \tilde{q}, r, \tilde{r}, \dots\}$;
 $c \in \{\vee, \&\}$; $n, n' \in |\mathbb{N}|$; $w, x, y, z = \text{optional values (may be NIL)}$.

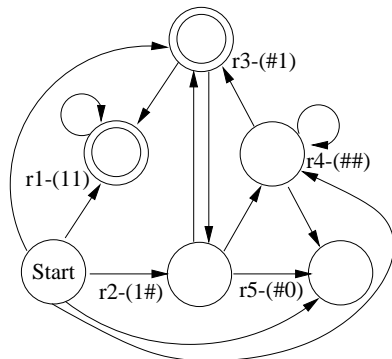
$$ST_S =_{def} \left\{ \left[\begin{array}{l} \text{prop: } \kappa \\ \text{ctn: } c \mu \\ \text{ctp:} \\ \text{prn: } n \end{array} \right], \{r1-(11), r2-(1\#), r3-(\#1), r4-(\#\#), r5-(\#0)\} \right\}$$

$$\begin{array}{l}
\text{r1-(11): } \begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } c \mu \\ \text{ctp: } w \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{prop: } \mu \\ \text{ctn: } x \\ \text{ctp: } \kappa c \\ n' \end{bmatrix} \Rightarrow \begin{bmatrix} \text{prop: } \mu \\ \text{ctn: } x \\ \text{ctp: } \kappa c \\ n' \end{bmatrix} \quad \{ \text{r1-(11), r2-(1\#)} \\
\text{r2-(1\#): } \begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \& \delta \\ \text{ctp: } w \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{prop: } \delta \\ \text{ctn: } x \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{bmatrix} \Rightarrow \begin{bmatrix} \text{prop: } \delta \\ \text{ctn: } x \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{bmatrix} \quad \{ \text{r3-(\#1), r4-(\#\#), r5-(\#0)} \} \\
\text{r3-(\#1): } \begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \vee \beta \\ \text{ctp: } w \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{bmatrix} \Rightarrow \begin{bmatrix} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{bmatrix} \quad \{ \text{r1-(11), r2-(1\#)} \} \\
\text{r4-(\#\#): } \begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \vee \delta \\ \text{ctp: } w \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{prop: } \delta \\ \text{ctn: } z \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{bmatrix} \Rightarrow \begin{bmatrix} \text{prop: } \delta \\ \text{ctn: } z \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{bmatrix} \quad \{ \text{r3-(\#1), r4-(\#\#), r5-(\#0)} \} \\
\text{r5-(\#0): } \begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \& \mu \\ \text{ctp: } w \\ \text{prn: } n \end{bmatrix} \begin{bmatrix} \text{prop: } \mu \\ \text{ctn: } z \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{bmatrix} \Rightarrow \begin{bmatrix} \text{prop: } \mu \\ \text{ctn: } z \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{bmatrix} \quad \{ \} \\
\text{ST}_F =_{\text{def}} \{ [\begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \\ \text{ctp: } \mu c \\ \text{prn: } n \end{bmatrix}, \text{rp1-(11)}], [\begin{bmatrix} \text{prop: } \kappa \\ \text{ctn: } \\ \text{ctp: } \mu \vee \\ \text{prn: } n \end{bmatrix}, \text{rp3-(\#1)}] \}
\end{array}$$

Final states showing the navigation to be consistent are defined by the application of rules r1-(11) or r3-(#1) in conjunction with suitable final proplets characterized by their empty continuation attributes. For backward navigation, the start states, rules, and final states of *LA-pccn* have to be complemented by a corresponding set of definitions.

The rules of *LA-pccn* calling rule packages constitute a finite state transition network, as in all LA-grammars:¹⁴

6.4 FINITE STATE TRANSITION NETWORK OF *LA-pccn*



¹⁴Other examples of finite state transition networks, characterizing LA-grammars for $a^k b^k c^k$ and for English and German, may be found in Hausser 1999/2001 on pp. 189, 365, 335, 333, and 338.

