[From: A.Zampolli, N.Calzolari (eds.) *Computational and mathematical linguistics*. Proceedings of the International Conference on Computational Linguistics, Pisa, 1973, vol.II (Firenze: Olschki, 1980)]

CHRISTIAN BOITET

TOWARDS AN ADAPTATIVE AND INTERACTIVE SYSTEM IN AUTOMATIC TRANSLATION

The models of the A.T. systems generally consist in two kinds:

— some divide the operation of translation in specialized, independent and successive phases;

— the others consider translation as a unique complex operation.

The models of the first kind present the advantage of easy conception and allow to implement successively the different informatic tools adapted to each stage. The C.E.T.A.'s stages (B. VAUQUOIS, 1967) for instance were the morphological analysis, the syntactical analysis, the transformation into an intermediate, or 'pivot language', the transfer and the synthesis. On the other hand, the models of this kind provide one algorithm for each stage, and the user can give the data (grammars, dictionaries) independently. The best example of this tendency is given by the TAUM project (1971), which uses a single informatic tool and a single algorithm, the Q-system. It is generally difficult to modify the algorithm in such systems.

However, the models of the second kind offer the advantage of flexibility; as a whole, the process of translation is described step by step, allowing exceptions to be treated case by case. This often forbids the separation of the linguistic tool from the linguistic data, hence one has to modify the program every time the data change. The models Fulcrum I et II due to P. L. GARVIN (1967) are good examples of this situation.

The aim of this paper is to propose a model which allies the conspicuousness and the relative ease of implementation of the models of the first kind to the flexibility of those of the second one, while adding some other properties.

As a matter of fact, our aim is to build a tool which might simulate a wide collection of different 'translation strategies', ranging from the word-by-word translation to the sentence by sentence, or paragraph by paragraph translation and including the 'quick translation', analogous to the operation performed by human interpreters.

In section 1 we'll present the basic notions, followed by a very anthropomorphic description of the proposed model and a sketch of the possible practical developments. Section 2. will give some precisions about networks in order to clarify section 3., (which will present in a more detailed manner the new tools of the models, especially the 'algogrammar').

1. FIRST APPROACH

1.1. *Basic notions.*

We are interested in the following problem: given a text written in a certain natural language, the 'source language', translate it automatically into another language, the 'target' language. We suppose the source text is given on a magnetic tape containing only the text and edition markers.

We don't allow any pre-analysis by hand, and don't care how we get this magnetic tape. One could note the increasing use of such tapes by editors, which somewhat decreased our interest in optical readers, for the sake of fiability. However, the task of translating a source text in which there are impression errors, as in a real situation, is very interesting and opens possibilities towards mechanical oral translation.

By "text" we design a string of characters, including the ordinary alphabet, the blank and special edition markers (type of alphabet, upper- and/or lower-case, paragraph ...)

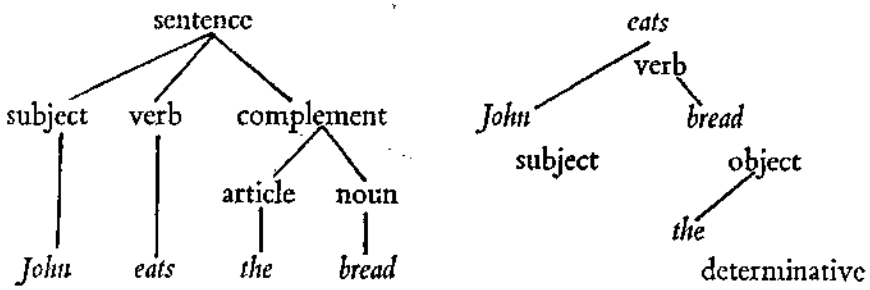
A "form" is a string of alphanumeric characters delimited by two blanks.

The "morphological analysis" of a form consists in finding in a dictionary the root(s) it is derived from, and the derivation type, characterised by the value of certain variables.

For example, the number variable *NBR* could have the singular and plural values, *SNG* and *PLU*, which we would schematically denote by *NBR*: = (*SNG*, *PLU*). If we consider that the lexical unit (*UL*) of a form is a value of the variable *UL*, the total result of the analysis is a "variable mask" containing the values of the different variables. A form can easily accept several morphological analyses (*can*, *use*, ...). Such a form is morphologically ambiguous.

The "syntactical analysis" of a group of forms consists in building one or several tree stuctures where the forms of the group appear at

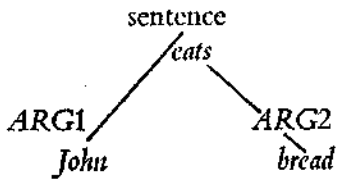
certain nodes, in order to represent the links between the different forms. In general, we can normally distinguish between the " constituent structures " where the forms appear on the leaves of the tree, and the " dependency structures ", where those forms appear on all the nodes. For example, the analysis of the group *John eats the bread* would give:



constituent structure dependency structure

One could remark that in the field of A.T., the search for such structures has more and more given place to the search of " deep " structures, since the sixties.

" To find the deep structure (' pivot ') of a group of forms " consists in building one or several tree structures representing the elementary enunciations appearing in the group and their links. The previous example would give only one such enunciation



(Not to be confused with Chomsky's deep structures).

Every node of all these tree structures is labelled by a variable mask. We only give here the values of one or two variables.

In order to transfer such a tree structure one must find, in a bilingual dictionary, the *UL* of the target language corresponding to the source *UL* appearing on the tree, replace the first ones by the second ones and if necessary execute some structural transformations (expressions).

The synthesis of the linguistic tree-structure is the converse of its analysis, that is the transformation of the tree in a string of forms, which is the 'surface' expression of the structure.

1.2. *Anthropomorphic presentation of the model.*

The proposed system can be thought of as a controller, or monitor, together with specialists (modules) of the different operations used in a translation process. We add an operation of automatic updating to the operations mentioned in 1.1., so as to avoid the necessity of modifying the linguistic data of the system after each try on a new corpus.

The monitor corresponds with the modules by sending them commands and receiving their responses. It can also communicate with an outside 'oracle', the user, in order to allow external interventions during the treatment.

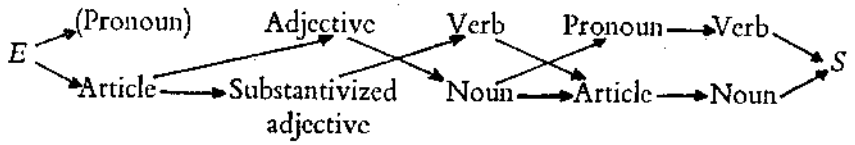
Let us present the different modules and the monitor.

