American Journal of Computational Linguistics *Microfiche 24*

THE SQAP DATA BASE FOR NATURAL LANGUAGE INFORMATION

Jacob Palme

Research Institute of National Defense
Operations Research Center
Stockholm 80, Sweden

ABSTRACT

The Swedish Question Answering Project (SQAP) aims at handling many different kinds of facts, and not only facts in a small special application area. The SQAP data base consists of a network of nodes corresponding to objects, properties, and events in the real world. Deduction can be performed, and deduction rules can be input in natural language and stored in the data base.

This report describes the data base, specially focusing on problems in its design, both problems which have been solved and problems which are not yet solved.

Specially full treatment is given to the data base representation of natural language noun phrases, and to the representation of deduction rules in the data base in the form of data base "patterns"

SWEDISH ABSTRACT

SQAP-projektet (Swedish Question Answering Project = Svenska projektet för frågebesvarande system) syftar till att kunna hantera många olika slags fakta i datorn, inte bara fakta inom ett litet speciellt tillämpningsområde. Databasen består av ett nätverk av noder som svarar mot objekt, egenskaper och händelser i verkligheten. Slutsatsdragning kan göras, och slutsatsdragningsregler kan ges i naturligt språk och lagras i databasen.

Denna rapport beskriver databasen, med speciell tonvikt pa problem vid dess konstruktion, bade sådana problem som vi löst och sådana som vi ännu inte löst.

Speciellt utforligt behandlas representation av substantivkonstruktioner i databasen, samt hur slutsatsregler kan representeras som monster i databasen.

FOAP rapport C 8376-M3(E5), September 1973, revised July 1975.

Contents: Page

0. Introduction

This paper describes the natural language data base structure used in the SQAP system (Swedish Question Answering System). Much of that system is already working, but the paper does not only describe the solutions to solved problems. Difficulties and unsolved problems are also presented, since I feel this is important to further progress.

One of the goals of the SQAP project was to create a question-answering system capable of handling facts of many different kinds. The system should thus not be restricted to a small special application area.

1. Natural language representation

There is an obvious need for computers with a capability to converse in natural human languages. Natural languages are more general-purpose than most artificial languages, which means that you can talk about a wider subject area if you use natural languages. Natural languages can be used by everyone without special training, so computers talking natural language can make more people able to use more different computer facilities. Finally, a rizing part of computer usage in the future will be unintelligent processing of natural language texts, and such systems can be improved if the processing is not wholly unintelligent.

There are also wellknown difficulties with natural languages for computers. Natural language is closely connected to human knowledge. Therefore, natural language sentences can only be understood by a man or a computer with factual knowledge about the subject matter and with the ability to reason with those facts. To disambiguate such wellknown examples as "The pig was in the pen" (Bar-Hillel 1964) or "He went to the park with the girl" (Schank 1969) the computer must have an underlying knowledge about various kinds of "pens", about where "the girl" was previously and so on.

Also, the same thing can be said in many different ways, and a computer with natural language capabilities must be able to understand this, so that for example it can see the similarity between "Find the mean income of unmarried women with at least two children." and "Search through the personell file. For each individual who is a woman, who is not married, and who has a number of children greater than two, accumulate income to calculate the mean."

Therefore, a computer understanding natural language must have a data base with basic factual knowledge about the world in general or about the subject matter which the computer is to be used for.

This data base is needed to understand ambiguous sentences, but also to interpret the sentences into executable data processing commands.

The requirements on such a data base are:

- You should be able to store a wide variety of different kinds of facts. Natural languages are very general-purpose. so the data base should also be general-purpose.

- You should be able to use this data base to make deductions. The capability to do simple and natural deductions fast is more important than the capability to make very advanced and long-range deductions. Since the data base will be large, an important part of deduction will be the selection of the relevant facts and rules out of the large mass of facts not needed for one special deduction.

The data base can be more or less close to natural language. A data base close to natural language makes input translation easier, and also the loss of nuances during the input translation

will be smaller. But the data base must on the other hand have a logical structure which is suitable for deduction and fact searching.

One model of natural language knowledge is the following: The knowledge consists of "concepts" and of rules relating these concepts to each other. A typical concept might be "John", "All young men", "The event when John meets Mary in the park" or "The month of July, 1973". The concepts are related by rules, which can be very simple relations (like the relation between "All young men" and the property "young") or complex patterns of concepts (like the rule "If Mary is weak and tired, and she meets a strong brutal man, then she will be frightened.") These rules form a network linking all concepts together.

This model of natural language is close to that often used by psychologists in trying to explain the working of the intelligence in the human mind.

The SQAP system uses a data base of that kind. The model may at first seem simple and straightforward. When you try to produce a working question-answering system, you will however find that there are many difficulties and complications with such a data base. This report presents the most important of the problems we have met, and in some cases also our solutions. I believe that other producers of natural language systems will sooner or later encounter the same problems, and they may then benefit from our experience as presented in this paper.

2. Introduction to our data base.

During the 1960:s, several researchers independently and simultaneously came up with the same basic idea of organizing such a data base - Sandewall 1965, Simmons 1971, Shapiro 1971. Some of them were influenced by the case grammar of Fillmore 1968.
The idea is that the data base is organized into nodes, each node representing a concept. In natural language, the prepositions are used to represent short simple and direct relations between concepts "John is in the bed", "The fire was lit by Mary"
In the data base, the idea of prepositions is extended so that all simple and direct relations between concepts are represented by implicit prepositions. (Just as you could say that there is an implicit preposition "by" in the phrase "Mary lit the fire".)

More complex rules or relations between concepts are represented by extra concepts. Thus there is a concept for the event "Mary lit the fire" and this concept is related to "Mary", "the fire" and "act of lighting" in a structure like that in figure 1.

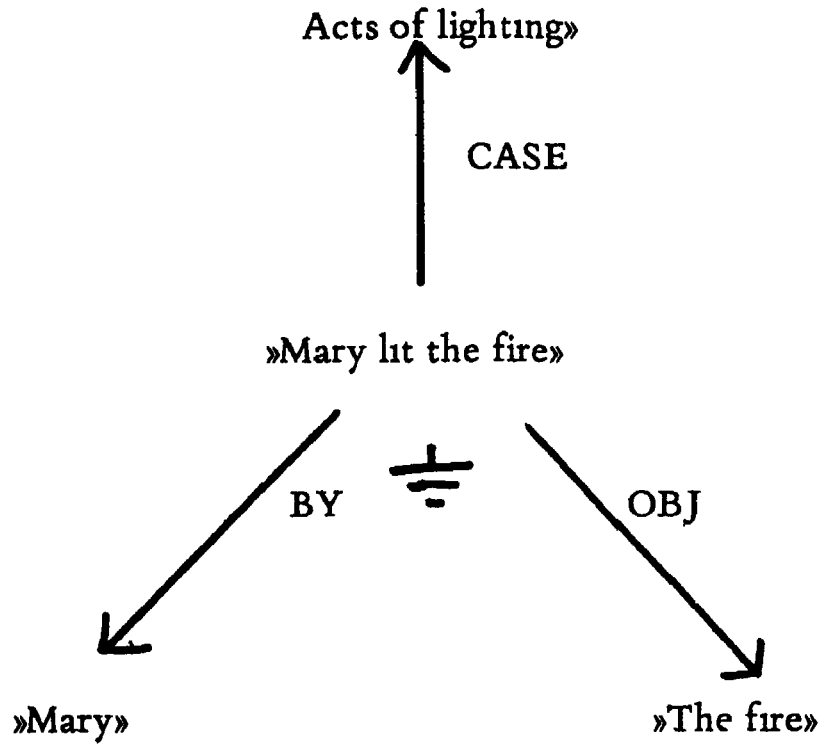


Figure 1

This structure has four concepts linked together by three "prepositional" relations: CASE, BY and OBJ. From now on, I will in this paper call such relations "short relations".

The data base is organized so that the deduction rules can follow the short relations in both directions, that is go from "Mary" to "Mary lit the fire" or from "Mary lit the fire" to "Mary".

3. Objects, events and predicates

Noun phrases in natural language usually refer to one or a set of objects in the real world, like "Stockholm" or "Every house in Sweden" or "The nice man with a bicycle". In our system each such concept is represented by a node in the data

base, which could be called an object node.

Each object node is associated with one or more predicate nodes expressing properties of that object. In our data base, we mark predicates with the postfix "*P". Thus, the phrase "An always happy girl" would in our data base be represented like in figure 2:

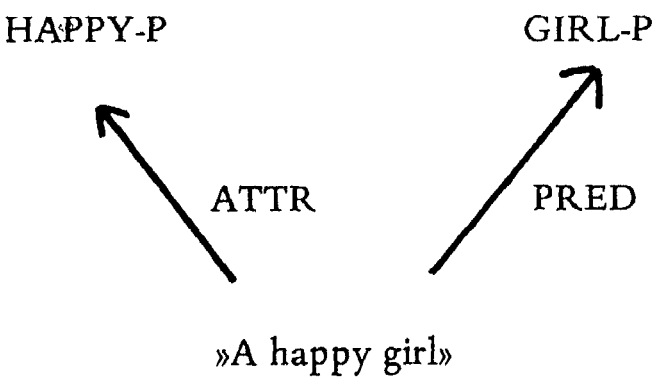


Figure 2

A statement like "There is an always happy girl" or "One girl is always happy" would be represented in the same way, with an object node and two short relations on it to the two predicates, ATTR to the adjectival predicate, PRED to the nominal predicate.

If we meet the natural language phrase "One girl is nice today", then we cannot represent it as simply. We have to affix a time to the relation between the girl and HAPPY*P". One way to do this would be to always have the ability to add extra short relations to existing short relations. This would require that short relations are represented in the data base in a form where there is a place to add a list of extra short relations, and this would triple the size of the data base. Instead we have an expanded form of those short relations to which we want to add other relations. This expanded form is only used when it is needed, the non-expanded form is used when there are no added short relations. The expanded form for the PRED short relation is a node of the type "event".

"One girl is happy today" will thus be represented like in figure 3.

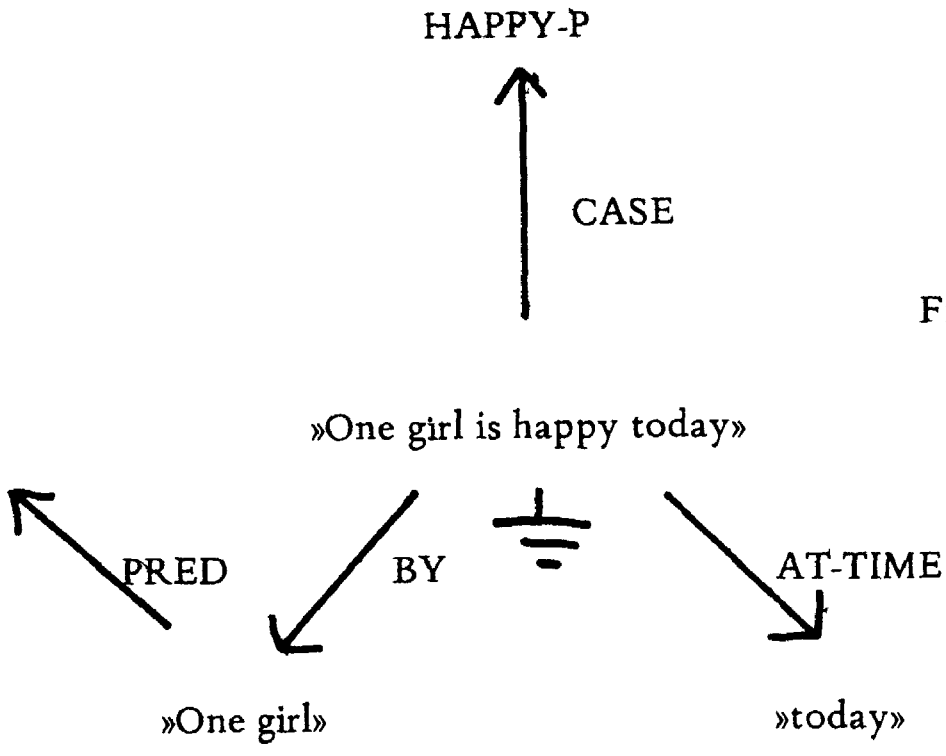


Figure 3

The advantage of having such an extra concept in the data base is that we can easily add more short relations to the event node "One girl is happy today", for example to represent "in the school" or "because of the weather" or "according to what Tom said".

Since we want to deal with true statements, hypothetical statements and statements belonging to some person's belief structure, we always add a relation to an event node indicating which belief structure it belongs to, true events belong to the set "TRUE*S" of all true statements. Since the relation "PART TRUE*S" is so common, we represent it in pictures with the earth sign of electric charts: ⏚ .

The statement "John believes that Mary loves him" would thus be represented like in figure 4:

»John believes that Mary loves him»

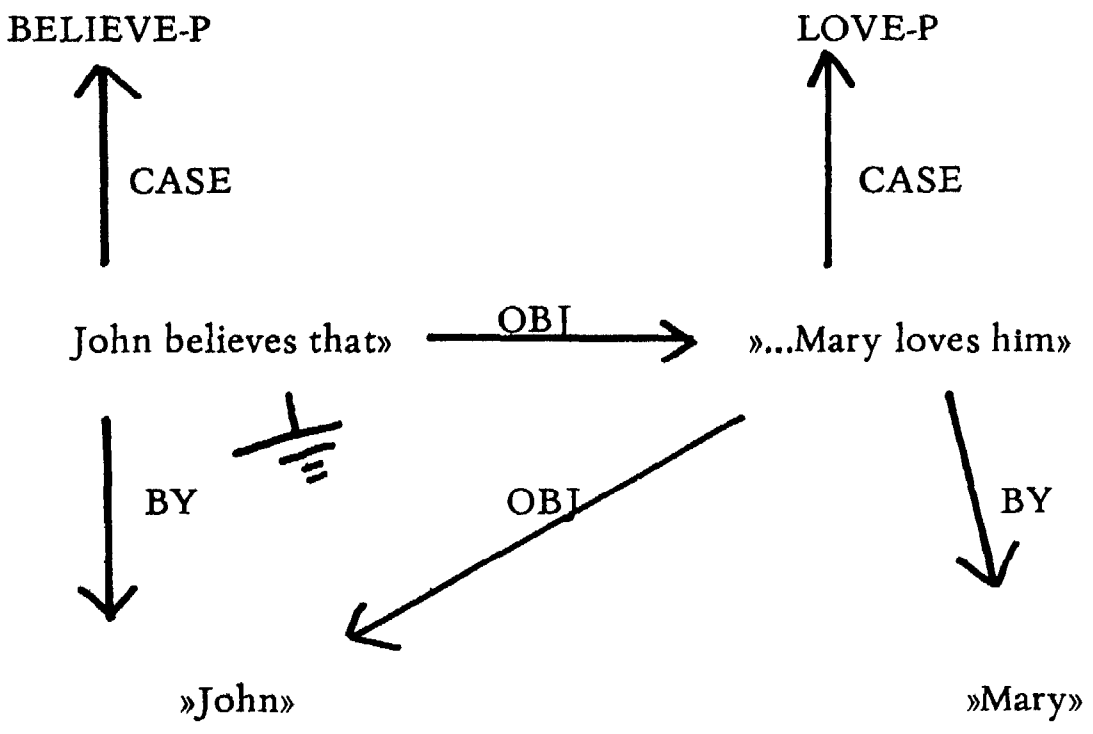


Figure 4

Note that there is an earth sign on the true event, but no earth sign on the event belonging to John's belief structure.

Note that predicates and relations in natural language are often not represented directly as short relations in our data base. "John is the father of Angelica" is thus not represented by a short relation "FATHER" from "John" to "Angelica" but rather with an event node like in figure 5:

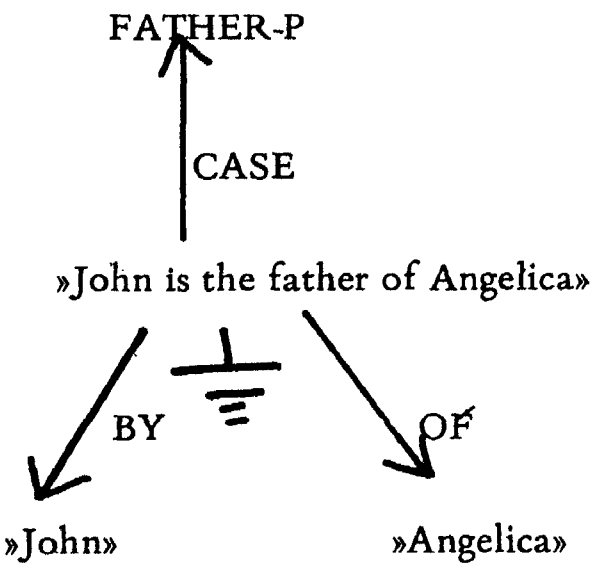


Figure 5

4. Quantifiers on the short relation

Every node in our data base can stand for a set of objects instead of for just a single object. Thus we can represent "All nice girls" with a node representing the set of all nice girls.

This means that we need quantifiers on the short relations, to be able to express relationships between sets.

If there is a short relation R between two sets A and B, then the relation R might not be true between any member of A and any member of B. We have several cases:

i) $\forall x \{x \in A\} (\forall y \{y \in B\} \; x \; R \; y)$

ii) $\forall x \{x \in A\} (\exists y \{y \in B\} \; x \; R \; y)$

iii) $\exists x \{x \in A\} (\forall y \{y \in B\} \; x \; R \; y)$

iv) $\exists x \{x \in A\} (\exists y \{y \in B\} \; x \; R \; y)$

These and other cases are represented in our data base with three quantifiers ALL, SOME and ITS. The difference between SOME and ITS is shown by the difference between the second and the third example. One quantifier is placed on each end of the relation. The four examples above will thus in our data base look like this:

i) (ALL A, R, ALL B)

ii) (ALL A, R, ITS B)

iii) (SOME A, R, ALL B)

iv) (SOME A, R, ITS B)

The difference between ITS and SOME can be understood if you look at the statement "Every man is in a car" This can mean that "Every man is inside one single car" or it can mean "For every man there is one car in which he is". The first phrase might in our data base be represented as

"Every man" ALL IN SOME "car"

while the second might be represented as

"Every man" ALL IN ITS "car"

There are simple rules to manipulate the quantifiers when the deduction rules chain from node to node in the data base. This is described in Sandewall 1969.

In the following, if no quantifier is marked on a short relation in a figure, then ALL is implicit.

5. Deduction in the data base

The data base does not contain all true statements explicitly, some of them have to be deduced when needed. Basically all deduction rules can be seen as pattern matching. You have a pattern saying for example that "If something hot is near something inflammable then the imflammable will catch fire". Then we have some actual situation, explicit or deduced, e.g. "The burning cigarette is thrown in the petrol tank". In our data base, as in figure 6.

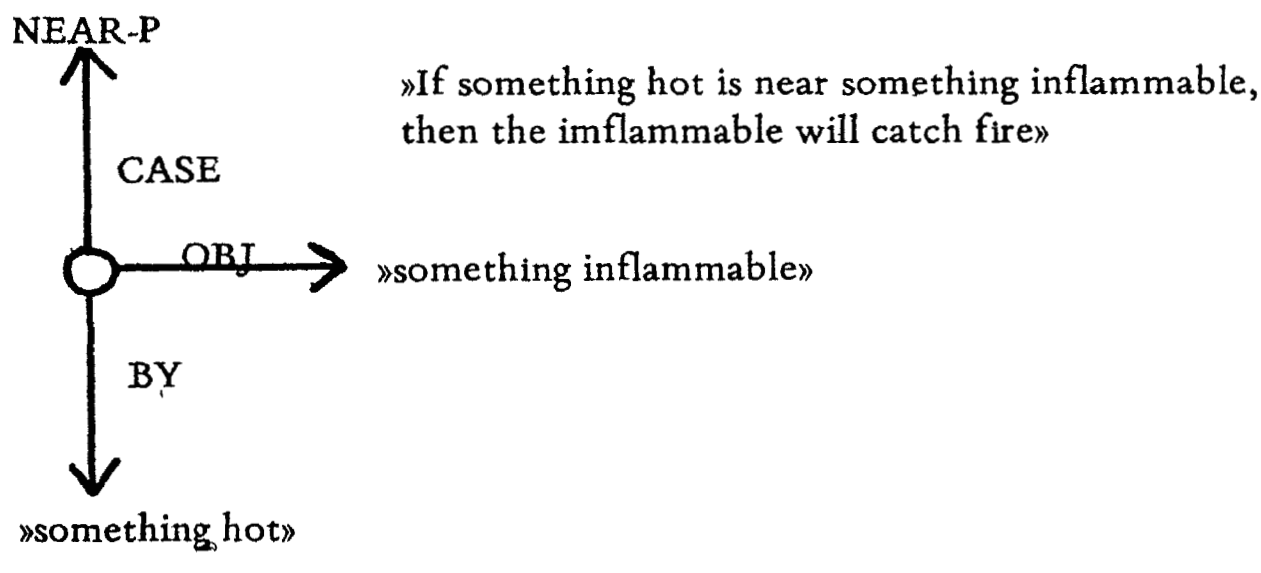
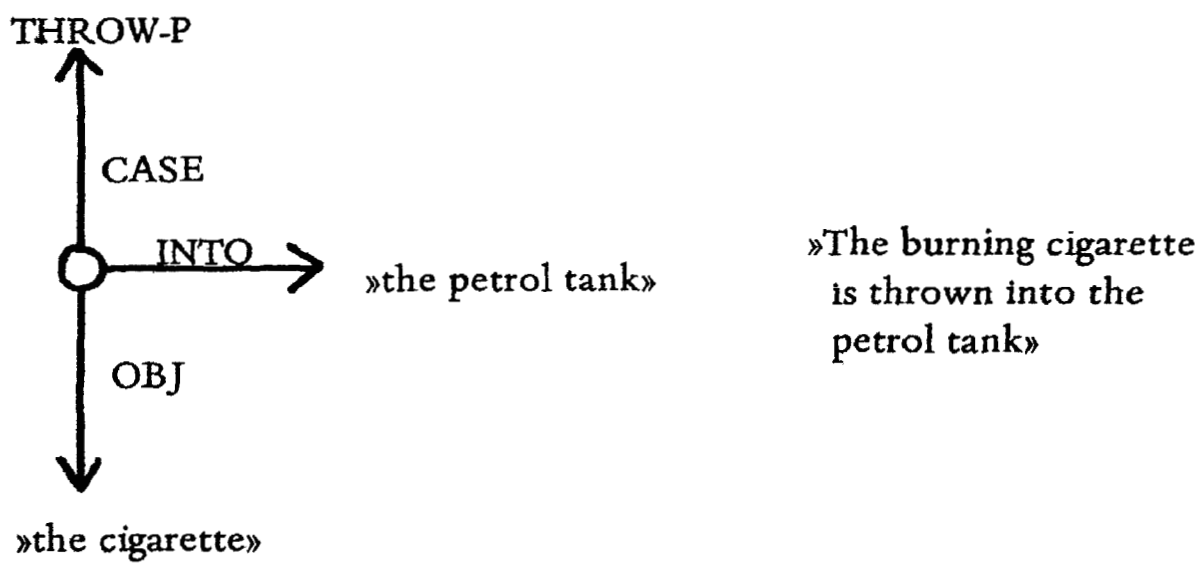


Figure 6



Before using the deduction rule, we must match the pattern to the actual situation. The pattern can contain many interconnected nedes, and the reality may not at first resemble the pattern directly, deduction may be necessary to see the resemblance.

