A User Friendly A T N Programming Environment (APE)

Hans Haugeneder, Manfred Gehrke
Siemens AG, ZT ZTI INF
W. Germany

APE is a workbench to develop ATN grammars based on an active chart parser. It represents the networks graphically and supports the grammar writer by window- and menu-based debugging techniques.

1. ATNs - Attractive, but....

Augmented Transition Network Grammars are one of the frameworks for developing natural language parsers that has been used sucessfully in a large number of natural language systems for many languages since its introduction in the early seventies by Woods [WOO 70].

Three aspects of ATNs, namely applicability in various types of natural language systems, suitability for different languages and the availability of efficient processing methods make ATNs an adequate framework for practically oriented development of natural language parsers.

Since the time of its introduction the core of the ATN formalism has proved to be astonishingly stable and the exposition and specification of ATNs given in [BAT 78] turned out to become a quasi-standard.

One of the five claims stated there , namely perspicuity, deserves some comments, because to us it doesn't seem justified strongly. On the contrary we feel that concerning perspicuity and descriptiveness ATN grammars do have some shortcomings. These come into play if you use the ATN formalism to develop a grammar for a nontrivial subset of a natural language.

The main reason for this insufficient perspicuity clearly lies in the possiblities ATNs offer with respect to local and global register setting and testing. These facilities, though practically very useful, give an ATN grammar a somewhat procedural character, the grammar writer has to cope with. In this sense ATNs can be seen as a programming language for writing nondeterministic programs (grammars). Thus for the development of any larger grammars (programs) some sort of programming environment for ATNs not only is necessary but also compensates the lack of perspicuity and it makes the development of ATN grammars a practicable task.

2. Design Considerations for an ATN Environment

Examining various ATN environments as [KEH 80], [GNE 82] and [CHR 83] for example we developed our ATN programming environment (APE) along following design principles.

1) The various tools the environment offers must be integrated allowing simultaneous grammar editing and testing.
2) The grammar editor has to represent the network structures graphically allowing the user to access the grammar via the contextfree skeleton of the various networks.
3) The design of the system should make use of techniques like multi-windowing, menue- and mouse-based interaction facilities, in order to make the system usable in an easy manner.

With this desiderata concerning the design of such a system, certain requirements concerning the hardware and software for such an implementation are necessary. We have chosen Interlisp-D (Trademark of XEROX) as basis of APE, which due to its comprehensive display- and interaction facilities proved to be an adequate starting point for the realisa-tion of our ideas.

3. Active Chart Parsing as a Framework for an ATN Environment

Active Chart parsing ([KAP 73]) is a highly general framework to implement parsers. The two main ideas of this approach are to represent the parser's control structure explicitly allowing high flexibility in scheduling the various paths to be followed and to prevent the parser from doing the same thing twice using a comprehensive bookkeeping mechanism. The interaction of these components is shown schematically in figure 1.

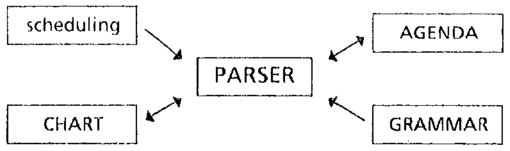


Figure 1

The possibilities of a flexible scheduling is achieved by means of an agenda, which at any state of the parser contains all the tasks that are induced by the grammar and not processed so far. The ordering of the agenda thereby determines the way, the search space is traversed. With this agenda-based scheduling facility the parser can apply various control structures like depth-first, breadth-first or heuristic scheduling, even changing it during one parse.

Such facilities are of interest for "tuning" the parser's behaviour in an intended way. Agenda-based task scheduling also offers the operational facilities for pruning parts of the search space which amounts to switching off certain parts of a grammar during a parse.

The second central concept in active chart parsing, the chart, is a graph structure, which does not only do the bookkeeping of the parsed constituents (the inactive edges). It also records each of the partial intermediate steps (the active edges), thus logically representing all the paths in work and all constituents parsed so far offering the possibility to inspect the uptothen parsing process.