1.2.1. *The morphological analyser.*

This analyser works on a string of forms. Its elementary action consists in analysing a form, denoted by *C*, and in linking the different solutions found for *C* to the solutions found for the preceding form, *P*1. It uses dictionaries and a grammar which can include (linking) conditions on the four preceding forms, *P*1, *P*2, *P*3 and *P*4, and on the following form, *S*. If a form admits no solution, one can modify it, add items to the dictionaries or continue.

Schematically, the output of the analyser can be represented as a graph where the nodes on the same vertical bear the different solutions produced for the form written below.

The sentence *Le grand lit le livre* would then yield, with appropriate grammar and dictionaries:



1.2.2. *The algogrammar.*

This tool allows the finding of a 'course' of a 'transition and ruption network' compatible with a chain-graph.

For the linguistic application, the chain-graph is the output of the morphological analyser. The 'transition and ruption network' is written by the linguist. It is analogous to W. A. WOODS's "augmented transition networks" (1970), in as much as its arcs bear conditions and actions. It differs from it by two features:

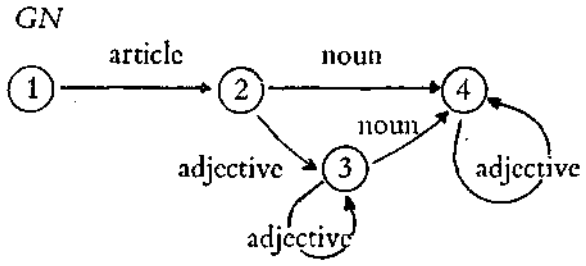
— it doesn't include its own searching and stack controlling algorithm: the monitor commands its running from the outside;

— it includes numerical registers and functions associated with the arcs enabling the linguist to control the search and to value the current analysis.

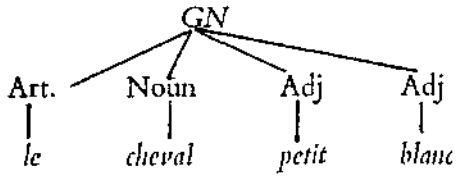
In contrast to W. A. WOODS's (1970) and T. WINOGRAD's (1971) systems, the algogrammar does not include its own algorithm: the search (that is, the way of progressing through the choice tree) is determined from the outside. Hence, the algogrammar can admit a whole family of such algorithms.

As a matter of fact, an elementary action of the algogrammar is a simple stack operation. The stack used is a linear representation of the partial analyses done so far.

Let us give a very simple representation of the noun phrase in French:



The algogrammar allows the building, step by step, of a relatively 'low' structure of the recognized groups: for instance, the group *le petit cheval blanc* could yield:

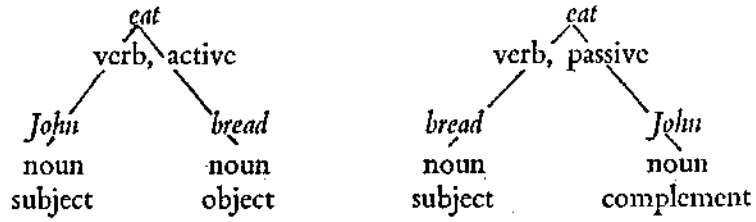


As in W. A. Woods (1970), we can already obtain a uniform representation for different surface structures at this level.

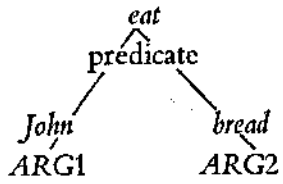
1.2.3. *The transformer.*

The transformer realizes changes in a tree by means of a transformational grammar. This tool allows the movement from the syntactical structure to the deep structure, or perhaps even directly from the surface string structure to the deep structure. A very classical example is the obtaining of the same deep structures from the (different) syntactical analyses of two equivalent sentences, the first in the active form and the second in the passive form.

Example:



The transformation consists in transforming subject and object into *ARG*1 and *ARG*2 if the verb is active, or into *ARG*2 and *ARG*1 if the verb is passive, so as to obtain:



The command of the transformer consists in giving it a transformational grammar, that is a set of transformation rules and the way to apply them, and a tree to be transformed.

It answers by giving the resulting tree.

1.2.4. *The translater.*

The translater is given by a tree representing the analysis of a group of forms at a certain (e.g. morphological, syntactical or deep) level. Using information about the n last groups given before, it lets the group wait or immediately transfers it.

Hence, the 'interpreter's tricks' can appear here: a human interpreter translates group by group, often before the whole sentence is finished, but in certain cases he modifies the order of the groups. Let us for example consider the German sentence: *den Feind hatte er schon gesehen, als er ankam.* The translater can translate it by *He had already seen the enemy when he arrived,* even if only the analysis of the groups but not of the whole sentence had been performed.

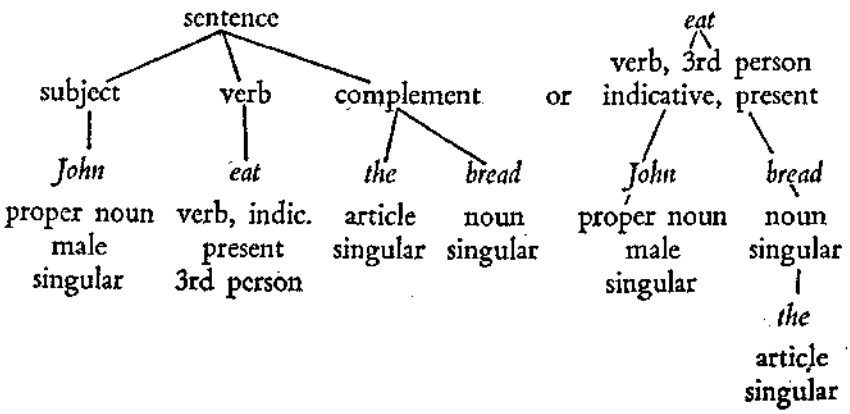
The transfer of a group consists in two stages. During the first one, possibly optional, one looks for the expressions of the group occuring in the bilingual dictionary, transfers the lexical units and executes the structure's transformations attached to the transition form the "source" expression to the 'target' expression. In the second stage, one transfers all the 'source' lexical units left and possibly modifies the value of certain variables according to the dictionary.

The translater answers that it will wait or outputs the result of the transfer, which is in input form for the generator.

1.2.5. *The generator.*

Once a group is analysed and transferred, it can again pass through structural transformations in order to prepare its output in the target language in a linear form. For example, one transforms the structure so that a certain string of nodes of the tree corresponds to the morphological analysis of the target group. This "certain string" can be, for instance, to 'word of the leaves', that is the string of the terminal nodes, or, using any enumeration of the nodes, the substring of nodes verifying a certain condition. This enumeration can be calculated using weights, as did the C.E.T.A.'s generator.