The simplest deduction rule possible is just a pattern of two short relations from which a third can be deduced: "If A R1 B and B R2 C then A R3 C", a simple example: "If A is subset of B, and B is subset of C, then A is subset of C". Since such rules link together nodes through a chain of short relations, they are called chaining rules. Some chaining rules require side relations on B to the fullfilled, for example "A BY B, and A CASE C implies B PRED C", but only if A is a true event.

This is described in Sandewall 1969 and in Makila 1972.

6. Variables

The simplest kind of deduction pattern involves just one node. Such a node is called a variable. For example, VARIABLES are used in the translation of "Every intelligent man is a bad soldier" which is represented like in figure 7;

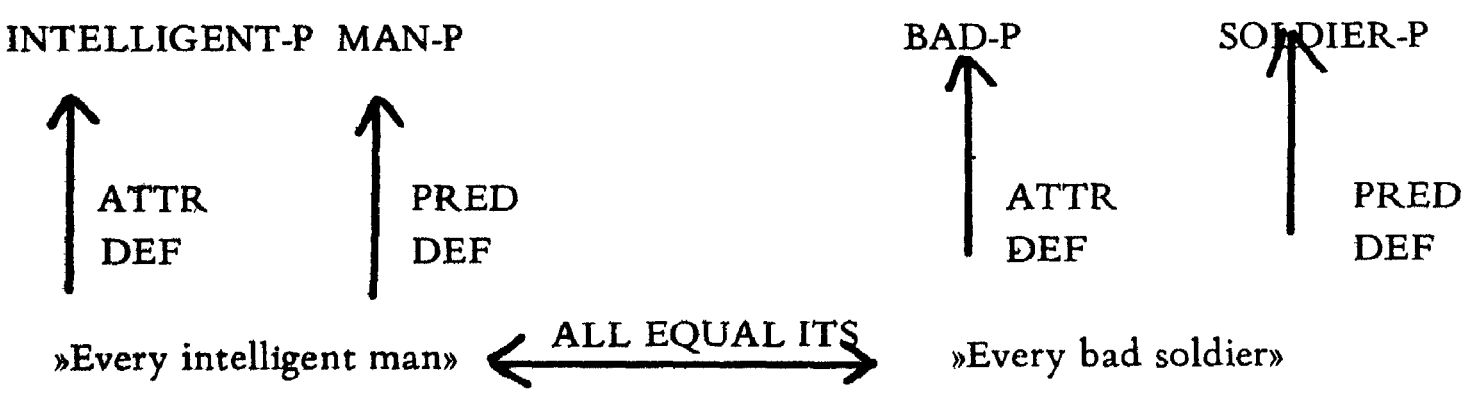


Figure 7

Variables have a new quantifier on them, DEF. This indicates that this short relation is part of the definition of that variable. The variable "Every intelligent man" above corresponds to the set of all objects which satisfy the definition. This means that as soon as we find an object in the data base for which we know or can deduce that it satisfies the definition, then we know that it belongs to the VARIABLE above, and we can thus deduce that it is a bad soldier.

7. Keys

Sometimes the deduction requires a pattern of more than one node. Such patterns are called keys. The sentence "If a motorboat meets a sailingboat, then the motorboat must steer away from the sailingboat" will in our data base be represented like in figure 8.

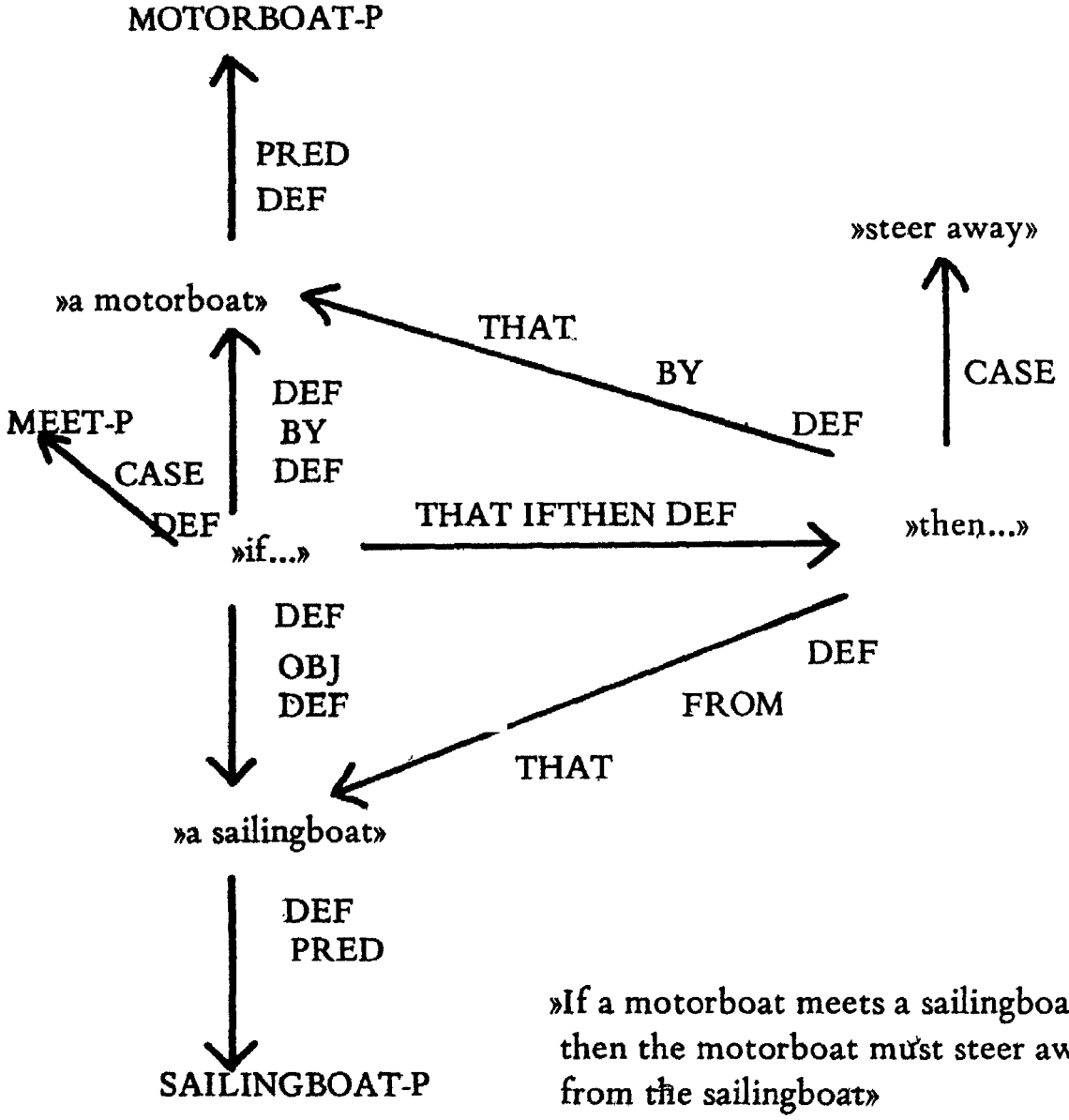


Figure 8

»If a motorboat meets a sailingboat, then the motorboat must steer away from the sailingboat»

To the left is the pattern of three nodes connected by short relations with DEF on both ends of the relations. This shows that they are part of a key. To the right is the deduced statement, connected to the center of the key with an IFTHEN short relation. Note that there is a new quantifier THAT above. This quantifier means that we should single out just that actual object which was matched to that part of the key. We do not want to say that "Every motorboat meeting a sailingboat must steer away from every sailingboat being met by a motorboat", and therefore we must single out the matched object only.

8. Dummies

Natural language sentences often refer to previously mentioned entities with constructs like "he" or "the old man". Our system will first translate a sentence into an independent data base fragment. An assimilation program will then merge this fragment with the data base. Other systems often make this assimilation during the input translation. They would then avoid some, but not all, of our problems, but would get some other problems instead.

For this merging we create temporary variables and keys during input translation. The sentence "The man always with the gun is in the forest" is thus translated into figure 9:

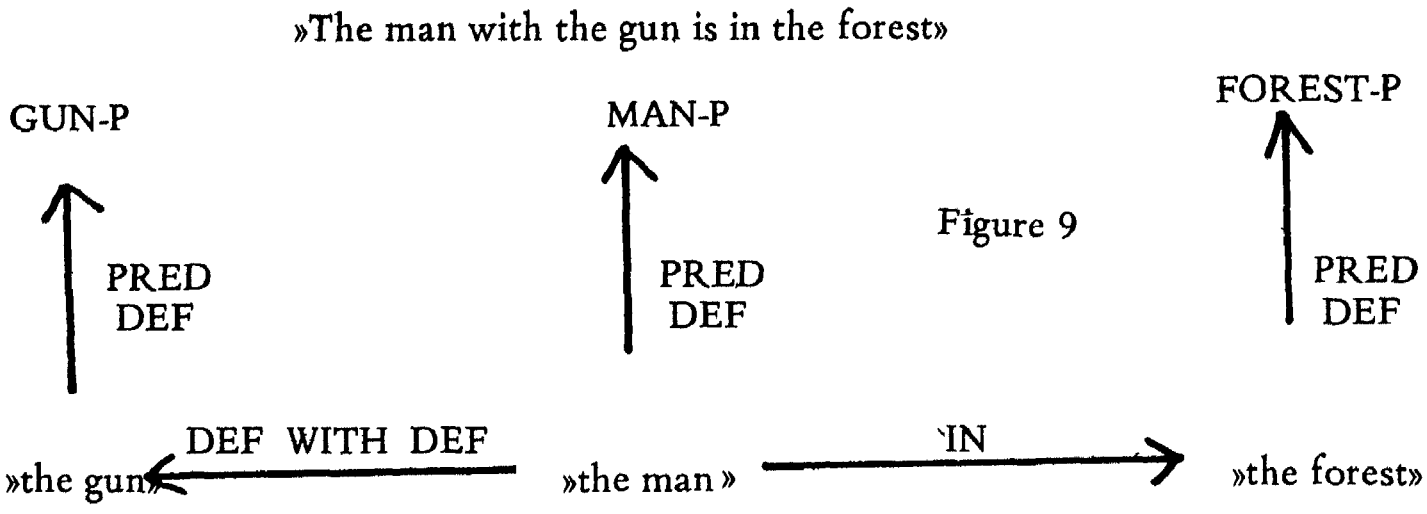


Figure 9

The merging program will use special deduction rules for these temporary variables, which we call DUMMIES. After merging, the DUMMIES usually merge with some previous node in the data base, and they thus become CONSTANTS or VARIABLES depending on the type of that previous node.

9. Questions

Questions to a computer can require short or long answers. There are for example yes-no questions like "Is a man with a baloon coming?" which in our data base will be represented like in figure 10.

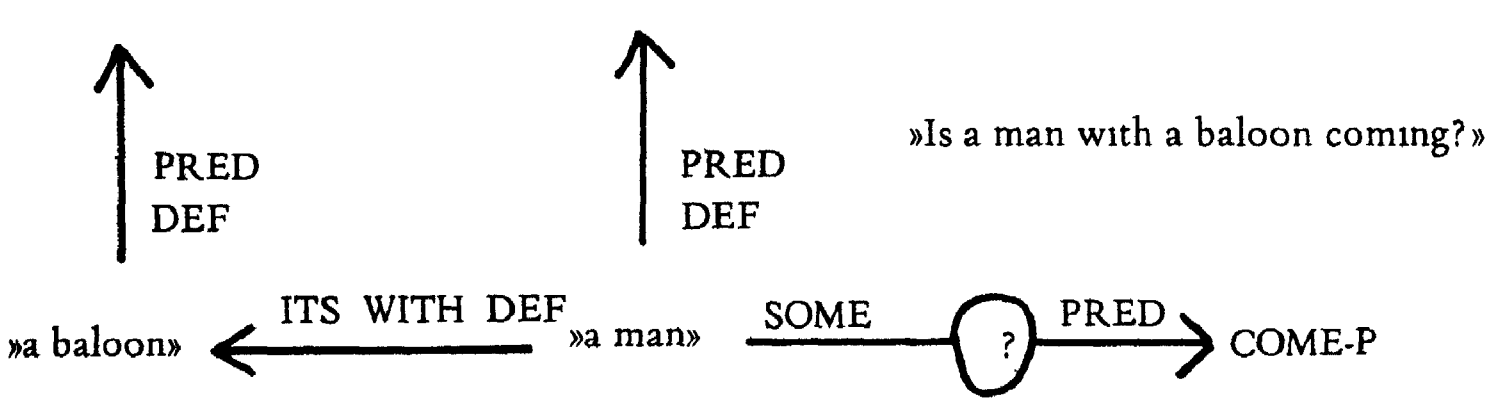


Figure 10

where we put a question-mark on one relation to show that the program shall try to deduce that relation from the previous knowledge.

Other questions require as answer a list of objects, for example "Which of you can drive a car" or a description of a deduction chain ("why" questions) or a description of an algorithm ("how" questions). We have not yet tried to represent such questions in our data base structure.

10. Example of what our system can do

This example shows a set of facts and questions, such that our system can answer the questions based on the facts.

The input language to our system is not full natural english. the language is slightly simplified. The sentences in the example are written in this simplified english.

A girl is a young woman. A boy is a young man. A woman is a human female. A man is a human male. Every young man is a boy.

Every sports car is fast and expensive . Every fast car is dangerous. Anybody - with an expensive car is rich. Every rich woman is frightened by every poor man.

If a woman is meeting a man and she is frightened by him but she is loved by him, then she will be despising him. If a man is loving a woman and she is despising him, then he is depressed. If a man is depressed and he is driving a car, then he is senseless and irrational. If a senseless man is driving a dangerous car, then he is dangerous, and all traffic is in deadly peril.

Everyone - on a public street is traffic.

Mary is a mature girl - with a sports car. Eliza is a pretty girl - with long hair. The mature girl is ugly.

Is the ugly girl, rich?

John is a man. He is young and poor. He is loving every fast car and every girl - with a fast car.

Is the pretty girl loved by John? Is the rich and ugly girl loved by John?

If Eliza had been a girl - with a fast car, then would she be loved by John?

Mary is meeting John and he is driving her car. Is the poor boy, dangerous?

11. The EQUAL relations

The EQUAL relation between two singular elements means that they are identical. However, since we can put quantifiers on the EQUAL relation, we can also use it for many set relationships. Some examples:

ALL A EQUAL ALL B means that the sets A and B are equal and contain not more than one element each.
ITS A EQUAL ALL B means that A is a subset of B.
ITS A EQUAL ITS B means that A and B overlap.
ALL A NOT EQUAL ALL B means that A and B are disjoint.
SOME A EQUAL ALL A means that A is not empty.
ALL A NOT EQUAL ALL A means that A is empty.
SOME A EQUAL ALL A means that A is singular, that is contains exactly one member.

Natural language noun phrases are translated into nodes marked as singular in the data base, if:

a) The noun phrase is not plural.

b) The noun phrase is not translated into a VARIABLE in the data base.

c) The noun phrase is interpreted in the special sense (like "A man is walking on the street") and not in the general sense (like "Every man is a male human").

Data base nodes are marked as non-empty if they are of the type predicates (like "the act of lighting" which we call LIGHT*P).

12. Natural language noun phrases

There are many data base constructs which correspond to natural language noun phrases. Noun phrases create many problems with their attributes, with composite objects which have several parts a.s.o. A number of chapters will discuss problems with representing that kind of facts. The necessary data base concepts are discussed, rather than the translation problems.

Singular noun phrases without conjunctions are usually translated into one object-type node. An exception to this is some simple sentences in which noun-phrases are translated to predicate-type nodes, see chapter 21.

This object-type node can be a CONSTANT, a DUMMY or a VARIABLE. CONSTANTs are created for simple positive sentences like "A man is walking on a street." Note however, that in an if-statement or a question, the noun-phrases instead must be interpreted as VARIABLEs, e.g. "If a man is walking on a street, then..." or "Is a man walking on a street?". In these cases, "a man" does not introduce a new constant, but represents a simple search pattern to be used in deduction, and VARIABLEs are used for such search patterns in our data base system.

For general-sense statements, the nouns are usually translated to VARIABLEs. Example: "Every good girl will kiss every brave soldier." These VARIABLEs can be used in later deduction, to find out what happens if a good girl meets a brave soldier.

Noun-phrases beginning with "the" or "this" or "that" or some similar determiner are usually translated to DUMMIES. Here a search must be made in the data base for some previously known node to merge the DUMMY with.

Pronouns like "he" or "it" or "her" are also translated into DUMMIES, for the same reason.

The noun word itself indicates a property of that noun (e.g. "man" indicates the sex and species of "a man"). A predicate MAN*P is therefore created, and "a man" gets a relation PRED to MAN*P

Adjectives do not always indicate properties which are generally true for the noun phrase. They can mean many things Examples:

The good teacher (The teacher which is good as a teacher),
The big ant (The ant which is big for an ant),
The red house (The house which is red).

Therefore, a weaker relation ATTR is used from a noun to its adjectives.

Names are a very special kind of predicates, and therefore a special relation NAME goes from a noun node to its name.

When a name such as "John" or "Cambridge" is used, we want to identify this with some previously known "John" or "Cambridge" in the data base. But we cannot give the node itself the name "John" or "Cambridge", since there may be more than one "John" and "Cambridge" in the data base. Therefore, the last-mentioned which fits the description is found, just as for other DUMMIES. "The always Old John" will therefore for example in our data base be translated into the DUMMY in figure 11.

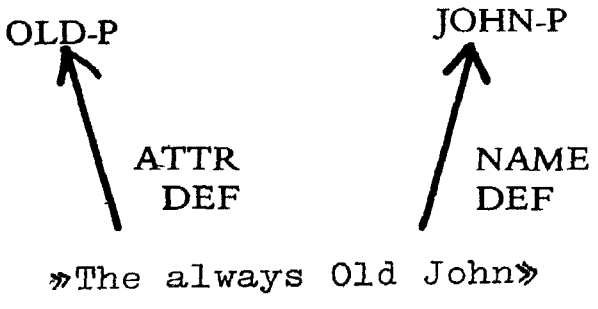


Figure 11

We have a special rule for VARIABLES with only one PRED as a definition, where this PRED goes to a predicate whose name comes from the input sentence. This variable gets the same name, but with "*S" in the end. Thus, "Every man" is translated to MAN*S $\xrightarrow[\text{DEF PRED}]{}$ MAN*P

This special rule is not really necessary, but has two advantages:

a) The data base becomes more readable,

b) The data base routines will immediately see that all MAN*S nodes created by several different sentences can be merged into one, without having to do any deduction.

You could say that MAN*P is the property of being a man, while MAN*S is the set of all men.

12b. Attributes on noun phrases

This section describes things which are not yet implemented in the SQAP program when this is written (May 1974)

For several reasons, attributes on a noun phrase cannot always be represented by a direct relation from the noun phrase to the attribute.

Sometimes two or more attributes on a noun phrase are related. If you say "A friend of Nixon" then this person is not always a friend (he may not be a friend of McGovern) and he is not always "of Nixon" (he may not be "a son of Nixon" although he is surely a son). If we represented the two attributes that he is a friend and that he is "of Nixon" as two separate independent relations on his object node, then the data base deduction rules would not properly understand sentences like "A friend of Nixon is an enemy of McGovern". The deduction

rules would wrongly deduce that since this object independently has the attribute of being a friend and the attribute of being "of McGovern", the object is a friend of McGovern.

