

ABOUT NATURAL LOGIC*

A. COLMERAUER and J.F. PIQUE

Abstract

The use of first-order logic in data bases raises problems about representing relations that are not defined everywhere. The solution to this problem is to use a 3-valued logic with the truth value "undefined". This paper is concerned with the naturalness of such a logic, its power and its connection with classical logic.

Introduction

Many problems which arise in knowledge representation with classical 2-valued logic come from the absence of the notion of presupposition. As Keenan (1972) has shown, at the least, a 3-valued logic must be introduced to deal with presuppositions in natural languages. We present here a 3-valued extension of classical logic which, from our point of view, is convenient and natural. After an informal introduction to the basis of our system, we study in detail the Boolean algebra which underlines it; this algebra is an extension of the one developed by D.A. Bochvar as described in Rescher (1969). Then we define precisely the whole system with its quantifiers. It turns out that every deduction can be transformed into a deduction in classical 2-valued logic. This is a very interesting result. We conclude by describing very briefly the question-answering (QA) system which originated this paper and was implemented by J.F. Pique (1978). This system, which deals with natural language, was influenced deeply by the work described in A. Colmerauer (1979) and in R. Pasero (1980). Readers who are interested by the treatment of presupposition in linguistics may consult L. Karttunen (1963) and R. Zuber (1972).

* réimpression de l'article paru dans «Advances in Data Base theory, vol. 1», Phenom Press 80, avec l'aimable autorisation des éditeurs.

*Presupposition and 3-Valued Logic**Natural Language and Presupposition*

Any assertion in natural language carries a certain number of presuppositions: if we assert

(1) toc saw mary

without knowing anything about toc, anyone will conclude that toc is a being capable of seeing *and* that mary was in his field of vision. But if we say

(2) toc did not see mary

in this case we conclude that toc is a being capable of seeing *and* that mary was *not* in his field of vision. Thus it appears that an assertion in natural language links two types of properties. A property which cannot be denied, which we shall call a presupposition, and which can be seen as the semantic field of the assertion, and a description property which is asserted as true or false (in the case of a negation). To represent such a behavior we shall introduce the presupposition operator “if” with the fundamental properties:

(3) $\text{if}(p,q) = \text{true} \Leftrightarrow \text{and}(p,q) = \text{true}$

(4) $\text{not}(\text{if}(p,q) = \text{true}) \Leftrightarrow \text{if}(p, \text{not}(q)) = \text{true}$

where p is a presupposition and q a description property. We notice that the operator “if” enables us to introduce quite naturally the domains used in question answering systems for limiting the scope of negative questions: to the questions

(5) who does not distribute the product U1?

(6) who distributes the product U1?

we are only interested in the answers for which certain properties (domains) are true (for example bolt 49, table 66... will not be considered as valid answers). The domains of the relationship “ x distributes the product y ” are respectively the persons for x and the products for y . These domains appear in fact as presuppositions on the arguments of the relationship. If we represent this by something like:

- (7) $\text{distribute}(x,y) = \text{if}(\text{and}(\text{person}(x), \text{product}(y)), \text{isdistributorof}(x,y))$

we can see that using (3) and (4), the questions

- (8) $x ? \text{not}(\text{distribute}(x,U1)) = \text{true}$
 (9) $x ? \text{distribute}(x,U1) = \text{true}$

will give respectively the correct answers to the questions (5) and (6).

What happens when a presupposition is not true? If we find out for example that *toc* is a second name for the object *bolt 49*, the assertion (1) cannot be true, but its negation (2) seems hardly any more acceptable. This is still clearer if we consider the property “work with”; the assertion:

- (10) *john works with himself*

is not true, but its negation

- (11) *john does not work with himself*

is not true either. It appears then that two values of truth are not sufficient to account for the properties of natural language. We shall therefore introduce the truth value “undefined” to represent the idea of “absurd” or “meaningless”. Thus the presupposition operator “if” is such that:

- (12) $p \neq \text{true} \Rightarrow \text{if}(p,q) = \text{undefined}$.

If we consider that the relation “*x works with y*” presupposes that *x* and *y* are persons and that *x* is different from *y*, from (12) we deduce immediately that (10) and (11) are undefined. If we ask questions like

- (13) *who works with john?*
 (14) *who does not work with john?*

with

$$x \text{ works with } y = \text{if}(\text{and}(\text{person}(x), \text{differentfrom}(x,y)), \text{isworkmate}(x,y))$$

we shall never obtain abnormal answers such as (10) or (11). One notices that a QA system provided simply with domains does not enable us to avoid such anomalies. From the above we define the

connector “if” and extend the connector “not” thus:

if (p,q) =
 undefined if p is not true
 q otherwise
 not (p) =
 undefined if and only if p is undefined.

If we remember that “undefined” stands for “meaningless”, it seems quite natural to say that the conjunction or the disjunction of a proposition with something meaningless yields a sentence which can neither be true nor false as for example:

john sells or eats cars

So we extend “and” and “or” to three values by stating:

and (p,q) = or (p,q) =
 undefined as soon as one of the propositions is undefined

One can notice that these extensions preserve De Morgan laws.

Quantifiers and Presupposition

Sentences in natural language contain articles. The latter may be considered as quantifiers joining a statement around a common noun introduced by the article, to a statement built around the verb of the sentence (Colmerauer (1979)). We shall represent them by quantifiers with three arguments: the first is the quantified variable, the second, the statement around the common noun, the third the statement around the verb. Let us consider the assertion:

(15) each salesman distributes U1

which we shall represent by

each (x,p1,p2)
 p1 = salesman (x)
 p2 = distribute (x,U1).