Arrows going into a state correspond to applications of the same rule, though from different rule packages. Arrows going out of a state correspond to different rules in the same rule package. Final states are indicated by double circles.

Next consider a schematic derivation, followed by three rule applications:

6.5 SCHEMATIC DERIVATION

<i>rules:</i>	r1-(11)	r2-(1#)	r4-(##)	r3-(#1)	r2-(1#)	r5-(#0)							
<i>continuations:</i>	p	∨	q	&	r̃	∨	ṡ	∨	t	&	ḡ	&	h
<i>values:</i>	1		#		#		1		#		0		

6.6 APPLICATION OF R1-(11) IN THE FIRST COMBINATION

$$\begin{array}{l}
 \text{r1-(11):} \\
 \left[\begin{array}{l} \text{prop: } \kappa \\ \text{ctn: } c \mu \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \mu \\ \text{ctn: } x \\ \text{ctp: } \kappa c \\ \text{prn: } n' \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \mu \\ \text{ctn: } x \\ \text{ctp: } \kappa c \\ \text{prn: } n' \end{array} \right] \quad \{\text{r1-(11), r2-(1#)}\} \\
 \\
 \left[\begin{array}{l} \text{prop: } p \\ \text{ctn: } \vee q \\ \text{ctp: } \\ \text{prn: } 1 \end{array} \right] \left[\begin{array}{l} \text{prop: } q \\ \text{ctn: } \& \tilde{r} \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } q \\ \text{ctn: } \& \tilde{r} \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{array} \right]
 \end{array}$$

6.7 APPLICATION OF R2-(1#) IN THE SECOND COMBINATION

$$\begin{array}{l}
 \text{r2-(1#):} \\
 \left[\begin{array}{l} \text{prop: } \kappa \\ \text{ctn: } \& \delta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \delta \\ \text{ctn: } x \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \delta \\ \text{ctn: } x \\ \text{ctp: } \kappa \& \\ \text{prn: } n' \end{array} \right] \quad \{\text{r3-(#1), r4-(##), r5-(#0)}\} \\
 \\
 \left[\begin{array}{l} \text{prop: } q \\ \text{ctn: } \& \tilde{r} \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{array} \right] \left[\begin{array}{l} \text{prop: } \tilde{r} \\ \text{ctn: } \vee \tilde{s} \\ \text{ctp: } q \& \\ \text{prn: } 3 \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \tilde{r} \\ \text{ctn: } \vee \tilde{s} \\ \text{ctp: } q \& \\ \text{prn: } 3 \end{array} \right]
 \end{array}$$

6.8 APPLICATION OF R4-(##) IN THE THIRD COMBINATION

$$\begin{array}{l}
 \text{r4-(##):} \\
 \left[\begin{array}{l} \text{prop: } \kappa \\ \text{ctn: } \vee \delta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \delta \\ \text{ctn: } z \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \delta \\ \text{ctn: } z \\ \text{ctp: } \kappa \vee \\ \text{prn: } n' \end{array} \right] \quad \{\text{r3-(#1), r4-(##), r5-(#0)}\} \\
 \\
 \left[\begin{array}{l} \text{prop: } \tilde{r} \\ \text{ctn: } \vee \tilde{s} \\ \text{ctp: } q \& \\ \text{prn: } 3 \end{array} \right] \left[\begin{array}{l} \text{prop: } \tilde{s} \\ \text{ctn: } \vee t \\ \text{ctp: } \vee \tilde{r} \\ \text{prn: } 4 \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{prop: } \tilde{s} \\ \text{ctn: } \vee t \\ \text{ctp: } \vee \tilde{r} \\ \text{prn: } 4 \end{array} \right]
 \end{array}$$

etc.

The upper level of 6.6 – 6.8 represents the rules, while the lower level shows the matching proplets of the database. The remaining rule applications are omitted.

