

Getting Started with the CAT2 Machine Translation System

Oliver Streiter, Antje Schmidt Wigger, Ute Hauck and Randall Sharp
IAI

Martin-Luther-Straße 14

D-66111 Saarbrücken

Germany

Tel. (0681) 39313

Fax (0681) 397482

email : catler@iai.uni-sb.de

First Edition

© IAI, 1994

March 19, 1996

Contents

1	Introduction	2
2	The Formalism	4
2.1	Feature Bundles	4
2.2	Objects and Rules	6
2.3	Unification	6
2.4	Rule types	10
2.5	The Order of Rule Application	16
2.6	The Levels	17
3	The CAT2 Command Language	18
4	The Installation of CAT2	21
4.1	Preliminaries	21
4.2	How to proceed	21
5	A Mini Example of an English to German Translation System	23
6	Solutions to the Problems	30

Chapter 1

Introduction

CAT2, a transfer-based MT system, was first developed in 1987 as an alternative prototype to Eurotra, the large-scale MT project sponsored by the CEC between 1983 and 1992. In 1985 the CEC developed the first MT formalism, the so-called $\langle C,A \rangle, T$ framework [AJJ⁺85] [AKM⁺86] (C=Constructors, A=Atoms, T=Translators). The idea to develop this prototype was soon rejected, as its inadequacy as a working system became apparent. Starting from 1987, the CEC was developing a second formalism, the so-called Engineering Framework [BN88], the development of which was finally stopped in 1992. In the meantime, a number of alternative MT prototypes has been developed in various Eurotra research centers, in order to do continuous research into machine translation. Among them was the CAT2 system, directly derived from the $\langle C,A \rangle, T$ framework mentioned above.

As any other NLP system, the CAT2 system consists of two parts, the CAT2 formalism and its implementation (software) and the CAT2 lexica and grammars (lingware). Both parts have been constantly improved and tested from 1987 on at IAI. The formalism may be seen as the programming language in which the linguist writes the lexica and grammars. Its formal properties are described in a number of publications [Sha88], [Sha91], [SS92], [Hal93], [SSH⁺94], [Str94], [SS95] reviews [AAB⁺91] and in the CAT2 Reference Manual [Sha94].

To date at IAI, experimental translation systems have been developed for English, German, French, Spanish, Greek, Italian, Portuguese, Russian, Dutch, Czech, Korean and Arab [Hal90], [Str90], [Hal91a], [Hal91b], [SW91], [Iom94], [Cho95]. The systems were widely used to test the adequacy of the formalism and the linguistic conceptualization.

Since 1994 the CAT2 system has been involved into two CEC-sponsored projects, ANTHEM and CAT2-EDS, and a national project, MULTILINT. The ANTHEM project makes use of the CAT2 formalism and the common modules of the CAT2 grammars. Its aim is to analyse medical diagnoses in French and Dutch and to translate them into Dutch and French respectively, as well as into German and to use the IS representation as an interface to an expert system, which checks the diagnosis for internal consistency and allows statistical evaluation of the data. The parametric language approach as exemplified by the CAT2 system is an ideal means to model sublanguages such as the medical sublanguage (cf. [CDS⁺94a], [CDS⁺94b]).

The CAT2-EDS project aims at the development of a fully automatic trilingual machine translation system for German as source language and English and French as target languages. In

cooperation with a major software company the CAT2 system has to be developed and validated in this industrial environment.

In the MULTILINT project CAT2 is integrated into the document administration system of the German automobile manufacturer BMW for different language-based tasks such as terminology checking and multilingual access and retrieval of service information based on linguistic intelligence. This project is funded by the German Federal Ministry of Commerce (Bundesministerium für Wirtschaft - BMWi).

This booklet is intended for those who want rapid access to the CAT2 system as a working tool for the creation of a monolingual or multilingual NLP system, reducing to a minimum the burden of what has to be studied. To make sure that the information is well understood and to stimulate the reader, some exercises have been added, in which the student can test his understanding of the material.

Chapter 2

The Formalism

2.1 Feature Bundles

To represent the (mostly linguistic) information that must be processed, we make use of feature bundles, abbreviated as FB.

A FB is an unordered set of attribute-value pairs.

A FB is represented by curled brackets, which in contrast to square brackets, are used to show that the objects contained in them are not ordered. The attribute value pairs are separated by kommas.

(1) FB: {attribute-value-pair1,attribute-value-pair2,...,attribute-value-pairN}

2.1.1 Attribute-Value Pairs

Attribute-value pairs are represented as **attribute=value**, e.g. an expression like

(2) {colour=red,door=4,type=ford,ps=150}

would represent a partial description of a car. It goes without saying that the order of pieces of information is not relevant for the object described.

The fact that a set is unordered means that e.g. the two FBs (3) and (4) are identical.

(3) {a=b,c=d} (4) {c=d,a=b}

2.1.1.1 Attributes

Attributes are strings of small characters:

colour =
type =

2.1.1.2 Values

Values are

- ATOMs:
 - strings of small characters e.g. colour=red
 - numbers, e.g. door=4
- FBs, e.g colour={inside=red,outside=green}

2.1.1.3 Exercise 1

Are the following attribute-value pairs well-formed?

- (1) COLOUR=red
 - (2) colour=red
 - (3) {type=ford}={ps=150}
 - (4) COLOUR={inside=red}
-

2.1.2 Tree structures

Most linguistic information is organized into tree structures. In a tree structure every FB is associated to a node of a tree structure. A tree structure consists of a Root and a Body, represented as

Root.Body.

where the Root is the FB representing the mother node and Body is the ordered set of FBs, each FB representing a daughter node. The ordered character of the set is shown by square brackets.

{...}. [{...},{...}].

The ordered character implies that the following tree structures are NOT identical:

{...}. [{a=b},{a=c}]. {...}. [{a=c},{a=b}].

A non-atomic node is represented by a tree structure containing at least one daughter (a), whereas an atomic node cannot have any daughters at all (b).

(a) {...}. [{...},...]. (b) {...}. [].

In linguistic terms, a non-atomic node corresponds to a phrasal node (e.g. NP, VP, etc ...) whereas an atomic (=terminal) node corresponds to a lexical unit (=word).

2.2 Objects and Rules

Tree structures can be of two types: **objects** and **rules**.