But more important. e.g. for perspicuity, the chart (i.e. its graphical representation) also can be seen as a descriptive representation of the parser's state from a naive grammar writers' point of view. It is a conceptually simple representation, whose atomic constructs, the graph's nodes, the active and inactive edges, have clear counterparts to the conceptual entities a grammar writer has a naive understanding of, namely the positions in the sentence to parse (i.e. the nodes), the partial parses spawning between two nodes (i.e. the active edges) and already analysed constituents (i.e. the inactive edges spawning the sequence of words between two nodes). Thus a graphical representation of the chart growing as the parser proceeds makes the parsing process easily perspicuable for the user.

4. Description of the Environment

4.1. The Grammar-Editor

The user interface to the ATN grammar is built on top of an active graph-like representation of the single networks, which is initiated by the user in a menu-based manner. This bird's eye view gives the user an overall first impression of the global structure of the whole grammar with the type of the arc (PUSH, POP, CAT, JUMP) and the specification of "categorial" information with CAT- and PUSH-arcs.

Thus the user is not beaten with an unnatural, artificially linearized (for example lispish) way to represent the basic graph-like concepts of ATNs. The benefits of such network-based grammar specification facilities have been pointed out by Grimes [GRI 75].

The networks, displayed in the way described above, additionally offers the user a number of operational facilities, such as getting more specific information on a certain arc as for example its actions or additional tests. The user can activate the displayed network's arc and nodes respectively by clicking the mouse.

Activating an arc hereby pops a menu with the following possibilities:

- info: Gives a detailed printout of the arc, including its status (broken vs. unbroken).
- delete: Deletes the arc from the network, causing a new graphical layout of the network.
- edit: Edits the complete arc in a mouse- and menu-oriented editor with all necessary facilities to modify various parts of the arc, such as tests, actions and forms as well as its weight. Leaving the editor several checks are performed, putting the user back into the edit mode, if the modified arc structure is incorrect (e.g. if it contains too many items or items of an incorrect type at the wrong place).
- break: Puts a break on the arc taking the user into the break mode with interactive facilities (as described below) after the broken arc's actions are performed.
- unbreak: Removes a break from the arc.

Activating a node in the network offers the following facilities:

- info: Gives a detailed printout of all the arcs starting at that node.
- insert: Allows the user to insert an arc starting at the node activated, the arc's ending node (except POP-arcs) being determined via the mouse. To introduce additional new nodes the user is prompted by the system for subsequent arcs until he specifies a POP-arc or an already existing node as ending node of the last prompted arc.
- merge: A new node N1 is inserted after node N with the leaving arcs of N now beginning at N1 and a new arc between N and N1.

4.2. Grammar-Debugger

The user can specify in advance certain constructions he wants to be parsed, thus having the possiblitiy to test certain NP-constructions for example without the overhead of parsing a whole sentence.

These debugging facilities can be involved in three ways: primarily while the parser is working in a stepper-mode by means of a user interaction, secondly during the parser's run by means of a break put on an grammar arc and thirdly system-initiated at the end of the parse giving the user the possibility to restart.

In the stepper-mode the user can cause a break while watching the chart growing as the parser processes one task after another in the following way. During the single steps of creating of the chart graphically the system is interruptable to give the user the opportunity to put APE's stepper into the break-mode by mouse-clicking the relevant menu's item).

In the break-mode the user is offered a number of operational facilities which can be accessed activating the chart nodes and edges with the mouse. When selecting an edge the user can get more detailled information as for example its weight, its register environment and its history, consisting of the path through the grammar each arc being augmented with additional information as its current inputword, its register environment and the number of the task being responsible for processing that arc, which directly reflects the way the scheduling is performed. But more importantly the grammar tester can also modify the edges in various dimensions, including the following options:

- registers: The user can change registers by employing the same language he is used to as a grammar writer, i.e. in terms of actions defined in the ATN formalism as for example SETRs, ADDRs or form to be evaluated such as BUILDQs .
- weight: Allows to change of the weight of an edge, affecting the order of further processing.
- ending edge: With this option an edge can be modified with respect to the part of the input being spanned by it.