7 Communicating with propositional calculus

During language production, the navigation driven by *LA-pccn* serves as the speaker's conceptualization. The conceptualization is mapped into language by (i) copying the proplets traversed into a buffer and (ii) mapping the buffer sequence of proplets into an equivalent CNF formula. Procedure (ii) is based on the following LA-grammar, called *LA-propositional_calculus_output*, or *LA-pco* for short:

7.1 DEFINITION OF *LA-pco*

$LX =_{def}$ proplets in a word bank

Variable definition: $\alpha, \beta \in \{p, \tilde{p}, q, \tilde{q}, r, \tilde{r}\dots\}$; $c \in \{\vee, \&\}$; $n, n' \in \mathbb{N}$;
 $w, x, y, z = \text{optional values (may be NIL)}$.

$$ST_S =_{def} \left\{ \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c\beta \\ \text{ctp:} \\ \text{prn: } n \end{array} \right] \{r-1\}, \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c\beta \\ \text{ctp:} \\ \text{prn: } n \end{array} \right] \{r-2\} \right\}$$

$$r-1: \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c\beta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \Rightarrow [\text{sur: } \alpha] [\text{sur: } c] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \{r-1, r-2\}$$

$$r-2: \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c\beta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \Rightarrow [\text{sur: } \alpha] [\text{sur: } c] [\text{sur: } \beta] \{ \}$$

$$ST_F =_{def} \left\{ [\text{sur: } \alpha] [\text{sur: } c] [\text{sur: } \beta], rp-2 \right\}$$

Language production may be illustrated by treating 4.1 as a buffer sequence of proplets resulting from an *LA-pccn* navigation through 4.2. The application of rule r-1 of *LA-pco* to the first two proplets of the sequence has the following form:

7.2 ILLUSTRATING APPLICATION OF *LA-pco* RULE R-1

$$r-1: \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c\beta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \Rightarrow [\text{sur: } \alpha] [\text{sur: } c] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn: } y \\ \text{ctp: } \alpha c \\ \text{prn: } n' \end{array} \right] \{r-1, r-2\}$$

$$\left[\begin{array}{l} \text{prop: } p \\ \text{ctn: } \vee q \\ \text{ctp:} \\ \text{prn: } 1 \end{array} \right] \left[\begin{array}{l} \text{prop: } q \\ \text{ctn: } \vee r \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{array} \right] \Rightarrow [\text{sur: } p] [\text{sur: } \vee] \left[\begin{array}{l} \text{prop: } q \\ \text{ctn: } \vee r \\ \text{ctp: } p \vee \\ \text{prn: } 2 \end{array} \right]$$

This rule application (i) realizes the name of the first proplet and (ii) its first ctn value, thus reversing the process of connective absorption shown in 3.3 and 3.5.

A similar effect results from the following application of rule r-2 of *LA-pco* to the last two proplets of 4.1:

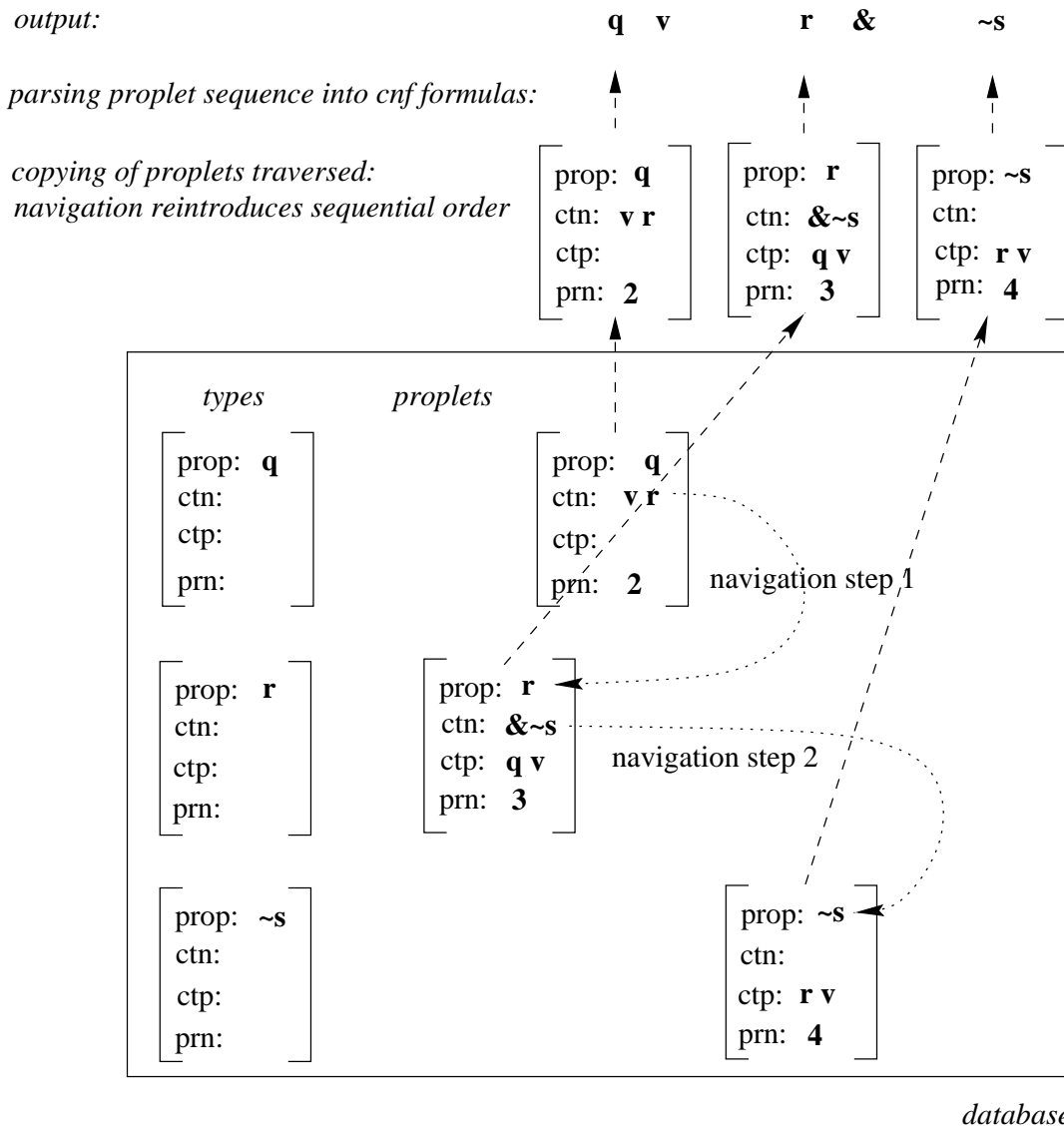
7.3 ILLUSTRATING APPLICATION OF *LA-pco* RULE R-2

$$\begin{array}{l}
 \text{r-2:} \\
 \left[\begin{array}{l} \text{prop: } \alpha \\ \text{ctn: } c \ \beta \\ \text{ctp: } w \\ \text{prn: } n \end{array} \right] \left[\begin{array}{l} \text{prop: } \beta \\ \text{ctn:} \\ \text{ctp: } \alpha \ c \\ \text{prn: } n' \end{array} \right] \Rightarrow [\text{sur: } \alpha] \ [\text{sur: } c] \ [\text{sur: } \beta] \ \{ \} \\
 \\
 \left[\begin{array}{l} \text{prop: } s \\ \text{ctn: } \vee \ q \\ \text{ctp: } \tilde{p} \ \vee \\ \text{prn: } 5 \end{array} \right] \left[\begin{array}{l} \text{prop: } q \\ \text{ctn:} \\ \text{ctp: } s \ \vee \\ \text{prn: } 6 \end{array} \right] \Rightarrow [\text{sur: } s] \ [\text{sur: } \vee] \ [\text{sur: } q]
 \end{array}$$

Rule r-2 realizes the name of the first proplet, its ctn connective, and the name of the second proplet, ending the derivation with an empty rule package.