Rules represent the linguistic knowledge of the system. Grammars and lexica are sets of different rules (i.e. a lexicon contains at least one lexical rule and a grammar contains at least one grammar rule). Rules of a special type can build objects. Such rules are called b-rules (building rules). Objects are tree structures which are generated by b-rules. Objects represent the intermediate state of the analysis of the linguistic entity processed. Other rule types than the b-rules modify or delete objects. The modification of tree structures occurs during the translation process. Objects which represent a false analysis are killed if their construction cannot be generally ruled out.

Rules and Objects interact via one mechanism, that of *Unification*. Rules apply to objects iff they unify with the objects. The result of the application of the rule is the unification of the rule and the object.

2.3 Unification

A rule and an object unify iff

- the tree structures unify
- the feature bundle unify

2.3.1 Unification of tree structures

The tree structure of a rule and an object unify if the number of FB in the Body are identical.

object={...}.[{...},{...},{...}]. rule={...}.[{...},{...},{...}]. Unification
object={...}.[{...},{...},{...}]. rule={...}.[]. No unification

The CAT2 formalism employs the following abbreviations for 'counting' tree structures.

- \sim unifying with 0 or 1 FB
- * unifying with $0 \rightarrow \infty$ FB
- + unifying with $1 \rightarrow \infty$ FB
- _ unifying with 1 FB

With exception of the _ these abbreviations can be used alone or as prefix to a feature structure, where the usage without feature structure corresponds to the specification of an empty feature structure.

- \wedge equals $\wedge\{\}$
- $*$ equals $*\{\}$
- $+$ equals $+\{\}$

2.3.1.1 Exercise 2

Do the object and the rule unify ?

object	rule
1) $\{\dots\}.\square$.	$\{\dots\}.[\wedge]$.
2) $\{\dots\}.\{\{\dots\},\{\dots\}\}$.	$\{\dots\}.[_]$.
3) $\{\dots\}.\{\{\dots\},\{\dots\}\}$.	$\{\dots\}.[*]$.
4) $\{\dots\}.\{\{\dots\},\{\dots\},\{\dots\}\}$.	$\{\dots\}.[_,\wedge]$.
5) $\{\dots\}.\{\{\dots\},\{\dots\},\{\dots\}\}$.	$\{\dots\}.\{\{\dots\},+\}$.
6) $\{\dots\}.\{\{\dots\}\}$.	$\{\dots\}.\square$.
7) $\{\dots\}.\square$.	$\{\dots\}.[+]$.
8) $\{\dots\}.\square$.	$\{\dots\}.[_]$.

2.3.1.2 Exercise 3

How many times do the rule and the object unify ?

object	rule
1) $\{\dots\}.\{\{\dots\}\}$.	$\{\dots\}.[\wedge,\wedge]$.
2) $\{\dots\}.\square$.	$\{\dots\}.[\wedge,\wedge]$.
3) $\{\dots\}.\{\{\dots\},\{\dots\}\}$.	$\{\dots\}.[*,*]$.
4) $\{\dots\}.\{\{\dots\},\{\dots\}\}$.	$\{\dots\}.[*,+]$.
5) $\{\dots\}.\{\{\dots\},\{\dots\},\{\dots\}\}$.	$\{\dots\}.[\wedge,+,*]$.

2.3.2 Unification of Feature Bundles

- Two FBs unify iff
 - for every attribute-value pair of FB1

- * the attribute is not present in FB2
 - * the attribute is present in FB2 and the values unify
 - * the values may be FBs or atoms
- Two atoms unify iff they are identical
 - A FB and an atom do not unify

2.3.2.1 Exercise 4

Do the following FBs unify ?

object	rule
1) {a=b, c={d=e, e=f}}	{a=b, c=d, e=f}
2) {a=b, c={d=e, e=f}}	{a=b, c={d=e, e=f, f=e}}
3) {r=s, k=j}	{a=b, b=c, k=j, b=c, r=c}
4) {k={h=j, l={k=m, r=q}}}	{k={p=q, k=n, r=q, l={a=b}}}

The result of the unification of an object FB1 and a Rule FB2 is the object FB3, where the features (attribute-value pairs) of FB2 not present in FB1 are added to FB1.

Example

object FB1	rule FB2	object FB3
{a=b, c={d=e}}	{d=e, c={e=f}}	{a=b, d=e, c={d=e, e=f}}

2.3.2.2 Exercise 5

Find the result of the unification

object	rule
1) {a={b=c}, d=e}	{a={c=h}, e=f}
2) {a={b={d={a=b}}}}	{a={b={d={a=b, b=c}}}}
3) {f=h, h={d=e}}	{a=b, h={e={f=g}}}

2.3.2.3 Constraints on Unification

The anonymous variable `_` unifies with any value (atoms or FBs). The alternation of values is indicated by `attribute=(value1;values2;...)`. The inequality is indicated by `attribute~value`

Example

object FB1	rule FB2	object FB3
{a=b}	{a~b}	no unification
{a={b=c}}	{a=_}	{a={b=c}}
{a~b}	{a=c}	{a=c}
{a=(b;c)}	{a~c}	{a=b}
{a=(b;c)}	{a=b}	{a=b}
{a=(b;{b=c})}	{a={b=c}}	{a={b=c}}
{a=(b;c)}	{a=(c;d)}	{a=c}
{a=(b;c)}	{a=(d;e)}	no unification

Disjunctions are also possible for the root of an object/rule.

Feature bundles and/or alternations of feature bundles can be coordinated by the '&' operator.

```
{a={b=c}&({c=d};{c=e})}
```

2.3.2.4 Exercise 6

Find the result of the unification

object	rule
1) {a=({a={b=c}};b)}	{a={a={b={c=a}}}}
2) {a={b=_}}	{a=({b={c=d}};b)}
3) {a=({a={b~c}};b)}	{a=({a={b=c}};b)}
4) {a={b=_}}	{a=_}

Variables can be used to establish a link between two atomic values or two feature structures; they will then share all the constraints they receive by unification. According to PROLOG conventions, variables are expressed with capital letters. The same variable used more than once indicates the same value, iff the variables appear in the same object or the same rule. Variable binding between an object and a rule is impossible.