This last option together with the possibilities of register modifying renders for example the simple simulation of the parser's behaviour under the assumption of a (effectively missing or due to not matching tests blocked) grammar arc by enlarging the span of an edge.

Another more powerful possibility in testing a grammar is the introduction of additional (in)active edges, connecting two arbitrary nodes, which can be achieved via an activation of the starting arc. This allows the specification of partial parses or parsed constituents, which - though missing due to some defect in the grammar - the user wants the to make use of in further parsing process.

Parallel to all the options presented so far the user can edit the grammar on the fly, thus being able to modify the grammar just when he recognises certain bugs.

Additionally APE gives the user the possibilitiy to manipulate the agenda offering him various actions to be performed on the single tasks like freezing and killing a task, or changing its weight. This facility provides an advanced grammar writer with very effective means to focus the parser on things that are interesting for him in a certain situation, abondoning with irrelevent paths or postponing them.

Finally, when the user has done all the things that seemed useful to him at this break point he can continue the parsing process leaving the stepper options as they are or changing them appropriately.

At the end of the parsing process the user again gets in a break mode giving him the opportunuity of inserting new edges with the facility to restart the parsing process with this new information. Thus adding a new inactive edge and restarting for example amounts to asking the parser "what would yours results have been with an additional constituent c_i from word w_j to word w_k?". With the facilities described above the user also can easily analyse a configuration when the parser did not succeed in parsing a certain construction.

This description, though sketchy, should give an impression of the facilitities of APE and the ideas behind it. An illustration of APE's environment is shown in the appendix.

5. Outlook

The described ATN programming environment gives substantial support to the user in building up a working grammar, but some of APE's aspects aren't completely satisfying. So a lot of polishing the user interface as well as improving the functionality is still to be done.

Primarily we are currently working on an user-friendly lexicon handling. Another augmentation will be the easier global specification of very fine-grained breaks.

We'd like to thank U. Hochgesand, C. Maienborn and L. Simon for implementing parts of the environment and the colleagues of our lab for many fruitful discussions.

7. Literature

[BAT 78]
Bates, M., "The Theory and Practice of Augmented Transition Network Grammars", in: Bolc, L. (ed), "Natural Language Communication with Computers", Berlin 1978

[CHR 83]
Christaller, T., "An ATN Prgramming Environment", in: Bolc, L. (ed), "The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks", Berlin 1983

[GNE 82]
Gnefkow, W., "Studien zu einer Programmierumgebung für Augmented Transition Networks", Memo ANS-3, Universität Hamburg, 1982

[GRI 75]
Grimes, J. (ed) "Network Grammars", Summer Institute of Linguistics, University of Oklahoma, 1975