To avoid this erroneous conclusion we must have only one single outgoing relation from the object to the composite property of being "a friend of Nixon". The statement "A friend of Nixon is an enemy of McGovern" might thus be represented like in figure 11b

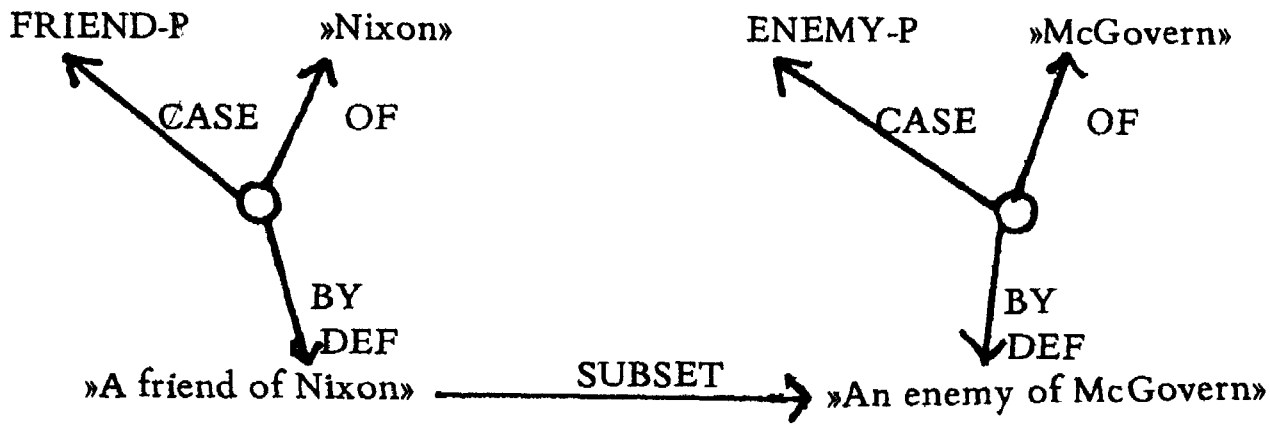


Figure 11b

where two new "event" nodes are introduced for being a friend of Nixon and being an enemy of McGovern. It seems as if only the preposition "of" and no other preposition in english creates this problem.

The same kind of representation can be used to correctly represent a statement like "A big ant is a small animal", see figure 11c.

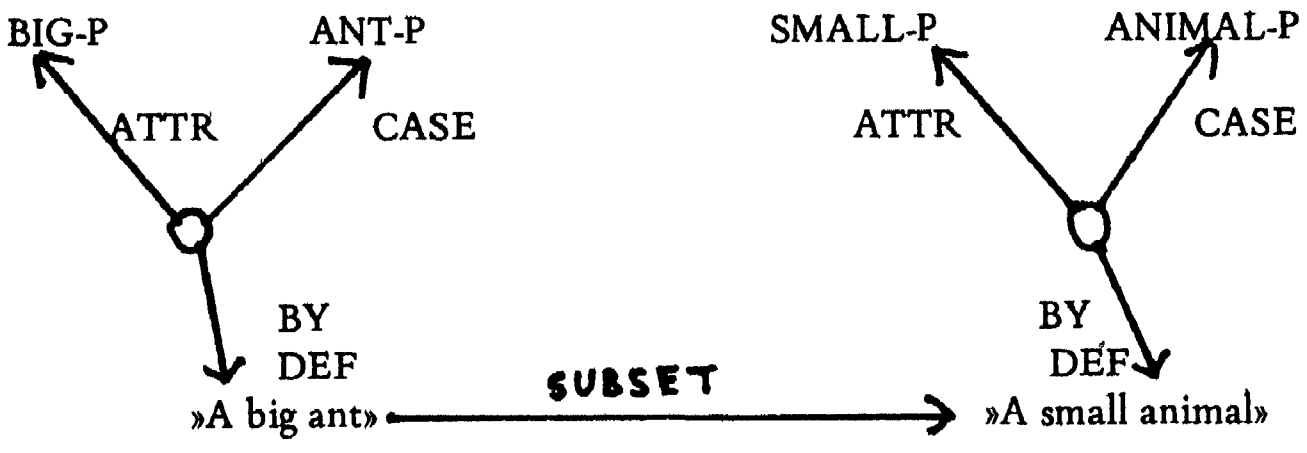


Figure 11c

Another reason why attributes cannot always be represented as direct relations on the noun is that the attribute may be restricted in time or space or in some other way. If we input the noun phrase "A hungry girl" then the computer creates an object for this girl. But we may thereafter learn that the girl eats and is not hungry any more. Thus the same object, at a later time, does not any more have the attribute of being hungry. Here again, we must introduce an EVENT node for the fact that the girl is hungry as shown in figure 11d.

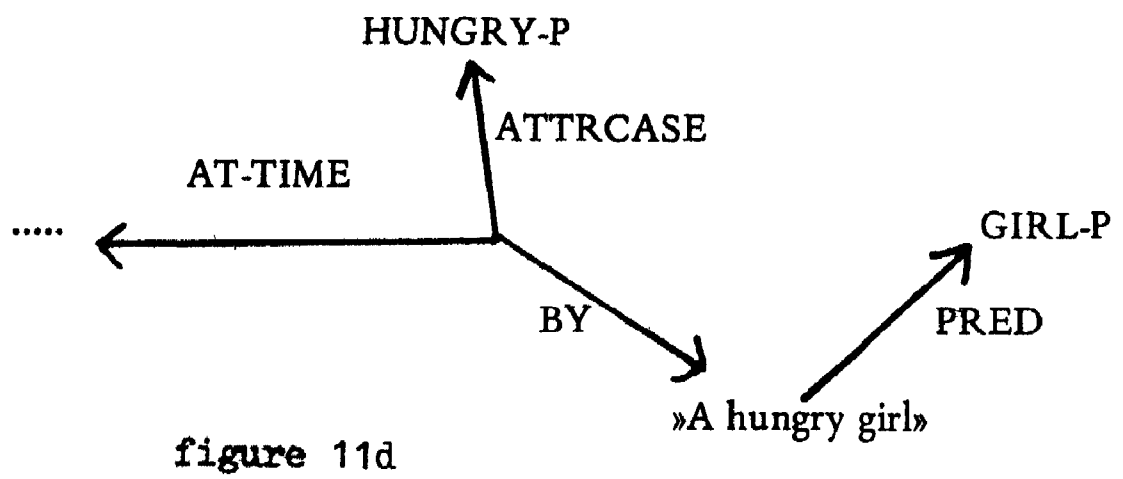


figure 11d

In most cases, the same time-restriction applies to attributes as to the main verb in the sentence. If we say "A hungry girl ate a cold buffet in a sundrenched meadow on a warm summer day" then the time and space restrictions are valid for all the attributes on the various nouns in the sentence. In such cases, the data base could be simplified if we introduced a special "situation" node to represent the time and space, and then used a new short relation SIT from the various event nodes to this situation. This would be even more useful if a series of sentences all apply to the same situation.

Prepositional attributes may also be situation restricted as for the sentence "An angry man with a gun is coming at ten o'clock", where the man at another time may not be "with a gun". This might be represented as shown in figure 11e.

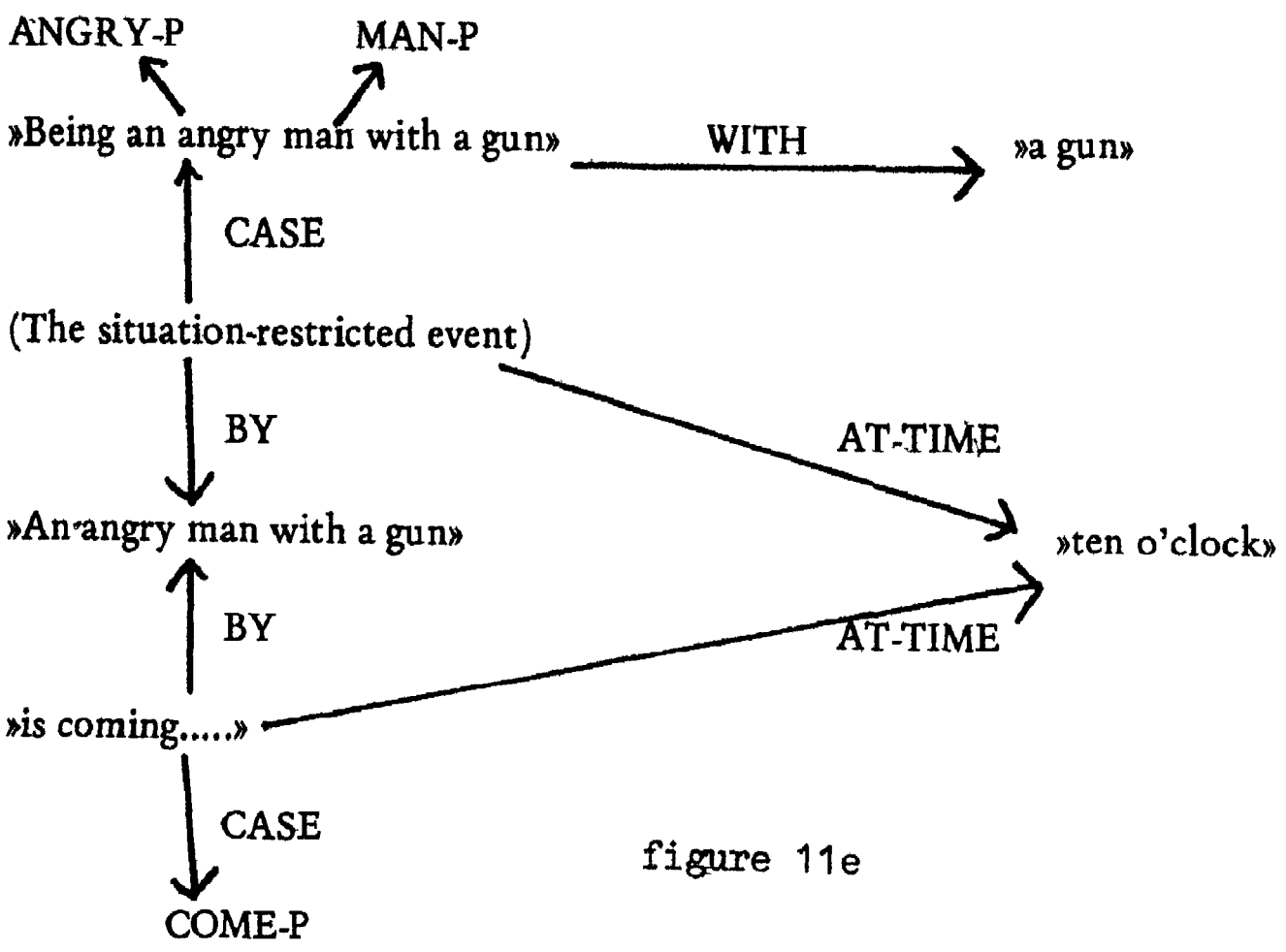


figure 11e

If the computer is told that "Every english spinster who comes into the church is awed" then the computer can deduce that Eliza is awed if it knows that Eliza is english, is a spinster, and is coming into the church, all at the same time. But if Eliza was an english spinster five years ago, and comes into the church today, then we cannot make this deduction. This can be solved in two ways. Either the data base representation of "Every english spinster who comes into the church is awed" is changed into "If at a certain time, an english spinster comes into the church, then she is awed" or else the deduction rules are changed so that the time-limitations are implicitly carried along and combined during deduction.

13. Composite objects

There is a need to describe the fact that objects can be parts of other objects. We therefore introduce the node type composite object. A composite object consists of a known or unknown number of elements, which may or may not be similar.

If we know which the elements of a composite object are, then we use the ELEMENT short relation from the composite to one or more of its parts.

Example: "John and Mary are married." would be translated into figure 12.

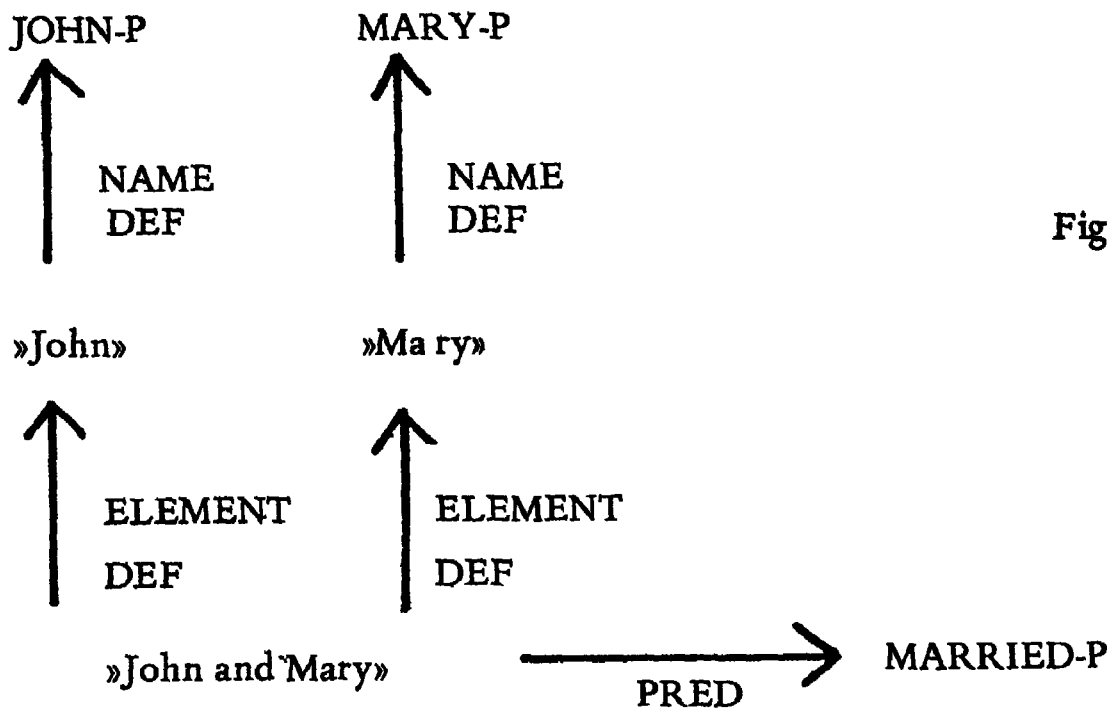


Figure 12

»John and Mary are married»

One might suggest that conjunctions between noun phrases are translated as two separate indenpendent object nodes. "John and Mary are married" would thus be translated in the same way as "John is married and Mary is married". However, this is obviously not the same thing. Sometimes the difference is perhaps not there, for example when we say "John and Mary are human". But the safest way is to create a composite object. This of course requires deduction rules to decide when a property on a composite can be transferred to its elementary parts. This can almost always be done, but not in some cases, for example if we say "John and Mary together are heavier than Peter." But in such cases there is usually some indication in natural language, like the word "together" indicating that the property of the composite cannot be transferred to its elements.

Look again at the picture above showing the translation of "John and Mary are married."

ALL the nodes with DEF on them above are DUMMIES. This means that we first search for a previous-mentioned node with a "NAME JOHN*P" relation on it. If one is found, "John" will merge with it, otherwise "John" will become a new constant and the DEF is changed to ALL. The same thing is done for "Mary"

Thereafter, when "John" and "Mary" have been found in the data base, we try to identify "John and Mary", that is to find in the data base a node whose two elementary parts are just "John" and "Mary". If such a node is found, "John and Mary" will merge with it, otherwise "John and Mary" becomes a new constant and the DEF quantifiers are changed to ALL.

This process ensures that if we first say "John and Mary are married." and then say "John and Mary are going to separate." then the two statements will refer to the same data base node "John and Mary" for both sentences.

There is a risk, however, if we say "John and Mary and their son are a family." and then say "John and Mary are going to London." Then the data base might wrongly identify "John and Mary" in the second sentence with the composite "John and Mary and their son" in the first sentence, and thus wrongly conclude that the son is coming along to London. To stop this, we might require that if there are ELEMENT relations on a node, these must point out all the elements, and not some of them. Thus the data base would know that "John" and "Mary" are the only elements of the composite "John and Mary", and therefore cannot identify this with "John and Mary and their son".

The data base also ought to have a deduction rule which automatically can conclude that there is a PART relation between two composites, if all elements of the first composite are also elements of the second.

14. Conjunctions between noun phrases in the general sense

In the previous chapter I pointed out the ambiguity between sentences like "John and Mary are married" and "John and Mary are human" where the first sentence says that the composite was married, while the second said that the elements individually were humans. I also said that such sentences could always be translated to composites, since properties of composites can in general be trasferred by deduction to the elementary parts.

This is not so easy in the general sense, see the following examples:

"All men and women are getting married."
"All men and women are happy."
"Every man and woman standing together are a married couple."
"All men and women are young people."

Noun phrases in the general sense are translated into VARIABLES in the data base, and these VARIABLES are later used during deduction. In most of the sentences above, the best translation is to create an individual variable for each element, but no variable for any composite. If we say "All men and women are happy" what we mean is "Create a VARIABLE containing all men, and another VARIABLE containing all women, and put the property of being happy on all members of both variables." We do certainly not mean "Create a VARIABLE of man-human couples, and put the property of being happy on all such couples."

In the general sense, conjuncted nouns are therefore not combined into composite objects. An exception is when there is some special indication that such a combination is wanted, like the word "together" in the third example sentence above.

15. Plural nouns

Plural nouns do not simply indicate a set of singular objects. There are also properties which belong to the composite of all the elements together. One of these is the property of being plural, that is of having more than one element. If we say "Two horses are running", then each horse is not plural, neither is each horse two, it is the composite which has these two properties.

Plural nouns must therefore often be translated into composite objects in the data base, since relations on sets in our data base always refer to the individual members of the set, not to the set as a whole.

An exception from this rule is phrases in the general sense like "All men are male humans". Here the plurality is of little importance, and no composite object is created at input translation.

Therefore, when a plural refers to all objects with a certain property, then a variable is created, but when the plural noun phrase refers to some special collection of objects, then a composite object is created.

We introduce the new relation NUM which goes from a composite object to the numeral of it. We also introduce the relation COMPLEX which goes from a composite object to a predicate which applies not necessarily to the composite, but which applies to all its elements. Examples in figure 13.

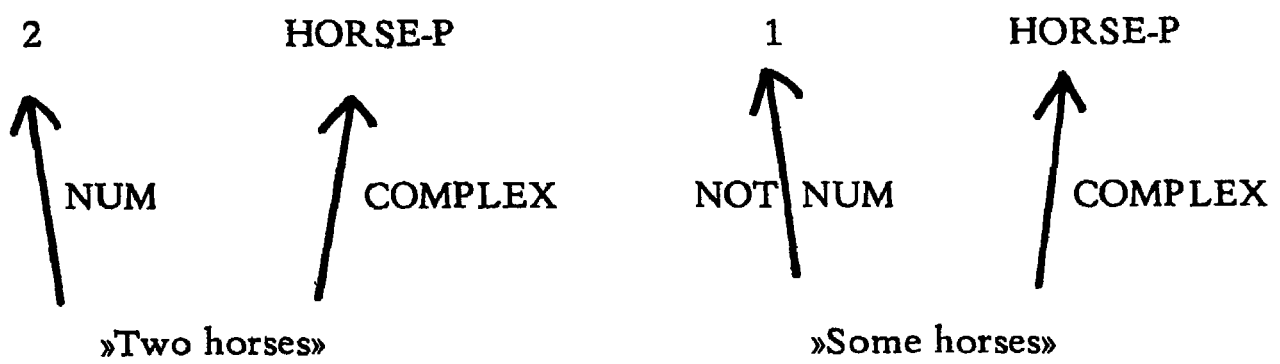


Figure 13

This means that the data base must be able to make deductions on numbers, e.g. to deduct that if a composite has the relation NUM 2, then the relation NOT NUM 1 can be deduced. This is necessary e.g. to merge these sentences into the data base in a correct way:

"Two horses are coming. One of the horses is sick."

To identify "the horses" in the second sentence with "two horses" in the first sentence, deduction must infer NOT NUM 1 from NUM 2.

There may also be a need to have in the data base management a routine for counting the number of elementary parts of a composite, so that the NUM numeral can be deduced if all parts are known.

16. Fitting composite objects into the sentence

The general rule is that when two conjuncted nouns have been translated into a composite object, then it is this composite objects and not its parts which is fitted into the sentence framework.

This is obviously the correct translation e.g. when you say "The road between Stockholm and Gothenburg" where "between" refers to the composite, but not to the elementary parts singularly. ("The road between Stockholm" is not right).

However, for a phrase like "Every man and woman in the city" we do not want to find only couples of men and women, so general sense noun phrases are not translated into any composites at all. The parts are fitted separately into the sentence framework instead.

If we say "The father and the mother of Mary is coming", then obviously it is the composite which is "coming", but it is not so obvious that "of" refers to the composite. Suppose that we previously in the data base have got a father of Mary and a mother of Mary, but no composite of these two. "The father" and "The mother" are translated into two DUMMIES, and we first search to identify these in the data base, before trying to identify the composite. But when we try to identify "The father" we do not want to find the closest previous-menthioned father, we want to find the closest previous-menthioned father of Mary. Therefore, the rule for prepositions is that to the left, they refer to the elementary parts but to the right they refer to the composite.

Se example in figure 14.

Example: "The road and railway between Stockholm and Gothenburg is blocked."