The process of navigating with *LA-pccn*, copying the proplets traversed into a buffer, and realizing the buffer sequence as equivalent cnf surfaces is shown schematically in 7.4:

7.4 STORAGE OF PROPLETS AS WELL AS NAVIGATION AND OUTPUT

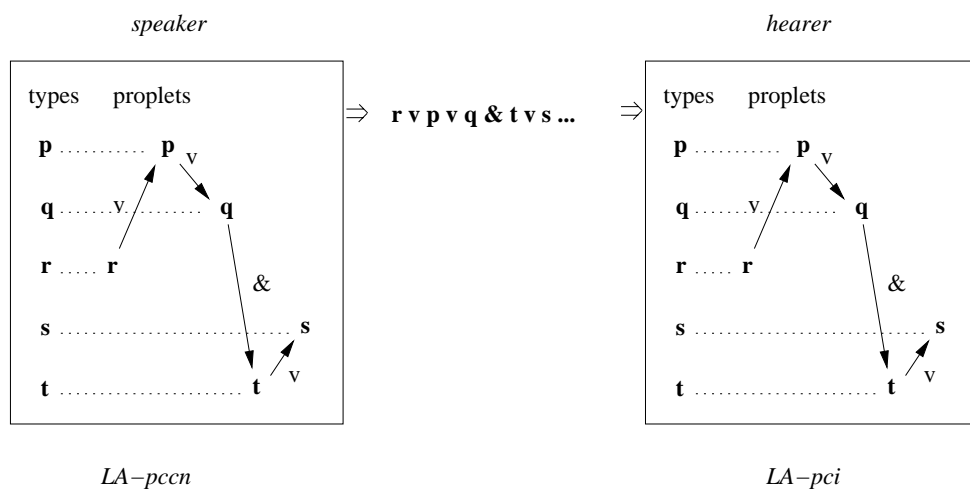


8 Summary

In DBS, communication between a speaker and a hearer is successful if a database content mapped into language by the speaker is reconstructed equivalently in the database of the hearer. Accordingly, the reconstruction of propositional calculus in DBS requires (i) a data structure for storing CNF formulas as concatenated propositions (common to the speaker and hearer), (ii) a procedure for mapping concatenated propositions into CNF formulas (speaker mode) and (iii) a procedure for mapping CNF formulas into concatenated propositions (hearer mode).

This interaction between speaker and hearer has been reconstructed using propositional calculus as a simplified form of natural language.

8.1 PROPOSITIONAL CALCULUS AS A COMMUNICATION LANGUAGE



The process of communication depicted in 8.1 is based on the data structure of a word bank, consisting of alphabetically ordered token lines (cf. 2.5 and 4.2), and three LA-grammars, called *LA-pccn* (defined in 6.3), *LA-pco* (defined in 7.1), and *LA-pci* (defined in 3.2).

In the speaker's database, CNF formulas are stored as concatenated propositions, each represented as a feature structure called proplet. This content is activated by navigating along the concatenations, using *LA-pccn* as the motor algorithm. During the navigation, consistency of the concatenated propositions traversed is checked. This process models a speaker's monitoring whether or not what he or she is currently thinking is consistent.

In addition, the navigation provides the conceptualization for language production. Thereby, the navigation is mapped into a corresponding CNF expression by (i) copying the proplets traversed into a buffer and (ii) mapping the buffer sequence into suitable surfaces. Procedure (ii) is provided by *LA-pco*.

The hearer parses the incoming surfaces using *LA-pci*. The semantic interpretation of *LA-pci* results in a set of proplets which are stored in the hearer's database. The interaction of *LA-pccn*, *LA-pco*, and *LA-pci* ensures that the content mapped by the speaker into CNF formulas is reconstructed by the hearer into an equivalent content, resulting in successful communication.