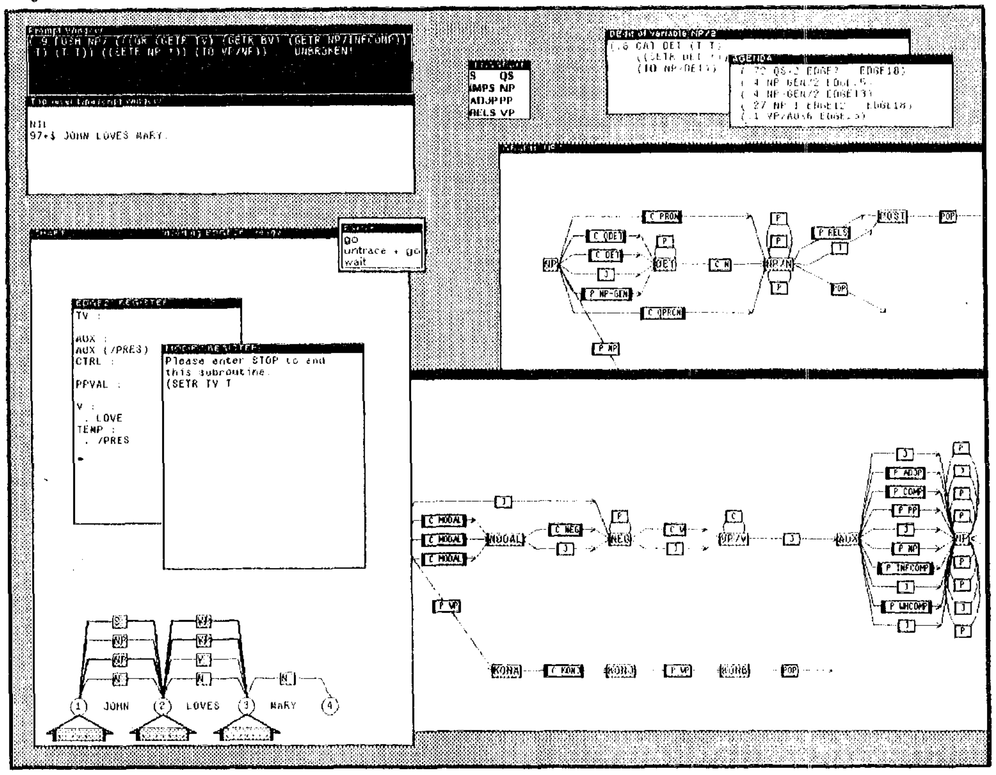
[KAP 73]
Kaplan, R. M., "A General Syntactic Processor", in: Rustin, R. (ed), "Natural Language Processing", New York 1973

[KEH 80]
Kehler, T.P., Woods, R.C., "ATN Grammar Modelling in Applied Linguistics", Proc. 19th ACL Meeting, 1980

[WOO 70]
Woods, W.A., "Transition Network Grammars for Natural Language Analysis", Comm.of the ACM 13(10), 1970

Appendix

Snapshot of the system in the breakmode.



```
( 9 PUSH NP/ (((OR (GETR TV) (GETR BV) (GETR NP/INFCOMP))
 T) (T T)) ((SETR NP *)) (TO VP/NP))   UNBROKEN!

NIL
97+$ JOHN LOVES MARY.

( 6 CAT DET (T T)
    ((SETR DET *))
    (TO NP-DET))

( 72 QS-2 EDGE7    EDGE18)
( 4 NP-GEN/2 EDGE15)
( 4 NP-GEN/2 EDGE13)
( 27 NP 1 EDGE12    EDGE18)
( 1 VP/AUX6 EDGE25)

TV :

AUX :
AUX (/PRES)
CTRL :

PPVAL :

V :
 . LOVE
TEMP :
 . /PRES

Please enter STOP to end
this subroutine.
(SETR TV T
```

A LANGUAGE FOR TRANSCRIPTIONS

paper submitted for
COLING 86
Bonn

by

Yves LEPAGE

GETA, BP 68
Université Scientifique et Médicale de Grenoble
38402 Saint-Martin-d'Hères, FRANCE

(final draft, April 1986)

ABSTRACT

To deal with specific alphabets is a necessity in natural language processing. In Grenoble, this problem is solved with help of transcriptions. Here we present a language (LT) designed to the rapid writing of passage from one transcription to another (transducers) and give some examples of its use.

KEY-WORDS

Transcriptions, transducers, multi-alphabet text processing, logical and physical processing of texts.

INTRODUCTION

In the general framework of natural language processing, the possibilities of interfaces provided by the current devices are rather poor, when considering, for example, the number of alphabets to be used. The problem of uppercase/lowercase letters, that of non-latin alphabets, not mentioning ideograms, is usually solved by the use of transcriptions in computer science circle dealing with natural languages <BOITET83>.

Our idea is to provide a rather simple device allowing rapid writing of programs performing the passage from one transcription to another (transducers, <KAIN72>), with help of a language (LT or Language for Transcriptions) based on an abstract automaton. The definition and the implementation of this language were initiated during an engineering school project <MENGA84>. The work on this Specialised Language for Linguistic Programming (SLLP) has led to a first version <LT85> in the context of a GETA/USMG project. It has then been extended in the frame of EUROTRA contract ETS-5 <ETS5>.

