STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-128

COMPUTER SCIENCE DEPARTMENT
REPORT NO. 166

REPRESENTING NATURAL-LANGUAGE INFORMATION IN PREDICATE CALCULUS

BY

ERIK J. SANDEWALL

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY

ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-128

COMPUTER SCIENCE DEPARTMENT
MEMO NO. CS166

July 1970

REPRESENTING NATURAL-LANGUAGE INFORMATION IN PREDICATE CALCULUS

by

Erik J. Sandewall

ABSTRACT: A set of general conventions are proposed for representing natural language information in many-sorted first order predicate calculus. The purpose is to provide a testing-ground for existing theorem-proving programs.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract SD-183.

INDEX
*****

0. INTRODUCTION

This paper proposes a set of general conventions for representing natural-language (= 'semantic') information in many-sorted first-order predicate calculus. The purpose of this work is, of course, to provide a testing-ground for existing theorem-proving programs, and to suggest a method for using theorem-provers for question-answering and other information retrieval. The purpose is NOT to propose any system of logic. We stress practicality, and the contents of this paper may well be quite trivial to a logician.

Our approach consists of specifying functions and relations that re-express commonly encountered constructions in natural language (e.g. 'kernel sentences', comparison of adjectives, subordinate sentences, etc.) as well as specifying the intended interpretation for, and some axioms for these functions and relations. The given set of axioms is probably incomplete, but hopefully consistent. No proof of either is given.

This approach should be contrasted with the 'monkey-banana' approach, where one selects one particular problem environment, and tries to write down a notation and a set of axioms that will handle this environment. Our reason for doing things the way we do is as follows:

When a question or a problem is given to an advice taker or another similar system, we clearly wish (in the long run) that the problem statement shall consist only of very specific statements ('Consider a room in which there is a monkey and a box'). More general statements ('If a monkey is at a box, he can climb it') shall not need to be part of the problem statement, but shall be known to the advice taker beforehand.

We should ask, therefore, what general axioms are necessary for such a system, and equally important, how we can select functions and relations so that the amount of knowledge that has to be stored away is held reasonably finite. We believe that these questions are best answered if we consider classes of semantic information first, and specific exercises afterwards.

BLANK PAGE

1. APPROACH

In this section, we shall outline in general form some of the problems that one encounters while expressing natural-language (NL) information in predicate calculus (PC). We shall also outline conventions which are claimed to handle these problems in a satisfactory manner. The details of the notation are left to later sections.

Higher-order operations

Several natural-language constructions are in a certain sense 'higher-order'. For example, if we represent 'm is expensive' (where m is an object) through

  Expensive(m)

(which is a reasonable, although not the only reasonable convention), then 'm is more expensive than n' might be well expressed through

  More(Expensive)(m,n)

where 'More' is a second-order function that maps a monary first-order predicate into a binary first-order predicate. Such a function 'More' is of course proper only if we assign an intensional interpretation to predicates such as 'Expensive'.

It is unfortunate, then, that although the technology of automatic theorem-proving has been developed rather far [see e.g. Green 1969, Allen 1978, Luckham 1978], there is very little work done on theorem-proving in higher-order logic. The paper by Darlington in the present volume is an exception. It has even been suggested [Robinson, 1969] to use present theorem-provers for simulating higher-order logic.

With this state of affairs, we propose that the 'higher-order' constructions in NL should be expressed directly in first-order PC. The method, of course, is to re-express what used to be predicates as individuals, and to use a single application predicate. Thus 'm is expensive' is to be expressed as

  IS(m,expensive)

where IS is the application predicate. We need to distinguish between individuals of two types: OBJECTS and PROPERTIES, exemplified by 'm' and 'expensive', respectively. Our other example, 'm is more expensive than n', is then expressed through

  IS(m,MORETHAN(expensive,n))

where MORETHAN is a function

  [properties * objects → properties]

with the obvious intended interpretation.

It might be objected that present theorem-provers have not been designed to handle many-sorted logic, and that a notation using many-sorted logic therefore is no better than a notation in

higher-order logic. The answer is that recent results [Luckham 1970...] indicate that under certain (generous) restrictions, ordinary resolution-based theorem-provers will handle many-sorted logic correctly without even knowing so (i.e. only correct unifications will ever be attempted).

So far, properties have only been specified intuitively as counterparts of adjectives or nouns. We shall not attempt to make the interpretation of properties more precise than this. One important point, however: we shall require that properties are something 'more than' the set of all objects that have the property (by the IS relation). In other words, we shall NOT have the following axiom:

[(∀m) IS(m,p) ≡ IS(m,q)] ⊃ [p = q]

This intentional usage of properties is necessary e.g. for our use of the function 'MORETHAN', above.

In this paper, we shall not be concerned with transformations between situations, or the logic of actions. If we were, we would probably propose that the predicate IS should have a third argument, which would be the situation in which the object has the property. For an introduction to the situation concept, see e.g. [McCarthy and Hayes, 1969]. As long as IS only has two arguments, we shall usually prefer to write it infixed, rather than prefixed. Thus we write

π IS expensive

synonymously with

IS(π,expensive)

Representation of attributes

Expressions such as 'John is the father of Peter' are represented as follows. We consider 'father' as a property, and we have a property modifying function OF of two arguments:

OF: [properties * objects → properties]

so that we can write

John IS father

and

John IS (father OF peter)

The same conventions and the same function OF are used for other similar constructions, e.g. 'son of', 'color of', 'telephone number of', etc.