Example

object FB1	rule FB2	object FB3
{a=a}	{a=A, b=A}	{a=a, b=a}
{a=a, b=(a;{c=d})}	{a=A, b=A}	{a=a, b=a}}
{a~a, a=(a;b)}	{a=A, b=A}	{a=b, b=b}
{a=a, b=c}	{a=A, b=A}	{a=a, b=c} (No unification)
{a=a, a=B}	{b=b, b=B}	{a=a, b=b}

2.3.2.5 Exercise 7

Find the result of the unification

object	rule
1) {a~b, a=X, b=X}	{b=b}
2) {c=C, d~d, d=C}	{e=d, c=F, e=F}
3) {c=A, c=d}	{d=A, d=e}

There is also a possibility to require that an object should have a particular structure by using the implication operator. If the condition part unifies with the object, the implication part must also unify with the object if the object is to be validated. The implication operator is notated like this:

{condition part}>>{implication part}

2.4 Rule types

The CAT2 formalism has 5 different types of rules.

- b-rules (building rules):
they confirm tree structures
- f-rules (feature rules):
they test the well-formedness of tree structures, add information or kill the structure
- t-rules (transfer rules):
they translate one tree structure into another tree structure.
- tf-rules (transfer feature rules):
they test the well-formedness of tree structures translated by t-rules. They add features or kill the structure.

- mw-rules (multi word rules):
they translate a list of feature bundles into a list of feature bundles

Rule types are indicated by a **rule type statement** in combination with the level declaration.

- @level(LEVEL).
@rule(b).
- @level(LEVEL).
@rule(f).
- @level(LEVEL1 ⇔ LEVEL2).
@rule(t).
- @level(LEVEL1 ⇔ LEVEL2).
@rule(f).
- @level(LEVEL1 ⇔ LEVEL2).
@rule(mw).

All rules which follow a rule type statement are interpreted as belonging to that rule type, until another rule type statement is found.

2.4.1 B-rules

B-rules confirm and hereby build up tree structures which are suggested

- by the parser
- by t-rules

The parser is a general algorithm to build up a tree structure from a linear string of words. The parser tries to build all possible syntactic structures. However, only those consolidated by the b-rules are definitely built.

linear input structure for the parser:

```
{lex=d,string=the,cat=d}, {lex=man,string=man,cat=n}, {lex=come,string=comes,cat=v}
```

Grammar A:

```
b_s = {cat=s}. [{cat=dp},{cat=v}].
b_dp= {cat=dp}. [{cat=d},{cat=n}].
b_vp= {cat=v}. [{cat=v},{cat=dp}].
```

tree structure:

```
{cat=s}. [{cat=dp}. [{lex=d,string=the,cat=d},
                  {lex=man,string=man,cat=n}],
          {lex=come,string=comes,cat=v}]] .
```

B-rules are unordered, i. e. b-rules can be written and applied in any order without affecting the final result.

2.4.1.1 Exercise 8

Write a grammar of b-rules which analyses the following structure:

```
{cat=s}. [{cat=dp}. [{cat=d},
                    {cat=np}. [{cat=a},
                                {cat=np}. [{cat=a},
                                            {cat=n}]]],
          {cat=v}. [{cat=v}. [{cat=v},
                              {cat=adv}],
                    {cat=neg}]] .
```

2.4.2 F-rules

F-rules control the well-formedness of tree structures: they add information to the structure or kill the structure.

F-rules are ordered: Changing the order of rules in their written form changes the order of rule application, which may give different results.

F-rules apply to objects after they have been built by b-rules.

If a normal f-rule does not unify with the object nothing happens. F-rules may be divided into a condition part and a consequence part. This second part may kill an object, if the first part unifies and the second part does not.

```
{condition part}>>{implication part}
```

2.4.2.1 Exercise 9

Add the following f-rules to grammar A . (You should presume that the value of 'num' is calculated for each atom by the morphological module and as such present in the linear input for the parser.) How different will the behavior of a) and b) be?

a)

```

@rule(f).
percol_num_dp = {cat=dp,num=NUM}. [_,{cat=n,num=NUM}].
percol_num_v  = {cat=v,num=NUM}. [{cat=v,num=NUM},_].
default_num_dp = {cat=dp,num=sing}. [_,_].
default_num_vp = {cat=v,num=plu}. [_,_].
subj_verb_agr = {}. [{cat=dp}>>{num=NUM},
                    {cat=v}>>{num=NUM}].

```

b)

```

@rule(f).
default_num_dp = {cat=dp,num=sing}. [_,_].
default_num_vp = {cat=v,num=plu}. [_,_].
percol_num_dp  = {cat=dp,num=NUM}. [_,{cat=n,num=NUM}].
percol_num_vp  = {cat=v,num=NUM}. [{cat=v,num=NUM},_].
subj_verb_agr  = {}. [{cat=dp}>>{num=NUM},
                    {cat=v}>>{num=NUM}].

```

2.4.2.2 Exercise 10

The CAT2 formalism assigns a word number to every word of a sentence. '#word'=1 for the first word, '#word'=2 to the second word etc Write a f-rule that percolates the word number from the left daughter to the mother node

2.4.3 T-Rules

T-rules translate one tree structure into another tree structure.

T-rules are unordered (there is however one type of t-rule which is ordered).

Examples:

```

t_tasse = {lex=tasse}. [] <=> {lex=cup}. [].
t_gross_angelegt = {}. [{lex=angelegt},{lex=gross}] <=> {lex='large-scale'}. [].

```

T-rules may be one-directional or bidirectional as in the upper examples. One directional t-rules can be used in order to have a 'default' translation in the case where more or two translations are possible, as show in the following example.

```

t = {lu=aendern,head={pref=ab}}. [] <= {lu=alter}. [].
t = {lu=aendern,head={pref=ab}}. [] <= {lu=change}. [].
t = {lu=aendern,head={pref=ab}}. [] <=> {lu=modify}. [].

```

When translating from German into English, the verb *abändern* is translated into *modify* only, since *to alter* and *to change* are not made accessible. If the same transfer lexicon is used for the transfer English to German, all English verbs, *modify*, *change* and *alter* are translated into the German verb *abändern*.

In t-rules special markers can be used. A marker is a string of small characters, separated from FB by ':'. It refers to the whole tree structure to the root of which it is attached. The marker indicates that the tree structure so marked must be translated by another rule. Trees which are not marked by a marker must be rewritten on the right-hand side, or they will not be translated. Consider the following example:

```
@rule(t).

t_rule_s = {cat=s}. [subj:{cat=n},
                    {cat=v}. [verb:{cat=v},
                              obj:{cat=n}]
                    {cat=punct}]
=>
{cat=s}. [verb,subj,obj].

t_atom = {cat=CAT,lex=LEX}. [] <=> {cat=CAT,lex=LEX}. [].
```

The subject, the verb and the object are not directly translated by the rule `t_rule_s`. Since these (sub)trees are marked, they must be translated by another rule. The subject, the verb and the object are all translated by the rule `t_atom`, the results of this translation are fetched by the markers and put into the order indicated on the right-hand side of the rule `t_rule_s`.