This paper presents:

- the semantics of LT in automata theory;
- the syntax of LT briefly described;
- indications on the implementation;
- some applications.

I. SEMANTICS OF LT IN AUTOMATA THEORY

1. "BASIC" TRANSDUCER

Transduction may be regarded as a simultaneous operation of reading and writing, writing being a function of reading <AHO,UL72>, <CHAUCHE74>. A transducer is a machine with an input tape and an output tape.

```
              -------------------         -------------------
input tape       ... a <str x>               ... a <str x>
READING HEAD  -------↑----------         ---------↑---------
                  ___!___                      ___!___
state             !  q  !         -->          !  q' !
                  ---!---                      ---!---
writing head  _______↓__________         _________↓_________
output tape    <str z>                    <str z> b
              ------------------         -------------------
```

Given a state and a character read, the transducer goes into another state and determines which character to write onto the output tape (transition).

The most simple transducer is _deterministic_ and _regular_ :

- it has only one input tape and only one output tape;
- there is only one way for reading and writing (rightwards);
- it reads only one symbol at a time; it writes only one symbol for one symbol read;
- there are no other objects such as stacks or balloons.

2. THE ABSTRACT AUTOMATON OF LT

The "basic" automaton is extended in LT into three directions

1. availability of the right context by means of two reading heads. The transition is function of the head ("forward" or "current") used in the reading of the input tape. A special transition performs the return of the "forward" head onto the position of the "current" one. This permits to simulates the reading of the empty string and places the abstract automaton of LT in the class of the "sequential transducers" as defined in <KAIN72>;
2. use of the notions of attributes in the states. A state is an etiquette with attributes. The values of some attributes are tested before a transition (condition) and the values of some attributes are changed after (actions). This theoretically increases the non-determinism of the automaton;
3. work on strings and not only on characters, which sets definitively the automaton in the class of "sequential transducers".

```
 ----------------------           -----------------------
  ... <str x> <str y>              ... <str x'> <str y'>
------↑-------↑--------          -----↑---------↑---------
      !_______!                       !_________!
       ___!___                         ___!____
       !  q  !            -->          !  q'  !
       !Ai=ai!                         !Ai=ai'!
       ---|---                         ----!---
__________↓__________            __________↓______________
    <str z>                          <str z'>
---------------------            -------------------------
```

The power of the LT automaton is restrained to a transducer with the following characteristics:

- one input tape and one output tape;
- determinism;
- states defined by etiquettes and attributes;
- two reading heads.

3. CLASS OF LANGUAGES ANALYSED BY LT

The abstract LT transducer may be under-used as a deterministic finite-state machine. So the class of languages which can be analysed by LT comprises the class of regular languages.

On the contrary to what we wrote in <ETS5>, LT can be used to define an acceptor of the famous context-dependent language anbncn. It is the semi-regularity which permits to simulate stacks. This means that the class of languages analysed by the abstract LT transducer comprises some of the context-dependent languages.

Using the Chomsky hierarchy we say that LT can analyse

- all the languages of class L3;
- some of the languages of class L2; to know if all languages in L2 can be analysed by LT is an open problem;
- some of the languages of class L1;

II. SYNTAX OF LT

SUMMARY

After a presentation of the syntax of the strings, we introduce the definition of conditions and actions based on the attributes.

With these three objects (strings, conditions and actions) we define the rules which serve to write the bundles.

Finally, we sketch the general structure of a LT program. Incidentally, the concrete syntax of LT has taken its inspiration from that of Ariane-78 <DSE1>.

1. THE STRINGS

A string is a concatenation of simple strings. A simple string may be a string of characters or hexadecimal codes or special symbols for the end of the line and the end of the file.

Any string of a certain length may be read with help of a special designator.

There exist three other conventions for the output tape to designate the same string as read in input, or with letters only in upper-case or in lower-case.

2. THE CONDITIONS AND ACTIONS

A condition is a first order predicate on the attributes, expressed in the usual syntax (logical connectors: no, and, or; parentheses allowed). The attributes belong to one of the three classes: scalar, set or arithmetic (inferior to an upper bound).