Representation of sentence kernels

The simplest kind of sentence with a subject and an intransitive verb is represented in the obvious way: the subject goes into an 'object' individual; the verb into a 'property' individual. Thus 'John is running' goes into

John IS running

For transitive verbs (see, give, etc.), we use the entire verb-object constellation as a property. Thus 'John sees Mary' goes into

  John IS OBJ(seeing,mary)

Here, 'OBJ' is a function

  [properties * objects → properties]

(similar in structure to 'MORE') which enable us to compound the property from individuals that correspond to natural-language words. In this particular case, it still makes sense to write

  John IS seeing

In some other cases, this may not be so (e.g. 'John IS opposing'). In such cases, we shall say that the verb-property itself ('opposing') is a property that no object can have. In principle, it would be more attractive to add to the number of sorts, and to let e.g. 'opposing' have the sort of a 'pre-property' which can be mapped into a property, using some suitable function, but at least for the moment, we shall not bother to introduce such tight-fitting sorts. - We shall later encounter several similar cases where we must again resist the temptation to introduce too many sorts.

For verbs with several objects ('give', 'lend'), we use several functions similar to 'OBJ'. It makes sense to have a function 'TO' for what is represented in our natural language as the indirect object of a verb. For example, 'John gives Fido to Mary' would be represented as

  john IS giving OBJ fido TO mary

Other similar functions (BECAUSE, FROM, etc.) can be introduced when needed.

Notice that terms in our PC formulas are intended to denote the 'meaning' (?) of NL phrases, rather than these phrases themselves. It follows that the convention of having functions 'OBJ' and 'TO' that correspond to NL direct-object and indirect-object constructions, is motivated by convenience, rather than by logical necessity. It is OK to represent phrases involving some verbs differently (e.g. by having more functions besides 'OBJ' and 'TO'), as long as we are prepared to undertake the heavier burden in translation.

Representation of subordinate sentences

Verbs that govern a subordinate sentence, such as 'knows [that]', 'knows [whether]', 'believes', 'claims [that]', etc. make it necessary to add some more conventions for handling these subordinate sentences. We propose the following conventions:

We introduce one more sort, EVENTS, and a function

  g: [objects * properties → events]

Let m be an object and p a property (either an elementary property, such as 'expensive', or a composite property, such as 'father of John'). We express 'm believes that m is p' through

  m IS Believing(g(m,p))

where 'Believing' is a function [events → properties]. The event

'g(m,p)' then expresses the possibility or the idea that m would have the property p.

It is a matter of discretion whether we use a single-argument function 'Believing' as defined here, or a (pre-)property individual 'believing', used like in
  n IS (believing THAT g(m,p))
where THAT is an infixed, binary function.

Here, again, it is important that the property p should carry more information than merely that of being the set of all objects that have the property p. For, the statement 'm believes that n is a unicorn' must be considered to be different from the statement 'm believes that n is a zubiahi', even though the set of all unicorns equals the set of all zubiahis equals the empty set.

It is hard to find a good English mnemonic for the function g. In other European languages, we would have selected the subjunctive of the present tense of the verb 'to be' (waere, solt, sera, vore, etc.). In English, by analogy, we should write 'were'. It is unfortunate, then, that 'were' is also used for past tense. In spite of this, we shall represent g as an infixed 'WERE', and we hope that the reader will get the right associations.

With these conventions, and some suitable priority conventions which make up for the suppression of parentheses, we can write 'n believes that m is p' through
  n IS Believing m WERE p

Other similar verbs (know, claim, etc.) are handled similarly to believe.

Representation of 'knows what'

Some properties (e.g. 'father OF peter') are only held by one single object. It is reasonable to have an operator 'The' which maps such properties into objects in the obvious way. Thus 'Peter's father is tall' would be expressed as
  'The father OF peter IS tall'
or more explicitly
  '(The (father OF peter)) IS tall'
The use of 'The' may be regarded as an input convention only. One would then eliminate 'The' before the theorem-prover is let loose on a statement or a question.

Consider now a statement such as 'John knows Peter's father' or 'John knows Peter's telephone number'. In the first statement, 'knows' probably has the meaning of 'is acquainted with'. If Dick is the father of Peter, then the first statement in synonymous with 'John knows Dick'. In this case, the PC translation of the first statement is

John IS Acquainted-with The (father OF peter)
where 'Acquainted-with' is a mapping [objects → properties].

By contrast, the second statement certainly means 'John knows what Peter's telephone number is'. If Peter's telephone number is in fact 321-5678, then the second statment is not equivalent to 'John knows 321-5678'. The use of properties enables us to handle this kind of sentence. We do it by introducing a function
Knowing [properties → properties]
so that we can write
John IS Knowing (telephone-number OF peter)
with the obvious meaning. It would seem that this approach is considerably more promising than the awkward 'idea-of-telephone-number' constructions proposed by McCarthy and Hayes [1969].

Referential opacity

In the notation proposed here, all functions and relations are referentially transparent (i.e. if x = y, then f(x) = f(y) etc.). The reason why we can permit this even for expressions involving knowledge, belief, etc. is of course that in this notation, some constructions which might be expressed using equality are expressed in other ways. For example, we express 'Sir Walter Scott is the author of Waverley' through
sir-walter-scott IS (author OF waverley)
or (since there is only one author) through
sir-walter-scott = The (author OF waverley)
but not through
sir-walter-scott = Authorof(waverley)

Deductions from beliefs

It is convenient to make certain assumptions about what it means for a person to 'believe' something. The first of these assumptions is that if a person believes a, and if he also believes b, then he believes any conclusion from (a^b). (The ^ sign should not be taken too literally). Similar assumptions apply to 'knows', etc.