2.4.3.1 Exercise 11

Given the object (1) and the t-rules (2), in which order do the different parts of rules apply: A, B, C or D ? lhs = left-hand side, rhs = right-hand side. Remember that t-rules decompose the structure to be translated top-down and rebuild it bottom-up.

(1)

```
{cat=pp,role=mod}. [{cat=p,role=funct},
                   {cat=dp,role=mod}. [{cat=d,role=funct},
                                       {cat=n,role=mod}]]
```

(2)

```
t_funct = {}. [{role=funct},x:{role=R}] => x:{role=R}.
t_atom = {cat=CAT,lex=LEX}. [] => {cat=CAT,lex=LEX}. [].
```

Orders:

A	B	C	D
t_funct lhs	t_funct lhs	t_funct lhs	t_atom lhs
t_funct rhs	t_funct lhs	t_funct rhs	t_atom rhs
t_atom lhs	t_atom lhs	t_funct lhs	t_funct rhs
t_atom rhs	t_atom rhs	t_funct rhs	t_funct lhs
	t_funct rhs	t_atom lhs	t_funct rhs
	t_funct rhs	t_atom rhs	t_funct lhs

2.4.4 TF-rules

Tf-rules test the well-formedness of tree structures translated by t-rules. They add information or, if combined with the '!!' operator, kill the structure. Like f-rules tf-rules are ordered.

Example:

```
t_num1 = {num=NUM}. [*] => {num=NUM}. [*].  
t_num2 = {num=NUM}. [*] => {}>>{num=NUM}. [*].
```

2.4.4.1 Exercise 12

What is the difference between t_num1 and t_num2? Which rule should be used in the transfer between languages, in view of the pluralia tantum and singularia tantum which exist in many languages?

2.4.5 MW-rules

MW-rules transfer word lists into word lists. They identify e.g. multiword-units in the input sentence and link them together into one node to be proposed to the syntactic parser. On the other hand, they can split of contracted elements for generalizing the syntax grammar.

Example:

```
t_mwu = [{lex=pomme},{lex=d},{lex=terre}] <=> [{lex=pomme_de_terre}].  
t_con = [{lex=p,string=am}] <=> [{lex=p,string=an},{lex=d,string=dem}].
```

2.5 The Order of Rule Application

Rules interact in a special order. B-rules confirm structures offered by t-rules. Thus t-rules apply before b-rules. After the translation to the new level the f-rules are applied after the tf-rules. The general order of rule application is as follows:

$t \gg b \gg tf \gg f$

2.5.0.1 Exercise 13

The structure (a) is translated into the structure (b). In which order are the rules of the grammar applied?

CSDE:

(a) {cat=np}. [{cat=a,role=mod,lex=schoen},
 {cat=n,role=gov,lex=haus}]

ISDE:

(b) {cat=n}. [{cat=n,role=gov,lex=schoen},
 {cat=a,role=mod,lex=haus}]

grammar:

@level(csde-isde).

@rule(t).

t_1 = {}. [mod:{role=mod},gov:{role=gov}] => {}. [gov,mod].

t_2 = {cat=CAT,lex=LEX}. [] => {cat=CAT,lex=LEX}. [].

@rule(f).

tf_1 = {role=R}. [*] => {role=R}. [*].

tf_4 = {cat=n,num=N}. [] => {}>>{num=N}. [].

@level(isde/relational/german).

@rule(b).

b_1 = {}. [{role=gov}, * {role~=gov}].

haus = {lex=haus,cat=n}. [].

schoen = {lex=schoen,cat=n}. [].

@rule(f).

f_1 = {cat=CAT}. [{cat=CAT},*].

2.6 The Levels

As it had been seen in the Rule chapter, rules are defined in combination with a level. CAT2 has 5 different level types, which can be divided in constructor levels and translator level.

Constructor levels are defined by the level name, the level type and the language they are connected to.

```
@level(LEVELNAME/LEVELTYPE/LANGUAGE).
```

The three constructor level types are:

- *morph*: On this level, sentences are represented by lists of feature bundles. It can only be defined once for each language.
- *syntactic*: On this level, a parser constructs in analysis a tree on the morphological word list. In synthesis, the level is used like a relational level. It can only be defined once for each language.
- *relational*: On this level, a weak tree proposed by a translator level is confirmed. CAT2 allows to define an indefinite number of relational levels.

Translator levels are defined by the names of the constructor levels they connect together.

```
@level(LEVELNAME<=>LEVELNAME).
```

If they connect a syntactic or a relational level with another relational level, they use t-rules and tf-rules for mapping trees on trees. If they connect a morph level with a syntactic level, they use mw-rules for mapping lists on lists. Connecting a morph level directly with a relational level is not possible.

Chapter 3

The CAT2 Command Language

(Antje Schmidt Wigger)

BASIC ENTITIES:

(Things in parentheses can be omitted.)

- OBJECTs (LEVELNAME)/(NUMBERs)
- RULEs (LEVELNAME):(RULETYPE):(RULENAME)
- NUMBERs can be lists NB,NB or ranges NB-NB
- FEATs can be listed FEAT,FEAT
a feature name should specify the feature hierarchy FEAT{FEAT}

PROMPTS, MODES and expected RESPONSES:

- Command? Command mode: Every command, followed by <RETURN>
- |: Input mode: An input sentence or ^D
- RULENAME..→ Trace mode: <RETURN> or a trace command
- OBJECT? Command mod: <RETURN> or q (stops the execution of the command)
or a (executes the command without asking anymore)
- | ?- you are in Prolog!
start. goes back to CAT2 while ^D leaves the program.
- if there is a problem: ^D leaves CAT2 for the Prolog level

BASIC COMMANDS FOR SIMPLE TRANSLATION:

- cat2 enters the system
- l(oad) FILE loads grammars, lexicons (and kepted objects)
- set source LANGUAGE chooses the input language
- set target LANGUAGE chooses the output language
- i(nput) FILE takes sentences from a file as input
i <RETURN> INPUT <RETURN> ^D or interactive input of sentences
- t(ranslate) OBJECT translates an object
- quit leaves the system