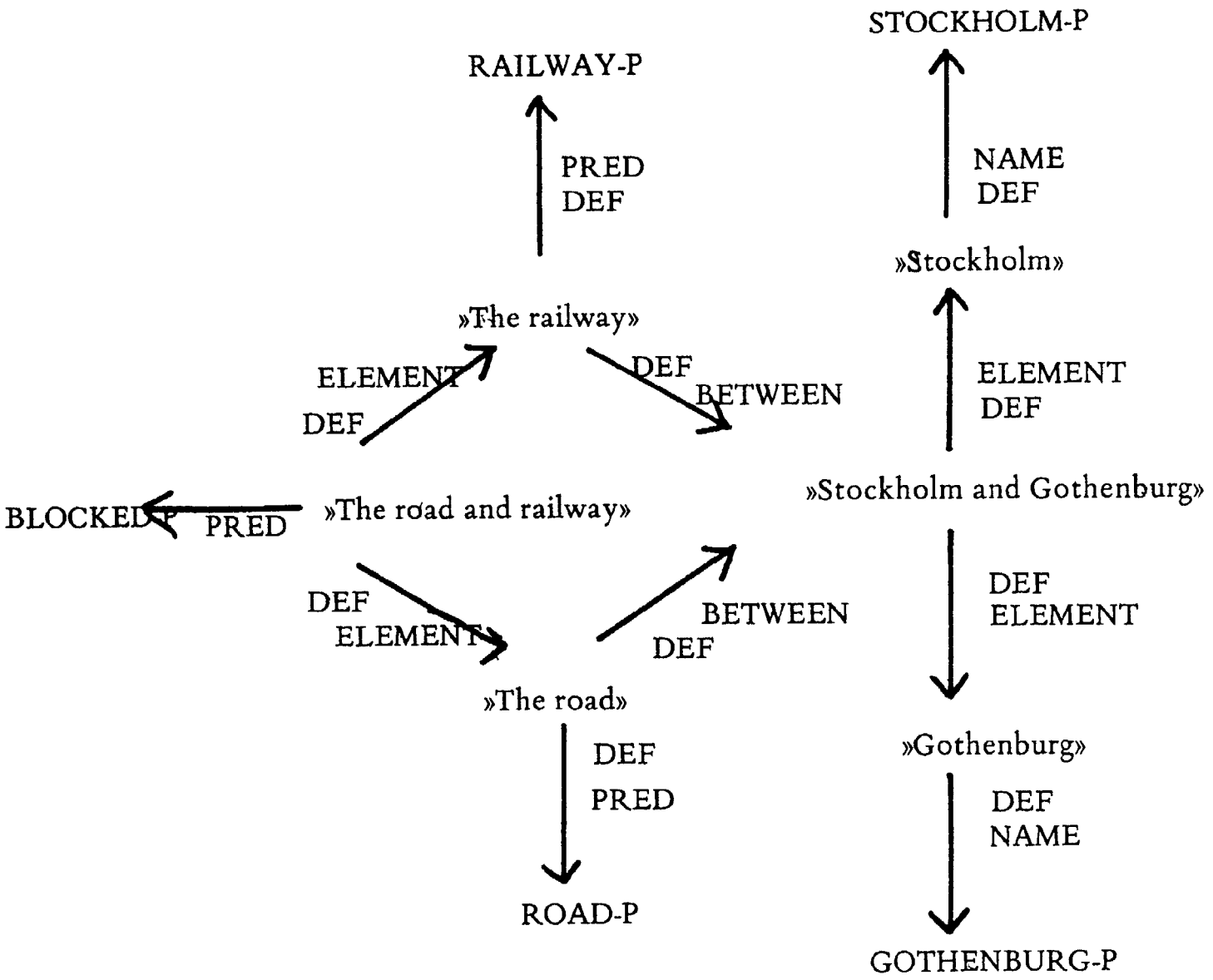


Figure 14

As seen from the picture, between goes from the elementary parts "the road" and "the railway" to the composite object "Stockholm and Gothenburg".

17. Noun phrases with just a number and nothing more

Some natural languages contains constructs where a noun phrase consists of only a number, usually followed by a preposition. Example "One of the horses" or "Two of the horses".
Here, as usual "one" creates a singular set, while any number except "one" creates a composite object. The relation "of" is in this case translated into

Noun phrase before the "of"	Noun phrase after the "of"	
	A composite object	No composite object
A composite object	PART	ELEMENT ITS
No composite object	REV ELEMENT	EQUAL ITS

Examples:

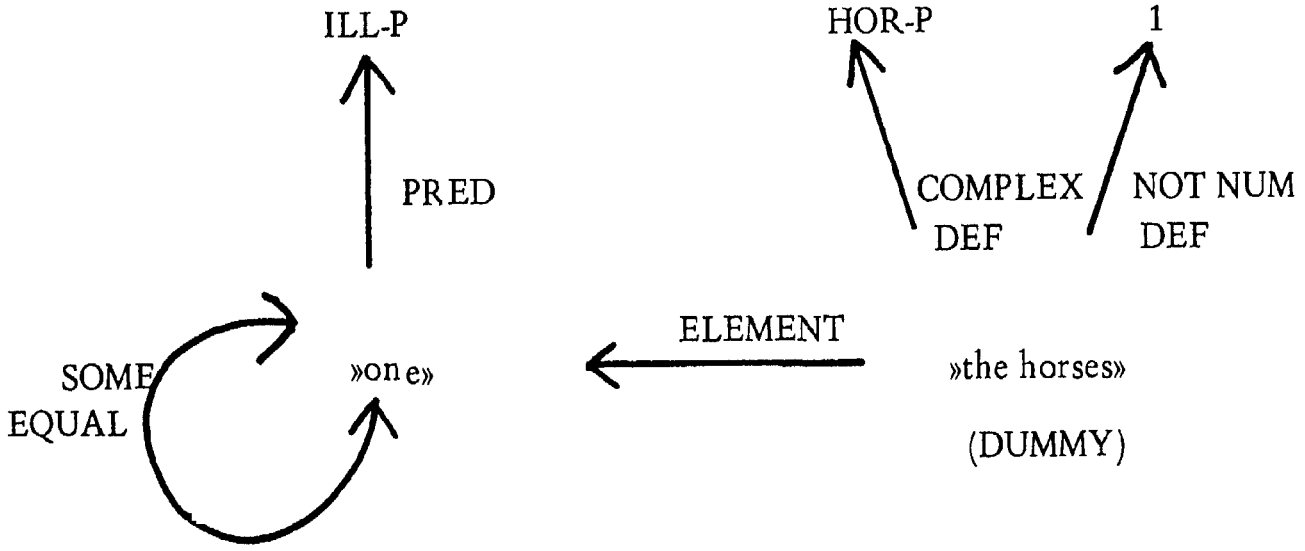
Two of the horses: PART

One of the horses: REV ELEMENT

Two of all horses: ELEMENT ITS

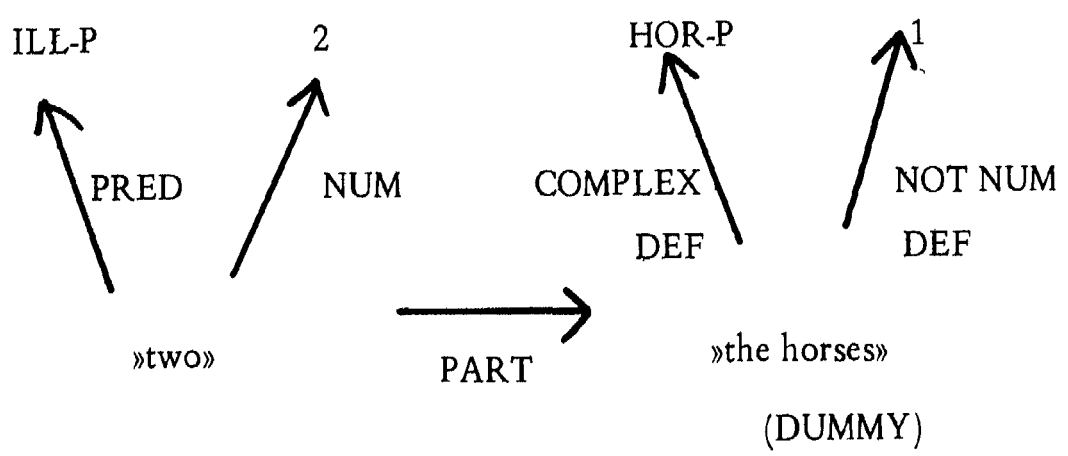
One of all horses: EQUAL ITS

18. Some examples of translations of sentences with plural nouns



»One of the horses is ill»

Figure 15



»Two of the horses are ill»

Figure 16

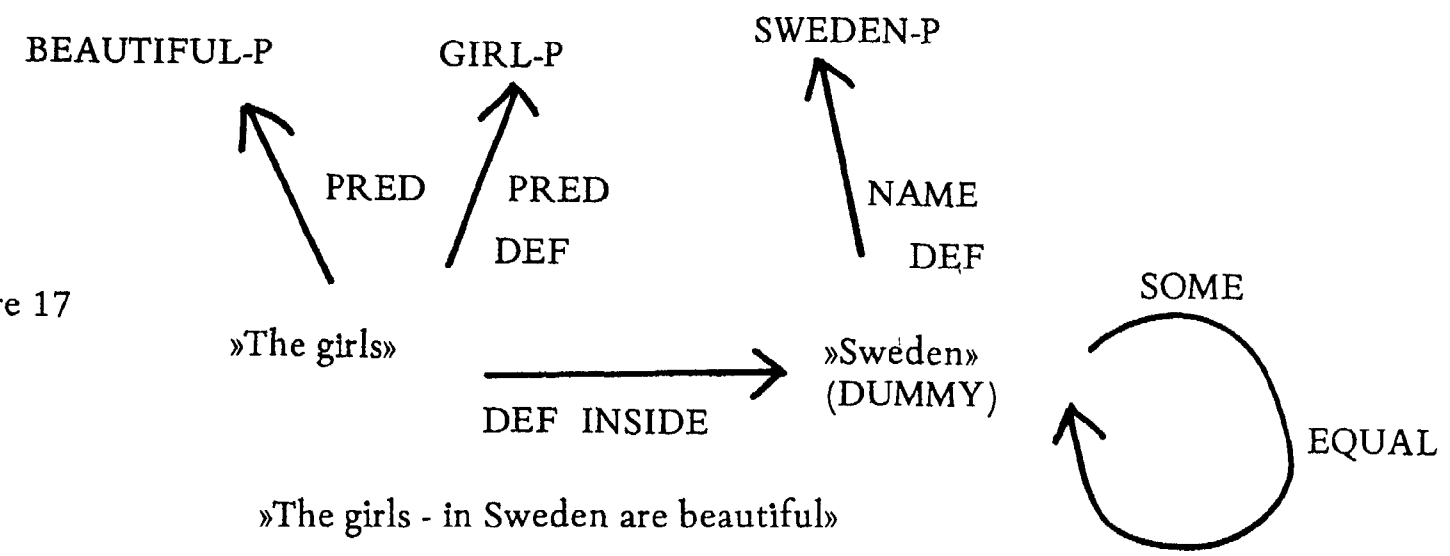
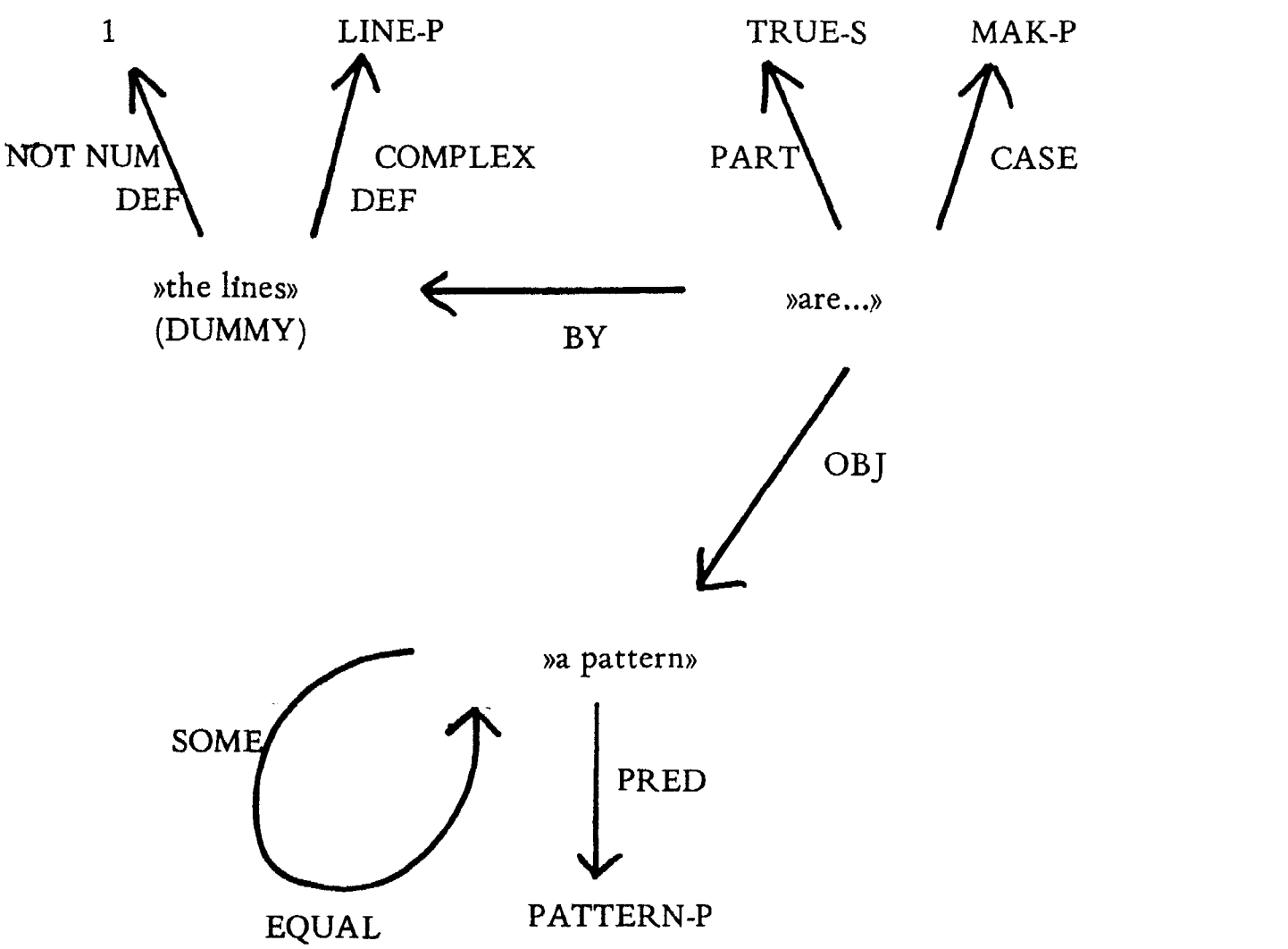


Figure 17

»The girls - in Sweden are beautiful»



»The lines are making a pattern»

Figure 18

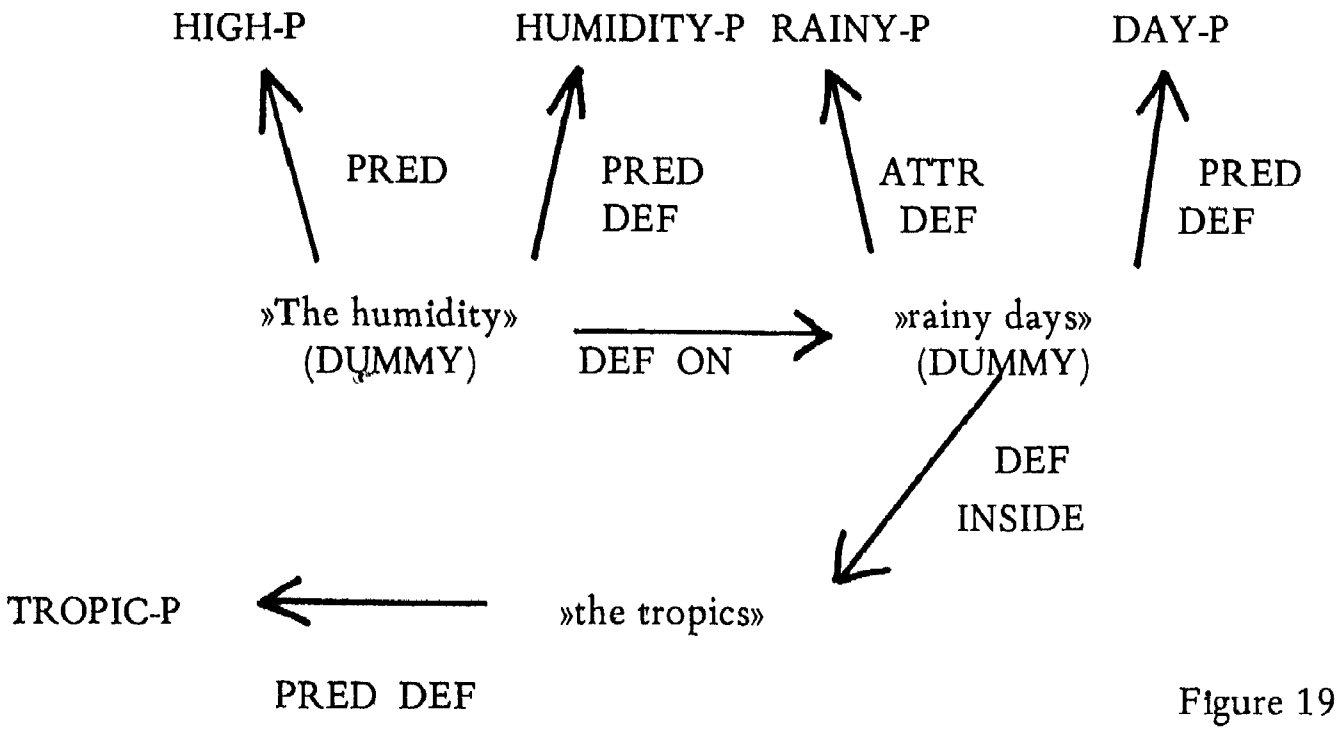
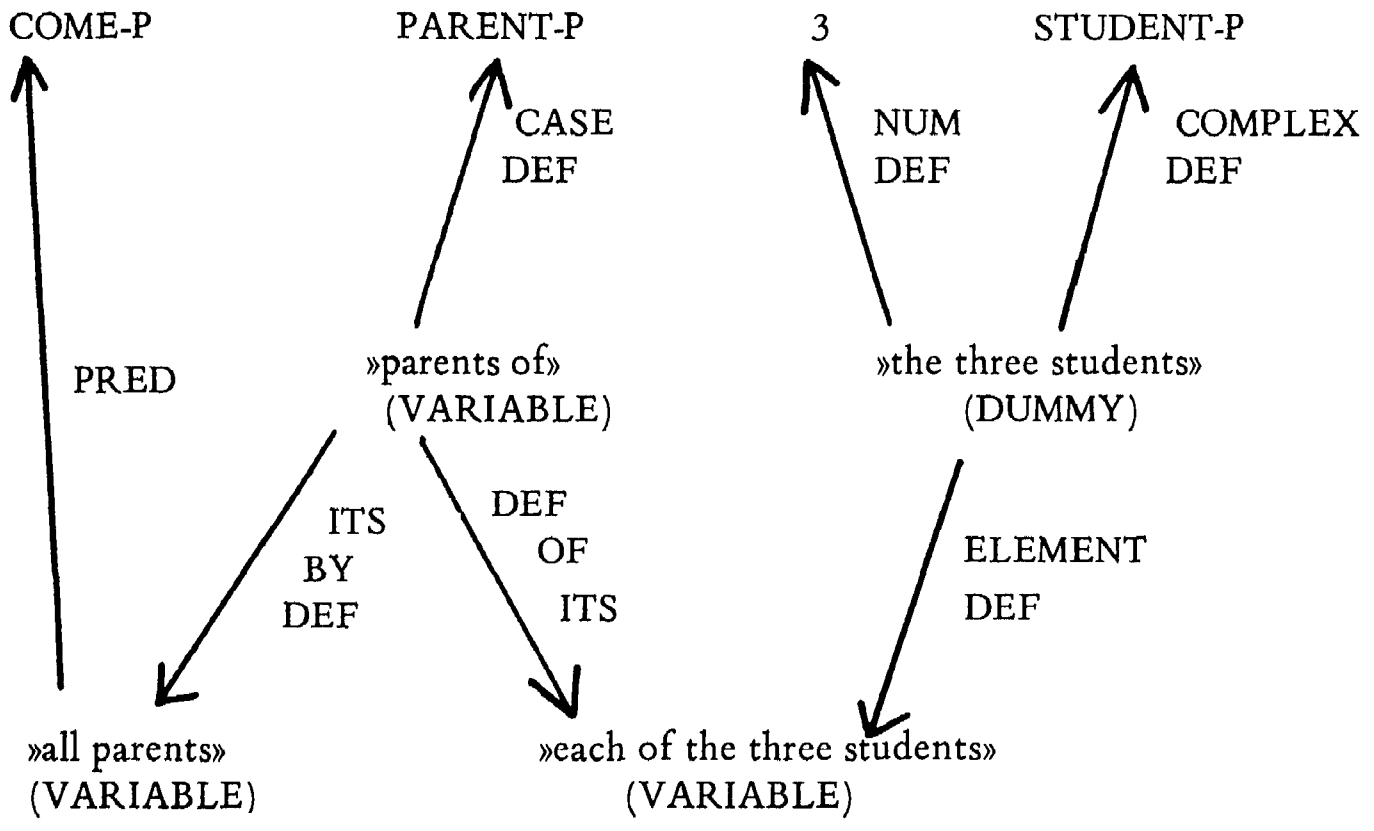


