AUTOMATIC TRANSLATION WITH ATTRIBUTE GRAMMARS

Werner Dilger
University of Kaiserslautern
Computer Science Department
D-6750 Kaiserslautern
Federal Republic of Germany

Summary

Starting from an ATN-grammar and translation rules assigning expressions of a predicate calculus language to the symbols of the grammar one can produce an attribute grammar for the translation of natural language sentences (here German) into expressions of the predicate calculus language. The paper illustrates that this can be done in a straightforward way and outlines further improvements of the resulting attribute grammar.

Introduction

An important component of the natural language information system PLIDIS, developped by my colleagues and myself at the Institut fuer deutsche Sprache in Mannheim (cf. [BW 78], [KL 79]), is the translation algorithm, which transduces natural language sentences into expressions of an augmented first order predicate calculus, called KS (cf. [DZ 78],[Zi 77]). Special features of KS going beyond ordinary predicate calculus are many-sorted domain of objects, λ-abstraction, and complex term-building facilities. The examples contained in this paper will illustrate these features. Input for the translation algorithm are the parsed sentence and a set of translation rules (in the following: TR-rules) (cf. [Wu 79]), which are defined for labels of the parse tree nodes, mainly for the labels of the terminal nodes, i. e. for the words of the input sentence. Working bottom up the parse tree the translation algorithm assigns a translation to each of the nodes of the tree by interpreting the TR-rules defined for the labels of the nodes. If a translation has been successfully assigned to the root of the tree, which is labelled by S, this translation is the translation of the whole sentence.

The advantage of the translation algorithm, the most important part of which is the interpretation of the TR-rules, is its rather simple structure, which facilitated implementation. But it also has several disadvantages. First of all the algorithm is not very efficient since it runs separately from parsing, i. e. it does not start before parsing has finished. The TR-rules must take care of the structure of the parse tree, that means, during their interpretation we must check which steps were made in parsing some relevant part of the tree. Next, the TR-rule for the label of a node must be completely evaluated, though it depends on the position of the node in the tree, which parts of the rule apply to the node or whether the rule applies as a whole. Finally it is difficult to detect circularities in the translation process on the basis of the TR-rules.

To avoid these disadvantages we can use attribute grammars for the translation. The content of the TR-rules must then be represented by attributes and semantic functions. But for this purpose we need a context free grammar as a basis of the parsing. In PLIDIS, however, we have no such grammar, parsing is done by means of an ATN-grammar (cf. [Wo 70], [Wo 73], [Ba 78], [KL 79]), adapted for German. Though the networks of the ATN-grammar are not context free productions, we can produce such productions out of them. At first glance, by doing so, the context-sensitivity of the networks - which is their main advantage - is lost. But we can regain it by providing the productions with appropriate attributes and semantic functions. If we take a simpler version of the ATN-grammar, namely the RTN-grammar ("recursive transition networks") (cf. [Wo 70]), then an ATN-grammar is nothing else but an attributed RTN-grammar; so we could read the letters "ATN" as "attributed transition networks" instead of "augmented transition networks". In the remainder of the paper we omit the attributes needed to express context conditions, we only deal with those needed for translation.

To summarize, we have to show how to obtain context free productions from the networks and attributes and semantic functions from the TR-rules. We will demonstrate by examples that the method is straightforward, and we will outline how the resulting attribute grammar can be improved.

ATN-grammars and TR-rules

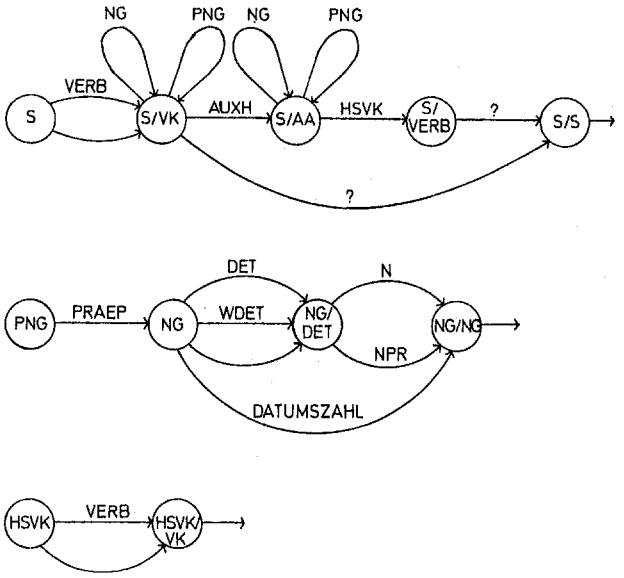
We want to parse the following questions asking for facts of the PLIDIS mini-world, i. e. the control of water pollution:

> Enthielt eine Probe im Jahr 1979 in Stuttgart Arsen?
> Did a sample in 1979 in Stuttgart contain arsenic?
>
> Welche Betriebe in Stuttgart hat Zimpel im Jahr 1979 geprüft?
> Which plants in Stuttgart Zimpel has inspected in 1979?
>
> Welche Betriebe hat Zimpel im Jahr 1979 in Stuttgart geprüft?
> Which plants Zimpel has inspected in 1979 in Stuttgart?