SOME ADDITIONAL COMMANDS:

- t(ranslate) OBJECT translates an object to the target language
- t OBJECT to LEVELNAME translates an object into an object on the named level
- s(how) (-g) OBJECT shows the tree-structure with numbered nodes
 - x OBJECT shows the tree-structure without lexemes on the leaves
 - h OBJECT shows the history of the object
 - f OBJECT shows the entire feature-structure of an object
 - f FEATs OBJECT shows only the named features
 - n (FEATs) OBJECT shows the (partial) feature structure of the node number 'n'
- t(race) (RULEs)
 - t will stop for all rules
 - t RULEs adds the rule(s) to the set of rules where the trace stops
- u(ntrace) (RULEs) unables trace-modus for translation
- possible trace commands:
 - f shows feature structure of the partial object
 - g shows tree structure of the partial object
 - j jumps to the next b/t-rule for this node or to the first b/t-rule of its sister node
 - t (RULEs) traces all (or the listed) rules
 - u untraces all rules
 - q stops the translation process
- set PARAMETER VALUE (for further parameters see the HELP command)
 - ask on(off) switches on/off the questioning before executing the command
 - trans all(first) produces all possible translations or only the first translation found
 - analysis extern chooses the morphological processor for the input
 - synthesis extern chooses the morphological processor for the output
- st(atus) displays informations of parameters and loaded levels
- c(ompare) (-f) OBJECTS compares the named objects two by two;
- k(eep) OBJECT FILE stores an object in a file
- del(ete) OBJECTS deletes objects
- counting continues if there are still objects at this level
- unl(oad) LEVELNAME deletes all rules at the named level
- COMMAND > FILE writes the output of a command in a file;
(mostly used for show,trace,translate and lex)
- e(dit) FILE enters VI-modus
- h(elp) COMMAND displays informations about most of the commands

Chapter 4

The Installation of CAT2

(Ute Hauck)

4.1 Preliminaries

For the installation of the Cat2 system you need a Sparc machine with at least 16 MB RAM. As operating system all versions from 4.1 upwards are possible.

4.2 How to proceed

- Ask your system manager to install a user named CAT2.
- login as CAT2 user
- put the cartridge into the streamer
- to make sure that the cartridge contains the right files and directories, show the content of the cartridge with

```
tar tvf /dev/rst8
```

or

```
tar tvf /dev/rst0
```

You should see something like this:

```
.cshrc
```

```
.login
```

```
.logout
```

```
/bin
```

```
/bin/Runtime
```

```
/grammars
```

```
/grammars/common
```

```
/grammars/de
```

```
/term
```

```
etc...
```

- copy the content of the cartridge onto the hard disk
tar xvf /dev/rst8
or
tar xvf /dev/rst0
- logout
- login as CAT2 user
All environment variables must be correctly instantiated. To make sure that the environment variables are o.k. type
setenv
Among others the variables CAT2 should have the value:
CAT2= ~cat2
If you use the runtime system, the variable SP_PATH should have the value:
SP_PATH ~cat2/Runtime

Chapter 5

A Mini Example of an English to German Translation System

(Randall Sharp)

```
/* CAT2 English--German Mini-Translation System
   IAI, Saarbruecken, Germany */

%=====
%
%           ENGLISH
%
%=====
%-----%
%           Morphological Structure
%-----%

@level(msen/morph/english).
@call(common_morph).
@call(en_lexicon).

%-----%
%           Constituent Structure
%-----%

@level(csen/syntactic/english).
@call(en_lexicon).

%%%%%%%%%% B-Rules
@rule(b).
text = {cat=text}. [ {cat=s,tense~=nil,mode=M}, {cat=punct,mode=M} ].
s     = {cat=s} . [ {cat=np}, ^{cat=aux}, {cat=vp} ].
np    = {cat=np,det=T,agr=A}. [ ^{cat=det,type=T,agr=A},
```

```

^ {cat=num, agr=A},
  {cat=n, agr=A} ].
vp = {cat=vp, agr=A, tense=T}. [ {cat=v, agr=A, tense=T}, *{cat=(np;pp)} ].

```

%%%%%%%%%% F-Rules

```

@rule(f).
np_vp_agr = {cat=s, tense=T}. [ {cat=np, agr=A}, {}>>{agr=A, tense=T}, * ].
declarative = {cat=s}>>{mode=decl}. [ {cat=np}, * ].
interrogative = {cat=s}>>{mode=interrog}. [ {cat=aux}, {cat=np}, * ].
det_abs = {cat=np}>>{det=abs}. [ +{cat~=det} ].

```

```

%-----%
%           Relational Structure           %
%-----%

```

```

@level(isen/relational/english).
@call(common_is).
@call(en_lexicon).

```

```

%-----%
%           English Lexicon           %
%-----%

```

```

@common(en_lexicon).
@rule(b).

```

```

council = {lex=council, cat=n}
>>
  ({string=council, agr={num=sing}}
  ;{string=councils, agr={num=plu}}). [].

```

```

decision = {lex=decision, cat=n}
>>
  ({string=decision, agr={num=sing}}
  ;{string=decisions, agr={num=plu}}). [].

```

```

adopt = {lex=adopt, cat=v,
  frame={arg1={role=agent, cat=n}, arg2={role=theme, cat=n}}}
>>
  ({string=adopt}>>({tense=nil, vform=infin}
    ;{tense=pres, vform=fin, agr~={per=3, num=sing}})
  ;{string=adopted}>>({tense=nil, vform=pastp}
    ;{tense=past, vform=fin})
  ;{string=adopting, tense=nil, vform=presp}
  ;{string=adopts, tense=pres, vform=fin, agr={per=3, num=sing}}). [].

```

```

article = {lex=art,string=the,cat=det,type=def}. [].
article = {lex=art,string=(a;an),cat=det,agr={num=sing},type=indef}. [].

```

```

punct   = {lex=end,string='.',cat=punct,mode=decl}. [].

```

```

%-----%
%           CSEN<=>ISEN Transformation           %
%-----%

```

```

@level(csen<=>isen).

```

```

%%%%%%%%%% T-Rules

```

```

@rule(t).

```

```

text = {cat=text}. [ s, {lex=end} ] <=> {}. [ s:{role=proposition} ].