For instance, the generator could be given such structures as:



The aim of the generator is to transform the given tree into a string of variable masks, and this string into a string of words of the target language, using rules and dictionaries written by the linguist and analogous to those of the morphological analyser. The rules must also allow the use of the restricted context, in order to take care of elisions or contractions (*l* in French, *don't* in English, *im* in German).

1.2.6. *The learner.*

This module has different tasks of different complexities. First of all, it allows the user to introduce new data during the treatment, that is for example new indexings of words, or new equivalents.

It can also increment certain counters referring to the use of certain parts of the other modules, so as to be able to speed up parts of the treatment. If, for instance, an analyser uses rules in a certain order, it can reorder them according to their frequencies.

A third task of the learner is to modify parameters related to the control of a module. Hence, the learner can modify weights associated to the transitions of the algogrammar, so as to adapt it to the corpus being translated, that is to give preference to the frequent constructions, which will be tried first.

Another task could be the construction of hypothetical transitions of the algogrammar with the use of the past failures, trying to generalize the 'break-points'. However, it seems now very difficult to realize, and, on the other hand, the philosophy of the system is more to begin with a linguistic basis as sure and as complete as possible and adapt it according to the results than to try to build grammars with the use of a collection of correct sentences.

1.2.7. *The conserver.*

This module is intended to keep certain results of the treatment of the *n* groups preceding the processed group, so as to allow the search for antecedents and transmissions information concerning, for example, the 'target words' used in the translation, so as to refine the translation's style!

Hence, the conserver may be used in the analysis as well as in the synthesis.

Another use of the conserver can also be to keep the two or three 'better' (from the system's point of view) translations of a group in case the user refuses the first proposed.

1.2.8. *The transmitter.*

This module transmits to the user the system's messages and is intended to receive the user's messages, to verify their correctness and in certain cases to remember them permanently.

If, for example, the user wants to index a new word, it must verify the correctness of the new dictionary entry, keep it and send it to the morphological analyser, which will compile it into a more compact form and add it to its data.

The transmitter must also provide the user, if desired, with a trace of the actions of a specified module.

1.2.9. *The monitor.*

We arrive at our last module, which determines the strategy of the translation, that is the way of chaining the actions of the previous modules according to the user's commands.

The monitor directs the system with the use of elementary actions such as the call of a module, and of tests on the module's answers. If one wants to be able to simulate different strategies of translation, they must correspond to certain commands of (or parameters given by) the user, so that the monitor must also be able to test these commands to determine its next move.

The linguist who writes the monitor begins hence with declarations of the names and the values of the commands: with exactly the same format as in the analysers and in the generator. He then writes the monitor in an ALGOL-like, but simple langage using only:

— elementary actions
— tests on the commands
— an interruption test
— branching facilities.

The interruption is a special signal indicating that the user wants to interfer with the treatment, for example by changing his commands.

In such a system, which is also in a certain sense a model of trans-

lation, the linguist could try different strategies and different degrees of learning with the use of the same linguistic data.

These strategies could be, for example, the word by word translation, the 'quick translation' (one begins to translate before having even reached the end of the sentence), the 'stage-translation' (the CETA's strategy) or 'heuristic translation' (one tries to find the first of all the correct best translations or whatever the linguist could imagine to do with these tools).

1.3. *Possibilities of implementation.*

This model tries to make use of the informatic tools developed by the GETA, in certain cases with minor modifications. On the other hand, certain modules, which will be briefly described in section 3., must be implemented.

The first group includes the morphological analyser, the transformer, the translater and the generator.

a) The morphological analyser currently in use in the GETA had to be slightly modified in order to allow the insertion of new entries to the dictionary during the analysis, the modification of an input form and the output as a 'chain-graph'. As a matter of fact, its output is now the set of all possible "morphological strings" of each sentence of the translation unit (about 500 words). These modifications are now in progress.

b) The transformer now being implemented can be used as it is.

c) The GETA's translater is not yet written. It is likely that one would have to add to it the possibility to change the order of the groups.

d) The generator can be that of the GETA.

The second group includes the algogrammar, the learner, the transmitter and the monitor.

a) The algogrammar, although its principle is directly inspired by W. A. WOODS's *Transition networks* (1970), differs sensibly from them and would require a new informatic tool allowing different algorithms to explore the tree generated by a 'transition and ruption network' and a chain-graph (cf. sections 2 and 3).

b) The learner would be more simple to implement, but for its last task. It essentially consists in tables of counters incremented

once a group is successfully parsed or translated, in a mechanism for periodically updating the linguistic data and/or the control parameters according to these tables, and in a set of files to keep trace of the supplementary information supplied by the user during the treatment.

c) The transmitter could be a very simple or very sophisticated program of man-machine communication.

d) The monitor requires the description and the compilation of a very simple language.

e) Of course one would have to write some auxiliary programs not mentioned here because the linguist doesn't use them directly. Essentially, they would be the compilers, the loaders and the programs to measure the efficiency of the system.

The modular conception of the system could allow the obtaining of partial results even before modules like the learner or the conserver are written.

2. NETWORKS

2.1. *First definitions.*

2.1.1. *Network element.*

A Network Element (N.E.) is a mapping ρ from $\mathbb{N}_+$ into $\mathbb{N}_+^2$ In the following text, we shall only consider finite N.E. (that is to say $Dom\rho$ is finite), with one exception in part 2.5.4.

By π_1 and π_2 we shall denote the two canonical projections from $\mathbb{N}_+^2$ into $\mathbb{N}_+$.

Then, $\sigma = \pi_1 \cdot \rho$ is the "source-mapping" associated with ρ, and
$\tau = \pi_2 \cdot \rho$ is the "target-mapping" associated with ρ.

$Dom\rho = A(\rho)$ is the set of the "arcs" of ρ.

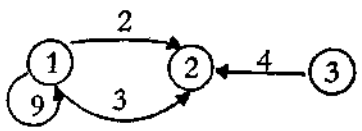
$\pi_1(Im\rho) \cup \pi_2(Im\rho) = S(\rho)$ is the set of the "nodes" of ρ, where $Im\rho$ stands for range ρ.

If $n \in A(\rho)$, we say that the arc n links $\sigma(n)$ to $\tau(n)$.

It is possible to represent a finite N.E. with a set of 3-uples of integers of the form $(n, \rho(n))$, or with a planar graph.

Example:

$$\rho \begin{cases} (2,\ 1,\ 2) \\ (3,\ 1,\ 2) \\ (4,\ 3,\ 2) \\ (9,\ 1,\ 1) \end{cases} \quad \text{or}$$



2.1.2. *Some particular nodes.*

The set of the predecessors of a node s, denoted by $PR(s)$, consists of the nodes of ρ linked to s. In the same way, the successors of s are the nodes to which s is linked.