9 Outlook

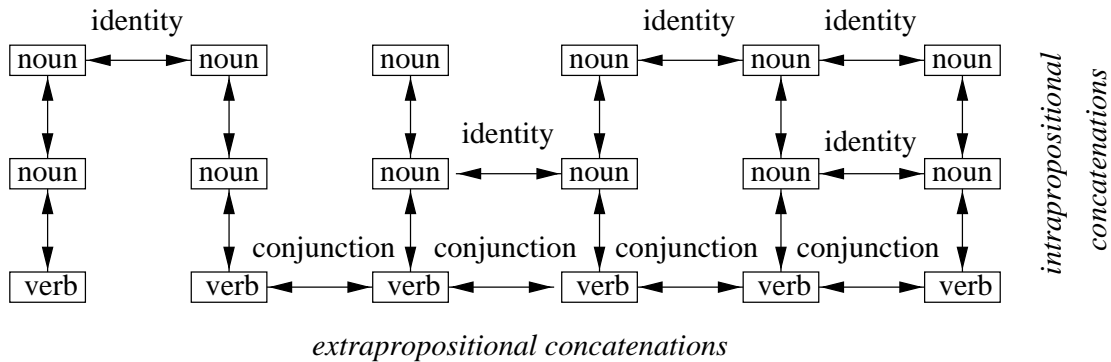
As a language, propositional calculus is especially simple in that it uses only one type of extrapositional relations, namely the logical connectives. As a consequence, there are

only two types of navigation, namely forward and backward.¹⁵

This situation changes significantly, however, when we move from propositional calculus to predicate calculus with its distinction between functors and arguments. These are represented as proplets for verbs and nouns, whereby the logical connectives are reconstructed as conjunctions between the proplets of verbs. In addition, a new kind of extrapropositional relation is defined, namely the identity relation between the proplets of nouns.

Consider the following schematic presentation of the railroad system of predicate calculus, assuming only elementary nouns and two-place verbs:

9.1 SCHEMATIC RAILROAD SYSTEM OF PREDICATE CALCULUS



The nouns and two-place verbs forming propositions are distributed all over the data structure. In line with the coding technique of DBS, all intra- and extrapropositional concatenations are established solely in terms of attributes, i.e., proplet name, proposition number, identity, and conjunction.

Extrapropositional relations are shown in 9.1 as horizontal double arrows, while intrapropositional relations are shown as vertical double arrows. Each proposition of predicate calculus may have up to three extrapropositional relations, one based on verbal conjunction and two based on nominal identity.

As a consequence, the navigation through a data structure containing proplets of predicate calculus is confronted at each point with a choice between several possible continuations – in contradistinction to propositional calculus. This choice may be made either at random or by constructing a suitable control structure. The latter requires a kind of component which is indispensable in DBS for a procedural definition of semantic primitives, but missing in old logic, namely contextual (or non-verbal) recognition and action.

10 Conclusion

The reconstruction of propositional calculus and the prospective reconstruction of predicate calculus may be regarded conceptually as a step by step upscaling from well-known formal languages to a model of natural language communication as defined in DBS. This approach is well-suited to integrate the important results of traditional logic into DBS. It also serves to highlight the differences between the metalanguage-based approach to natural language meaning in terms of truth conditions (Montague 1974) and the procedural approach of Database Semantics, designed as a functional model of communication based on the cognitive operations of agents which use language as speakers and hearers.

¹⁵For simplicity, language production based on *LA-pco* is defined here only for forward navigation.

References

- Frege, G. (1879) "Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought," in J. van Heijenoort (ed.) *Frege and Gödel, Two Fundamental Texts in Mathematical Logic*, translated into English by S. Bauer-Mengelberg, Harvard University Press, Cambridge, Mass.
- Hausser, R. (1989) *Computation of Language, An Essay on Syntax, Semantics and Pragmatics in Natural Man-Machine Communication*, Springer-Verlag, Berlin-New York.
- Hausser, R. (1992) "Complexity in Left-Associative Grammar," *Theoretical Computer Science*, Vol. 106.2:283-308, Elsevier, Dordrecht.
- Hausser, R. (1999/2001) *Foundations of Computational Linguistics: Human-Computer Communication in Natural Language*, Springer-Verlag, Berlin-New York.
- Hausser, R. (2001a) "The Four Basic Ontologies of Semantic Interpretation," in H. Kargassalo et al. (eds) *Information Modeling and Knowledge Bases XII*, IOS Press Ohmsha, Amsterdam.
- Hausser, R. (2001b) "Database Semantics for Natural Language," *Artificial Intelligence*, Vol. 130.1:27-74, Elsevier, Dordrecht.
- Hopcroft, J.E. & Ullman, J.D. (1979) *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass.
- Montague, R. (1974) *Formal Philosophy*, Yale University Press, New Haven.