What shall we say about this assertion if there is no salesman? If we say that it is true, then to the question

(16) which products does each salesman distribute?

we shall obtain the answer U1 (in fact all the products) since (15) is true, which is unacceptable. If we say it is false, then the assertion

(17) a salesman does not distribute U1

ought to be true since it is the negation of (15). As it is obviously not true, it appears that the assertion (15) is undefined in this case. So the article "each" in (15) introduces the presupposition that an entity satisfies the property "to be a salesman". The presupposition is even stronger because:

(18) each bolt distributes U1

will also be undefined because, even if entities satisfying the statement p_1 exist, the statement p_2 will not be defined for any of these. Moreover, one cannot impose that p_2 is defined for all entities satisfying p_1 , since the sentence

(19) each person works with john

would be undefined because, for the person john, the property "works with" is undefined. So it appears that "each(x, p_1, p_2)" presupposes that there exists at least one entity for which p_1 is true and p_2 is defined. The value of a proposition "each(x, p_1, p_2)" will then be

- undefined if there exists no entity such that p_1 is true and p_2 is defined
- false if there exists an entity such that p_1 is true and p_2 is false,
- true if there exists an entity such that p_1 and p_2 are true and, for all entities, p_1 true implies p_2 true or undefined.

We notice that to a question like

(2) who distributes each product?,

represented by,

(21) $x? \text{ each}(y, \text{product}(y), \text{distribute}(x, y)) = \text{true},$

we can only obtain the answer "nil" if the products and persons likely to distribute them actually exist. If not, the question will be undefined and we can reply "there is a false presupposition in your question".

N.B. In a question like (20) where the question deals with an

argument of the proposition, and not with its truth value, the role of "each" can be used to introduce a multiple question (see Pique [1978] for a discussion on this subject). In order to remain clear we shall restrict ourselves to sentences outside these cases, which does not alter in any way the validity of our system.

The quantifier " $a(x,p1,p2)$ " will enable us in the same way to express assertions like

(22) "a car is blue"

$p1 = \text{car}(x)$

$p2 = \text{isblue}(x)$

we shall define the value of " $a(x,p1,p2)$ " as

- undefined if there exists no entity such that $p1$ is true and $p2$ is defined,
- true if there exists an entity such that $p1$ and $p2$ are true,
- false if there exists an entity such that $p1$ is true and $p2$ is false and, for each entity, when $p1$ is true $p2$ is false or undefined.

The value of the proposition " $no(x,p1,p2)$ " is deduced immediately by observing that it is the negation of " $a(x,p1,p2)$ ":

$$no(x,p1,p2) = \text{not}(a(x,p1,p2)).$$

One can notice moreover that

$$\text{each}(x,p1,p2) = no(x,p1,\text{not}(p2)).$$

Let us now see if it is possible, with some extensions, to express the natural quantifiers "a", "each", "no" by the quantifiers "exist" and "all" of classical logic. If we look at the definition of the value of the proposition " $\text{each}(x,p1,p2)$ " for example, one can note that this definition can be formulated solely in relation to the value of " $\text{if}(p1,p2)$ ", instead of those of $p1$ and $p2$ separately (for example " $\text{each}(x,p1,p2)$ " will be undefined when " $\text{if}(p1,p2)$ " is undefined for any x). This leads us to generalize the quantifier " $\text{all}(x,p)$ " to three values as follows:

$\text{all}(x,p)$ will be

- undefined iff for all x , p is undefined,
- false iff there exists x such that p is false,

- true iff whatever x is, p is true or undefined, and
 there exists x such that p is true.

We have, therefore:

$$\begin{aligned}\text{each}(x, p_1, p_2) &= \text{all}(x, \text{if}(p_1, p_2)), \\ \text{no}(x, p_1, p_2) &= \text{all}(x, \text{if}(p_1, \text{not}(p_2))).\end{aligned}$$

We shall also generalize the existential quantifier by:

- $\text{exist}(x, p)$ will be
- undefined iff for all x , p is undefined,
 - true iff there exists x such that p is true,
 - false iff whatever x is, p is false or undefined, and
 there exists x such that p is false.

This leads us to the relationship,

$$a(x, p_1, p_2) = \text{exist}(x, \text{if}(p_1, p_2)).$$

Note that the relations of classical logic are maintained:

$$\begin{aligned}\text{not}(\text{exist}(x, p)) &= \text{all}(x, \text{not}(p)), \text{ and} \\ \text{not}(\text{all}(x, p)) &= \text{exist}(x, \text{not}(p)),\end{aligned}$$

but that the quantifiers “exist” and “all” cannot be seen as a generalization of “and” and “or”. Various types of quantifiers have been defined and studied in A. Mostowski [1957].

The Adequate 3-Valued Boolean Algebra

Before formalizing a complete logical system with quantification, we will first define and study the 3-valued algebra which we have introduced informally.