Hence,

$$PR(s) = \{t \mid (\exists n)\ [\sigma(n) = t\ \&\ \tau(n) = s]\}$$
$$SU(s) = \{t \mid (\exists n)\ [\tau(n) = t\ \&\ \sigma(n) = s]\}$$

The next two properties are related to ρ as a whole. The origins of ρ, $D(\rho)$, are the nodes with no other predecessor than themselves, and the ends of ρ, $F(\rho)$, are the nodes with no other successor than themselves.

Thus,

$$D(\rho) = \{s \mid PR(s) \subset \{s\}\}$$
$$F(\rho) = \{s \mid SU(s) \subset \{s\}\}$$

The entries of ρ, $I(\rho)$, are the nodes without predecessors, and the exits of ρ, $O(\rho)$, are the nodes without successors.

In the above example, 1 is an origin, 2 an exit and 3 an entry.

2.1.3. *Ways and chains.*

A "way" (path) of a N.E. ρ is an application γ from an initial segment of $\mathbb{N}$ into $A(\rho)$, that is a finite or infinite sequence of arcs of ρ, such that:

$$(\forall i \in Dom\gamma - \{0\})\ [\tau \cdot \gamma(i\text{-}1) = \sigma \cdot \gamma(i)]$$

A "chain" of a N.E. ρ is a way without loops, that is an one-one way of ρ.

In the above example, (9, 9, 2) is a way but is not a chain.
The length of a way γ is of course set equal to $|\ Dom\gamma\ |$.

2.2. *Network.*

2.2.1. *Isomorphisms of N.E.*

One can generally define a mapping f from F^E into $F'^{E'}$ using two mappings $f_1 : E \to E'$ and $f_2 : F \to F'$ (but not all f are definable in that manner).

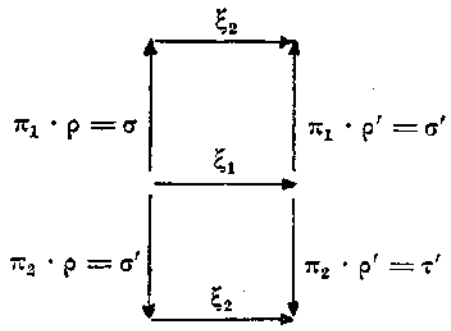
Then, $f(\alpha)\ (f_1(e)) = f_2(\alpha(e))$.

Hence, we will say that two N.E. ρ and ρ' are "isomorphic" if there is a pair (ξ_1, ξ_2) of 1-1 mappings from $\mathbb{N}$ into $\mathbb{N}$ such that:

1) $Dom\rho' = \xi_1(Dom\rho)$
2) $\xi_2 \cdot \sigma = \sigma' \cdot \xi_1$ & $\xi_2 \cdot \tau = \tau' \cdot \xi_1$

If furthermore ξ_1 and ξ_2 are recursive, ρ and ρ' are "recursively isomorphic".

The property (2) only asserts that the image of the source is the source of the image, and the target of the image is the image of the target, or that the following diagram is commutative.



The (recursive) isomorphy is clearly an equivalence relation on N.E.

A "network" is a class of recursive isomorphy of N.E.

An "ordered network" is a non decreasing class of recursive isomorphy of N.E. (we add the condition that ξ_1 preserves the order between the arcs).

2.2.2. *Invariance.*

All the properties introduced for N.E. or nodes of N.E. are clearly preserved by isomorphisms of N.E. In particular, the isomorphic image of a way (resp. a chain) is also a way (resp. a chain), and the predecessors and successors of a node as well as the origins and ends of an N.E. are preserved by isomorphy. This follows from the fact that ξ_1 and ξ_2 are one-one.

2.3. *Particular networks.*

The following properties (of N.E.) are compatible with the isomorphy of N.E., hence we will say a network verifies the property P if it contains a N.E. verifying P.

If ρ is one-one, ρ is an "unredundant" N.E.

If τ is one-one, ρ is divergent, or "arborescent".

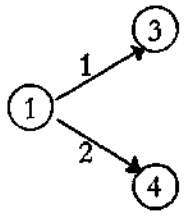
If σ is one-one, ρ is convergent (or antiarborescent).

If ρ and τ are one-one, ρ is a "linear" N.E.

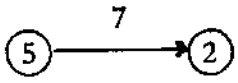
If every way of ρ is a chain, ρ is loopfree. To verify this property, it is enough to test the ways of length less than $|S(\rho)|$.

An arborescent network can be represented using only its nodes, as a 'forest' labelled by its nodes.

Example:



will be represented by: 1(3, 4), 5(2)



Let be the transitive closure of the successor relation. In an arborescent network, a "section" is a sequence $\mathscr{S}$ of nodes such that

1) it contains no pair of nodes related by D

2) every leave (exit) of is an (extended) successor of a node in $\mathscr{S}$:

$$O(\rho) \subset D(\mathscr{S})$$

A loop-free network with exactly one entry and one exit is a chain-graph. In a chain-graph there is only one origin, the entry, and one end, the exit. Furthermore, every node belongs to a chain linking the entry to the exit. Such a chain will be called complete.

2.4. *Labelled networks.*

A N.E. labelled on a (finite) alphabet E is a 3-uple $(\rho, \varepsilon_1, \varepsilon_2) \in \mathbb{N}_+^{2^{\mathbb{N}}+} \times (E^\star \ \mathbb{N}_+)^3$ where $| Dom\rho | < \infty$, $Dom\varepsilon_1 = A(\rho)$ and $Dom\varepsilon_2 = S(\rho)$

An isomorphism of labelled N.E. ρ and ρ' is an isomorphism of N.E. preserving the labels, that is to say:

$$\varepsilon'_1 \cdot \xi_1 = \varepsilon_1 \quad \& \quad \varepsilon'_2 \cdot \xi_2 = \varepsilon_2$$

The labelled (ordered or unordered) networks are thus defined as in 2.2. They will be denoted by $E(\rho)$.