How can this assumption be axiomatized? We propose to do this in the following manner:

(a) We introduce functions AND, OR, Not, etc. which map events (or pairs of events) into events;

(b) We introduce one more type, that of a 'subordinate variable', which is used syntactically like a constant, but which should only occur in subordinate expressi should on The purpose of subordinate variables is to act like variables in a simulated logic that is performed on the arguments of 'Believing' (etc.).

(This is another case where we may later wish to add to the number o' types to make them fit tighter).

(c) Suppose we are planning to use the resolution operator [Robinson 1965] for deductions. We then invent a function
RESOLVE: [events * events → events]
which resolves all pairs of 'clauses' from the first and the second argument, and forms the 'conjunction' (using the function AND on events) of the 'resolvents'. The function RESOLVE must of course do 'unification' on subordinate variab- les, etc. We then have the axiom
m IS Believing e ∧ m IS Believing f ⊃
m IS Believing RESOLVE(e,f)
If we use some other inference rules instead of or together with the resolution rule, then similar functions on events and similar axioms for Believe (etc.) are introduced.

(d) During the deduction process, the function 'RESOLVE' is handled with immediate evaluation. Cf. [ ].

This would seem to be a satisfactory way of formulating the convention that 'if m believes a and m believes b, then m believes the conclusions from a^b'. It must be understood, of course, that this convention is a rather crude approximation to the psychological reality. (Even if m is a computer, rather than a human being, it is still an approximation for any reasonably interpretation of 'believes'). - The detailed development of these suggestions is left to a later paper, and shall not bother us here any further.

Analytic vs. empirical facts

We shall make another, similar convention which approximates reality. Namely, we shall attempt to distinguish between 'analytical' and 'empirical' facts. An 'analytical' fact is a fact such as 'all men are mammals'; an 'empirical' fact is a fact such as 'John is asleep'. The difference between the two is critical because of the following convention: if a is an empirical fact, and b is an analytic fact, and m believes a, then m believes any conclusion from a∧b. In other words, analytic facts are assumed to be built into all agents who are capable of believing (and knowing, etc.).

From these conventions, it immediately follows that analytical facts can not be subjected to belief, knowledge, etc. We shall therefore adopt the convention that empirical facts are exactly those facts which are expressed with the relation IS (which means they can be expressed as events, using the function WERE). Analytical facts are expressed with other relations. In particular, we need a binary relation SUB between properties, used e.g. as in
elephant SUB mammal
This relation obeys the axioms
(p SUB q) ⊃ (m IS p ⊃ m IS q)

and

(p SUB q) ⊃ (Believing(m WHERE p) SUB Believing(m WHERE q))

Notice that we do NOT have the stronger axiom

(p SUB q) ≡ (∀ m)(m IS p ⊃ m IS q)

We do not, because we want the relation SUB to express that the relationship between p and q is an analytic one.

In summary, only empiric facts can be subject to knowledge, belief, etc., and only analytic facts may be expressed with SUB (and other, similar relations, which will be introduced later on).

The distinction between analytic and empiric statements obviously has some potential philosophical overtones. We hope to avoid most of them by formulating the distinction in terms of an assumption on the verbs believe, know, etc., rather than in terms of philosophical considerations.

The predicate 'Holds'

The 'connectedness' of our set of functions and relations requires that there should be some monary relation 'Holds' such that

Holds(m WHERE p) ≡ m IS p

We shall find frequent use for this relation.

It might be argued that 'Holds(e)' is in essence an empirical fact, and that it should therefore be expressed through e.g.

e IS true

(where 'true' is a property on events). However, the only advantage would be that we could write terms of the form

e WHERE true

But this is a very dispensable feature, since we have anyway that

e WHERE true = e

We shall therefore prefer to use the predicate 'Holds'.

Summary

In this section, we have introduced the following relations and functions:

IS	[objects * properties]
OF	[properties * objects → properties]
OBJ	[properties * objects → properties]
TO	[properties * objects → properties]
WERE	[objects * properties → events]
The	[properties → objects]
RESOLVE	[events * events → events]
SUB	[properties * properties]
AND, OR	[events * events → events]
Not	[events → events]

Holds [events]

plus some specialized functions (whose definitions will later be modified):

Believing	[events → properties]
Acquainted-with	[objects → properties]
Knowing	[properties → properties]
MORETHAN	[properties → objects → properties]

These functions and relations are intended for expressing NL information in a many-sorted, first-order predicate calculus. We have given the intended interpretation of these functions and relations, and outlined the reasons for selecting these conventions.

2. NOTATION AND OTHER CONVENTIONS

Before we proceed, we shall specify the notational conventions that we will use (and which we have in fact already tacitly used).

Ortography

Binary functions and relations are usually written infixed, and with capital letters throughout: OF, WERE, IS. Functions have higher priority than relations.

Monary functions and relations (and operators, see below) are written prefixed, and with an initial capital letter: Knowing, Holds, The, Any. Monary functions have higher priority than binary ones. The arguments are not necessarily enclosed by parentheses.

Parentheses are used freely to clarify or modify the order of application of functions or relations.