Basic Functions

Let us take the three truth-values:

- 1 to represent “true”
- 0 to represent “false”
- ω to represent “undefined” or “meaningless”

The classical boolean functions are extended to three values by considering that it is sufficient that one operand is undefined to produce an undefined result:

$$\begin{aligned} \text{not } [1] &= 0, \text{ not } [0] = 1, \text{ not } [\omega] = \omega, \\ \text{and } [p, q] &= \min \{p, q\} \text{ with } \omega < 0 < 1, \\ \text{or } [p, q] &= \max \{p, q\} \text{ with } 0 < 1 < \omega. \end{aligned}$$

It follows that the usual equalities still hold:

$$\begin{aligned} \text{not } [\text{not } [p]] &= p, \\ \text{not } [\text{and } [p, q]] &= \text{or } [\text{not } [p], \text{not } [q]], \\ \text{not } [\text{or } [p, q]] &= \text{and } [\text{not } [p], \text{not } [q]]. \end{aligned}$$

To deal with presuppositions we introduce:

$$\begin{aligned} \text{if } [p, q] &= \\ &\quad q \text{ if } p = 1 \\ &\quad \omega \text{ if } p \neq 1. \end{aligned}$$

The behavior of this function with regard to negation is characteristic:

$$\text{not } [\text{if } [p, q]] = \text{if } [p, \text{not } [q]].$$

It is convenient to be able to go back to a two-valued logic. The two simplest functions allowing this to be done are:

$$\begin{array}{ll} \text{true } [p] = & \text{defined } [p] = \\ 1 \text{ if } p = 1 & 1 \text{ if } p \neq \omega \\ 0 \text{ if } p \neq 1 & 0 \text{ if } p = \omega \end{array}$$

Completeness

It is interesting to show that it is possible to express any n-ary 3-valued Boolean function by the means of composition of our previous defined functions and constants: *not*, *true*, *defined*, *and*, *or*, *if*, 1, 0, ω .

First we prove that it is possible to express the functions:

$$\begin{aligned} \text{case } [\omega, p, q, r] &= p \\ \text{case } [0, p, q, r] &= q \\ \text{case } [1, p, q, r] &= r. \end{aligned}$$

We have:

$$\begin{aligned} \text{case } [k, p, q, r] = \\ \text{or } [pick[undef[k], p], \text{ or } [pick[false[k], q], \\ pick[true[k], r]]] \end{aligned}$$

with;

$$\begin{array}{lll} pick[1, p] = & undef[k] = & false[k] = \\ p \text{ if } 1 = 1 & 1 \text{ if } k = \omega & 1 \text{ if } k = 0 \\ 1 \text{ if } 1 \neq 1 & 0 \text{ if } k \neq \omega & 0 \text{ if } k \neq 0 \end{array}$$

We can now represent these three functions by:

$$\begin{aligned} pick[1, p] &= if[or[not[1], defined[p]], and[1, true[p]], \\ undef[k] &= not[defined[k]], \\ false[k] &= true[not[k]]. \end{aligned}$$

To express any n -ary function $f[p_1, \dots, p_n]$, we note that in the special case where $n = 0$ it is reduced to 1 , 0 , or ω and that in the other cases it is possible to decrease n as much as one wants by decomposing the function according to the scheme:

$$\begin{aligned} f[p_1, \dots, p_n] = \\ \text{case } [p_n, f[p_1, \dots, p_{n-1}, \omega], f[p_1, \dots, p_{n-1}, 0], \\ f[p_1, \dots, p_{n-1}, 1]]. \end{aligned}$$

A stronger result can be obtained by considering the equalities:

$$\begin{aligned} 1 &= not[0] \\ 0 &= true[if[not[p], p]] \\ \omega &= if[not[p], if[p, p]] \\ or[p, q] &= not[and[not[p], not[q]]] \\ and[p, q] &= if[defined[p], if[defined[q], true[if[p, q]]] \\ true[p] &= defined[if[p, p]] \\ defined[p] &= not[true[if[not[true[not[p]]], not[true[p]]]]. \end{aligned}$$

This result is:

Any n -ary 3-valued Boolean function ($n > 0$) can be expressed using only functions from the set $\{if, not, true\}$ or using only functions from the set $\{if, not, defined\}$

Moreover the sets $\{if, not, true\}$ and $\{if, not, defined\}$ are minimal, which means that they contain no strict subset which allows one to express every function. To prove this last point let f be a unary function. It has the properties:

$$\begin{aligned} f[\omega] &= \omega & \text{if } f \text{ is constructed only from } \{if, not\} \\ f[I] &= I & \text{if } f \text{ is constructed only from } \{if, true, defined\} \\ f[0] &\neq \omega & \text{if } f \text{ is constructed only from } \{if, true, defined\}. \end{aligned}$$

The function

$$\begin{aligned} f[p] &= \\ &0 \text{ if } p = \omega \\ &\omega \text{ if } p \neq \omega \end{aligned}$$

has none of the three properties and so it cannot be constructed only from one of the three last sets and a fortiori not from a strict subset of $\{if, not, true\}$ or $\{if, not, defined\}$.

We conclude by noting that if we had taken as primitive the function:

$$\begin{aligned} undef[p] &= \\ &I \text{ if } p = \omega \\ &0 \text{ if } p \neq \omega \end{aligned}$$

the equalities

$$\begin{aligned} defined[p] &= not[undef[p]] \\ not[p] &= if[undef[if[undef[p], undef[p]]], undef[if[p,p]]] \end{aligned}$$

would have allowed us to reduce the minimal subset $\{if, not, defined\}$ to $\{if, undef\}$

Definition of our Logical System

We now have all the elements to define, in a very precise way, the logic with quantification introduced at the beginning of the chapter. As usual, we distinguish two aspects: syntax, which deals with the way formulae look, and semantics, which deals with what they mean.