```

```

s     = {cat=s}. [ 1, {}. [ 2, *3 ] ] <=> {cat=v}. [ 2, 1, *3 ].

```

```

np    = {cat=np,det=T}. [ ^{lex=art,type=T},
                    ^1:{cat=num},
                    2:{cat=n} ] <=>
        {cat=n}. [ 2, ^1 ].

```

```

atom  = ATOM. [] <=> ATOM. [].

```

```

%%%%%%%%%% TF-Rules

```

```

@rule(f).

```

```

% Copy NP agreement between CS and IS:

```

```

np_agr = {cat=np}>>{agr=A,det=T}. [ + ] <=> {cat=n}>>{agr=A,det=T}. [ + ].

```

```

%-----%
%           English <=> German Transfer           %
%-----%

```

```

@level(isen<=>isde).

```

```

%%%%%%%%%% T-Rule

```

```

@rule(t).

```

```

cons = {}. [ +x ] <=> {}. [ +x ].

```

```

atom = {lex=council}. [] <=> {lex=rat}. [].

```

```

atom = {lex=decision}. [] <=> {lex=beschluss}. [].

```

```

atom = {lex=adopt}. [] <=> {lex=verabschieden}. [].

```

```

unknown = {lex='@unknown',string=L}. [] <=> {lex='@unknown',string=L}. [].

```

```

num     = {lex='@num',string=L}. [] <=> {lex='@num',string=L}. [].

```

```

%%%%%%%%%% TF-Rules

```

```

@rule(f).

```

```

cat = {}>>{cat=C}. [] <=> {}>>{cat=C}. [] . % Copy category

```

```

agr = {cat=n}>>{agr=A}. [*] <=> {cat=n}>>{agr=A}. [*] . % Copy noun agreement

```

```

det = {cat=n}>>{det=T}. [*] <=> {cat=n}>>{det=T}. [*] . % Copy det type

```

```

tns = {cat=v}>>{tense=T}. [*] <=> {cat=v}>>{tense=T}. [*] . % Copy verb tense

```

```
role = {}>>{role=R}.[*]      <=> {}>>{role=R}.[*].      % Copy semantic roles
```

```
%===== %  
%                                     %  
%                GERMAN                %  
%                                     %  
%===== %
```

```
%----- %  
%                Morphological Structure                %  
%----- %
```

```
@level(msde/morph/german).  
@call(common_morph).  
@call(de_lexicon).
```

```
%----- %  
%                Constituent Structure                %  
%----- %
```

```
@level(csde/syntactic/german).  
@call(de_lexicon).
```

```
%%%%%%%% B-Rules
```

```
@rule(b).  
text = {cat=text}      . [ {cat=s,tense~=nil}, {cat=punct,lex=end} ] .  
s     = {cat=s,tense=T} . [ {cat=np,agr=A},  
                           {cat=v,agr=A,tense=T,vform=fin},  
                           *{cat=(np;pp)},  
                           ^{cat=v,tense=nil} ] .  
np    = {cat=np,agr=A,case=C,det=T}  
        . [ ^{cat=det,agr=A,case=C,type=T},  
            ^{cat=num,agr=A},  
            {cat=n,agr=A,case=C} ] .
```

```
@rule(f).  
det_abs = {cat=np}>>{det=abs}. [ +{cat~=det} ] .
```

```
%----- %  
%                Relational Structure                %  
%----- %
```

```
@level(isde/relational/german).  
@call(common_is).  
@call(de_lexicon).
```

```
%----- %  
%                German Lexicon                %  
%----- %
```

```

@common(de_lexicon).
@rule(b).

rat = {lex=rat,cat=n,agr={gen=masc}}
>>
({string='Rat',agr={num=sing},case~=gen}
;{string='Rats',agr={num=sing},case=gen}
;{string='Raete',agr={num=plu},case~=dat}
;{string='Raeten',agr={num=plu},case=dat}). [].

beschluss =
{lex=beschluss,cat=n,agr={gen=masc}}
>>
({string='Beschluss',agr={num=sing},case~=gen}
;{string='Beschlusses',agr={num=sing},case=gen}
;{string='Beschluesse',agr={num=plu},case~=dat}
;{string='Beschluessen',agr={num=plu},case=dat}). [].

verabschieden =
{lex=verabschieden,cat=v,
 frame={arg1={role=agent,cat=n,case=nom},
        arg2={role=theme,cat=n,case=acc}}}
>>
({string=verabschieden}>>({tense=nil,vform=infin}
;{tense=pres,vform=fin,agr={per=3,num=plu}})
;{string=verabschiedet}>>({tense=nil,vform=pastp}
;{tense=pres,vform=fin,agr={per=3,num=sing}})
;{string=verabschiedete,tense=past,vform=fin,agr={per=(1;3),num=sing}}
;{string=verabschiedeten,tense=past,vform=fin,agr={per=(1;3),num=plu}}). [].

article =
{lex=art,cat=det,type=def}
>>
({string=der,agr={num=sing,gen=masc},case=nom}
;{string=die,agr=({num=sing,gen=fem};{num=plu}),case=(nom;acc)}
;{string=das,agr={num=sing,gen=neut},case=(nom;acc)}
;{string=den,agr={num=sing,gen=masc},case=acc}). [].

article =
{lex=art,cat=det,type=indef,agr={num=sing}}
>>
({string=ein,agr={gen=masc},case=nom}
;{string=eine,agr={gen=fem},case=(nom;acc)}
;{string=einen,agr={gen=masc},case=acc}). [].

punct = {lex=end,string='.',cat=punct,mode=decl}. [].