Figure 19

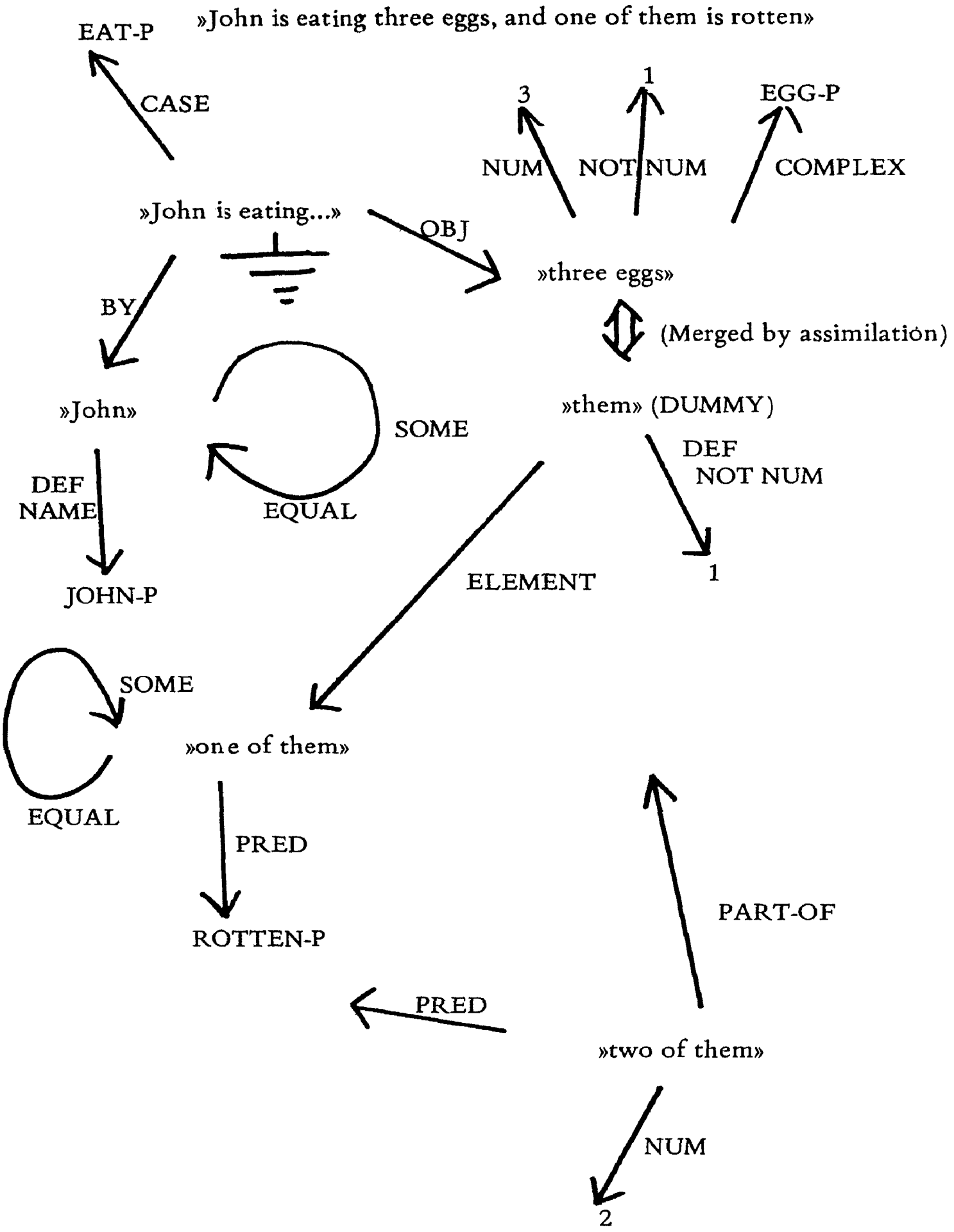
»The humidity on rainy days - in the tropics is high»



»All parents of each of the three students are coming»

Figure 20

Figure 21



»John is eating three eggs, and two of them are rotten»

Figure 22

19. Problems with the dual representation of nouns

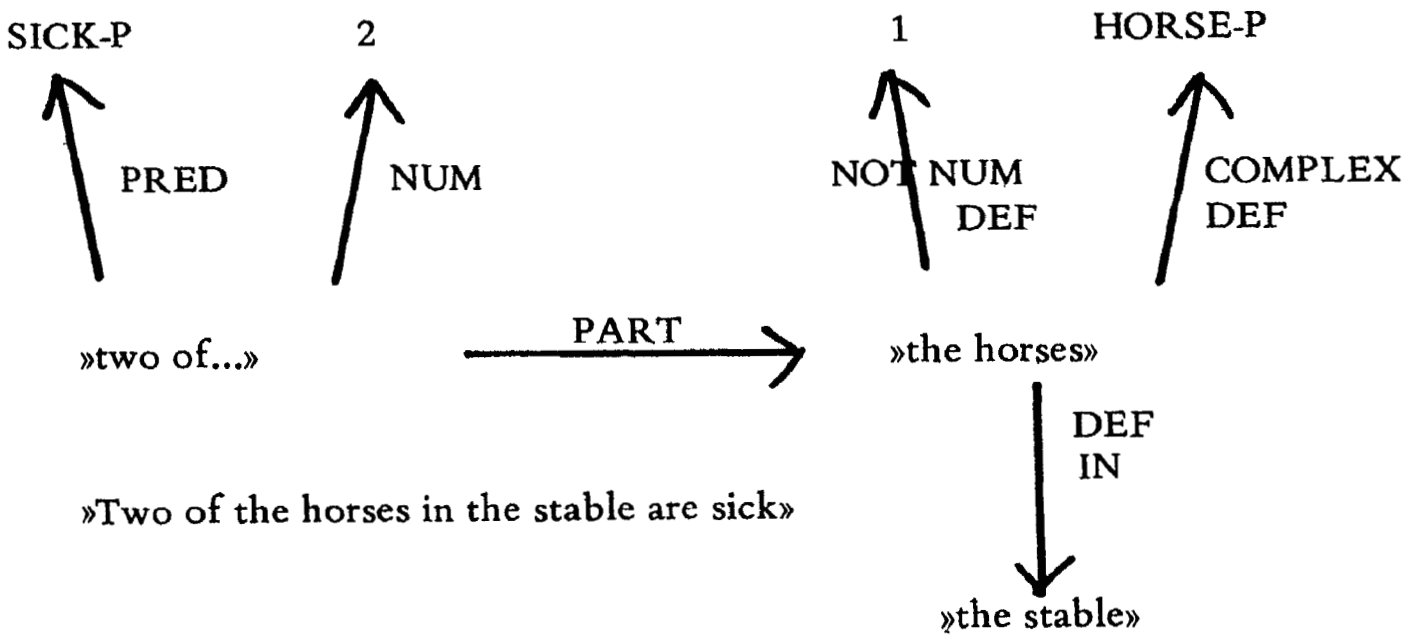
As has been explained above, nouns must sometimes be translated to singular sets (for special sense singular nouns), to defined sets (for general sense nouns) or to composite objects (for most plural nouns). This duality is necessary, but it also will make deduction more difficult, since the deduction rules must be able to make inferences from the composite to its parts. The deduction rules must also sometimes be able to create auxilliary composites or auxilliary non-composites.

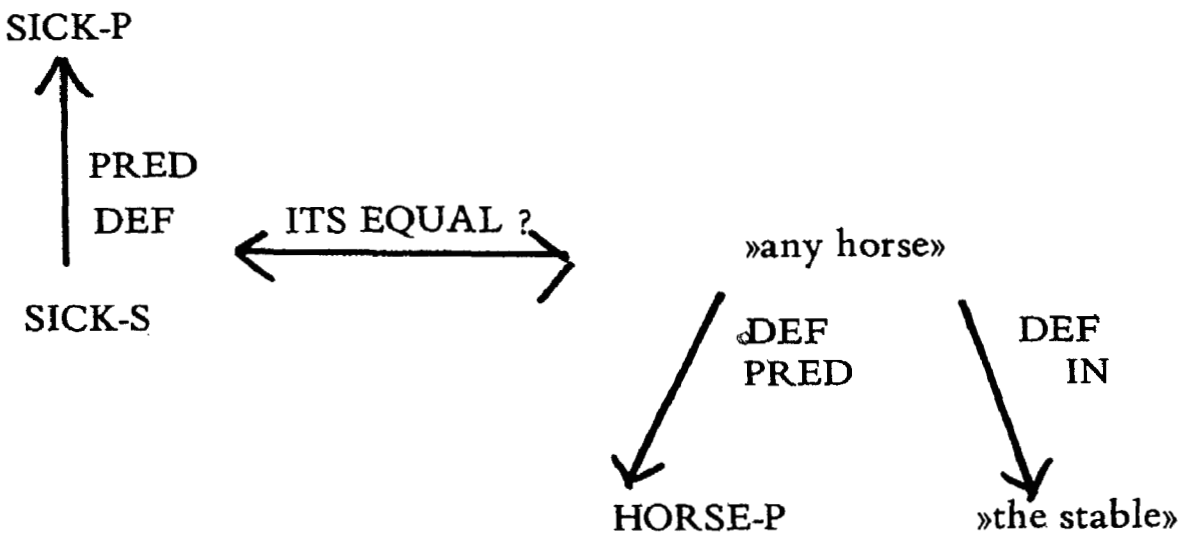
Example I: "One man and three women are coming. How many men and how many women are coming?" Here, deduction will probably have to create a help-composite for the single man, since only composites have numeral on them, and the question asks for this numeral.

Example II: "One or more men is coming." The natural translation of this is into a composite object with NUM to "One or more". But if we later learn, or can deduce from the data base, that it is only a single man, then a singular non-composite node probably has to be created.

Example III: "Soldiers are cruel. This is because they are scared." In the first sentence, "soldiers" is used in the general sense and thus a defined set is created. But in the second sentence, the translation will first translate "they" into a DUMMY looking for a composite object. When the routine for merging the second sentence into the data base finds the defined set for "soldiers", it must recognize that a DUMMY looking for a composite object can merge with a non-composite defined set in the data base.

An even more complex problem for the deduction routines will occur if we say "Two of the horses in the stable are sick. Is any horse in the stable sick?" which will be translated like in figure 23.





»Is any horse in the stable sick?»

Figure 23

The question-answering routine will be asked to answer the question "ITS EQUAL" (that is: SUBSET) and to do this it must in some way recognize that a member of the defined set SICK*S is an element of the composite "two of", and thus is sick.

20. Equality between composite objects

The natural language phrase "The father and the mother is John and Mary" cannot be translated with an EQUAL relation between the two composites for "the father and the mother" and "John and Mary". EQUAL says that all members of two sets

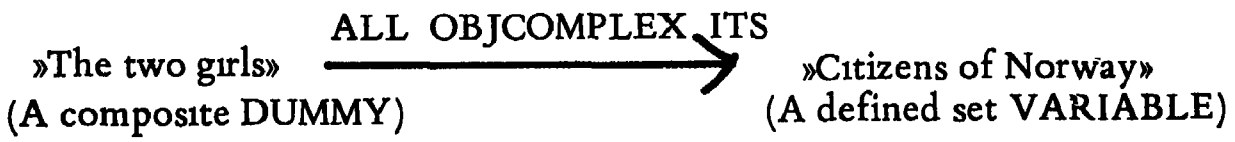
are the same so such a translation would say that there is a composite object which has four elementary parts: "the father", "the mother", "John" and "Mary".

Therefore a new relation SAME is introduced into the data base. SAME goes between two composite objects, or between a composite object and a non-composite object. SAME says that both object nodes represent the same reality, but viewed from different viewpoints, described by a different set of descriptions.

"John and Mary are a married couple" will also be translated with a SAME relation between the two nodes. "John and Mary" is a composite object, and "a married couple" is a non-composite, and EQUAL would therefore mean that a node can be both composite and non-composite at the same time. To avoid this confusion, the SAME relation is used.

SAME is thus used to indicate a relation between two different descriptions of the same reality. But SAME cannot be used between a composite object and a defined set containing its elementary parts. ELEMENT might be used here, but ELEMENT refers to one of the parts, not to all the parts. Therefore a new relation OBJCOMPLEX is used. OBJCOMPLEX refers from a composite object to a set of all its parts.

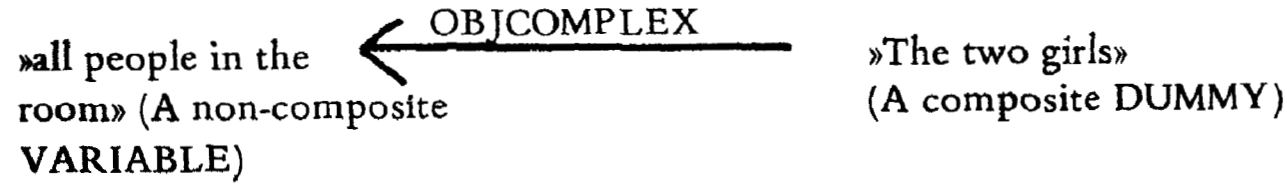
Example: "Two girls are citizens of Norway" would be translated like in figure 24.



»Two girls are citizens of Norway»

Figure 24

Second example: "All people in the room are the two girls" is translated into figure 25;



»All people in the room are the two girls»

Figure 25

Subject noun	Predicate complement noun: composite	Predicate complement noun: non-composite
composite	SAME	OBJCOMPLEX
non-composite	REV OBJCOMPLEX	EQUAL

21. Relations between predicates

Predicates form an hierarchical structure, e.g. VERTEBRATE*P is a special case of ANIMAL*P, HUMAN*P is a special case of VERTEBRATE*P, KING*P is a special case of HUMAN*P a.s.o. To indicate this we use the relation SUBPRED, as in figure 26.

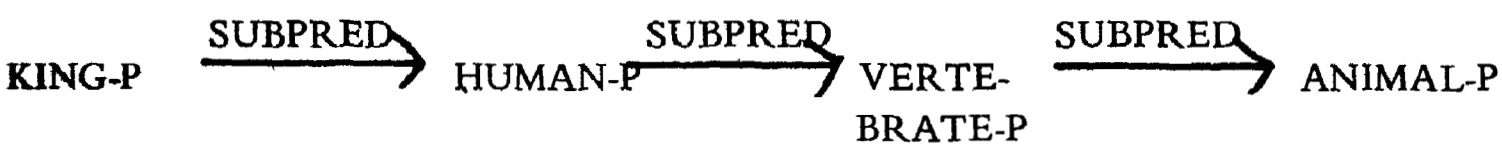


Figure 26

This means that some simple sentences can be translated as relations between predicates. For example, the sentence "Every man is a human" can be translated like in figure 27.

Figure 27

which is much simpler than the other translation, in figure 28.

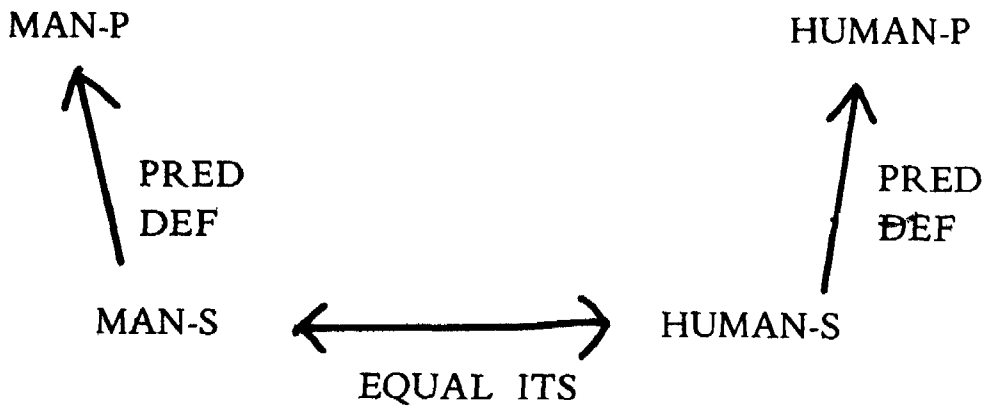


Figure 28 »Every man is human»

To be able to give this simple translation to adjectival predicates, we also have the short relation SUBATTR, so that "Every man is a male human" can be translated to figure 29.

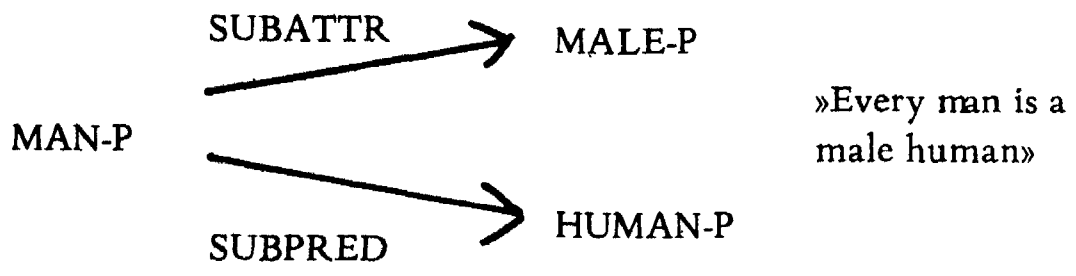


Figure 29

The difference between SUBATTR and SUBPRED is the same as between ATTR and PRED in figure 11. In the above case, there is no semantic difference.

22. Event nodes

Many natural language phrases combine several nodes (objects, defined sets, properties) into a statement which can have limitations in space, in time, in its truth value, and which can have a cause, a result etc.

The central node in the translation of such phrases is the event node. Event nodes are used in our system not only for typical events like "John went to the cinema with Mary" but also for more sustained "events" like "John is the fiancee of Mary" or even "John is the father of Mary"

The most important relations on an event node are BY to the subject, CASE to the predicate, OBJ to the object, and PART to the validity environment . Example: "John is riding the bike" is translated as in figure 30.

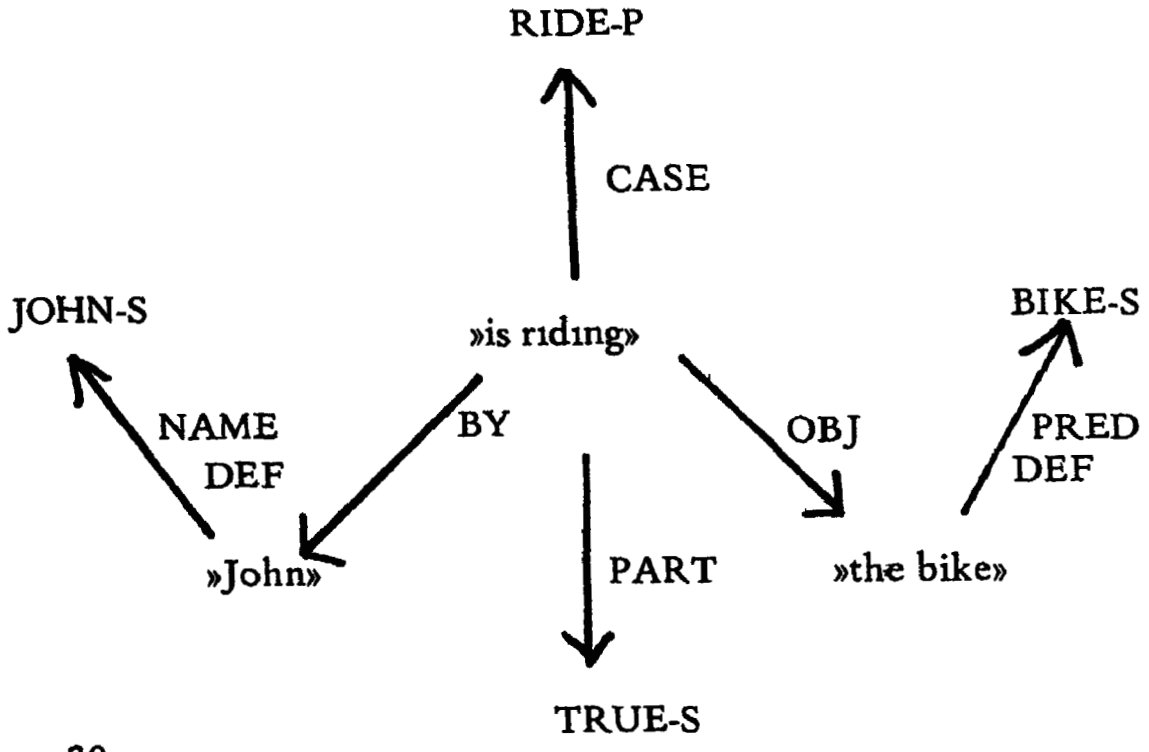


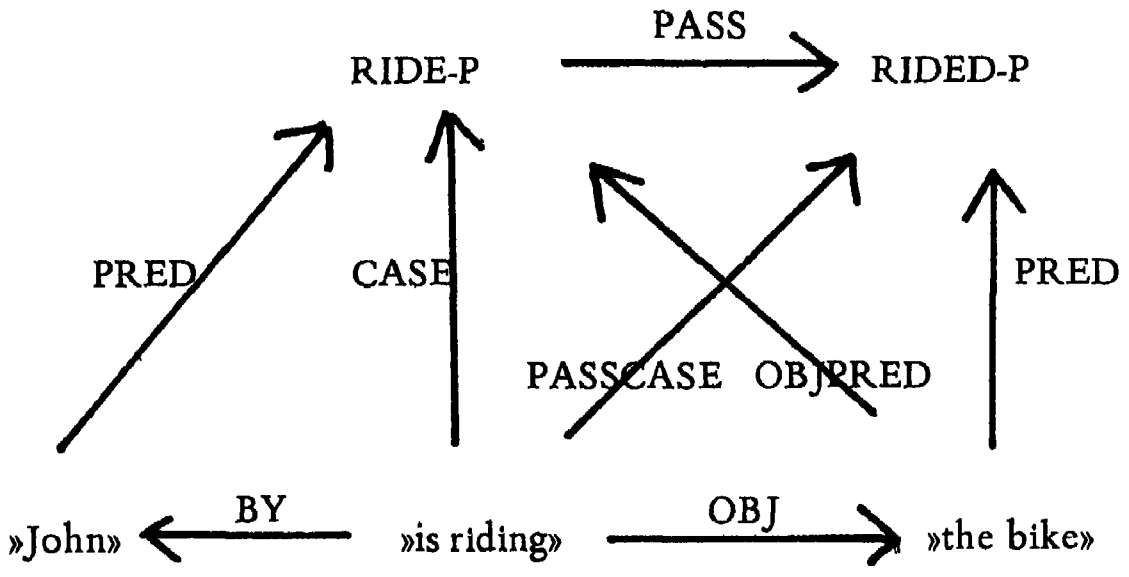
Figure 30

»John is riding the bike»

From a valid event node (that is at the time and place of the event etc.) the deduction procedures can deduce e.g. a PRED relation from "John" to 'RIDE*P', and these deduced relations are very useful in later deduction. There is also a symmetric relation OBJPRED from the object to the predicate. If we can deduce that some object has OBJPRED to a predicate like RIDE*P, then we can deduce that that object is being ridden, that is that the predicate RIDED*P (the passive of RIDE*P) is appliable to the object.