Welche Betriebe der Firma Lauxmann in Stuttgart hat Zimpel geprüft?
Which plants of the firm Lauxmann in Stuttgart Zimpel has inspected?

Welcher Probenehmer hat bei der Firma Lauxmann im Jahr 1979 Proben gezogen?
Which sampler has taken a sample from the firm Lauxmann in 1979?

We will give a small section of the ATN-grammar used within PLIDIS, reduced to an RTN-grammar, which allows the parsing of these questions. The networks and word classes of fig. 1 are needed for this purpose.



AUXH = {hab}
have

DET = {der, die, das, ein, eine}
the, a

N = {Probe, Probenehmer, Jahr, Betrieb, Firma, Arsen}
sample, sampler, year, plant, firm, arsenic

NPR = {Stuttgart, Zimpel, Lauxmann}

PRAEP = {in, bei}
in, at

VERB = {enthalt, prüf, zieh}
contain, inspect, take

WDET = {welch}
which

Figure 1

We obtain by means of this grammar for the question

Welche Betriebe in Stuttgart hat Zimpel im Jahr 1979 geprüft?
Which plants in Stuttgart Zimpel has inspected in 1979?

the parse tree of fig. 2.

We will now give TR-rules for the labels of some of the nodes occurring in fig. 2, using small

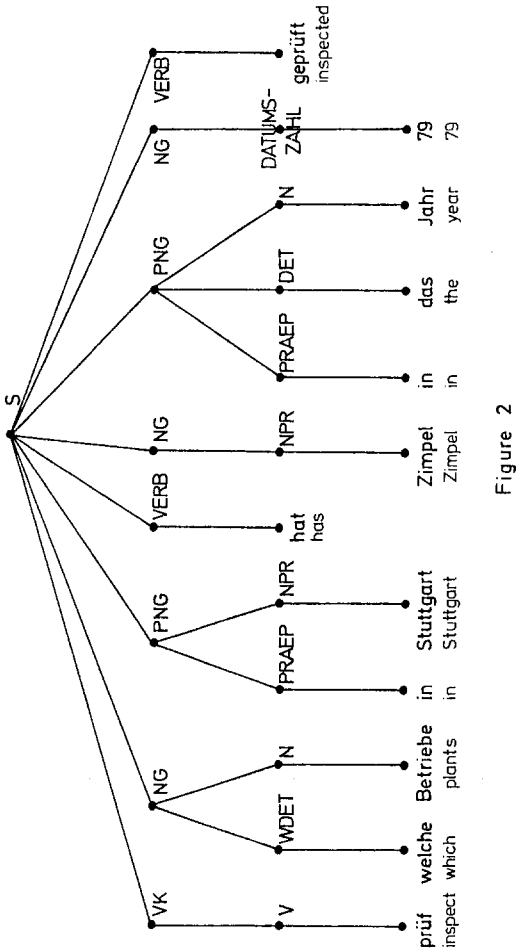
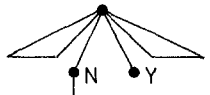
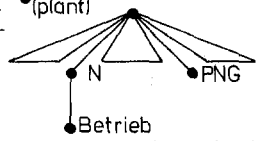


Figure 2

diagrams which represent sections of possible parse trees. The triangles in these diagrams denote arbitrary (perhaps empty) subtrees. Because the TR-rules are defined for single symbols of the grammar regardless of their occurrences in parse trees, all possible natural language occurrences of the symbols must be described and dealt with in the TR-rules. Therefore in the following TR-rules not all conditions will be immediately clear. But that doesn't matter since we are not interested in the details of the TR-rules, rather we will show below how the TR-rules can be transformed into expressions containing attributes and semantic functions.

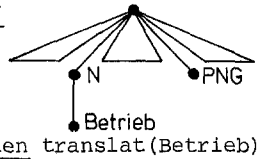
Betrieb (plant):

if  and sort(translat(Y)) = INDUSTRIE

then if  and sort(translat (PNG)) = ORT

then translat(Betrieb) = [LAMBDA X.BETR [BETRIEB translat(Y) translat(PNG) X.BETR]]

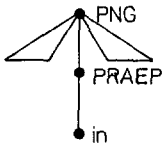
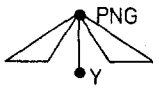
else translat(Betrieb) = [LAMBDA X.BETR [BETRIEB translat(Y) X.ORT X.BETR]]

else if  and sort(translat (PNG)) = ORT

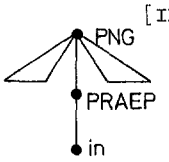
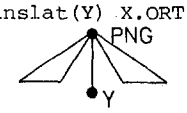
then translat(Betrieb) = [LAMBDA X.BETR [BETRIEB X.INDUSTRIE translat(PNG) X.BETR]]

else translat(Betrieb) = [LAMBDA X.BETR [BETRIEB X.INDUSTRIE X.ORT X.BETR]]