Syntax

Let F be a set of functional symbols, R a set of relational symbols and V an infinite enumerable set of variables. F_n and R_n represent respectively the set of n -ary functional and n -ary relational symbols. We define two types of formulae: terms, which are related to individuals, and propositions, which are related to truth-values.

Definition 1

The set T of terms is the smallest set of words over the alphabet $V \cup F \cup \{ (,), \text{comma} \}$, which are of one of the 3 forms:

- (1) x with $x \in V$
- (2) f_0 with $f_0 \in F_0$
- (3) $f_n(t_1, \dots, t_n)$ with $t_1, \dots, t_n \in T$ and $f_n \in F_n$.

Definition 2

The set P of propositions is the smallest set of words over the alphabet $V \cup F \cup R \cup \{ 1, 0, \Omega, \text{not}, \text{true}, \text{defined}, \text{and}, \text{or}, \text{if}, \text{exist}, \text{all}, (,), \text{comma} \}$, which are of one of the 13 forms:

- (1) 1
- (2) 0
- (3) Ω
- (4) r_0 with $r_0 \in R_0$
- (5) $r_n(t_1, \dots, t_n)$ with $r_n \in R_n$ and $t_1, \dots, t_n \in T$
- (6) $\text{not}(p)$ with $p \in P$
- (7) $\text{true}(p)$ with $p \in P$
- (8) $\text{defined}(p)$ with $p \in P$
- (9) $\text{and}(p, q)$ with $p, q \in P$
- (10) $\text{or}(p, q)$ with $p, q \in P$
- (11) $\text{if}(p, q)$ with $p, q \in P$
- (12) $\text{exist}(x, p)$ with $x \in V, p \in P$
- (13) $\text{all}(x, p)$ with $x \in V, p \in P$.

Semantics

The meaning of a formula will be the values it takes in different interpretations of the variables, the functional symbols and the

relational symbols.

To define an interpretation we need:

- the set of truth-values: $\{1, 0, \omega\}$
- a *nonempty* set D : the domain of interpretation.

Definition 3

An *interpretation* I is a mapping from $V \cup F \cup R$ to a set constructed from $\{\omega, 0, 1\}$ and D , which is such that:

- $Ix \in D$ if $x \in V$
- $If_0 \in D$ if $f_0 \in F_0$
- If_n is a mapping from D^n to D if $f_n \in F_n$
- $Ir_0 \in \{0, 1\}$ if $r_0 \in R_0$
- Ir_n is a mapping from D^n to $\{0, 1\}$ if $r_n \in R_n$.

It should be noted that the truth value ω is not really used here, which means that relational symbols are interpreted as classical 2-valued relations. We will come back to this subject later. As done by Lyndon [1964] in his nice little book, *Notes on Logic*, we introduce the notion of interpretations which differs from one another just on one variable. This helps to define the meaning of quantification:

Definition 4

The set $S[I, x]$ is the set of all interpretations that agree with I on $V \cup F \cup R$ except possibly on x .

For every formula f we can now define its value $\phi_I[f]$ in a given interpretation I .

Definition 5

The value $\phi_I[t] \in D$ of a term t in an interpretation I is defined recursively on the 3 possible forms of t :

- (1) $\phi_I[x] = Ix$
- (2) $\phi_I[f_0] = If_0$
- (3) $\phi_I[f_n(t_1, \dots, t_n)] = [If_n][\phi_I[t_1], \dots, \phi_I[t_n]]$.

Definition 6

The value $\phi_I[p] \in \{\omega, 0, 1\}$ of a proposition p in an interpretation I is defined recursively on the 13 possible forms of p :

- (1) $\phi_I[1] = 1$
- (2) $\phi_I[0] = 0$
- (3) $\phi_I[\Omega] = \omega$
- (4) $\phi_I[r_0] = Ir_0$
- (5) $\phi_I[r_n(t_1, \dots, t_n)] = [Ir_n][\phi_I[t_1], \dots, \phi_I[t_n]]$
- (6) $\phi_I[\text{not}(p)] = \text{not}[\phi_I[p]]$
- (7) $\phi_I[\text{true}(p)] = \text{true}[\phi_I[p]]$
- (8) $\phi_I[\text{defined}(p)] = \text{defined}[\phi_I[p]]$
- (9) $\phi_I[\text{and}(p, q)] = \text{and}[\phi_I[p], \phi_I[q]]$
- (10) $\phi_I[\text{or}(p, q)] = \text{or}[\phi_I[p], \phi_I[q]]$
- (11) $\phi_I[\text{if}(p, q)] = \text{if}[\phi_I[p], \phi_I[q]]$
- (12) $\phi_I[\text{exist}(x, p)] =$
 1 if there exists $J \in S[I, x]$ with $\phi_J[p] = 1$
 0 if there does not exist $J \in S[I, x]$ with $\phi_J[p] = 1$ but there
exists $K \in S[I, x]$ with $\phi_K[p] = 0$
 ω if for all $J \in S[I, x]$ we have $\phi_J[p] = \omega$
- (13) $\phi_I[\text{all}(x, p)] =$
 1 if there does not exist $J \in S[I, x]$ with $\phi_J[p] = 0$ but there
exists $K \in S[I, x]$ with $\phi_K[p] = 1$
 0 if there exists $J \in S[I, x]$ with $\phi_J[p] = 0$
 ω if for all $J \in S[I, x]$ we have $\phi_J[p] = \omega$.