We can therefore draw the following figure 31 of relations:

Figure 31



»John is riding the bike» & »The bike is ridden by John»
including implicit short relations

All of these relations do not have to be produced for every sentence, since some of them can be deduced from some others by chaining rules like:

X BY Y & Y CASE Z implies X PRED Z

X PASS Y & Z PASSCASE Y implies Y CASE X

Several more triangles in the figure form such chaining rules, although not all of them. (Even if John is riding and the bike is ridden, we cannot therefore conclude that John is riding just that bike). All the chaining rules involving the event node are true only when that event node is true, or valid when the event node is valid.

If the data base contains a verb both in active and passive form, then there must be a relation PASS between them to permit deduction. Since passive forms are less common than active, this PASS relation is generated whenever a passive verb appears.

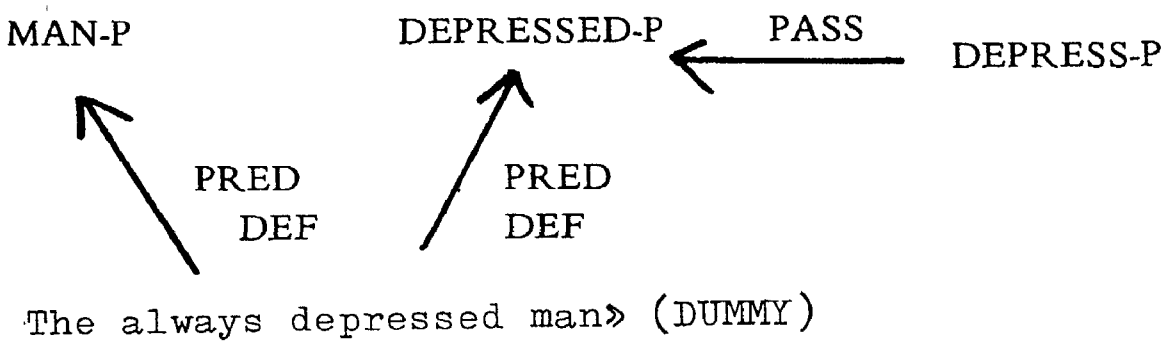


Figure 32

The direct relation OBJPRED from "the always depressed man" to DEPRESS*P is thus not created by input translation, but it can of course easily be deduced.

In the same way, "The bike is ridden by John" is translated like in figure 33.

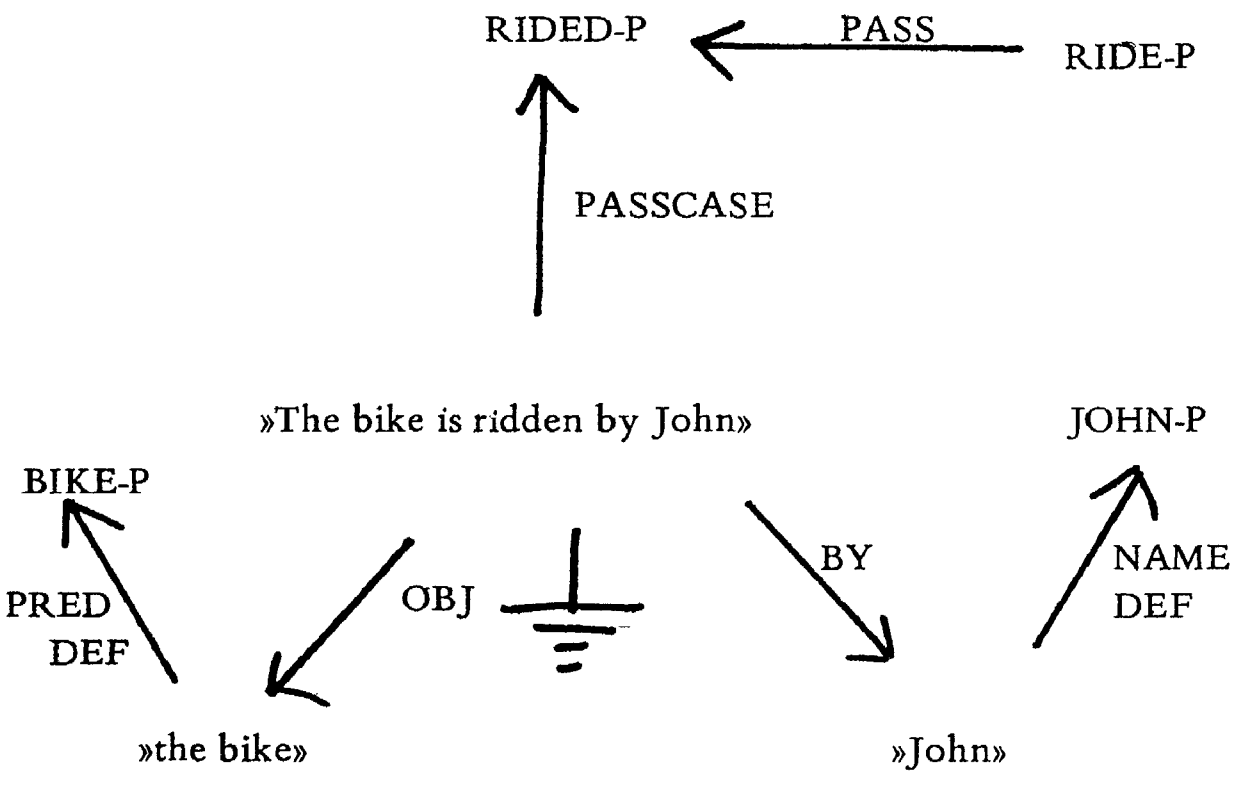


Figure 33

The CASE relation from the event to RID*P is not output explicitly, but can of course easily be deduced.

One could argue that we could avoid passive verbs altogether in our data base by always using the CASE and OBJPRED relations. There are two arguments against this:

a) It is valuable always to have the relation PRED from a noun to all properties on that noun. It is not systematic to need the relation OBJPRED to some properties.

b) Our representation makes it very easy to represent statements like "Someone who is killed, is dead" simply by KILLED*P SUBATTR DEAD*P which otherwise would have to be represented by a VARIABLE in the way in figure 34.

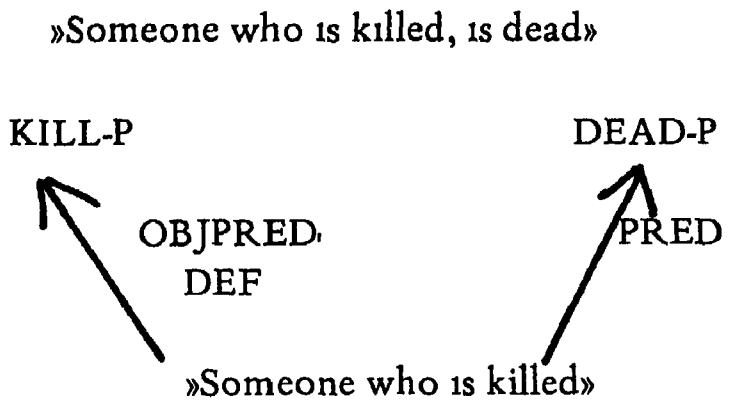


Figure 34

23. Putting restrictions on equality

One can see event nodes as a way of adding restrictions in time and space etc on PRED relations. The event nodes are necessary because our data base does not permit us to put short relations on short relations.

Sometimes there is a need to extend the SUBSET relation in the same way. PRED and SUBSET are very similar relations,

although PRED goes to a predicate, SUBSET to an object set.

Since SUBSET is a special case of the EQUAL relation, it is really the EQUAL relation which we want to extend into an event. We therefore introduce a new relation OBJCASE so that Y BY X & Y OBJCASE Z implies X EQUAL Z whenever the event Y is true or valid. The relations thus form the triangle in figure 35.

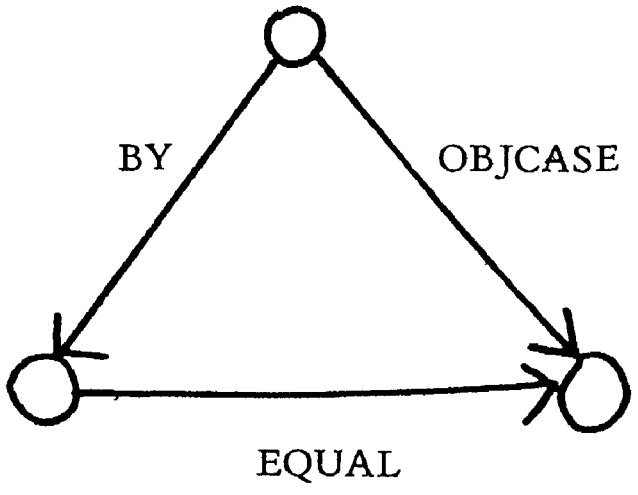


Figure 35

The OBJCASE relation is used when natural language equality has to be translated into relations between object nodes.

An example is given in figure 36.

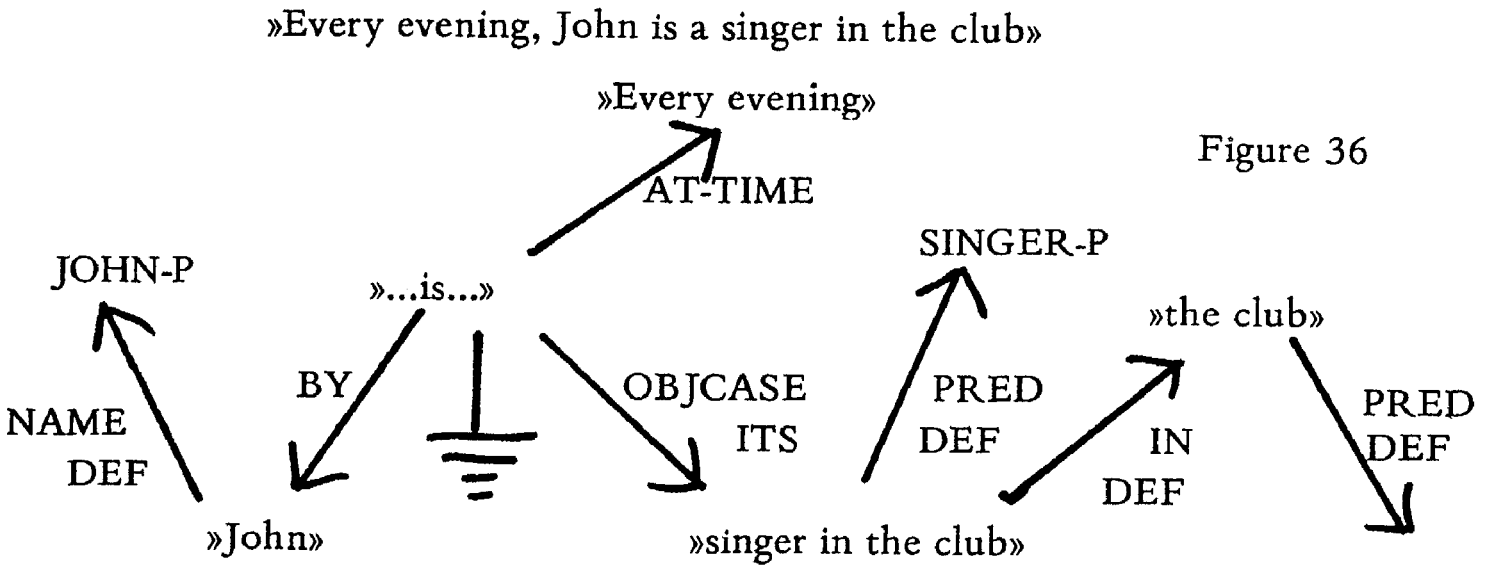


Figure 36

From this network, we can deduce that if the event is valid, e.g. in the evening, then "John" is a SUBSET of "singer in the club".

Since all relations expand into a chaining rule with EQUAL: "X R Y & Y EQUAL Z implies X R Z" and since EQUAL can be expanded into an event node using BY and OBJCASE, this can be used to expand any short relation into an event node. For example, "After 1972, Britain is a part of EEC" requires us to expand the PART relation between Britain and EEC into an event, to be able to add a time limitation to that PART relation. This can easily be done by expanding EQUAL into BY x OBJCASE in the way in figure 37.

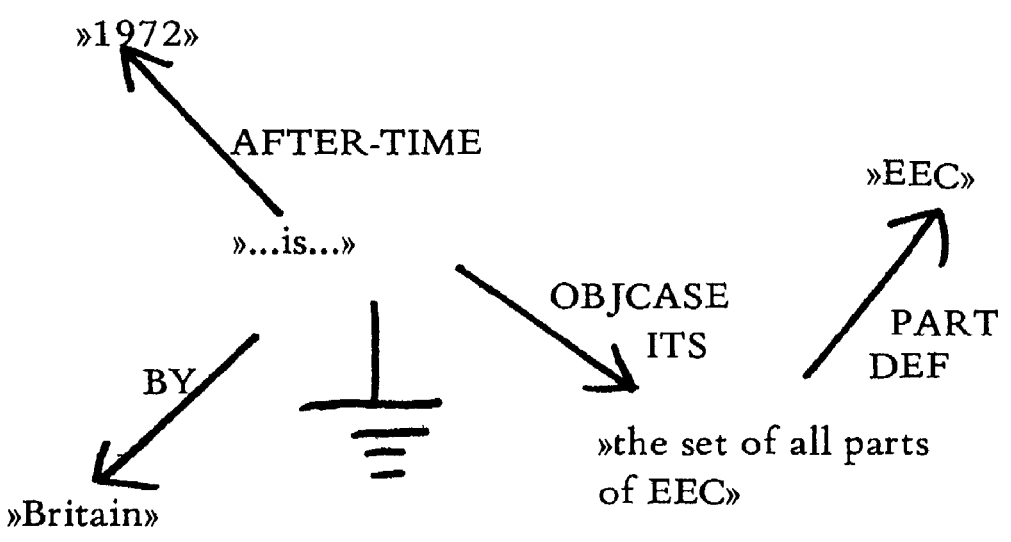


Figure 37

23b. Quantifiers on event nodes

Consider the sentence "A girl is giving every man a flower". This sentence could be interpreted in the following way: "There is a set of events, one for each man. One and the same girl is giving a different flower in each such event". In our data base, this is represented like this:

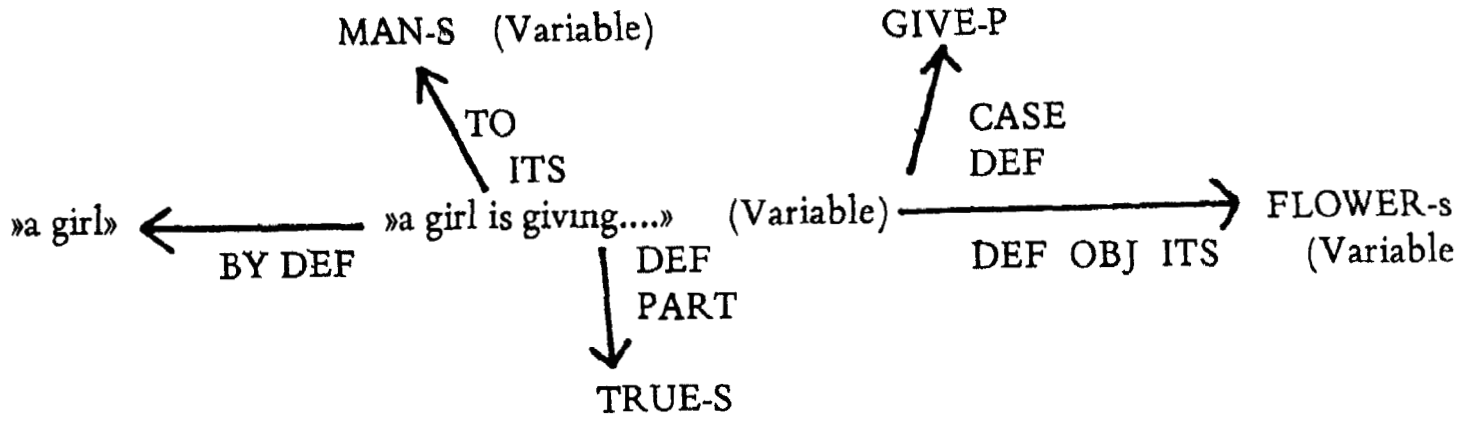


figure 37a

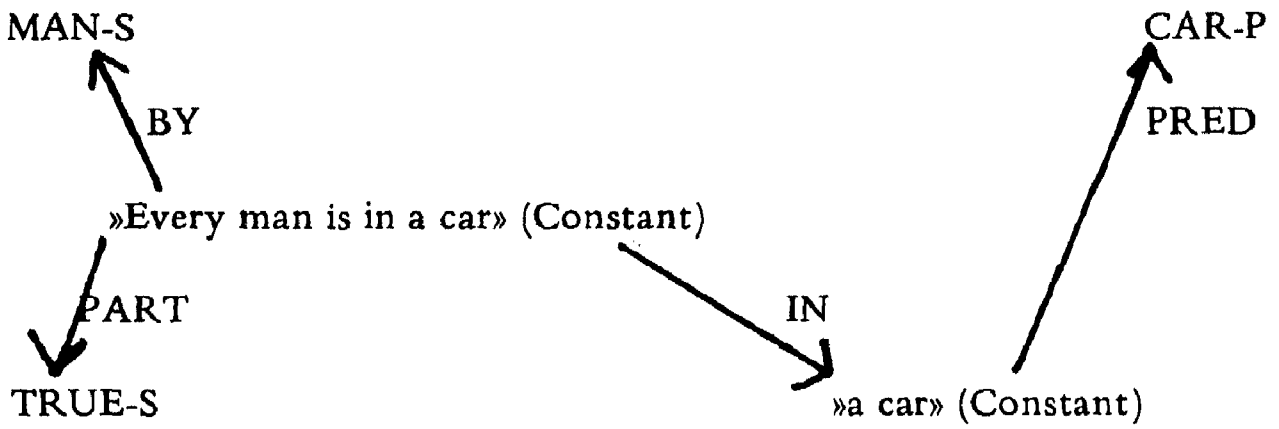
In the translation above, the two noun phrases "a girl" and "a flower" are interpreted in different ways. "a girl" is interpreted as one single girl, while "a flower" is interpreted as a set of different flowers, one for each man.

These two interpretations of "a" are called respectively the singular sense and the distributed sense. Other determiners than "a" have the same ambiguity, for example "some".

"a car" in the sentence "Every man is in a car" can be interpreted in the singular sense (one single car) or in the distributed sense (one car for each man).

In the singular sense the interpretation will be:

figure 37b



And in the distributed sense the interpretation will be:

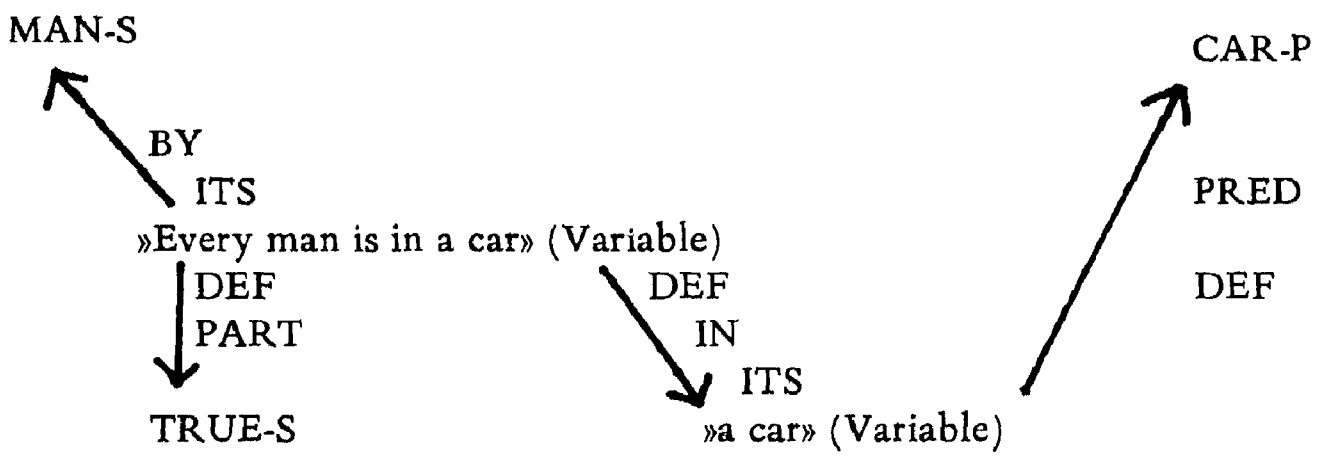


figure 37c

One can note that we can later refer back to the car only with the singular sense interpretation. Example: "Every man is in a car. The car drove away". This also corresponds to the interpretations in the figures above, where only the singular sense provides a node to refer back to.

Such a back-referencing could thus be used to disambiguate this kind of ambiguous sentence.

If you compare the two figures above, an important difference is that the event node is a constant in the singular sense, a variable in the distributed sense.

The translation rule is that if all the noun phrases marked with "a" or "some" are to be interpreted in the singular sense, then the event can become a constant. If, however, one of the noun phrases marked with "a" or "some" is to be interpreted in the distributed sense, then we must have one copy of the event node for each copy of the distributed noun, so the event must become a variable.

If the event is translated as a variable, then the quantifiers on the relations between the event and the noun phrases should be:

DEF-ALL to singular sense and constant nouns,
DEF-ITS to distributed sense nouns,
ITS-ALL to nouns marked with a general quantifier like "every" or "all" or "each".