An action can be an assignment of a value to a variable, a list of actions carried out conditionally, or, a block containing a list of actions.

This notion is extended to three predefined actions. The first has no impact at all on the semantics of the transduction (displaying a message on an auxiliary file); the two others, on the contrary, are significant for the transduction (displaying a return code on the error file and stopping the transduction; moving the "forward" head back to the position of the "current" head (semi-regularity)).

3. THE RULES

A rule describes a class of transitions of the shape:

input string / condition == output string / actions .

the symbol ? at the head of the rule signifies that the input string is to be read under the "forward" head.

The philosophy of LT is to put together the possible passages from one etiquette to another into a bundle of the shape:

```
de <etiquette1> a <etiquette2> via
  rule1 rule2 ... ruleN
```

4. GENERAL STRUCTURE OF A LT PROGRAM

A LT program is divided into sections.

- One must give the initial state of the automaton.
- Others give the definition of attributes and their initialisation.
- Other optional sections define conditions, actions and rules which can be referred to by their names in the bundles.
- The other sections give the bundles explicitely.

III. IMPLEMENTATION

In order to facilitate programming in LT, an environment for this language was written in Prolog-Criss <PROLOG85>:

- The manager allows the manipulation of LT programs. The usual functions of an interactive environment (PROLOG, APL) are defined: loading, saving, editing, listing, ...
- The compiler was implemented with use of a generator of analysers inspired from METAL <METAL82>, but less powerful.
- The interpreter is a mock-up in Prolog which works on the abstract trees resulting from compilation.

  The user must specify the files which will be the input and output tapes, and the LT program to be interpreted. Interactive traces are possible.

  The design of a Pascal version of the interpreter in order to increase the rate of execution is currently in work.

IV. APPLICATIONS

1. EXAMPLE OF PROGRAM

To illustrate the syntax of LT, we give a piece of the program for the analysis of AnBnCn on the next page.

```
+--------------------------------------------------------+
!                                                        !
!etat initial <read_AB> .                                !
!                                                        !
!variables                                               !
!  lastchar : (A, B, C, EOL) . -- last character read    !
!                                                        !
!** Reading AnBn, reading alternatively A and B under    !
!   the current and the forward heads respectively.      !
!de <read_AB> a <read_AB> via                            !
!  'A' / lastchar = B == / lastchar := A .               !
!? 'B' / lastchar = A == / lastchar := B .               !
!de <read_AB> a <read_BC> via                            !
!? 'C' / lastchar = A == / lastchar := C .               !
!de <read_AB> a <fail> via                               !
!      / lastchar = B == '*** Rejected string.' eol .    !
!                             -- if not A                !
!?     / lastchar = A == '*** Rejected string.' eol .    !
!                             -- if neither B nor C      !
!                                                        !
!** Reading of BnCn : same principle as for AnBn.        !
!...                                                     !
!                                                        !
!** Final section for the analysis.                      !
!de <read_C> a <success> via                             !
!'C' / lastchar = EOL == '*** Valid string.' eol .       !
!de <read_C> a <fail> via                                !
!                   == '*** Character C missing.' eol .  !
!                                                        !
+--------------------------------------------------------+
```

2. TRANSCRIPTIONS FOR DIACRITICS LETTERS

There exists in French a lot of diacritics and accents. In the frame of Eurotra, a transcription for the diacritics was proposed. Here is a text in the Eurotra Short Transcription and its responding form in the actual French orthograph. The passage between the two forms was performed by a LT program.

```
+--------------------------------------------------------+
!   Un certain Montflory (1533), puis l'imprimeur        !
! Etienne Dolet (qui publia en 1540 De la Punctuation    !
! de la Langue Franc!5oyse, plus des accents d'ycelle    !
! ) l'employe!2rent  notamment pour marquer la chute     !
! d'un  e  à  l'intel1rieur  d'un  mot  : vrai!3ment,    !
! pai!3ra, etc.                                          !
! (Gre!1visse, Le Bon Usage)                             !
+--------------------------------------------------------+
```

```
+--------------------------------------------------------+
!   Un certain Montflory (1533), puis l'imprimeur        !
! Etienne Dolet (qui publia en 1540 De la Punctuation    !
! de la Langue Françoyse, plus des accents d'ycelle )    !
! l'employèrent  notamment pour marquer la chute d'un    !
! e à l'intérieur d'un mot : vraîment, paîra, etc.       !
! (Grévisse, Le Bon Usage)                               !
+--------------------------------------------------------+
```