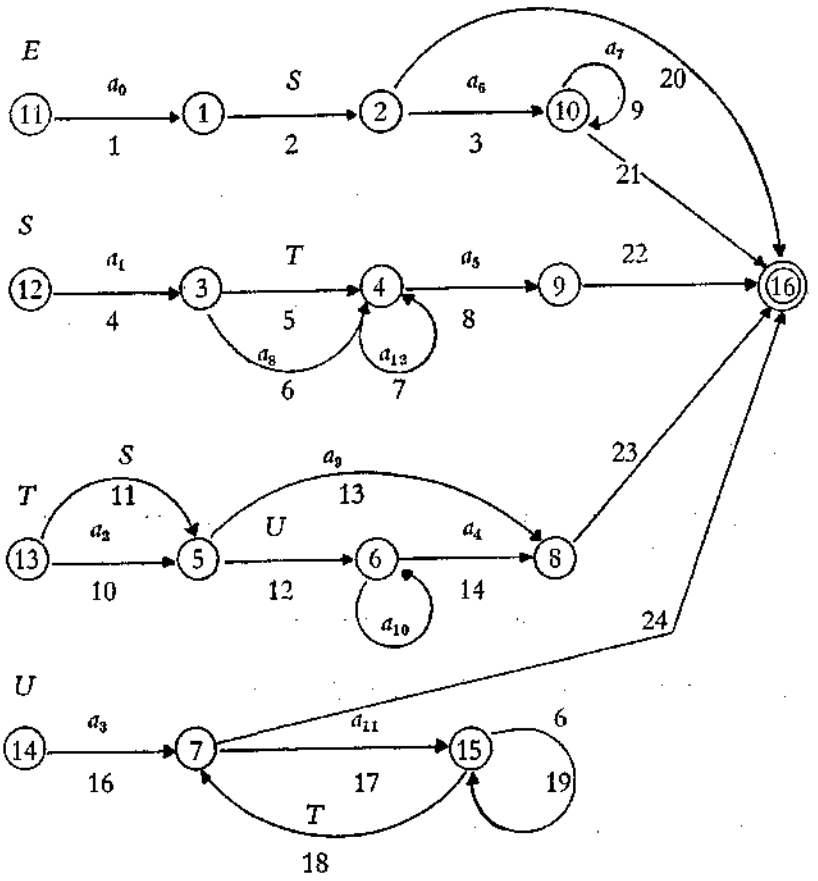
2.5. *Transition networks.*

2.5.1. *Definitions.*

The predecessors of the ends of a network will be called the " doors " of the network, $P(\rho) \cdot \wedge$ will denote the empty word in $E^\star$. A " transition network " is a labelled network with one exit and possibly many exits, such that the nodes with nonempty labels are exactly the origins and that the labels of the arcs pointing to the exit are not labels of node:

1) $| O(\rho) | = 1$
2) $(\forall n)\ [\varepsilon_2(n) \neq \wedge \Rightarrow n \in D(\rho)]$
3) $\varepsilon_1 \cdot \tau^{-1}\ (O(\rho)) \cap \varepsilon_2(S(\rho)) = 0$

Example:



2.5.2. *Course of a transition network.*

A course of a transition network is an arborescent network labelled by $A(\rho)$ such that, if K is a binary relation on E,

1) every way $\gamma = a_\varnothing, a_1, \ldots, a_n$ of ρ is a course of ρ with "origin" $D(\gamma) = \sigma(a_\varnothing)$ and "end" $F(\gamma) = \tau(a_n)$

2) If δ is a course of ρ, if $\varepsilon_1(q)\ K\ \varepsilon_2(D(\delta))$ and if $F(\delta)$ is a door of ρ, then $q(\delta)$ is a course of ρ with origin $D(q(\delta)) = \sigma(q)$ and end $F(q(\delta)) = \tau(q)$

3) If δ and δ' are courses of ρ and if $F(\delta) = D(\delta')$, then δ, δ' is a course of ρ with origin $D(\delta, \delta') = D(\delta)$ and end $F((\delta, \delta') = F(\delta')$.

Example: Taking the above example, if K is the equality,

$$\delta = 1,\ 2\ (4,\ 5\ (10,\ 12\ (16),\ 14),\ 8),\ 3 \quad \text{is a course of } \rho$$

If we represent the same course using as labels not the names but the labels of the arcs, we obtain a clearer, but possibly ambiguous representation:

$$a_0,\ S(a_1,\ T(a_1,\ U(a_2),\ a_4),\ a_5),\ a_6$$

$\mathscr{C}(\rho,\ E)$ will denote the set of all the courses of ρ.

2.5.3. *Course compatible with a chain-graph.*

Let ρ be a transition network with labels in $E^\star$.

χ be a chain-graph with labels in $F^\star$, and $R \subset E^\star \times F^\star$ a relation between labels.

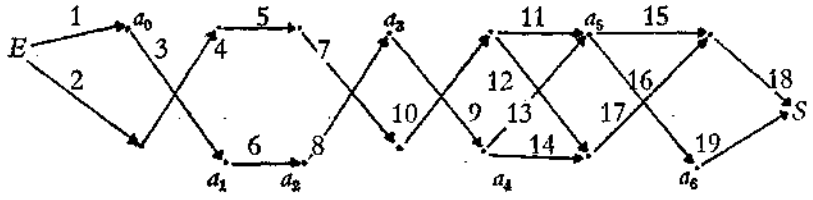
Let δ be a course of ρ and $\delta_0, \delta_1, \ldots, \delta_n = T(\rho)$ be its terminal word (the sequence of its leaves).

δ will be said " compatible with " χ if there is a way γ of χ linking $I(\chi)$ to $O(\chi)$ denoted by $\gamma_0, \gamma_1, \ldots, \gamma_{n+1}$, such that the labels of the arcs of $T(\rho)$ and the labels of the nodes of γ are in R, that is to say:

$$(\forall i \in [O,\ n])\ [\varepsilon_1(\delta_i)\ R\ \varepsilon_2 \cdot \tau(\gamma_i)$$

The set of the courses of ρ compatible with χ with reference to χ will be denoted by $\mathscr{C}(\rho,\ \chi,\ E,\ F,\ R)$.

Example: $E = F$, R is the equality. Let us take the same course as in 2.5.2. and the following chain-graph, where only the pertinent nodes are labelled.



The way γ we looked for is $\gamma = 1,\ 3,\ 6,\ 8,\ 9,\ 13,\ 16,\ 19$.
The sequence of the labels of the nodes is $a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6$, that is exactly the terminal word in the second representation of δ in 2.5.2.

2.5.4. *Tree of choices generated by a transition network and a chain-graph.*

$\mathscr{T}(\rho, E)$ and $\mathscr{T}(\rho, \chi, E, F, R)$ are generally infinite. As a matter of fact, it is enough to have a configuration such as

in ρ in order to increase δ. Hence, we will define an acceptable course of ρ as an element of $\mathscr{T}(\rho, E)$ which contains no 'left chain' of length greater than $|D(\rho)|$ linking arcs p such that $\varepsilon_1(p) \in \varepsilon_2(D(\rho))$.

A 'left-chain' of an arborescence is a chain of nodes without elder (left) brothers. We will denote by $\alpha(\rho, E)$ the acceptable courses of ρ.

Hence, using the above example, one would accept

$$T(a_\varnothing, T(a_\varnothing, b_\varnothing)), \text{ but not } T(T(b_\varnothing)).$$

$\alpha(\rho, \chi, E, F, R)$ will denote the set of the acceptable courses of $\rho(E)$ compatible with $\chi(F)$ with reference to R.