Constants and variables for objects and properties are written with small letters throughout. Variables are written with only one letter.

We shall sometimes use b i f i x functions. The Algol construction 'if x then y' is an example of a bifix. A function is bifixed if it is introduced in the form

  More .. THAN ..

In such cases, we really mean to have one binary function MORETHAN of two arguments, and we write

  More tall THAN peter

when we mean

  MORETHAN(tall,peter)

An infix-to-prefix translator (in LISP) which also takes care of bifixes is available from the author.

Sorts

In the sequel, we shall need two more sorts. Thus we use first-order predicate calculus with the following sorts:

1. Objects (for physical objects, persons, etc.)
2. Properties (for counterparts of nouns (except proper names), adjectives, and some verbs)
3. Events (for hypothetical or real events in the world, e.g.
   "that IJk is peter's tel-nr"
   "that the monkey is under the bananas"
   "that the monkey jumps to the ceiling"
4. Integers
5. Locations (for spatial positions, e.g. 'in the room', 'under the

table')

'Declarations' of variables

We shall use different variable symbols for different sorts, according to the following conventions:

k,m,n	objects
p,q,r	properties
d,e,f	events
v,w	integers
l,h	locations

Finally, we use the following notation:

Q,R	modification functions (see below)
Rprop	property function corresp. to R
S(x)	literal where x is one occurrence of a term

3. AMENDMENTS TO FUNCTIONS OF SECTION 1

In this section, we shall give some additional comments on the functions that were introduced in section 1. We shall also introduce some useful additional functions which are closely related to those of section 1. The things in the present section are minor details. They were not described in section 1 because that section was supposed to give an overview of the general approach in this paper.

Boolean algebras for properties and events

We use the functions AND, OR, and Not, the relation SUB, and the constants truth and falsity in a Boolean algebra in the obvious way. (The direction of SUB is such that

e AND f SUB e

et cetera). 'e SUB f' is intended to mean that f follows analytically from f. AND, OR, and Not are the functions we need for the function RESOLVE that was outlined in last section. Axioms for this algebra can be taken from any textbook and will not be iterated in this paper.

The following axioms are more or less obvious:

Holds(Not e) ≡ ¬ Holds(e)
Holds(e AND f) ≡ Holds(e) ∧ Holds(f)

We easily obtain theorems such as

e SUB f ∧ Holds(e) ⊃ Holds(f)
Holds(e OR f) ≡ Holds(e) ∨ Holds(f)

If we have a function RESOLVE like in previous section, we also need an axiom

e AND f SUB RESOLVE(e,f)

It is convenient to have a relation EXCLUDES, defined by

e EXCLUDES f ≡ e SUB Not f

For example, we have

m WERE male EXCLUDES m WERE female

A similar algebra is set up for properties, using the same symbols for the functions and relations. Thus Not is a function

[(events → events) ∪ (properties → properties)]

and similarly for the others.

We relate the two algebras through the following axioms

m WERE Not p = Not(m WERE p)
m WERE p AND m WERE q = m WERE (p AND q)

and obtain as theorems

m WERE p OR m WERE q = m WERE (p OR q)
p SUB q ⊃ m WERE p SUB m WERE q

The last theorem agrees with our intuitive idea that the relation SUB

on properties should be used like in
boy SUB male

Property functions: Ofprop, Atprop,...

Functions like OF, OBJ, TO, etc. shall be called modification functions. They shall be assumed to obey certain axioms; e.g. if FF and GG are two arbitrary modification functions, we shall have

m FF p GG q ≡ m GG q FF p

In order to handle e.g. 'Peter knows when John goes to school', we have for each modification function OF an associated property function Ofprop [events → properties], satisfying

m IS (p OF n) ≡ n IS Ofprop (m WHERE p)

Example: John IS giving OBJ fido TO mary
is equivalent to
fido IS Objprop (John WHERE giving TO mary)
is equivalent to
mary IS Toprop (John WHERE giving OBJ fido)

In natural language, the last phrase would be 'mary is the one John gives fido to' (or, more precisely, 'mary is one that John gives fido to').

Example: peter IS Knowing Toprop (John WHERE giving OBJ fido) In natural language: Peter knows whom John gives fido to.

Knowledge and belief

Let us make the functions for expressing knowledge and belief slightly more precise. We use the following functions:

Believing	[events → properties]
Knowing-whether	[events → properties]
Knowing-that	[events → properties]
Knowing	[properties → properties]
Acquainted-with	[objects → properties]

Starting from believing (the intention of which is left unspecified), we say that a person knows 'that' an event, iff he believes it, and it holds. A person is said to know 'whether' an event, iff he either knows that the event, or knows that not the event. Furthermore, we say that a person knows a property p, iff he can determine for every object m (given by its name (assumed to be unique), rather than by a description), whether (m WHERE p). This knowledge could conceivably be implemented e.g. by main- taining a list of all objects that have the property (or of those that do not have it).

Finally, a person is acquainted with an object iff he knows, for every property p, whether the object has this property.