3. PHYSICAL AND LOGICAL PROCESSING OF TEXTS

The use of the LT language is not limited to the transcriptions; one of its interesting features, and not the least one, is that physical and logical processing of texts could be carried out with its help.

```
+--------------------------------------------------------+
!.sp 2                                                   !
!.us on                                                  !
!Avant-dernier exemple:                                  !
+--------------------------------------------------------+
```

In the previous text, the first two lines correspond to formatting commands of SCRIBERE (a text formatting software developped at GETA and based on SCRIPT, an IBM text formatting software, <SCRIBERE85>) Transducers have been written which reflect tables of informations about punctuation, formatting commands and structural separators. Here is the result of the application of the sequence of those transducers written in LT on the following text.

```
+--------------------------------------------------------+
!.sp 2 : type=format,format=paragraph,level=1,start=no!
!.us on : type=format,format=beg_underli,level=7,        !
!           start=yes,ovl=end_underli                    !
!Avant                                                   !
!- : level=9,start=no,content=hyphen                     !
!dernier                                                 !
!exemple                                                 !
!:  : level=3,start=no,content=colon                     !
+--------------------------------------------------------+
```

CONCLUSION

The language LT defined and implemented as above was tested on various examples:

- passages from transcriptions to others (Russian, Thai, Greek, ...);
- logical and physical processing of texts;
- analysis of the context dependent language AnBnCn.

Though we intentionally limited the syntax of LT and forced non-determinism in the interpretation to fit our purpose, the power of this language seems to be rather sufficient for the applications it is specialised in.

REFERENCES

<AHO,UL72> AHO Alfred V., ULLMAN Jeffrey D.
The Theory of Parsing, Translation and Compiling
Prentice Hall series in Automatic Computation, 1972.

<BOITET83> BOITET Christian
Conventions de transcription pour la saisie et pour la révision de textes sous Ariane-78
Documentation du système russe-français version RUB-FRB
Rapport DRET n° 41, GETA, Grenoble, décembre 1983.

<CHAUCHE74> CHAUCHE Jacques
Transducteurs & Arborescences
Etudes et réalisation de systèmes appliqués aux grammaires transformationnelles
Thèse d'Etat, Grenoble, décembre 1974.

<DSE1> BOITET Christian, editor
Le point sur Ariane-78 début 1982
(Volume 1, Partie 1 : le logiciel) avril 1982
Convention ADI n° 81/423
Cap Sogeti Logiciel - GETA-Champollion

<ETS5> LEPAGE Yves, VAUQUOIS Philippe
Logical and physical processing of texts
Eurotra contract ETS5, Part B
Intermediate report number 2, September 1985

<KAIN72> KAIN Richard Y.
Automata Theory : Machines and Languages
Mac Graw-Hill Computer science-series 1972

<LT85> LEPAGE Yves
LT, un langage de transduction, manuel utilisateur
Internal document, GETA, September 1985

<METAL82> MELESE Bertrand
METAL, un langage de spécification pour le système MENTOR
T.S.I. vol.1, n°4, 1982, pp 275-285

<MENGA84> MENGA Daniel
Le langage de transduction LT
Rapport de troisième année ENSIMAG, Juin 1984

<PROLOG85> CRISS-Université II
PROLOG CRISS, une extension du langage Prolog (Version 4.0)
CRISS-Université II Grenoble, juillet 1985

<SCRIBERE85> BACHUT Daniel, VERASTEGUI Nelson
SCRIBERE
Internal document, GETA, April 1985