Of course, *not*, *true*, *defined*, *and*, *or*, *if* are defined by

$$\begin{array}{lll}
 \text{not}[1] = 0 & \text{true}[p] = & \text{defined}[p] = \\
 \text{not}[0] = 1 & 1 \text{ if } p = 1 & 1 \text{ if } p \neq \omega \\
 \text{not}[\omega] = \omega & 0 \text{ if } p \neq 1 & 0 \text{ if } p = \omega \\
 \\
 \text{and}[p, q] = \min\{p, q\} \text{ with } \omega < 0 < 1 & & \\
 \text{or}[p, q] = \max\{p, q\} \text{ with } 0 < 1 < \omega & & \\
 \text{if}[p, q] = & & \\
 \quad q \text{ if } p = 1 & & \\
 \quad \omega \text{ if } p \neq 1. & &
 \end{array}$$

Two remarks are necessary before ending this section.

It is obvious that the value of a proposition, in general, depends on the interpretation of free-occurring variables, but not on the interpretation of bound-occurring variables.

(The notion of free and bounded occurrences of variables is assumed to be familiar to the reader.)

We interpret relational symbols by 2-valued relations instead of 3-valued relations for the following reasons:

- for the sake of simplicity and to be able, later on, to switch back smoothly to 2-valued logic,
- because it is not a restriction: any 3-valued relation r can be expressed by two 2-valued relations r' and r'' using the fact that if

$$r'[t_1, \dots, t_n] = \text{defined}[r[t_1, \dots, t_n]]$$

$$r''[t_1, \dots, t_n] = \text{true}[r[t_1, \dots, t_n]]$$

then

$$r[t_1, \dots, t_n] = \text{if}[r'[t_1, \dots, t_n], r''[t_1, \dots, t_n]].$$

Some Equivalences

It is natural to introduce the following equivalence relation between propositions:

$$p \equiv q \quad \text{iff} \quad \text{in every interpretation } I \\ \phi_I[p] = \phi_I[q]$$

It must be noted that this equivalence relation has the same property as equality with respect to substitution, that is:

if in a proposition p one replaces a subproposition q by a proposition equivalent to q , one obtains a new proposition equivalent to p .

The following equivalence rules concern the three unary connectors: “not”, “true” and “defined”. They are directly deducible from the definition of ϕ_I . The notations are those of the syntactic definition of a proposition.

$$\text{not}(1) \equiv 0$$

$$\text{not}(0) \equiv 1$$

$$\text{not}(\Omega) \equiv \Omega$$

$$\text{not}(\text{not}(p)) \equiv p$$

$$\text{not}(\text{and}(p, q)) \equiv \text{or}(\text{not}(p), \text{not}(q))$$

$$\text{not}(\text{or}(p, q)) \equiv \text{and}(\text{not}(p), \text{not}(q))$$

$$\text{not}(\text{if}(p, q)) \equiv \text{if}(p, \text{not}(q))$$

$\text{not}(\text{exit}(x,p)) \equiv \text{all}(x, \text{not}(p))$
 $\text{not}(\text{all}(x,p)) \equiv \text{exist}(x, \text{not}(p))$

$\text{true}(1) \equiv 1$
 $\text{true}(0) \equiv 0$
 $\text{true}(\Omega) \equiv 0$
 $\text{true}(r_0) \equiv r_0$
 $\text{true}(r_n(t_1, \dots, t_n)) \equiv r_n(t_1, \dots, t_n)$
 $\text{true}(\text{not}(\text{true}(p))) \equiv \text{not}(\text{true}(p))$
 $\text{true}(\text{true}(p)) \equiv \text{true}(p)$
 $\text{true}(\text{defined}(p)) \equiv \text{defined}(p)$
 $\text{true}(\text{and}(p,q)) \equiv \text{and}(\text{true}(p), \text{true}(q))$
 $\text{true}(\text{or}(p,q)) \equiv \text{and}(\text{and}(\text{defined}(p), \text{defined}(q)), \text{or}(\text{true}(p), \text{true}(q)))$

$\text{true}(\text{if}(p,q)) \equiv \text{and}(\text{true}(p), \text{true}(q))$
 $\text{true}(\text{exist}(x,p)) \equiv \text{exist}(x, \text{true}(p))$
 $\text{true}(\text{all}(x,p)) \equiv \text{and}(\text{exist}(x, \text{true}(p)), \text{not}(\text{exist}(x, \text{true}(\text{not}(p)))))$

$\text{defined}(1) \equiv 1$
 $\text{defined}(0) \equiv 1$
 $\text{defined}(\Omega) \equiv 0$
 $\text{defined}(r_0) \equiv 1$
 $\text{defined}(r_n(t_1, \dots, t_n)) \equiv 1$
 $\text{defined}(\text{not}(p)) \equiv \text{defined}(p)$
 $\text{defined}(\text{true}(p)) \equiv 1$
 $\text{defined}(\text{defined}(p)) \equiv 1$
 $\text{defined}(\text{and}(p,q)) \equiv \text{and}(\text{defined}(p), \text{defined}(q))$
 $\text{defined}(\text{or}(p,q)) \equiv \text{and}(\text{defined}(p), \text{defined}(q))$
 $\text{defined}(\text{if}(p,q)) \equiv \text{and}(\text{true}(p), \text{defined}(q))$
 $\text{defined}(\text{exist}(x,p)) \equiv \text{exist}(x, \text{defined}(p))$
 $\text{defined}(\text{all}(x,p)) \equiv \text{exist}(x, \text{defined}(p))$