Although $\alpha(\rho, E)$ can be infinite, it should be clear that $\alpha(\rho, \chi, E, F, R)$ is always finite, and that there is a simple algorithm for enumerating it, namely to enumerate the complete chains of χ (linking $I(\chi)$ to $O(\chi)$) and, at every node, to construct the beginnings (in the sense of the linear representation of courses) of courses of $\rho(E)$ so far acceptable and compatible with $\chi(F)$.

At $O(\chi)$ one gets $\alpha(\rho, \chi, E, F, R)$.

Example: A 'beginning' of δ (as in 2.5.2. is 1, 2 (4, 5 (10)). It corresponds to the beginning 1, 3, 6 of a complete chain of χ.

The tree of choices of a transition network $\rho(E)$ is the (possibly infinite) arborescent network labelled with the beginnings of $\alpha(\rho, E)$ so that the label of each node is an immediate – that is to say, maximal and proper-prefix of the labels of its successors.

The elements of $\alpha(\rho, E)$ are the labels of the leaves of this tree, which will be denoted by $\mathscr{C}(\rho, E)$.

The tree of choices generated by a transition network $\rho(E)$ and a chain-graph $\chi(F)$, denoted by $\mathscr{C}(\rho, \chi, E, F, R)$, is the subtree of $\mathscr{C}(\rho, E)$ such that the labels of its nodes are exactly the prefixes of the elements of $\alpha(\rho, \chi\ E, F, R)$. Hence the elements of $\alpha(\rho, \chi, E, F, R)$ appear on the leaves of $\mathscr{C}(\rho, \chi, E, F, R)$.

Furthermore, $\mathscr{C}(\rho, \chi, E, F, R)$ is finite.

$\mathscr{C}(\rho, \chi, E, F, R)$ is a priori partially ordered (through the successor relation). It is equivalent to find a total ordering compatible with this partial ordering and to give an enumeration of the beginnings of the courses compatible with the prefix relation. A 'strategy' on an algogrammar will be the choice of such a possibly partial enumeration.

2.5.5. *Transition and Ruption networks.*

A Transition and Ruption Network, or TRN, is a transition network $\rho(E)$ together with two sets of labels G and H such that:

1) $G \subset E \;\&\; \varepsilon_1^{-1}(G) = 0$ (only on nodes)

2) $H \subset E \;\&\; \varepsilon_1^{-1}(H) = 0$ (only on arcs)

3) every H-labelled arc is on a way linking a G-labelled origin to the exit.

The courses of a TRN are defined as in 2.5.2. using rules (1), (2) (3) and the additional rule:

4) Let K' be a fixed binary relation on E, and let ρ and ρ' be two courses admitting the sections:

$$\mathscr{S} = x_\varnothing, x_1, \ldots, x_p, c_\varnothing, \ldots, c_n, \gamma_\varnothing, \gamma_1, \ldots, \gamma_q$$
$$\mathscr{S}' = a_\varnothing, a_1, \ldots, a_n, h, b_\varnothing, b_1, \ldots b_m$$

such that:

I) $\varepsilon_1(a_i)\ K'\ \varepsilon_1(c_i) \qquad 0 \leqslant i \leqslant n$

II) $\varepsilon_2 \cdot \sigma(a_\varnothing) \in G \quad \&\quad \varepsilon_1(h) \in H$

Then we obtain a new course δ'' by inserting in δ the sequence of trees of $\mathscr{S}'$ of roots $h, b_\varnothing, \ldots b_m$ to the right of c_n. That is to say: if

$$\delta = \alpha c(\beta, u, c_n, v, \gamma)\ \delta, \text{ then}$$
$$\delta'' = \alpha c(\beta, u, c_n, h(\ldots), b_\varnothing(\ldots), \ldots, b_m(\ldots), v, \gamma)\ \delta$$

where α, β, γ, δ stand for strings of symbols appearing in the linear writing of δ.

Intuitively, under certain conditions, we can 'jump' to h and return later in the normal course.

$\mathscr{C}(\rho, E)$ will denote the set of courses of the TRN, $\rho(E, G, H)$. As in 2.5.2., we define $\sigma(\rho, E)$, $\mathscr{C}(\rho, E, G, H)$ and $\mathscr{C}(\rho, \chi, E, G, H, F)$ which is again finite.

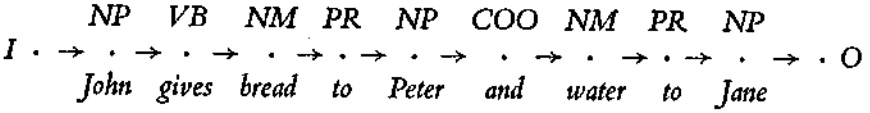
Example: Let us anticipate section 3. and give a small example on an English sentence:

John gives bread to Peter and water to Jane.

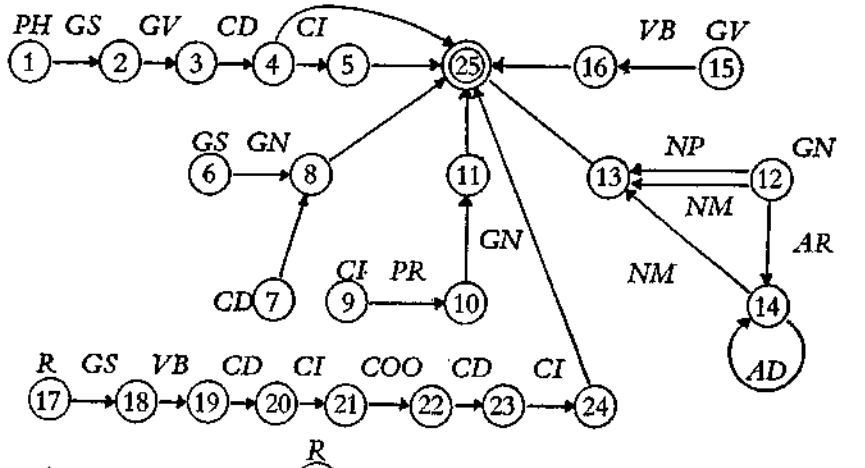
Let us take:

$G = \{R\}, \quad H = \{COO\}$
$E = \{PH, GS, GV, CD, CI, GN, PR, NP, NM, AD, AR, VB\}$

Suppose we have obtained the following chain-graph representation:



Let ρ be:



The part with entry (17) corresponds to the 'ruption'. Applying (4), one verifies that

$\delta'' = PH(GS(GN(NP)), GV(VB), CD(GN(NM)), CI(PR, GN(NP)), COO, CD(GN(NM)), CI(PR, GN(NP)))$

is a compatible and acceptable course, with:

$\delta = PH(GS(GN(NP)), GV(VB), CD(GN(NM)), CI(PR, GN(NP)))$
$\delta' = GS(GN), VB, CD(GN(NM)), CI(PR, GN(NP)), COO, CD(GN(NM)), CI(PR, GN(NP))$

with: $a_\emptyset$, a_1, a_2, a_3, h, $b_\emptyset$, b_1, $b_2 = GS, VB, CD, CI, COO, CD, CI$
$x_\emptyset$, x_1, ..., $x_p = y_\emptyset$, ..., $y_q = 0$
$c_\emptyset$, c_1, c_2, $c_3 = GS, VB, CD, CI$
K' the equality relation.