Examples: peter IS Knowing (tel-nr OF john)
peter IS Knowing-whether (321 WERE tel-nr OF john)
peter IS Knowing-that (321 WERE tel-nr OF john)
dick IS Knowing-that (peter WERE Knowing
(tel-nr OF john))

Paradoxes

In a previous section, we proposed that one should introduce a counterpart of variables in the event structure. When this is done (we shall not do it in this paper), it becomes possible to construct expressions which essentially involve

Holds (m WERE Knowing-that Not e)

where e is made to reference back to the WERE-expression. This is then our incarnation of the classical paradox: it is impossible to attribute a truth-value to such an expression. We can see two ways of dealing with the matter, both of which have some advantages:

(a) The ostrich (= head-in-the-sand) approach: It will be a while until mechanical theorem-provers discover this paradox. If we can trust each other with not telling the computer about it, then its theorem-prover will retain its sanity.

(b) The three-valued logic approach: The axioms above are weakened into

Holds(Not e) ⊃ ¬ Holds(e)
Holds(e AND f) ⊃ Holds(e) ∧ Holds(f)

With these (and possibly some other) conventions, we do not have any longer that

Holds(e) ∨ Holds(Not e)

so we obtain a three-valued logic on the event level (since we account for events e where

Holds(e)
Holds(Not e)
neither)

Under this approach, the function RESOLVE will have to perform resolution in a three-valued logic as described in Hayes (1969).

The function 'The' and the operators 'Any', 'Some', and 'No'

In those examples where we translate simple natural-language statements into our notation, we can gain much convenience by using the functions or operators The, Any, Some, and No.

The function The [properties → objects] assumes that the argument is a property which is satisfied by exactly one object, and has this object as value.

The operators Any and Some are used for those cases where the 'argument' is not guaranteed to satisfy the uniqueness criterium.

The expression 'Any p' (where p is a property) is used like a free variable ranging over all m such that m IS p.

The expression 'Some p' is equivalent to writing a new constant symbol (generated in a gensym-like manner) pn, and stating somewhere that pn IS p.

The expression 'No p' will only be used in a context of the form 'No p IS q', and is taken as an abbreviation for

m IS p ⊃ ¬(m IS q)

There is an obvious algorithm for rewriting expressions that involve The, Any, Some, and No into pure predicate calculus.

'The' is obviously similar, although not identical, to Church's iota operator.

The function Sizeof

We need some means of specifying when the operator 'The' may be used. It is proposed to do this through a function

Sizeof [properties → properties]

where the value ranges over properties on integers. (It is possible that this should be a separate sort, but we shall not delve into this matter). We intend

v IS Sizeof p

to mean 'exactly v different objects have the property p'. The reason why we use this formulation, rather than e.g.

Size(p,v)

is of course that we consider size to be an empirical property.

Whenever an expression of the form

S(The p)

is used, with the sub-expression 'The p' used on any level, we shall feel entitled to deduce

1 IS Sizeof p

This will later be given as a rule of inference.

The function Whatis

Finally, we need some way of handling situations where a person knows (or believes,...) something about an object which he knows by its description only. We introduce the function 'Whatis' for this purpose. If p is a property, then 'Whatis p' is taken to mean 'the object (whatever it is) that has property p', or more crudely, 'the idea of an object with property p'. The function Whatis eliminates the need for constructions such as 'idea-of-telephone-number' which are used in (McCarthy & Hayes, 1969).

Example 1: 'John's telephone number is next to Johanna's', viz. 'peter believes that John's telephone number is next to Johanna's' can be represented as:

```
The (tel-nr OF John) IS Next-to The
   (tel-nr OF Johanna)

peter IS Believing [Whatis (tel-nr OF John)
   WERE Next-to Whatis (tel-nr OF Johanna)]
```

Notice that peter may hold this belief without knowing John's or Johanna's telephone numbers. Therefore, we should not write 'The' instead of 'Whatis' in the second expression.

Example 2: Consider the two expressions

```
        peter IS Knowing-whether (Whatis (tel-nr OF John) WERE
          tel-nr OF dick)
  and
        peter IS Knowing-whether (The (tel-nr OF John) WERE
          tel-nr OF dick)
```

If John's actual telephone number is 321, then the first of the above sentences says that Peter would be able to answer correctly the question

'Do John and Dick have the same telephone number?'

whereas according to the second sentence, he would be able to answer the question

'Is 321 the telephone number of Dick?'.

In vague words, if 'The' is used, then the description is 'evaluated' during the conversation between you and me, whereas the 'Whatis' function performs a kind of quoting.

Inference rules

We shall obviously need some conventional inference rules (e.g. the resolution operator) and a rule for handling equality. It may or may not be a good idea to have special inference rules for the operators The, Any, and Some. (The alternative is to eliminate these before the deduction starts). In case we want to have such inference rules, here they are:

(1) x = y, S(x) |- S(y)

(2) m IS p, S(The p) |- S(m)

(3) S(The p) |- 1 IS Sizeof p

(4) m IS p, S(Any p) |- S(m)

(5) S(Some p) |- (∃m) m IS p ∧ S(m)

In each of these rules, we assume S to be a literal. We extend the rules to inference rules on clauses in the obvious way.

Remark: In (5), only ONE occurrence of 'Some p' in S can be substituted for at a time, 'm' can be selected as any variable which does not occur in S or p.

Some axioms

Finally, let us specify some axioms for the general ('system') functions and relations that have been introduced in this section. Axioms for more special-purpose functions (e.g. the knowledge functions) are postponed to next section. The axioms for the boolean algebras for properties and events are omitted altogether.