Examples:

"A man and a woman are everyone - in the house"

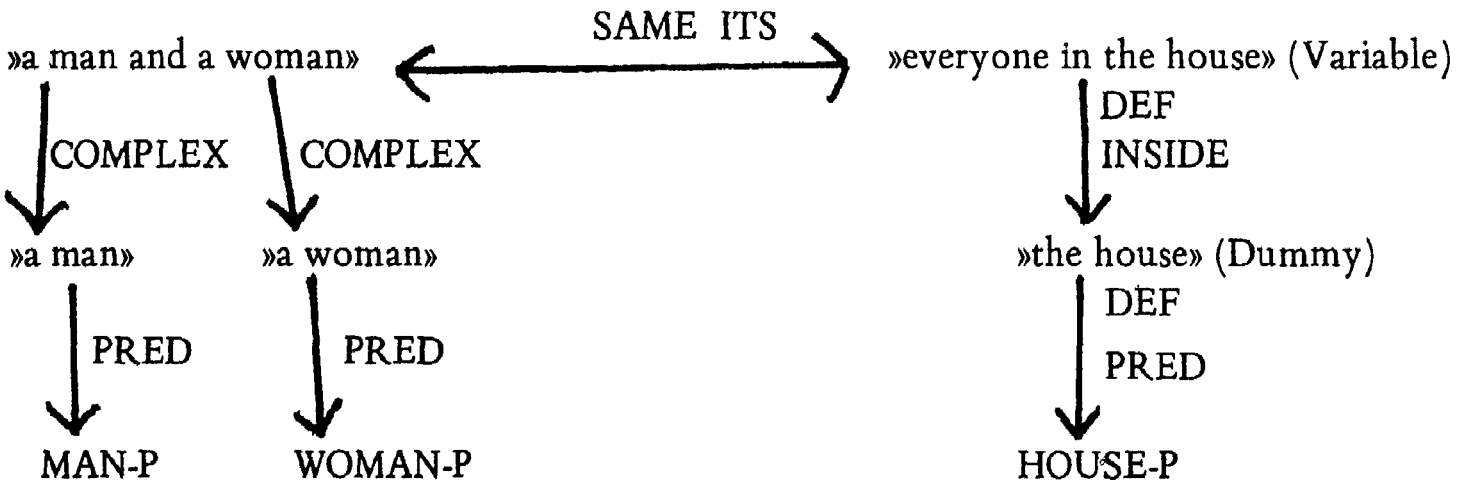


figure 37d

"In every city, some woman is in a hospital"

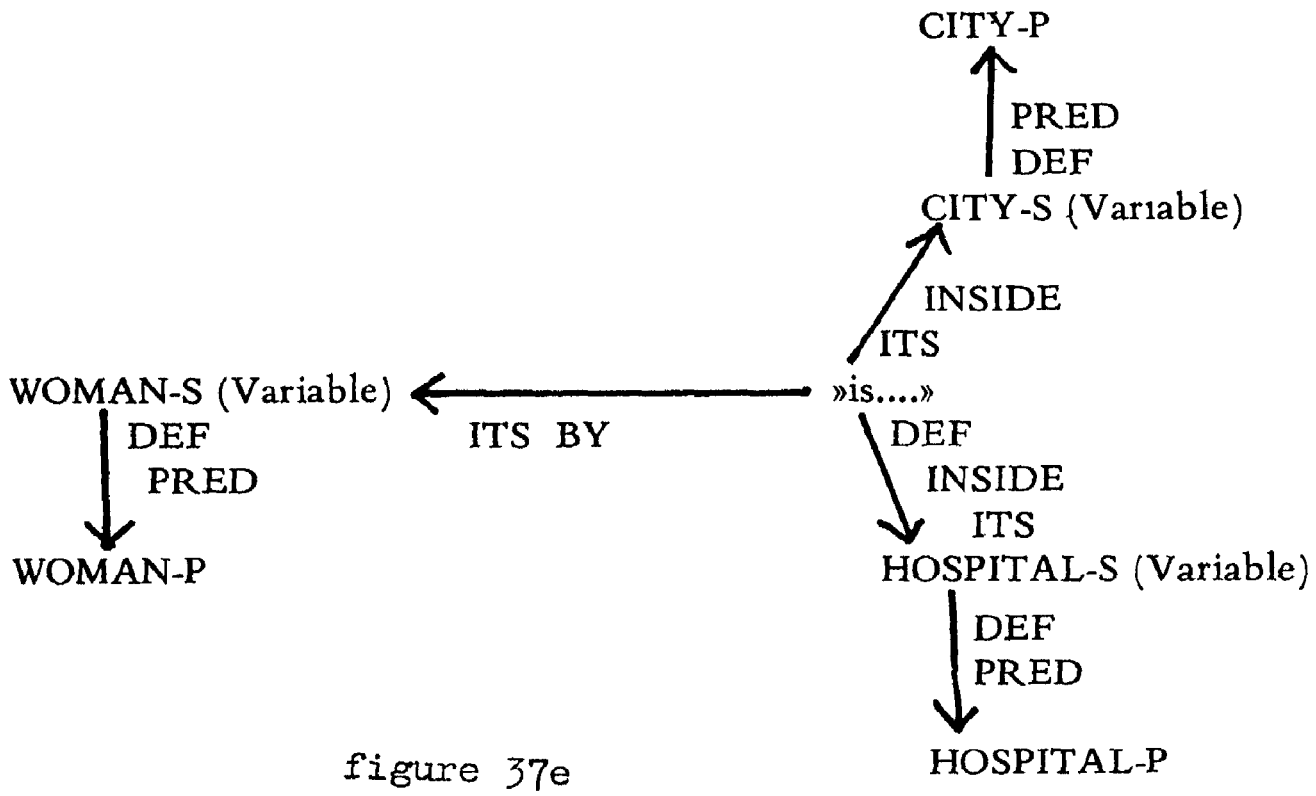
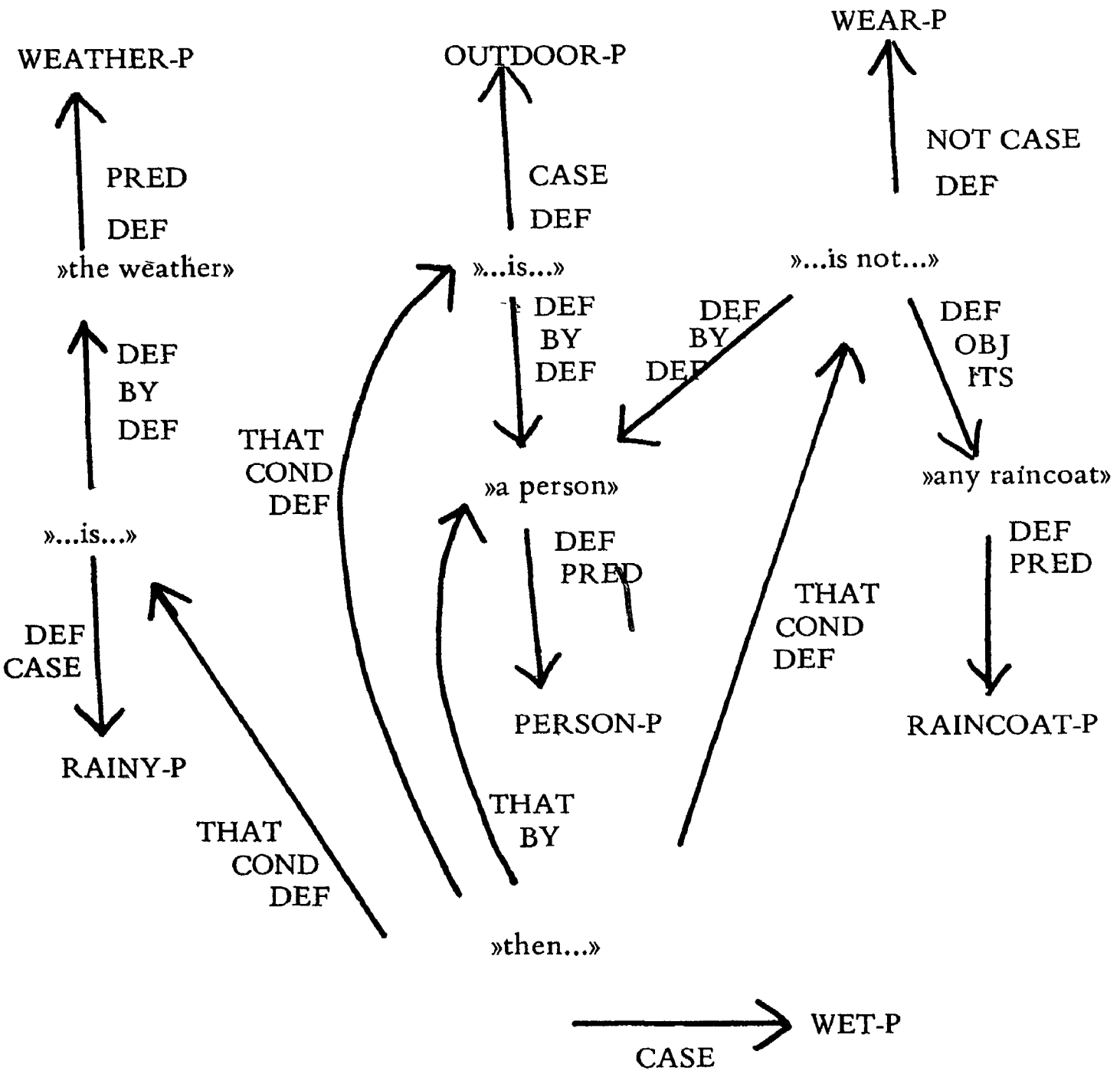


figure 37e

24. Deduction patterns and natural language if-clauses

A natural language statement like "If the weather is rainy and a person is outdoors and the person is not wearing any raincoat, then the person will become wet." introduces deduction rules into the data base. These rules are only valid if a pattern of facts in the data base can fit into the pattern created by the deduction rule.

Such deduction rule patterns are called keys in our system. After merging into the data base, the statement above may look like in figure 38.



»If the weather is rainy and a person is outdoors and the person is not wearing any raincoat, then the person will become wet»

Figure 38

A new quantifier "THAT" is introduced above. The reason for this is that if there are two different persons, one who is outdoors, and another who is not wearing a raincoat, then we do not want to conclude than any of them necessarily will become wet. We therefore have to single out in the data base one person and two events in which this person is the subject. One of the events should say that he is outdoors, the other

that he is not wearing a raincoat. We therefore have a key of one person and two events, which have to be fitted with facts in the data base when the deduction rule is used.

The quantifier "THAT" refers from a conclusion to the deduction pattern key. It means to single out that member of the referred set to which the whole pattern has been matched.

In the figure above, "the weather is rainy" is not part of the pattern. But in reality, there is in the natural language text an implicit time and place indication: "If, at a certain place, at a certain time..." and this place and time will fit the weather into the pattern key.

Our program does not yet handle such implicit time and place indications.

There are two short relations for "imply" in our data base: COND and IFTHEN. COND refers to necessary conditions, IFTHEN to sufficient conditions. To handle cause and effect patterns, and the resulting structure of situations depending on each other, probably more such relations are necessary, but we have not introduced them yet.

A somewhat simpler notation is available for some simple cases. This is the COP short relation, which refers to a hypothetical copy. Thus, "If the bridge is low and weak, then it will break" can be translated like in figure 39.

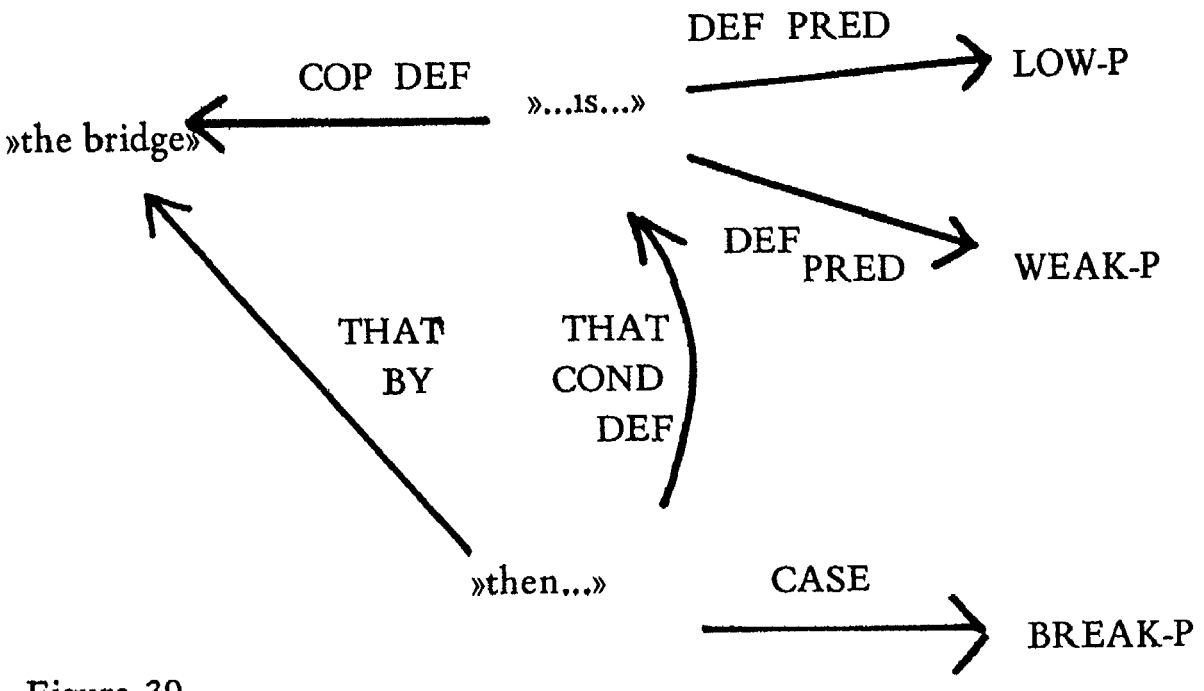


Figure 39

»If the bridge is low and weak, then it will break»

Thus, if-statements in natural language introduce a pattern key of variables, connected with DEF-DEF relations. And the conclusion refers to this pattern with relations with the quantifier THAT on the pattern end.

A natural language if-statement in a question is translated in a quite different way. The statement "If the weather is rainy and John is outdoors, will he then be wet?" is translated like this: "Add the temporary facts that the weather is rainy and that John is outdoors into the data base. Thereafter try to deduce if he will be wet. When the question has been answered, then remove the temporary facts from the data base again.

Compared to other natural language systems, one characteristic of our system is the representation of deduction rules as variable patterns in the data base. Other often used representations are

a) Predicate calculus clauses.

b) Executable programs in some special programming language.

The advantage with our system is that the representation of deduction rules is so closely integrated with the representation of facts. The simplest deduction rules, the chaining rules, simply are rules for traversing the data base graph from node to node. The more complex deduction rules are patterns very similar to the data base facts which these patterns are to match during deduction.

If a predicate calculus representation is used, then efficient deduction requires some algorithm for selecting those clauses which might match the clauses in the deduction rule. Thus, the pattern matching problem is not avoided, and an efficient deduction algorithm probably will have to have an underlying network pattern similar to ours, although not so visible.

The advantage with predicate calculus representation is however that the theory of decidability is much fuller developed for that representation than for ours.

Executable programs in some special programming language is potentially a more powerful representation than ours. Heuristic rules guiding the order of the deduction search are easier to include into such a deduction rule. However, the power in an actual system is of course limited to the set of programs which the input translator can generate. Many of the programs will probably in reality not contain anything else than our chaining rules, variables and patterns, and such system will also require some more or less hidden underlying network to select rules and facts of interest during a certain deduction process.

25. Questions

On the outermost surface level, we have until now only implemented yes-no questions in our system. Other kinds of questions can however appear as sub-questions during the deduction process. A question is in many ways similar to a natural language if-statement. In both cases, a pattern of variables is created, and we want to identify this pattern with the data base.

A typical question like "Is John father of a blond girl" will thus be translated like in figure 40.

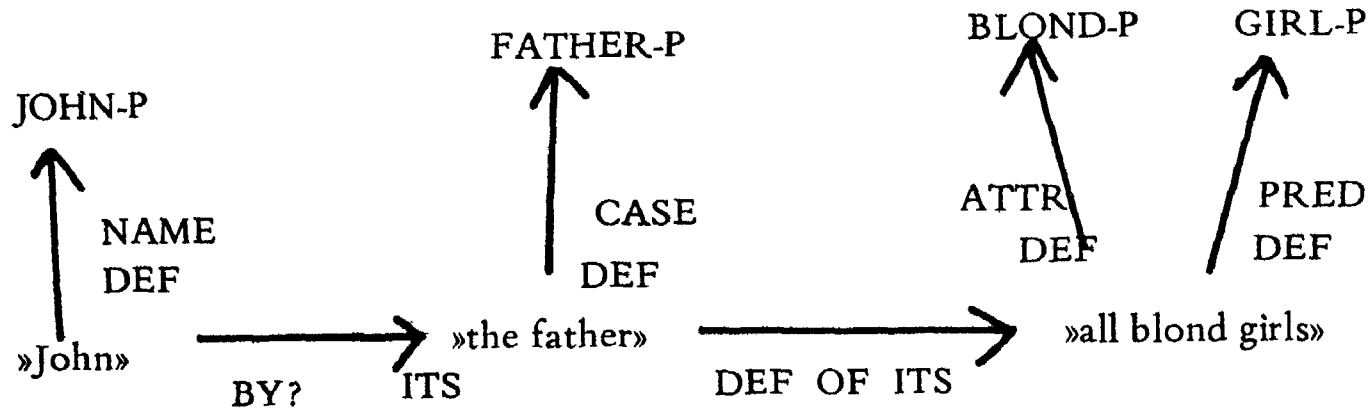


Figure 40 »Is John father of a blond girl?»

In this simple case, there was no need to introduce pattern keys of more than one variable, so the translation was very simple. One central relation, in this case the BY relation, is marked with a question-mark, which means that it is this relation which deduction should try to prove.

The processing of a question therefore usually begins with the introduction of temporary data (in this case the VARIABLE for "all blond girls" and the VARIABLE for "all fathers of blond girls") and then on a single question relation to prove.

This is what our system is capable of today. However, some complex questions will create patterns where part of the pattern refers to other patterns, just as for if-statements in the previous section of this paper.

Look for example at the question "Is the father of all the children of any of John's daughters married to that daughter?" In the translation of this question there will be a VARIABLE for "the father" and a VARIABLE for "that daughter". And these two VARIABLES must pairwise match. It is not enough to find that the father is married, not even enough to find that he is married to one of John's daughters. He must be married to just that daughter whose children are all also his children.

The translation will therefore have to be something like in figure 41.

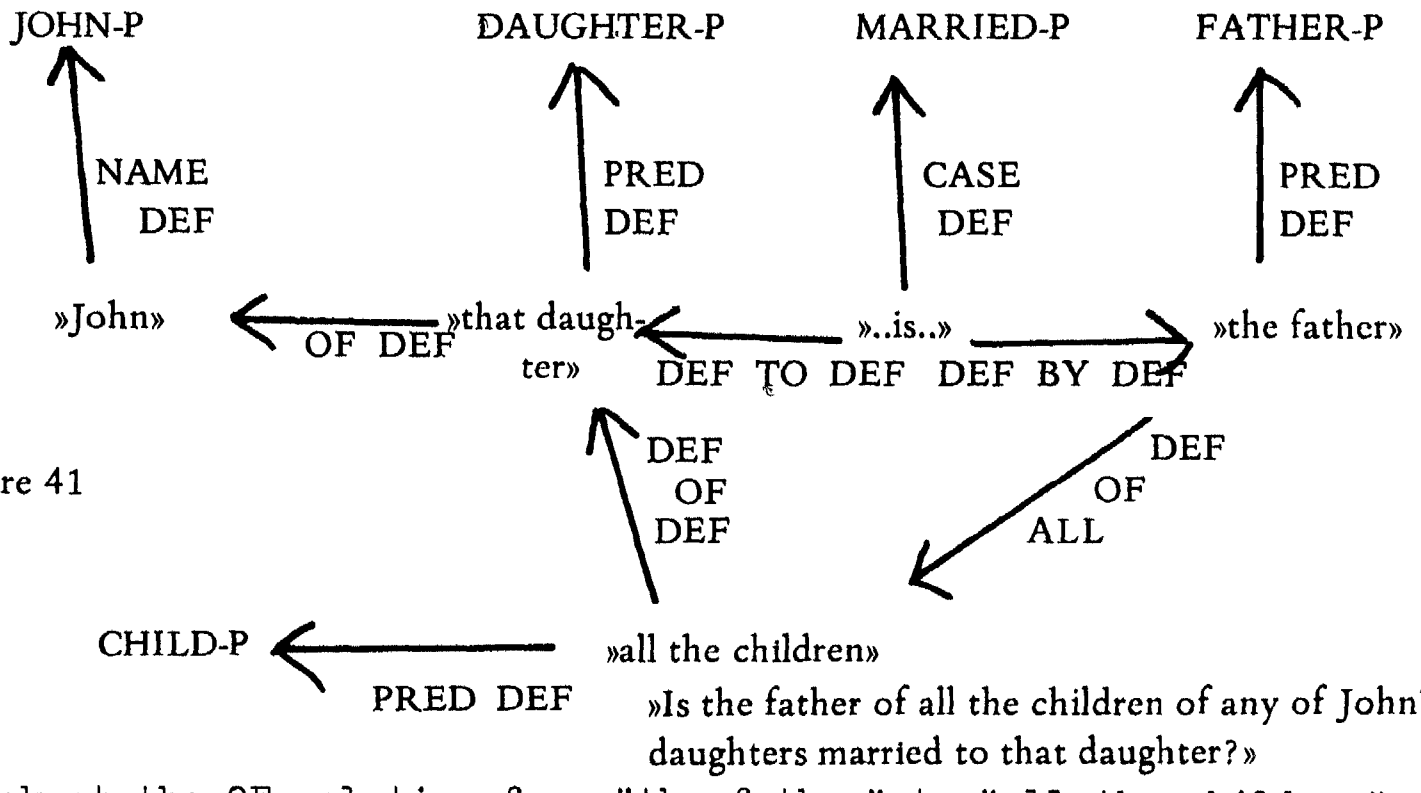


Figure 41

»Is the father of all the children of any of John's daughters married to that daughter?»

Look at the OF relation from "the father" to "all the children". This OF relation should single out just the children of his wife, not the children of all her sisters. We have not yet found out how to do this. We hope that this complex kind of questions will not be common.