3. THE NEW MODULES

3.1. *The algogrammar.*

The morphological analyser was presented elsewhere (ATEF, 1973). We admit here it is possible to modify it in such a way that its output is a chain-graph, constructible step by step reading the input text from left to right. Hence we immediatly go ahead with the formal description of the algogrammar.

3.1.1. *Definitions.*

An algogrammar is a Transition and Ruption Network (TRN) where the labels (in E) are of the form (c, v, t, a, M, C, A) and $\varepsilon_2(O(\rho)) \subset \pi_5(E)$. An arc will be called a "transition".

c, v, t, a are four recursive primitive functions on registers, M is the (proper) "label", generally a variable mask, C is a condition and A an affectation.

The four functions are the control on the algorithms using the algogrammar.

The remaining is the linguistic part of the transition.

A strategy on an algogrammar is an algorithm for enumerating the tree of choices generated by the algogrammar and any M-labelled chain-graph. Such a strategy is essentially independent of the algogrammar.

3.1.2. *Registers and labels.*

The algogrammar is to be written by a linguist. He begins by declaring registers R_0, R_1, R_2, R_3, N_1, ..., N_q, L_1, ..., L_r.

Registers R_i contain numerical values associated to the current

course. Hence, it is possible to value partial analyses (beginnings of courses), and the chosen strategy may use the weight of an analysis in order to decide to go ahead or to drop it (permanently or not).

Registers R_0, R_1, R_2, R_3 are mandatory. However, registers N_j and L_k are free. registers N_j contain numerical values (which may be, for example, used to limit the depth of the analysis), and registers L_k contain names (addresses) of groups.

At every step of the construction of an 'analysis' of a chain-graph (compatible course), R_0 contains the address of the node causing the last transition,

R_1 contains a weight C to be given to the transition to be tried

R_2 contains the weight of the current partial analysis

R_3 contains the weight of the last transition.

c: $\{c_\varnothing\} \times R_1 \times \pi_j N_j \rightarrow R_1$ is used to calculate the "initial weight" to give to the transition to be tried. $c_\varnothing$ is the "proper weight" of the transition.

v: $\pi_j N_j \rightarrow R_3$ calculates the weight of the course consisting of this transition.

t: $R_1 \times R_2 \times R_3 \rightarrow R_2$ calculates the weight of the new partial analysis obtained by adding this transition to the preceding partial analysis.

a: $R_1 \times R_2 \times R_3 \times \pi_j N_j \rightarrow \pi_j N_j$ is used to modify the numerical registers.

M: is homogeneous with a part of the labels of the chain-graph, for instance with the 'syntactical category' part of the variable-mask.

C is a condition on the values the variables of the group which 'causes' the transition.

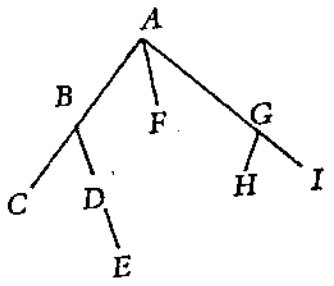
A is an affectation, that is a function from $R_0 \times \pi_k L_k$ into $\pi_k L_k$ allowing the storing of parts of a structure and the building of new structures. A is analogous to W. A. Woods's *BUILDQ* function (1970).

3.1.3. *Linear representation of the search in a tree of choices.*

We will call "search in a tree" any (possibly partial) enumeration of the nodes of the tree compatible with the successor relation. It is possible to write linearly any search in $\sigma(\rho, E)$ as a string of 3-uples (α, n, m), where

α is the current node (corresponding to an arc of the algogrammar)

n is the index of the 3-uple
m is the index of its unique predecessor
Example:



$\binom{A}{1}\binom{B}{2}\binom{F}{3}\binom{D}{4}\binom{G}{5}\binom{H}{6}\binom{E}{7}\binom{C}{8}$ is such a search going,
$\binom{0}{}\binom{1}{}\binom{1}{}\binom{2}{}\binom{1}{}\binom{5}{}\binom{4}{}\binom{2}{}$

through the paths: (A, B, C), (A, F), (A, B, D, E), (A, G, H).
Remark that the enumeration is partial.

3.1.4. *The Mechanism.*

A strategy on an algogrammar uses a stack which is the linear representation of the partial analyses already tried. The stack can only increase during the analysis of a chain-graph, and is emptied before analysing a new one. For instance, the input text can be represented by a string of chain graphs corresponding to sentences or paragraphs.

The elements of the stack consist of:

— the node Q of the chain graph where the course is arrived;
— the arc α of the algogrammar used to attain it;
— the node S of the algogrammar from which the course will go on;
— a flag indicating if Q is the last of its brothers;
— a flag indicating if α is the last arc issued from $\sigma(\alpha)$;
— a flag indicating if the analysis is complete or not;
— the values of the registers;
— a flag indicating whether the strategy drops the analysis at this point permanently or temporarily or if all the possibilities have been tried.

The elementary actions of a strategy are of 3 kinds: to empty the stack, to write a new element on its top or to go backward to an element and possibly to modify its flags. If one adds an element, it can be the immediate successor of the topmost element (one continues the current analysis) or of one of the elements in the stack (one drops the current analysis and continues with another one at the point where it was previously dropped).

The *commands* of the algogrammar are:
— to apply the first possible transition;
— to go to the first possible node of the chain-graph;
— to drop temporarily an analysis;
— to return to the last analysis temporarily dropped;
— to return to the last 'branching point' (where another analysis was possible);
— to give the mask of the group recognized, if any;
— to go on from the current point;
— to 'put in the data' the recognized group;
— to stop.

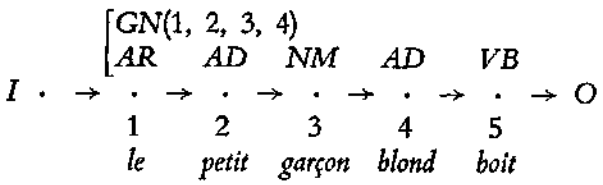
The *answers* of the algogrammar are:
— the last node, the last transition and the weight of the (partial) analysis;
— the mask of the recognized group, if any;
— the fact that one arrives to a 'pseudo-exit' of the chain-graph (recall it is constructed step by step), that is to say the morphological analysis has not yet reached the end of the sentence (of the paragraph).