(1) Holds (m WHERE p) ≡ m IS p

(2a) Holds(Not e) ≡ ¬ Holds(e)

(2b) Holds(e AND f) ≡ Holds(e) ∧ Holds(f)

(3a) m WHERE Not p = Not(m WHERE p)

(3b) m WHERE (p AND q) = (m WHERE p) AND (m WHERE q)

(4) p R m SUB p

(5) p R m Q n = p Q n R m

(6) m IS p R n ≡ n IS Rprop (m WHERE p)

(7) 1 IS Sizeof p ∧ m IS p ∧ n IS p ⊃ m = n

(8) 0 IS Sizeof p ⊃ ¬(m IS p)

(9) No p IS q ≡ [(∀m) m IS p ⊃ ¬(m IS q)]

Axioms (7) and (8) need to be supplemented with more general axioms for the function Sizeof, and with an axiomatization of integers.

WERE-ification of axioms

In the next section, where axioms for special environments are given, we shall e.g. see the axiom

m IS MORETHAN(p,k) ∧ k IS MORETHAN(p,n) ⊃
m IS MORETHAN(p,n) ... (a)

This axiom is of course perfectly equivalent to

Holds(m WERE MORETHAN(p,k) AND k WERE MORETHAN(p,n) IMPLIES
m WERE MORETHAN(p,n)) ... (b)

where 'e IMPLIES f' is defined as 'Not e OR f'. However, we also want to use this axiom in deductions about beliefs: If a person believes that m i taller than k, and that k is taller than n, then certainly he believes that m is taller than n. Neither of the above axioms permits us to make this deduction about his beliefs.

For belief, we shall use an axiom

e SUB f ⊃ Believing e SUB Believing f

It is therefore reasonable to strengthen (b) into

m WERE MORETHAN(p,k) AND k WERE MORETHAN(p,n) SUB
m WERE MORETHAN(p,n) ... (c)

A 'clause form' equivalent of (c) is

[Not(m WERE MORETHAN(p,k) OR
Not(k WERE MORETHAN(p,n) OR
m WERE MORETHAN(p,n)] = truth ... (d)

Clearly, then, 'e = truth' is our way of saying 'Necessarily e' or (less mystically) 'everybody knows that e'. We should not be surprised that all analytic facts come out as identical, for the reason for introducing events was to have some object for belief, knowledge, etc., and we have already stated that analytic facts are those which are not subject to belief.

In principle, it would be preferable to state all analytic axioms in the stronger form exemplified in (c) and (d), rather than the weaker form of (a) and (b). In the sequel, we shall simply refer to these as the stronger and the weaker form, respectively. Since we consider the weaker form more natural and more legible, we shall prefer to use it. To fill the gap, we specify here the procedure whereby an axiom can be 'strengthened', i.e. transformed from the weaker to the stronger form. The procedure operates on clauses:

Let (L1, ... Ln) be a clause in the weaker form. We define a function r on literals as follows:

r('Holds e') = 'e'
r('¬ Holds e') = 'Not e'
r('m IS p') = 'm WERE p'
r('¬ m IS p') = 'Not(m WERE p)'

and undefined for other arguments. Let L1, L2, ... LJ (J ≥ 1) be those literals fr which r is defined. Construct the clause

(r(L1) OR r(L2) OR ... OR r(LJ) = truth, L(J+1),... Ln)

(the notation is impure, but the intention should be clear). This is then the desired, strengthened clause. If J = 0, the clause

can not be strengthened.

Most axioms in the sequel do not need strengthening, but a few do. Axiom (7) above must not be strengthened.

4. FUNCTIONS, THEIR INTENDED INTERPRETATIONS, AND AXIOMS FOR VARIOUS DOMAINS

In this section, we shall work through various types of NL information, and suggest a notation and a set of axioms that reproduce this kind of information. In every case, we shall rely on the general framework that was set up in previous sections.

Axioms for knowledge

Following the discussion in previous sections, we use the following axioms:

Believing	[events → properties]
Knowing	[properties → properties]
Knowing-whether	[events → properties]
Knowing-that	[events → properties]
Acquainted-with	[objects → properties]

It is convenient to start out from the function 'Believing', and to define the others in terms of it.

(KNOW 1) m IS Knowing-that e $\equiv$ Holds e $\wedge$ m IS Believing e

(KNOW 2) Knowing-whether e = Knowing-that e OR Knowing-that Not e

(KNOW 3) m IS Knowing p $\equiv$ [(∀k) m IS Knowing-whether (k WHERE p)]

(KNOW 4) m IS Acquainted-with n $\equiv$ [(∀p) m IS Knowing-whether (n WHERE p)]

(KNOW 5) m IS Knowing p $\wedge$ m IS Knowing q $\supset$ m IS Knowing-whether (Whatis p WHERE q)

(KNOW 6a) (Believing e AND Believing f) = Believing(e AND f)

(KNOW 6b) e SUB f $\supset$ Believing e SUB Believing f

(KNOW 6c) e SUB f $\supset$ Knowing-that e SUB Knowing-that f

(KNOW 7a) Believing e EXCLUDES Believing Not e

(KNOW 7b) Knowing (Not p) = Knowing p

(KNOW 7c) Knowing-whether (Not e) = Knowing-whether e

(KNOW 7d) Knowing-that e EXCLUDES Knowing-that Not e

These axioms are not independent. (7c) is a direct corollary of (2); (7d) can be deduced from (7a) and the strengthened version of (1); etc.

Axioms for the connectives ET and ZU.

The function ET is used to construct composite objects from simple objects, for use e.g. in constructions such as 'Peter and Mary are married'.