Back to Two Truth-Values

Let us look at the syntactic definition of the set P of propositions described in the section DEFINITION OF OUR LOGICAL SYSTEM, Syntax. Consider the subset P' of propositions obtained by

suppressing the forms (3), (7), (8) and (11). There are therefore only 9 forms left:

- (1) 1
- (2) 0
- (4) r_0 with $r_0 \in R_0$
- (5) $r_n(t_1, \dots, t_n)$ with $r_n \in R_n$ and $t_1, \dots, t_n \in T$
- (6) $\text{not}(p)$ with $p \in P'$
- (9) $\text{and}(p, q)$ with $p, q \in P'$
- (10) $\text{or}(p, q)$ with $p, q \in P'$
- (12) $\text{exist}(x, p)$ with $x \in V$ and $p \in P'$
- (13) $\text{all}(x, p)$ with $x \in V$ and $p \in P'$.

We see now that the value of a proposition $p \in P'$ is never "undefined". This results on the one hand, from the fact that the relational symbols are interpreted as 2-valued relations and on the other hand, from the absence of the proposition " Ω " and the connector "if". Thus the definition of ϕ_I is simplified into:

- (1a) $\phi_I[1] = I$
- (2a) $\phi_I[0] = 0$
- (4a) $\phi_I[r_0] = Ir_0$
- (5a) $\phi_I[r_n(t_1, \dots, t_n)] = [Ir_n][\phi_I[t_1], \dots, \phi_I[t_n]]$
- (6a) $\phi_I[\text{not}[p]] =$
 I if $\phi_I[p] = 0$
 0 if $\phi_I[p] = I$
- (9a) $\phi_I[\text{and}(p, q)] = \min\{\phi_I[p], \phi_I[q]\}$ with $0 < I$
- (10a) $\phi_I[\text{or}(p, q)] = \max\{\phi_I[p], \phi_I[q]\}$ with $0 < I$
- (12a) $\phi_I[\text{exist}(x, p)] =$
 I if there exists $J \in S[I, x]$ with $\phi_J[p] = I$
 0 if for all $J \in S[I, x]$ we have $\phi_J[p] = 0$
- (13a) $\phi_I[\text{all}(x, p)] =$
 I if for all $J \in S[I, x]$ we have $\phi_J[p] = I$
 0 if there exists $J \in S[I, x]$ with $\phi_J[p] = 0$.

It is pleasant to see then that things are settled back to classical two-valued first order logic, the signs $\neg, \wedge, \vee, \exists, \forall$ being respectively "not", "and", "or", "exist", and "all". We will now consider three interesting mappings from P to P' .

Definition 7

The mappings T, F, D from P to P' are defined recursively on the 13 possible forms of a proposition:

- (01a) $T[1] = 1$
- (02a) $T[0] = 0$
- (03a) $T[\Omega] = 0$
- (04a) $T[r_0] = r_0$
- (05a) $T[r_n(t_1, \dots, t_n)] = r_n(t_1, \dots, t_n)$
- (06a) $T[\text{not}(p)] = F[p]$
- (07a) $T[\text{true}(p)] = T[p]$
- (08a) $T[\text{defined}(p)] = D[p]$
- (09a) $T[\text{and}(p, q)] = \text{and}(T[p], T[q])$
- (10a) $T[\text{or}(p, q)] = \text{and}(\text{and}(D[p], D[q]), \text{or}(T[p], T[q]))$
- (11a) $T[\text{if}(p, q)] = \text{and}(T[p], T[q])$
- (12a) $T[\text{exist}(x, p)] = \text{exist}(x, T[p])$
- (13a) $T[\text{all}(x, p)] = \text{and}(\text{exist}(x, T[p]), \text{not}(\text{exist}(x, F[p])))$
- (01b) $F[1] = 0$
- (02b) $F[0] = 1$
- (03b) $F[\Omega] = 0$
- (04b) $F[r_0] = \text{not}(r_0)$
- (05b) $F[r_n(t_1, \dots, t_n)] = \text{not}(r_n(t_1, \dots, t_n))$
- (06b) $F[\text{not}(p)] = T[p]$
- (07b) $F[\text{true}(p)] = \text{not}(T[p])$
- (08b) $F[\text{defined}(p)] = \text{not}(D[p])$
- (09b) $F[\text{and}(p, q)] = \text{and}(\text{and}(D[p], D[q]), \text{or}(F[p], F[q]))$
- (10b) $F[\text{or}(p, q)] = \text{and}(F[p], F[q])$
- (11b) $F[\text{if}(p, q)] = \text{and}(T[p], F[q])$
- (12b) $F[\text{exist}(x, p)] = \text{and}(\text{exist}(x, F[p]), \text{not}(\text{exist}(x, T[p])))$
- (13b) $F[\text{all}(x, p)] = \text{exist}(x, F[p])$
- (01c) $D[1] = 1$
- (02c) $D[0] = 1$
- (03c) $D[\Omega] = 0$
- (04c) $D[r_0] = 1$
- (05c) $D[r_n(t_1, \dots, t_n)] = 1$
- (06c) $D[\text{not}(p)] = D[p]$
- (07c) $D[\text{true}(p)] = 1$
- (08c) $D[\text{defined}(p)] = 1$