```

```

%-----%
%           CSDE<=>ISDE Transformation           %
%-----%

@level(csde<=>isde).

%%%%%%%% T-Rules
@rule(t).

text = {cat=text}. [ s, {lex=end} ] <=> {}. [ s:{role=proposition} ].
s     = {cat=s}. [ 1, 2, *3 ] <=> {cat=v}. [ 2, 1, *3 ].
np    = {cat=np,det=T}. [ ^{lex=art,type=T},
                        ^1:{cat=num},
                        2:{cat=n} ] <=>
        {cat=n}. [ 2, ^1 ].
atom  = ATOM. [] <=> ATOM. [].

@rule(f).
npAgr = {cat=np}>>{agr=A,case=C,det=T}. [+] <=>
        {cat=n}>>{agr=A,case=C,det=T}. [+] .

%-----%
%           COMMON MORPHOLOGICAL LEVEL           %
%-----%

@common(common_morph).
@rule(f).

num = {lex='@num',cat=num}
    >>
    ({string=1,agr={num=sing}};{string~=1,agr={num=plu}}). [].

per3 = {cat=(n;det),agr={per=3}}. [].

%-----%
%           COMMON RELATIONAL LEVEL             %
%-----%

@common(common_is).
%%%%%%%% B-Rules
@rule(b).
text = {role=text}. [ {role=proposition,cat=v} ].
pred = {role~=text,cat=C}. [ {role=gov,cat=C,frame={arg1=A1,arg2=A2}},
                            ^A1, ^A2, *{role=mod} ].

%%%%%%%% F-Rules
@rule(f).
frame1 = {frame={arg1=nil}}. [].

```

```
frame2 = {frame={arg2=nil}}.[].
```

```
complete1 = {}. [ {role=gov,frame={arg1={role=R}}}, *,{}>>{role=R},* ].
```

```
complete2 = {}. [ {role=gov,frame={arg2={role=R}}}, *,{}>>{role=R},* ].
```

Chapter 6

Solutions to the Problems

(Oliver Streiter)

6.0.0.2 Exercise 1

Are the following attribute-value pairs well-formed?

- (1) no, attributes must be written in small characters, the value is well-formed
 - (2) yes
 - (3) no, attributes can not be FB, the value is well-formed
 - (4) no, attributes must be written in small characters, the value is well-formed
-

6.0.0.3 Exercise 2

Do the object and the rule unify ?

- 1) yes, `\verb#^#` is 0 FB
 - 2) no, 2 FBs in the object, one FB in the rule
 - 3) yes, 2 FBs in the object, `*` is 2 FBs
 - 4) no, 3 FBs in the object, max. 2 FBs in the rule
 - 5) yes, 3 FBs in the object, `_` is 1 FB, `+` is 2 FBs
 - 6) no, 1 FB in the object, no FB in the rule
 - 7) no, no FB in the object, one or more FBs in the rule
 - 8) no, no FB in the object, one FB in the rule
-

6.0.0.4 Exercise 3

How many times do the rule and the object unify ?

- 1) 2 times: 1st solution $\hat{=0} \hat{=1}$
2nd solution $\hat{=1} \hat{=0}$
 - 2) once $\hat{=0} \hat{=0}$
 - 3) 3 times 1st solution $*=0 *=2$
2nd solution $*=1 *=1$
3rd solution $*=2 *=0$
 - 4) 2 times 1st solution $*=0 +=2$
2nd solution $*=1 +=1$
 - 5) 5 times 1st solution $\hat{=0} +=1 *=2$
2nd solution $\hat{=1} +=1 *=1$
3rd solution $\hat{=0} +=2 *=1$
4th solution $\hat{=0} +=3 *=0$
5th solution $\hat{=1} +=2 *=0$
-

6.0.0.5 Exercise 4

Do the following FBs unify ?

- 1) no (c=feature bundle, c=atom)
 - 2) yes
 - 3) no (r=s, r=c)
 - 4) yes
-

6.0.0.6 Exercise 5

Find the result of the unification

- 1) $\{a=\{b=c, c=h\}, d=e, e=f\}$
 - 2) $\{a=\{b=\{d=\{a=b, b=c\}\}\}\}$
 - 3) $\{f=h, a=b, h=\{d=e, e=\{f=g\}\}\}$
-

6.0.0.7 Exercise 6

Find the result of the unification

- 1) no unification
 - 2) $\{a=\{b=\{c=d}\}\}$
 - 3) $\{a=b\}$
 - 4) $\{a=\{b=_\}\}$
-

6.0.0.8 Exercise 7

Find the result of the unification

- 1) no unification
 - 2) no unification
 - 3) $\{c=d, d=e\}$
Variable binding (e.g. $A=A$) is possible within a rule or within an object. There can be no variable binding between object and rule.
-

6.0.0.9 Exercise 8

Write a grammar of b-rules which analyses the following structure

One possibility to write a grammar which generates the structure is as follows. Note however, that this grammar generates far more than this one structure.

$$b_s = \{cat=s\}. [\{cat=dp\}, \\ \{cat=v\}].$$
$$b_dp = \{cat=dp\}. [\{cat=d\}, \\ \{cat=np\}].$$
$$b_np = \{cat=np\}. [\{cat=a\},$$

`{cat=(np;n)}`].

`b_vp= {cat=v}. [{cat=v},
 {cat=(adv;neg)}]`].

6.0.0.10 Exercise 9

Add the following f-rules to grammar A . How different will the behavior of a) and b) be?

Both sets of f-rules differ with respect to the order of f-rules. The order of application of f-rules is relevant: In the first case the number of the dp and vp is calculated from the n or v respectively. If no values have been calculated, default values are assigned. The last rule tests the number of the dp and the vp. If the values do not unify, the structure is killed. With the second set of f-rules, first default rules are assigned to the dp and vp. Since different values are assigned, the object necessarily gets killed by the last f-rule.

6.0.0.11 Exercise 10

The CAT2 formalism assigns a word number to every word of a sentence. '#word'=1 for the first word, '#word'=2 to the second word etc Write a f-rule that percolates the word number from the left daughter to the mother node

`percol_num = {'#word'=WORD}. [{'#word'=WORD}, *]`.

an alternative solution, if you assume all branchings to be binary:

`percol_num = {'#word'=WORD}. [{'#word'=WORD}, _]`.

6.0.0.12 Exercise 11

Given the object (1) and the t-rules (2), in which order do the different parts of

rules apply: A, B, C or D ? lhs = left hand side, rhs = right hand side

B
t_funct lhs
t_funct lhs
t_atom lhs
t_atom rhs
t_funct rhs
t_funct lhs

Explanation:

t-rules apply top down. The first rule to apply is t_funct which analyses the PP as a P which has not to be translated and a DP whose translation is the result of the translation of the PP. Again the left hand side of the t_funct rule applies, which analyses the DP as a D and a N, which must be translated by another rule and whose translation is the result of the translation of the DP. The lhs of the t_atom rule unifies with the N and translates it (rhs of the t_atom rule). The result of this is the result of the translation of the DP iff the role of the result unifies with the role on the lhs. The rhs of t_funct gets role=mod. The result of this is the result of the translation of the PP iff the role of the result unifies with the role on the lhs. The rhs of t_funct gets role=mod.