In English (like in several other European languages) there is a number of equivalent formulations such as

Peter and Mary are married ≡
Peter is married to Mary ≡
Mary is married to Peter

Peter and Mary are quarrelling ≡
Peter is quarrelling with Mary ≡ ...

Peter and Mary meet in the city ≡
Peter meets Mary in the city ≡ ...

We shall make universal use of the connective ZU for the various prepositions used in natural English (to, with, _, ...). Thus we would write e.g.

peter IS married ZU mary

peter IS (meeting IN the city) ZU mary ≡
peter ET mary IS meeting IN the city

Moreover, we use a special (analytic) predicate Zuable to mark those properties (married, meeting, ...) which can be treated in this way.

We easily jot down the following axioms:

(ETZU 1) m ET n = n ET m

(ETZU 2) (m ET n) ET k = m ET (n ET k)

(ETZU 3) m ET m = m

(ETZU 4) Zuable p ⊃
(m IS p ZU n ≡ m ET n IS p)

Axioms for spatial location

We introduce a new sort, LOCATION, and the following functions:

Loc	[locations → properties]
Inside	[objects → locations]
Outside	-"-
Near	-"-
Farfrom	-"-
Atinside	-"-
At	-"-
Upon	-"-
Under	-"-
Above	-"-
Below	-"-
Beside	-"-
Between	[objects → locations]

A location is thought of as having an EXTENSION in space, and optionally, having an EDGE. An object is thought of as having an EXTENSION and (always) an EDGE. We write 'm IS Loc l' (where m is an object, l is a location) iff [1] the extension of m is contained in the extension of l, and [2] the edge of m has some segment in common with the edge of l, if l has one. (These ideas have been taken from [Schank - Tesler - Weber, 1970]).

The following functions generate locations with an edge: Atinside, At, Upon, Under. The other functions do not. The meaning of all functions should be rather obvious: Inside(m) has the same extension as m, and no edge; Atinside(m) has the same extension, but it also has the edge of m as its edge; etc.

The function 'Between' is supposed to take an argument of the for 'm ET n' or 'k ET m ET n'.

We use a relation SUBL on [locations → locations] to describe analytic location-inclusion.

(LOC 1) [l SUBL h] ≡ [Loc l SUB Loc h]

(LOC 2) Near m SUB_L Outside m

(LOC 3) Farfrom m SUB_L Outside m

(LOC 4) Atinside m SUBL Inside m

(LOC 5) At m SUBL Near m

(LOC 6) Upon m SUBL At m

(LOC 7) Above m SUBL Near m

(LOC 8) Upon m SUBL Above m

(LOC 9) Under m SUBL At m

(LOC 10) Below m SUBL Near m

(LOC 11) Under m SUBL Below m

(LOC 12) m IS Loc R n ≡ n IS Loc R m
(WHERE R = Near, Farfrom, At, Beside)

(LOC 13) m IS Loc Upon n ≡ n IS Loc Under m

(LOC 14) m IS Loc Above n ≡ n IS Loc Below m

(LOC 15) m IS Loc R n ⊃
¬ k IS Loc Between m ET n
(WHERE R = At, Inside)

For some of the further axioms, it is convenient to have an auxiliary relation EXCLUDEL on [locations * locations], saying that two locations are mutually exclusive:

(LOC 16) l EXCLUDEL h ≡ Loc l EXCLUDES Loc h
(LOC 17) Inside m EXCLUDEL Outside m
(LOC 18) Near m EXCLUDEL Farfrom m
(LOC 19) Below m EXCLUDEL Above m
(LOC 20) Below m EXCLUDEL Beside m
(LOC 21) Above m EXCLUDEL Beside m

Deduction using Loc axioms certainly needs to be supported by a natural model!

It may or may not be a good idea to use functions 'Locinside', 'Locnear', etc. which map directly from an object to the property of having a location related to the object. We would then avoid treating locations as separate sorts. Having a special sort for locations is probably a good idea, if we plan to support the theorem-prover with some kind of natural model.

Axioms for the comparison of adjectives

We use the following functions:

More .. THAN .. [properties * objects → properties]
As .. AS .. - " -
Less .. THAN .. - " -
Most .. AMONG .. [properties * properties → properties]
Least .. AMONG .. - " -

The meaning of these functions should be clear. Examples:

peter IS More tall THAN John
John IS Less tall THAN peter
John IS As tall AS dick
peter IS Most tall AMONG (brother OF dick)

The last expression is intended to say that Peter is a brother of Dick, and that no brother of Dick is taller than Peter, although some may be as tall as Peter. If Peter is strictly the tallest of Dick's brothers, we can write the stronger statement

peter = The (Most tall AMONG (brother OF dick))

To explain that two properties are each other's opposites, we introduce a monary function

Un: [properties → properties]

to be used like in

peter IS Un(old)
young = Un(old)

In cases where a certain natural language permits several opposites of an adjective (e.g., both 'tall' and 'long' are English opposites of 'short'), we shall take the standpoint that this is a case of lexical ambiguity (for 'short') or of imposed redundancy ('tall' vs. 'long'), and that the set of properties must be smoothened by using two different individuals to resolve the ambiguity ('short1' and 'short2') or by merging the two redundant properties into one ('long+tall'). With such arrangements, the function 'Un' can be made unambiguous.

Notice the difference between 'Not p' and 'Un p'. If the kind of property expressed by 'p' and 'Un p' is not at all applicable to an object, then the object has the property Not(p), but not the property Un(p). For example, we say that a stone is 'Not(happy)', but not that it is 'Un(happy)'.