$$(09c) \quad D[\text{and}(p,q)] = \text{and}(D[p], D[q])$$

$$(10c) \quad D[\text{or}(p,q)] = \text{and}(D[p], D[q])$$

$$(11c) \quad D[\text{if}(p,q)] = \text{and}(T[p], D[q])$$

$$(12c) \quad D[\text{exist}(x,p)] = \text{exist}(x, D[p]).$$

$$(13c) \quad D[\text{all}(x,p)] = \text{exist}(x, D[p]).$$

We notice that if “ $T[]$ ”, “ $F[]$ ”, “ $D[]$ ”, “ $=$ ” are respectively replaced by “ $\text{true}()$ ”, “ $\text{true}(\text{not}())$ ”, “ defined ”, “ \equiv ” these definitions are transformed in equivalences directly deducible from those described in the section PROPERTIES OF OUR LOGICAL SYSTEM, Some Equivalence. It follows that:

$$T[p] \equiv \text{true}(p)$$

$$F[p] \equiv \text{true}(\text{not}(p))$$

$$D[p] \equiv \text{defined}(p)$$

and of course $T[p]$, $F[p]$ and $D[p]$, contrariwise to p , are necessarily propositions of classical 2-valued logic.

Consider now a set of propositions p_1, \dots, p_n . Let W be the set of all the interpretations in which each p_i has the value “true”, that is, I . We may consider that W represents a partially known “world”. It is interesting to be able to deduce all the statements q which have the value “true” in this “world”. We write them $\{p_1, \dots, p_n\} \models q$. More precisely, we have the following

Definition 8

The *deduction relation* \models is defined by:

$$\begin{aligned} \{p_1, \dots, p_n\} \models q \quad \text{iff} \quad & \text{for every interpretation } I \\ & \phi_I[p_1] = I, \dots \text{ and } \phi_I[p_n] = I \\ & \text{imply } \phi_I[q] = I. \end{aligned}$$

Definition 8 is the classical definition of semantic deduction. We have:

$$\begin{aligned} \{p_1, \dots, p_n\} \models q \quad \text{iff} \\ \{\text{true}(p_1), \dots, \text{true}(p_n)\} \models \text{true}(q) \quad \text{iff} \\ \{T[p_1], \dots, T[p_n]\} \models T[q]. \end{aligned}$$

This proves that every deduction in our 3-valued logic can be reduced easily to a deduction in classical 2-valued first order logic.

Let us take an example. Consider the assertions

- P_1 : A is a table
- P_2 : B is a person
- P_3 : nobody is laughing
- P_4 : A is not laughing
- P_5 : B is not laughing.

We suppose that only persons can laugh. So we take:

- x is a table $\Leftrightarrow \text{table}(x)$
- x is a person $\Leftrightarrow \text{person}(x)$
- x is laughing $\Leftrightarrow \text{if}(\text{person}(x), \text{laugh}(x))$.

We obtain:

- $P_1 = \text{table}(A)$
- $P_2 = \text{person}(B)$
- $P_3 = \text{all}(x, \text{not}(\text{if}(\text{person}(x), \text{laugh}(x))))$
- $P_4 = \text{not}(\text{if}(\text{person}(A), \text{laugh}(A)))$
- $P_5 = \text{not}(\text{if}(\text{person}(B), \text{laugh}(B)))$.

And so:

- $T[P_1] = \text{table}(A)$
- $T[P_2] = \text{person}(B)$
- $T[P_3] = \exists x (\text{person}(x) \wedge \neg \text{laugh}(x)) \wedge \forall y (\neg \text{person}(y) \vee \neg \text{laugh}(y))$
- $T[P_4] = \text{person}(A) \wedge \neg \text{laugh}(A)$
- $T[P_5] = \text{person}(B) \wedge \neg \text{laugh}(B)$.

We want to prove that:

$$\{P_1, P_2, P_3\} \models P_5.$$

That is,

$$T[P_1], T[P_2], T[P_3] \models T[P_5].$$

We can do a proof by inconsistency (that is, deny the conclusion and show that a contradiction arises):

$$\{T[P_1], T[P_2], T[P_3], \neg T[P_5]\} \models 0$$

We transform the propositions into clauses:

- (1) table (A)
- (2) person (B)
- (3a) person (K)
- (3b) \neg laugh (K)
- (3c) \neg person (x) \vee \neg laugh (x)
- (5) \neg person (B) \vee laugh (B)

The proof by resolution gives:

- (2) and (3c) \Rightarrow (6) \neg laugh (B)
- (6) and (5) \Rightarrow (7) \neg person (B)
- (7) and (2) $\Rightarrow \square$.

By the same technique it would not have been possible to prove:

$$\{P_1, P_2, P_3\} \models P_4$$

This is reasonable since it is hard to assert that a table is not laughing, at least in data bases.

An Example Application

In this section we illustrate the application of our system to querying a data base. The example used represents a specific interpretation (closed world assumption as in Reiter [1978] and describes in a 2-valued system the relationships existing between teachers, students, language courses and residence. The true facts are represented explicitly, the false ones implicitly. The queries are made in natural language and translated by a metamorphosis grammar (Colmerauer [1978]) into a formula p of the 3-valued system. As we have a finite world and only one interpretation, we can evaluate p to obtain the answer by following recursively the definition of $\phi_I[p]$. The value of a formula can also be obtained from the evaluation of several formulas: the value of p can be deduced from the values of the three formulas "true(p)", "true(not(p))", "not(defined(p))" (which are respectively true if and only if $\phi_I[p] = 1$, $\phi_I[p] = 0$, $\phi_I[p] = \omega$). We notice that, for each value of p , one and only one of the three formulas can be true. For example we can deduce the value of p from the value of the

first and the second one. This is interesting because, as stated in the section Back to Two Truth-Values, values of the formulas "true(p)" and "true(not(p))", are respectively equal to those of the formulas $p'_1 = T[p]$ and $p'_2 = F[p]$ of the 2-valued logic. Thus we can restrict ourselves to the simpler and more classical evaluations of $\phi_I[p'_1]$ and $\phi_I[p'_2]$ of 2-valued logic.