6.0.0.13 Exercise 12

What is the difference between tf_num1 and tf_num2? Which rule should be used in the transfer between languages, in view of the pluralia tantum and singularia tantum which exist in many languages?

tf_num1 transfers the values of number from one language to another. If the number values is determined by the target language (as is the case for pluralia and singularia tantum) and does not unify with the value of the source language, this rule does not apply. The second rule however applies and could kill the target object:

```
{lex=information,string='Informationen',num=plu}. []  
=>  
{lex=information,num=sing}. []
```

Therefore only the first rule should be used in the transfer component between languages. If no number value is pre-determined in the target language, the value is taken from the source language, otherwise the rule does not apply.

6.0.0.14 Exercise 13

The structure (a) is translated into the structure (b). In which order are the rules of the grammar applied?

t1 lhs (np)
t2 lhs (a)
t2 rhs (a)
lexical entry a
tf1 (a)
t2 lhs (n)
t2 rhs (n)
lexical entry n
tf1 (n)
tf2 (n)
t1 lhs (np)
b1 (np)
t3 (np)
f1 (np)

Bibliography

- [AAB⁺91] H. Alshawi, Doug J. Arnold, Ralf Backofen, D.M. Carter, J. Lindop, Klaus Netter, Steve G. Pulman, J. Tsujii, and Hans Uszkoreit. ET6/1 formalism study - final report. DG XIII, CEC, Luxembourg, 1991.
- [AJJ⁺85] Doug Arnold, Lieven Jaspaert, Rod Johnson, Steven Krauwer, Mike Rosner, Louis des Tombe, G.B. Varile, and Susan Warwick. A MU1 view of the <C,A>,T framework in EUROTRA. In *Proceedings of the Conference on Theoretical and Methodological Issues of Natural Languages, Colgate University*, pages 1–14. Hamilton, NY, 1985.
- [AKM⁺86] Doug Arnold, S. Krauwer, Rosner M., Louis des Tombe, and G.B. Varile. The <C,A>,T framework in EUROTRA: A theoretically committed notation for MT. In *COLING-86*, pages 297–303. 1986.
- [BN88] Anneliese Bech and A. Nygaard. The E-framework: A formalism for natural language processing. In *Proceedings of COLING'88*, pages 36–39. Budapest, 1988.
- [CDS⁺94a] Werner Ceusters, Guy Deville, Oliver Streiter, Emmanuel Herbigniaux, and Jos Devlies. A computational linguistic approach to semantic modeling in medicine. In *Belgo-Dutch Congress on Medical Informatics '94*, pages 311–319, Veldhoven, 1994.
- [CDS⁺94b] Werner Ceusters, Guy Deville, Oliver Streiter, Emmanuel Herbigniaux, and Jos Devlies. When Machine Translation meets automatic encoding. In *Proceedings of the 1st Language Engineering Convention, held at Paris, 6-7 July 1994*, Paris, 1994.
- [Cho95] Sung-Kwon Choi. Unifikationsbasierte Maschinelle Übersetzung mit Koreansisch als Quellsprache. IAI WP 34, 1995.
- [Hal90] Johann Haller. Die semantische Interface-Struktur von EUROTRA im Transfer zwischen Deutsch und Portugiesisch. In H. Lüdtke and J. Schmidt-Radefeld, editors, *Linguistika contrastiva: Deutsch versus Protugiesisch und Spanisch*, Tübingen, 15.-17.November 1990. Akten des 2.kontrastiven Kolloquiums an der Christian-Albrechts-Universität Kiel, G.Narr Verlag.
- [Hal91a] Johann Haller. EUROTRA – Das Forschungs- und Entwicklungsprojekt der EG zur Maschinellen Übersetzung: Französisch- Deutsche Übersetzung mit der Seitenlinie CAT2. In Rolshoven and Seelbach, editors, *Romanistische Computerlinguistik, Linguistische Arbeiten*. Max Niemeyer Verlag, Tübingen, 1991.

- [Hal91b] Johann Haller. EUROTRA – O projeto de pesquisa e desenvolvimento em tradução automática da Comissão Europeia. Exemplos de Tradução Português-Alemão. *Coletânea da revista 'Letras de Hoje'*, 1991.
- [Hal93] Johann Haller. CAT2, Vom Forschungssystem zum präindustriellen Prototyp. In Horst P. Pütz and Johann Haller, editors, *Sprachtechnologie: Methoden, Werkzeuge, Perspektiven*, pages 282–303, Hildesheim, 1993. GLDV, Georg Olms AG. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.
- [Iom94] Leonid Iomdin. Automatic syntactic analysis of Russian in the CAT2 MT system. IAI WP 32, 1994.
- [Sha88] Randall Sharp. CAT2 - implementing a formalism for multi-lingual MT. In *Proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*. Carnegie Mellon University, Pittsburgh, PA, 1988.
- [Sha91] Randall Sharp. CAT2: An experimental Eurotra alternative. *Machine Translation*, 6(3):215–228, 1991.
- [Sha94] Randall Sharp. CAT2 Reference Manual Version 3.6. IAI WP n.27, 1994.
- [SS92] Randall Sharp and Oliver Streiter. Simplifying the Complexity of Machine Translation. *Meta-92*, pages 681–692, 1992. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.
- [SS95] Randall Sharp and Oliver Streiter. Applications in Multilingual Machine Translation. In *Proceedings of The Third International Conference and Exhibition on Practical Applications of Prolog, Paris, 4th-7th April, 1995*. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.
- [SSH⁺94] Oliver Streiter, Randall Sharp, Johann Haller, Catherine Pease, and Antje Schmidt-Wigger. Aspects of a unification based multilingual system for computer-aided translation. In *Proceedings of Avignon '94, 14th International Conference*, 1994. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.
- [Str90] Oliver Streiter. Construction d'un système de traduction automatique néerlandais → allemand dans le formalisme CAT2. Rapport de stage, Jussieu, Université de Paris VII, Paris, 1990.
- [Str94] Oliver Streiter. Komplexe Disjunktion und Erweiterter Kopf: Ein Kontrollmechanismus für die MÜ. In *KONVENS '94*, 1994. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.
- [SW91] Antje Schmidt-Wigger. Traitement du grec moderne. Rapport de stage, Jussieu, Université de Paris VII, Paris, 1991.