26. DUMMIES = temporary variables för data base merging

A short presentation of the concept of a DUMMY was made in section 8. DUMMIES and problems with them will be more fully treated here.

When natural language uses constructs like "He" or "the man" or "this object in the sky" then this usually refers to something which the reciever is supposed to know already. Often, the thing referred to has been mentioned a short time ago in the previous natural language input.

We therefore introduce a special kind of VARIABLE. This is called a DUMMY. An ordinary VARIABLE is kept in the data base to be used at some later time for deduction. A DUMMY causes an immediate search in the data base for a matching previously known object.

The order of this search is important. If there are several previous objects matching the descriptions, the last-mentioned one shall usually be found. However, the subject usually goes before other noun phrases. If we say "If a card is below another card, then it cannot be seen." then "it" refers to the subject "a card", not to the prepositional "another card" even though this was mentioned later.

Our program will therefore make a list, the so-called CURRENT list, of previous-mentioned objects. This is searched backwards.
We have at present two search routines for DUMMY matching, the "the" routine and the "this" routine. One difference between them is that if no matching node is found, the "this" routine will ask the user to rephrase his statement. The "the" routine will in that case accept that this is something which the user knows, but not the computer. It will therefore enter a new node if no previous-mentioned is found.

One problem which we so far have not completely solved is how to do with patterns of DUMMIES. If we say "the man behind John" then there are two natural ways to translate this into our data base:

a) Two DUMMIES, one independent (for "John") and another dependent (for "the man") as in figure 42.

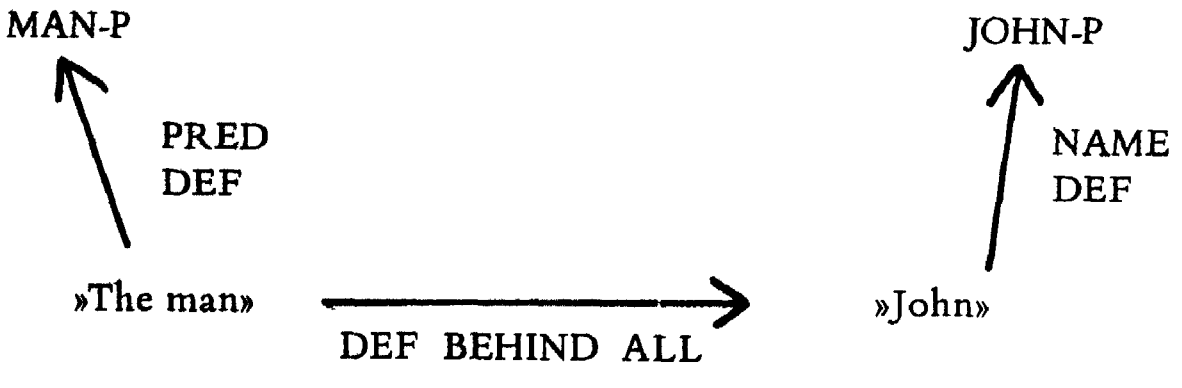


Figure 42 »The man behind John»

b) A pattern key of two mutually dependent DUMMIES, where "DEF BEHIND ALL" in the figure 42 is changed to "DEF BEHIND DEF".

The first translation is necessary in those cases where only one of the DUMMIES has a match in the data base, e.g. for a statement like "If a man is late, then the man behind him is even later." Here, there is no previously known man, and the second translation with the pattern key would not match "a man" in the if-statement at all.

However, if solution a) is adopted, this text will not be treated correctly "A man with a dog is coming. Another dog is barking. The man with the dog is frightened." Solution a) will first find the other dog, and then create a new man who is with that other dog, and let that other man be frightened. A more complex algorithm may be necessary to solve this problem.

Another example which will cause difficulty is "John and his brother are in the wood. His brother is leaving." If no DUMMY pattern key is created, then "his" in the second sentence will identify with "brother" in the previous sentence. "His brother" in the second sentence will then identify with "His brother's brother" in the first sentence, which is not correct.

27. DUMMIES which refer to VARIABLES

Look at the natural language sentence "If a lion meets an elephant, then the elephant will run to the forest."

There are two DUMMIES in the main clause, "the elephant" and "the forest". "The elephant" will match the VARIABLE created by "an elephant" in the if-clause. "The forest" will match a previously known, probably CONSTANT forest.

In general, only after doing the refer-back search in the data base will we know whether a DUMMY will match a VARIABLE or a CONSTANT.

If a DUMMY matches a VARIABLE, then that DUMMY may be adding definitions to that VARIABLE. Look for example at the sentence "If a lion meets an elephant, and if the lion sees the elephant, then..." Here, the DUMMIES in the second phrase will add to the pattern key being built up, and thus add to the definitions of the VARIABLES "a lion" and "an elephant".

This means that there are two kinds of DEF-marked relations on DUMMIES. The first of them are those which are to be used during the refer-back search. And the second are those which are to be added to the VARIABLE, if the DUMMY matched a variable. In our system, we intend to distinguish between these by first giving the relations which are to be used in the refer-back search. Then the refer-back search is done, and thereafter the relations are given which add DEF-s to the definition of the matched VARIABLE.

Another interesting case is where there are two DUMMIES, one dependent on the other, and one of them matches a VARIABLE. Look for example at the sentence "If a girl is in trouble, then her mother will be angry." Here, "her" becomes an independent DUMMY, while "her mother" becomes a dependent DUMMY. The "her" DUMMY will match the VARIABLE "a girl" in the if-clause. The DUMMY "her mother" will not find any match at all. And the interesting thing is that because the independent DUMMY matched a VARIABLE, the dependent DUMMY "her mother" which matches nothing should in this case not create a new CONSTANT but a new VARIABLE. For every different girl, there is a different mother who will be angry, so a CONSTANT will not do.

This means that the "the" dummy algorithm must be able to decide if a CONSTANT or a VARIABLE is to be created when a DUMMY finds no explicit match.

28. The problem of dual representation

We have of course during the writing of the SQAP system encountered many problems. For some of them we have found solutions, for some not. Many of the problems have already been presented in this paper, and those problems which belong more to input translation or to deduction than to data base structure do not fit into the subject of this paper.

Looking at the problems we have met, there seems to be one problem which recurs several times. This is the fact that the same natural language construct can be represented in several ways in our data base. We have found that this is unavoidable, since one representation is necessary in some cases and another in other cases. But on the other hand, this difference in representation will make the deduction difficult, including the deduction during the merging of new text into a previous data base.

One solution to this problem is that when there is two different representations, then for sentences giving one of them, both of them is created including the relationship between them.

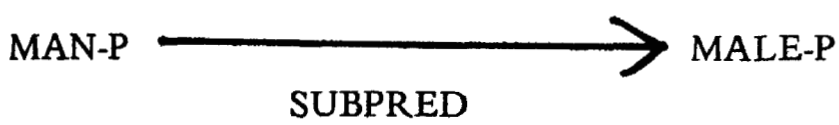
This solution is used for the duality of the representation of nouns. The noun "book" corresponds in our data base both to the predicate BOOK*P (=the property of being a book) and to the defined set BOOK*S (=the set of all books). But whenever BOOK*S is created in input translation, BOOK*P is also created and the relation BOOK*S DEF PRED BOOK*P is created. (If this already exists in the data base, then of course the same thing is not put there twice.)

This means that whenever both BOOK*P and BOOK*S occurs in our data base, the relation between them also exists.

Another example where the same solution is used is active and passive verbs. Whenever a passive predicate, e.g. KILLED*P is put into the data base, we also put in the active form KILL*P and the relation between them: KILL*P PASS KILLED*P. In this way we ensure that if both KILL*P and KILLED*P are in our data base, then the relation PASS between them is also there.

The same solution could be used, but would be cumbersome and memoryconsuming in other cases. For example, a number of objects can be regarded both as a composite object and as a set, for which we have two different representation. There is a short relation, OBJCOMPLEX, in our system, from a composite object to a set of all its parts. But this relation cannot solve the whole problem, and it would also be very cumbersome always to have to put out both representations for certain phrases. This is discussed further in section 19 of this paper.

Another problem of this kind is that our system is very much based on the idea that simple facts should be stored in a simple way and more complex facts in a more complex way. "A man is a male" is therefore in our data base stored like in figure 43.



»A man is a male»

Figure 43

In this case, a relation between the predicates was enough. But for the slightly more complex statement "Every human male is a man", a defined set is necessary as in figure 44.

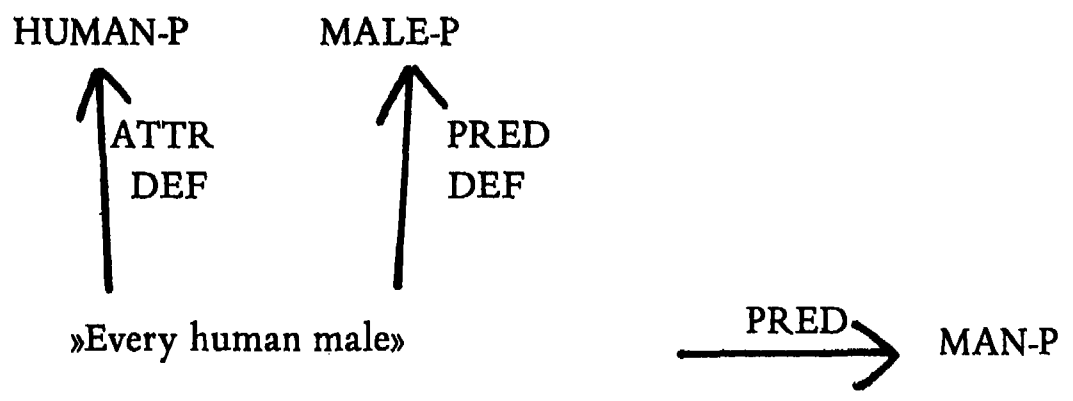


Figure 44

If there is some limitation in truthfulness or validity, e.g. a time-limit, then the PRED must be expanded to REV BY times CASE, e.g. for the phrase "Every human male was that year a soldier", in figure 45.

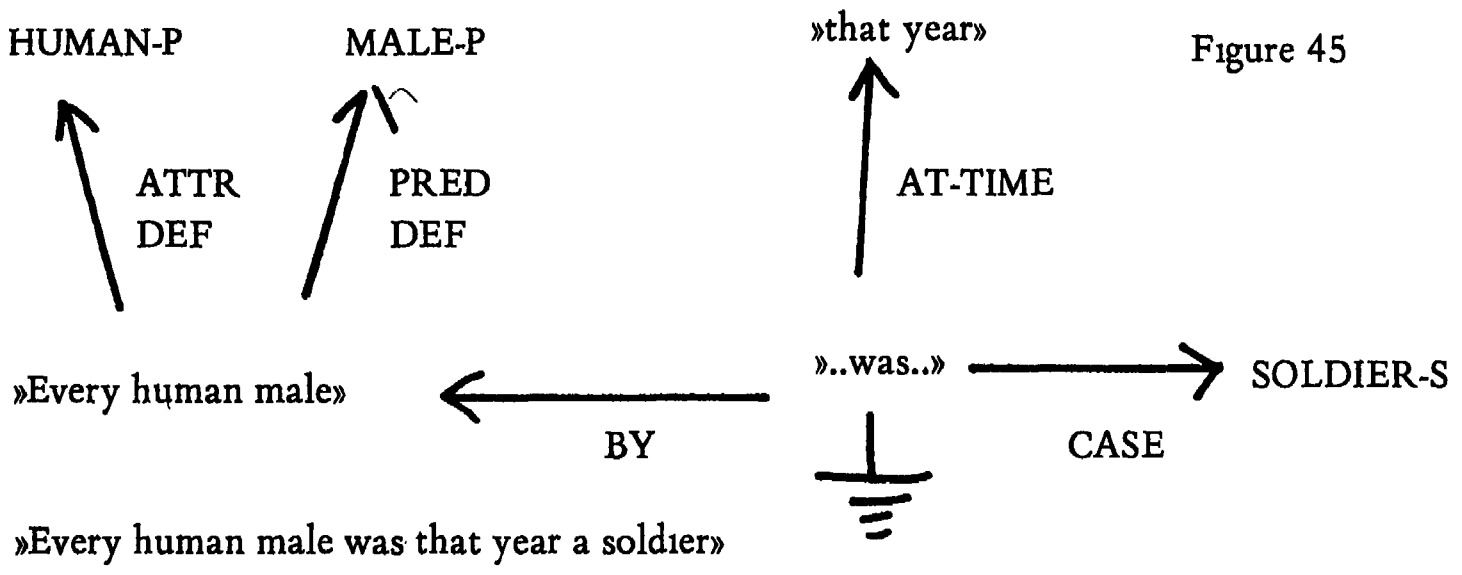


Figure 45

The difficulty with this is that when a new fact is going to be added to old facts, then the expanded version may be necessary. Also, a question may be asking for the expanded version, and the deduction routines may then have to do the expanding during deduction, which is surely possible; but difficult to manage in an efficient way.

Example: "Every male is an animal. If he is human, then he is also a man."

Here, "he" in the second sentence creates an object, the data base merging routine will find it difficult to understand that this refers to the "male" in the previous sentence, since this "male" was translated as a predicate, not as an object.

29. What our system can do and cannot do

Our system can at least partly manage the following natural language constructs: Nouns, articles, quantifiers, adjectives, numerals, most pronouns, the conjunction "and", passive and active verbs, objects, predicate complements, genitive, prepositional attributes and adverbials, if-clauses, yes-no questions.

Some of the things we are not ready with yet are other conjunctions than "and", relative pronouns, interrogative pronouns, negation, auxilliary verbs other than "be", comparative adjectives.

We do not yet try to resolve ambiguity by reference to the data base.

The kind of facts which our system can handle are basically a passive description of a true set of facts about the world. We can thus not yet handle properly a description of a sequence of events changing the world step by step. Neither can we handle properly facts which are part of someone's belief structure. Statements about statements cannot be handled (e.g. "This is a difficult problem" or "This should not be construed to mean that...").

30. A short comparison with other systems

Shapiro 1971, Simmons 1971 and others have presented systems very similar to our. Most other systems do not have quantifiers on the short relations as we have, and we feel that this is an addition which adds to the power of the representation. Special in our system may also be that one short relation can be extented when necessary into an event. This saves much memory compared to representations where the fullest form is always used, even though it is in most cases not needed. It is for example true that for a statement like that in figure 3, there may be doubt about only the BY relation, or only the AT-TIME relation, or only the CASE relation. (We may be sure that "One girl is happy", but not so sure about the day, or we may be sure that there is happiness today, but not sure where.) A full representation would therefore require a place to insert doubt on any short relation, whether there is doubt or not, and this would double the data base size.

In our system, the deduction rule can for any node in the data base find all outgoing and incoming short relations directly, and follow them. In-spite of this, we can store a whole short relation in just 64 bits(two 24-bit adresses plus 16 additional bits). This compact representation increases the efficiency of systems storing the data base in virtual memories.

31. Acknowledgements

The basic ideas for our system were initially conceived by Erik Sandewall and were presented in his papers in the bibliography.

Our system was developed as a team-work between me, Erik Sandewall and Kalle Mäkilä. I have been working with input translation, Kalle Mäkilä with data base managment and deduction, and Erik Sandewall has guided us in our work. It is difficult to pinpoint who solved each of our problems, since they were solved through discussions from which a solution sooner or later emerged.

Siv Sjögren has been working with the problem of adapting our system to the swedish and esperanto languages.

32. Bibliography

The most important papers for understanding our work are marked with an asterisk.

Bar-Hillel, Yehoshua, 1964: Language and Information, Addison-Wisley, Reading, 1964.

Fillmore, Ch. J.: The Case for Case. In Universals in Linguistic Theory, ed Back, E. et al, Holt Rinehård and Winston Inc., 1968.

* Mäkilä, Kalle 1972: Deduction procedures in a question answering system. FOA P rapport C 8310-M3(E5), January 1972.

Mäkilä, Kalle 1973: Experience of assimilation and deduction in a semantic net (to be published).

* Palme, Jacob 1970A: Making Computers Understand Natural Language. FOA P rapport C 8257-11(64), July 1970, also in Artificial Intelligence and Heuristic Programming (ed. Findler, Meltzer) Edinburgh University Press 1971.

* Palme, Jacob 1970B: A simplified English for Question Answering FOA P rapport C 8256-11(64), December 1970.

* Palme, Jacob 1971A: A Natural Language Parsing Program for Question Answering. FOA P rapport C 8268-11(64), February 1971.

Palme, Jacob 1971B: Internal Structure of the SQAP Natural Language Parser, FOA P rapport C 8286-11(64), April 1971.

* Palme, Jacob 1972A: Syntax and dictionary for a computer english. FOA P rapport C 8312-M3(E5), February 1972.

Palme, Jacob 1972B: From parsing tree to predicate calculus a preliminary survey. FOA P rapport C 8313-M3(E5), February 1972.

* Palme, Jacob 1973: The SQAP data base for natural language information. FOAP Rapport, September 1973.

Sandewall, Erik 1965: Representation of facts in a computer question answering systems. Uppsala University, Computer Science dept. 1965.

* Sandewall, Erik 1969: A set-oriented property structure representation for binary relations, SPB. In Machine Intelligence 5, Edinburgh University Press, 1970, also as Uppsala University Computer Sciences department Report nr 24.

* Sandewall, Erik and Mäkilä, Kalle 1970: A Data Base Structure for a Question-Answering System. FOA P rapport C 8265-11(64), Nove mber 1970.

* Sandewall, Erik 1971: Formal Methods in the Design of Question-Answering Systems. Artificial Intelligence vol. 2 (1971) pp 129-145.

Schank, R.C. and Testler, L.G., 1969: A Conceptual Dependency Parser for Natural Language. International Conference on Conputational Linguistics, 1969.

Shapiro, Stuart Charles 1971: The MIND system: A Data Structure for Semantic Information Processing. Rand, Santa Monica, Ca. 90406 USA, report R-837-PR.

Simmons, R.F., 1971: Natural Language for Instructional Communication. In Artificial Intelligence and Heuristic Programming, ed Findler, N.V. et al, Edinburgh University Press 1971.

Sjögren, Siv 1970: En syntax for datamaskinell analys av esperanto. FOA P rapport C 8264-11(64), Oktober 1970.

Sjögren, Siv 1971: Utkast till en syntax for data-maskinell analys av svenska. FOA P rapport C 8314-M3(E5), Februari 1971.

33. Index

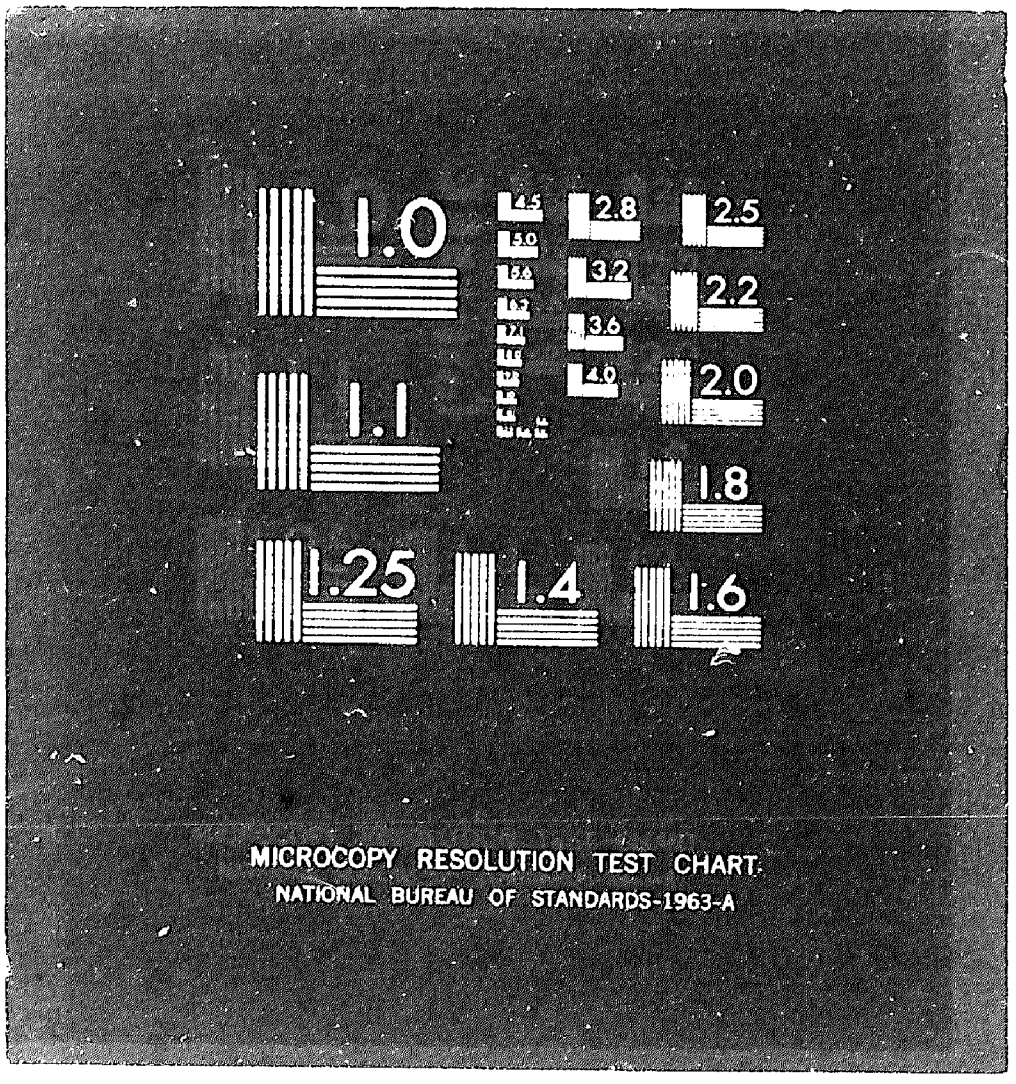
--- D ---

--- E ---

--- T ---

--- U ---

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A