We obtain the formulas p'_1 and p'_2 with the mapping defined also in the section Back to Two Truth-Values. Moreover, if the value of $p'_1 = T[p]$ turns out to be true, it is unnecessary to evaluate $F[p]$. Thus in most cases only one evaluation is needed since queries are true propositions in general.

The entire system is written in PROLOG (Roussel [1975]), a very high level programming language based on first order logic, and runs with virtual memory on a MOTOROLA 6800 micro computer with two floppy disks. In the trace below, the querying session is in bold capitals, English translation in italic and commentary in this font.

```

computer: J'ECOUTE
           I listen
           The system waits for a query to be entered.
user:      OU HABITE YAN?
           Where does Yan live?
computer:  NICE
           This is the answer, Nice.
computer:  J'ECOUTE
           I listen
user:      OU N'HABITE PAS YAN?
           Where doesn't Yan live?
computer:  AIX
           ARLES
           NIMES
           The answers are solely those for which the relationship
           "live" has a meaning (the only residences known
           from the system are Aix, Arles, Nice, Nimes).
user:      J'ECOUTE
           EST-CE QUE CHAQUE ETUDIANT DE GREC HABITE A NIMES?
           Does each student of Greek live in Nimes?

```

- computer: NON
no
The system is not successful in the evaluation of the transposed formula $T[p]$, but is in $F[p]$: it concludes that some student of Greek doesn't live in Nimes.
J'ECOUTE
- user: QUELS SONT LES ETUDIANTS DE GREC QUI N'HABITENT PAS NIMES?
Who are the students of Greek who do not live in Nimes?
- computer: LUC
LEON
J'ECOUTE
- user: QUELS SONT LES ETUDIANTS D'ALLEMAND QUI HABITENT A AIX?
Who are the students of German who live in Aix?
- computer: VOTRE QUESTION N'A PAS DE SENS: IL DOIT Y AVOIR UNE PRESUPPOSITION FAUSSE.
Your question is meaningless: these must be a false presupposition.
The system evaluates to false both $T[p]$ and $F[p]$ since there are no students of German in the data base; thus p has the value "undefined". We would have obtained the same answer to the query "does a student of German live in Aix?".
- computer: J'ECOUTE:
- user: EST-CE QU'UN ETUDIANT APPREND L'ALLEMAND?
Does a student learn German?
- computer: NON
no
As the presupposition is now only related to student existence, we obtain a defined answer.
- computer: J'ECOUTE:
- user: BONSOIR
Good evening.
The end of the session.

Summary

We have shown how presuppositions introduce problems in data base systems. To handle these problems in a clean manner, we defined a complete three valued logic. We then proved that it is possible to transpose any deduction in the three-valued logical system into a deduction in classical first-order logic. This made it possible for us to use techniques, devised for first order logic, for deduction based on a three-valued logic. Finally, a natural language question-answering system was implemented in the language PROLOG using the techniques described in this chapter.

REFERENCES

- [1] Colmerauer A. [1978] "Metamorphosis Grammars." In *Natural Language Communication with Computers* (L. Bolc Ed.) Lecture Notes in Computer Science 63, Springer-Verlag, 1978, 133-189.
- [2] Colmerauer A. [1979] "Un Sous Ensemble Intéressant du Français" *R.A.I.R.O.* 13, 4 (Oct. 1979), 309-336. In English as: "A Useful Subset of French," *Proceedings of the Workshop on Logic and Data Bases*, Toulouse, 1977, xxiv-1.
- [3] Karttunen L. [1963] "Implicative Verbs." *Language* 47, 2 (1963).
- [4] Keenan E.L. [1972] "On Semantically Based Grammar", *Linguistic Inquiry* 3, 4 (Fall 1972), 413-459.
- [5] Lyndon R.C. [1964] *Notes on Logic*, Van Nostrand Mathematical Studies #6, Van Nostrand Co., London, 1964.
- [6] Mostowski A. [1957] "On a Generalization of Quantifiers", *Fundamenta Mathematicae* 44, 1 (Jan. 1957), 12-36.
- [7] Pasero R. [1980] Thèse d'état, On a Logical Model for Natural Languages, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II (to appear in 1980).
- [8] Pique J.F. [1978] *Interrogation en Français d'une Base de Donnée Relationnelle*, mémoire de DEA, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II, 1978.
- [9] Reiter R. [1978] "On Closed World Data Bases." In *Logic and Data Bases* (H. Gallaire and J. Minker, Eds.), Plenum Press, New York, 55-76.
- [10] Rescher N. [1969] *Many-Valued Logic*, McGraw-Hill, New York, 1969.
- [11] Roussel Ph. [1975] *PROLOG, Manuel de Référence et d'Utilisation*, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II, 1975.
- [12] Zuber R. [1972] *Structure Presuppositionnelle du Langage*, Dunod, Paris, 1972.