Now some axioms:

(CADJ 1) m IS More p THAN k ∧ k IS More p THAN n ⊃ m IS More p THAN n

(CADJ 2) ¬ m IS More p THAN m

(CADJ 3) m IS Less p THAN k ≡ k IS More p THAN m

(CADJ 4) m IS More p THAN k ⊃ m IS More p THAN Any (As p AS k)

(CADJ 5) m IS More p THAN k ⊃ Any (As p AS m) IS More p THAN k

(CADJ 6) m IS As p AS n ⊃ n IS As p AS m

(CADJ 7) n IS As p AS n

(CADJ 8) m IS Most p AMONG q ≡ (m IS q ∧ No q IS More p THAN m)

(CADJ 9) m IS Least p AMONG q ≡ (m IS q ∧ No q IS Less p THAN m)

(CADJ 10) p = Un(Un(p))

(CADJ 11) Un(p) SUB Not(p)

(CADJ 12) As p AS n = As Un(p) AS n
(CADJ 13) As p AS n EXCLUDES More p THAN n

(CADJ 14) More p THAN n = Less Un(p) THAN n

(CADJ 15) Most p AMONG q = Least Un(p) AMONG q

Axioms for measures on adjectives

We use the following functions

Very	[properties → properties]
Rather	- " -
Slightly	- " -

We also assume that every object has exactly one of the following properties:

Very p
Rather p
Slightly p
Not p

for every 'basic property' p. (Very p, Rather p, etc. are not 'basic properties', so we do not assume constructions like 'Very Rather p').

Some axioms are:

(MADJ 1) Very p EXCLUDES Rather p
(MADJ 2) Rather p EXCLUDES Slightly p
(MADJ 3) Very p EXCLUDES Slightly p
(MADJ 4) Very p SUB p
(MADJ 5) Rather p SUB p
(MADJ 6) Slightly p SUB p
(MADJ 7) Any Very p IS More p THAN Any Rather p
(MADJ 8) Any Rather p IS More p THAN Any Slightly p
(MADJ 9) Any Very p IS More p THAN Any Slightly p

(MADJ 10 a-c) m IS As p AS n ∧ m IS Op p ⊃
n IS Op p
(WHERE Op = Very, Rather, Slightly)

5. CONCLUSION

In this paper, we have proposed a set of functions and relations that we claim are good for re-expressing a cross-section of typical NL constructions. We have also given some 75 axioms for these functions and relations. This is an order of magnitude bigger than the axiom sets that (to our knowledge) have before been used in theorem-proving programs, and it should present a challenge to them.

Some of the problems that should be treated next are:

a. Automate the translation from a simplified natural language to the notation presented here.

The reader will have noticed that in developing the notation, we took considerable delight in staying close to NL concepts and formulations. There are good and bad aspects to this; one good aspect is certainly that it should simplify translation.

b. Polish up and extend the axiom sets.

The axiom sets that have been given in this scetchy paper are somewhat haphazard, and they need debugging. We submit that this debugging can best be performed in interactive experiments on a computer, and that human think power is not sufficient. We submit, further, that the criteria for selecting an axiom set must be those of power and of computational efficiency, and that the criteria usually used in logic (elegance, minimal set of axioms, etc.) are largely irrelevant.

c. Develop short-cut methods whereby a theorem-prover can manipulate the algebras on properties and events in an efficient way.

d. Try to get some handle on those sentences in NL which are not intended to convey the information of their 'face value' assertion, and which are not either intended as informaton requests (questions).

Many of the sentences that we use (even in regular, non-fiction prose) are pronounced only in order to focus the listener's attention on some fact that he already knows, or to tell the listener that the speaker knows a certain fact and has accounted for it, etc. Statements of this kind are not adequately handled if we merely translate them into PC and shuffle them into a data base. They must be treated quite differently. We consider this the most important (and also the most evasive) problem in NL processing today.

REFERENCES

Allen 1978

J. Allen and D. Luckham
An Interactive Theorem-proving Program
In: Machine Intelligence 5
(B. Meltzer, D. Michie, eds)
Edinburgh University Press, 1978

Green 1969

C.C. Green
Theorem-Proving by Resolution as a Basis for
Question-Answering Systems
In: Machine Intelligence 4
(D. Michie, ed.)
Edinburgh University Press, 1969

Hayes 1969

P. Hayes
A Machine-Oriented Formulation of the
Extended Functional Calculus
Stanford A.I. Project Memo # 86

Luckham 1978

D. Luckham
Refinements in Resolution Theory
Proc. IRIA Symposium on Automatic Demonstration
Springer Verlag, 1978, p. 163-198

McCarthy 1969

J. McCarthy and P. Hayes
Some philospophical problems from the standpoint
of artificial intelligence
In: Machine Intelligence 4
(D. Michie, ed.)
Edinburgh University Press, 1969

Robinson 1965

J.A. Robinson
A Machine-oriented Logic Based on the Resolution
Principle
Journal of the ACM, Vol. 12 (1965), pp. 23-41

Robinson 1969

J.A. Robinson

In: Machine Intelligence 5
(B. Meltzer, D. Michie, eds)
Edinburgh University Press, 1978

Schank 1978

R. Schank, L.Tesler, S.Weber
SPINOZA II: Conceptual Case-based Natural
Language Analysis
Stanford A.I. Project Memo # 189