3.1.5. *Storing in the data.*

Essentially, the algogrammar examines one possibility at a time: there is no parallel search. Hence, it is useful to have a way for storing the (partial) results of an analysis in order not to repeat the same calculation in several analyses. This is modeled by labelling the chain graph on $(F^*)^*$ rather than on F^*, that is to say the nodes can bear lists of variable-masks.

On the other hand, a group can 'cover' several nodes on a way in the chain-graph. Hence, the labels of the chain-graph are rather elements of $(F^* \times S(\chi)^*)^*$, that is to say we describe the way covered by the group (with label in F^*).

Example:



After having recognized a (sufficiently complex) group, one stores it in the data, that is in the chain-graph.

There could be a problem if two group structures with the same name can cover the same group.

The strategy must then look for another structure with the same name. This search is possible because the ' history ' of the past analyses for the group are in the stack. Furthermore, we can associate a flag with a certain label in the label-list of a node in order to avoid looking for another analysis satisfying the same conditions – if for example this has already been tried.

3.1.6. *Names for groups.*

Every time a group is found and used (that is, it is ' heavy ' enough), the algogrammar gives it a number. The different noun phrases of a sentence would hence be called *GN*1, *GN*2, ...

This makes it possible to have the same name for the different structures of different levels associated with a group, and to use translations already obtained in a (dropped) preceding analysis.

3.2. *The learner.*

3.2.1. *Preliminary discussion.*

E. M. Gold (1967) has shown that it is impossible to find the grammar of a regular language using only an enumeration of this language. However, if one knows a language to be context-sensitive, one can find a grammar of this language, ' in the limit ', using an enumeration of all the possible strings together with the value of the characteristic function on each string (' informant presentation ').

As he points out, learnability is a property of classes of languages. To the best of my knowledge, nobody has yet given the ' class of natural languages ' – perhaps it is even impossible in terms of formal grammars, and one has to formalize very precisely the notions of semantics and communication to approach it.

For this reason as well as for praticability we do not consider a system learning all from nothing, but one where linguistic data (dictionary, grammars, weights ...) representing a great bulk of knowledge in a condensed form can be locally modified.

3.2.2. *Learning in dictionaries.*

3.2.2.1. *Updating.*

At running time the user must have the possibility of adding new items to the dictionaries (on to auxiliary dictionaries). One of the tasks of the learner is to merge these additions with the preceding dictionaries.

3.2.2.2. *Optimization.*

Certain dictionaries can have a very simple structure, as for instance the morphologic dictionaries. However, bilingual dictionaries have a fairly complex structure, they can be viewed namely as trees labelled on ' strata ' (see C. BOITET, 1973) called " astrabonds " when the labels include weights.

The search for an equivalent may be oriented by these weights and not exhaustive: one could be satisfied with the first correct (w.r.t. given conditions) equivalent without going to the end of an item.

The learner may then increment counters associated to given nodes in the items when they are used in the treatment and finally change the weights and reorder of the dictionary after a long enough pass on a corpus.

Furthermore, if the dictionaries are very large, it can be useful to have dictionaries specialized to certain domains and a ' basic ' dictionary. If the system works on a mathematical text, one would load the mathematical and the basic dictionaries. Only in case a word is not found

would the system load the appropriate page of the large original dictionary, and in case of success the learner can add the new item to the specialized dictionary.

Let us finally remark that operations on dictionaries such as merging, specialization, ..., can be realized by the transformer (see C. BOITET, 1973).

3.2.3. *Learning in the algogrammar.*

Labels of transitions in the algogrammar contain a weight or a "weight function" C including a constant, the "proper weight" $C_\emptyset$.

The learner contains a set of counters associated with the transitions. It can then increment them if the associated transitions have been used in a successful analysis and finally calculate a new weight for a transition using $C_\emptyset$ and the value of the counters associated to the transitions issued of the same source.

Hence, this modification of the weight will modify the order in which the analyses are tried so that correct analyses are generally tried first, which is very interesting for strategies looking for a 'first correct solution'.

3.3. *Conserver.*

It is only a storing mechanism using a given space cyclically.

3.4. *Transmitter.*

It is a program working as an interface between the monitor and the user. It can be fairly simple if the messages are predetermined or use analysers and possibly deductive algorithms in a sophisticated version, as in recent man-machine systems (T. WINOGRAD, 1971).

3.5. *Monitor.*

It is a finite state automaton written by a linguist. Its language contains:

— names and values of the commands (declared by the linguist);
— labels;
— logical operators $\wedge$, $\vee$, $\neg$, $=$, $\neq$;
— conditionals and go to statements;
— elementary actions: tests or commands or operations on the variables;
— numerical variables (cut-points).

Example of a command declaration:

MODE : = (*AUTO, CONV*)
QUALITY : = (*HI, LO, MI*)
OPTION : = (*LEARNING, NOLEARN*)
STRATEGY: = (*WORD BY WORD, STRING, QUICK, INTERACT*)
CONTEXT : = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Example of instruction:

If MODE = AUTO then go to FIRST else TRANSMITTER (MSG)

To write a monitor is to write a whole set of strategies of translation corresponding to different values of the commands.

CONCLUSION

To implement such a model would make it possible to use the flexibility of programming at the highest level without the difficulties of first generation systems, and to allow specialists in translation to try different ' models of translation '.

Let us finally remark that we don't know if the splitting of the process (corresponding to the different modules) described here corresponds in any way to the human way of translating. We can only rely on linguistic studies and intuition as we do not know of any physiological results in this field. But it is our hope that such a model, once implemented, could give new insight into the very process of translating.

REFERENCES

C. BOITET, *Astrabondes et Dictionnaires*, Document G.E.T.A., Grenoble, Février 1973.

P. L. GARVIN, *The Fulcrum Syntactic Analyser for Russian*, in 2° conference internationale sur le Traitement Automatique des Langues, rapport n. 5, Grenoble, 23-25 Août 1967.

E. M. GOLD, *Language Identification in the Limit*, I & C, X (1967), pp. 447-474.

B. VAUQUOIS, *Le Système de Traduction Automatique du* C.E.T.A., Document CETA, 1967.

T. WINOGRAD, *Procedures as a Representation for Data in a Computer Program for understanding natural languages*, Cambridge (Mass.), 1971.

W. A. WOODS, *Transition Network grammars for natural language analysis*, in « Communications of the ACM », XIII (1970) 10.

ATEF, *Système d'Analyse de Textes en États Finis*, Document G.E.T.A., Grenoble, Février 1973.

TAUM 71, *Groupe de Recherches pour la Traduction Automatique*, Université de Montréal, CNR, Janvier 1971.