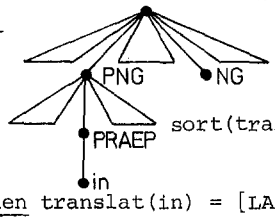
in:

if  and  and Y ∈ {N,NPR} and sort(translat (Y)) = ORT

then translat(in) = [LAMBDA X.ORT [IN translat(Y) X.ORT]]

else if  and  and sort(translat(Y)) ≦ INT

then translat(in) = [LAMBDA X.INT[INTEMP translat(Y) X.INT]]

else if  and sort(translat(NG))≦INT

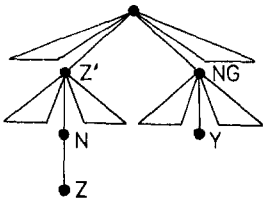
then translat(in) = [LAMBDA X.INT [INTEMP translat(NG) X.INT]]

else translat(in) = ω

Stuttgart:

translat(Stuttgart) = STUTTGART

N:

if  and Z' ∈ {NG,PNG} and

((Z ∈ {Monat,Jahr} and cat(translat(Y)) = KONST and sort(translat(Y)) = INT)
or(Y ∈ {N,NPR} and cat(translat(Y)) = TERM and cat(translat(Z)) = LTERM and sort(translat(Y)) = sort(translat(Z))))

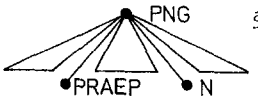
then translat(N) = ω

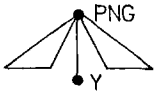
The else-part here is assumed to be substituted by the general rule prescribing that whenever a TR-rule does not apply, the translation of the node will be the translation of one of its daughter nodes, e.g. here we could write:

else translat(N) = translat(Z)

From the TR-rule for PNG we will only give some part:

PNG:

if  and cat(translat(N)) ≠ KONST

then if  and Y ∈ {DET,QDET,WDET,NEG-DET,ZAHL} and cat(translat(Y)) = QUANT

then if cat(translat(PRAEP)) = LTERM
then translat(PNG) = [translat(Y) translat(PRAEP)]
else translat(PNG) = [translat(Y) translat(N)]
else if cat(translat(PRAEP)) = LTERM
then translat(PNG) = [[QUANT EIN] translat(PRAEP)]
else translat(PNG) = [[QUANT EIN] translat(N)]

As with the rule for N the else-part is omitted here too.

Applying these TR-rules to the section of the parse tree of fig. 2 represented in fig. 3, we get as translation of this section:

```
[LAMBDA X.BETR
[BETRIEB X.INDUSTRIE
        [LAMBDA X.ORT[IN STUTTGART X.ORT]]
        X.BETR]]
```

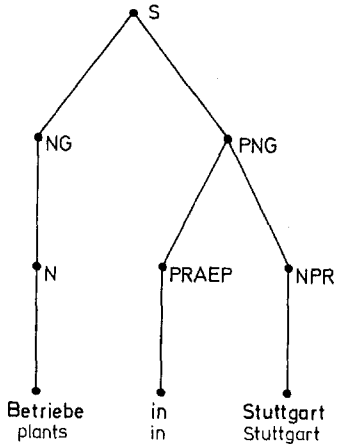


Figure 3

Transformation of the networks into context free productions

The first step is to produce sets of productions by means of the inscriptions of the nodes and edges, the right hand sides of which consist of at most two symbols.

```
S → VERB S/VK | S/VK
S/VK → NG S/VK | PNG S/VK | AUXH S/AA | ? S/S
S/AA → NG S/AA | PNG S/AA | HSVK S/VERB
S/VERB → ? S/S
S/S → ε

PNG → PRAEP NG
NG → DET NG/DET | WDET NG/DET | NG/DET |
     DATUMSZAHL NG/NG
NG/DET → N NG/NG | NPR NG/NG
NG/NG → ε

HSVK → VERB HSVK/VK | HSVK/VK
HSVK/VK → ε
```

In the next step the sets of productions are combined for each network to a single production the right hand side of which consists of a regular expression. The set for PNG is duplicated, omitting PRAEP, to get a separate production for NG.

```
S → (VERB|ε)(NG|PNG)*(AUXH(NG|PNG)*HSVK ? | ?)
PNG → PRAEP((DET|WDET|ε)(N|NPR)|DATUMSZAHL)
NG → (DET|WDET|ε)(N|NPR)|DATUMSZAHL
HSVK → VERB|ε
```

Now these productions are transformed step by step into "disjunctive normal form", where sequencing (represented by iuxtaposition) corresponds to the logical "and", | corresponds to the logical "or".

```
S → (VERB|ε)(NG*PNG*)*(AUXH(NG*PNG*)*HSVK ? | ?)
S → (VERB|ε)(NG*PNG*)*AUXH(NG*PNG*)*HSVK ? |
    (VERB|ε)(NG*PNG*)*?
S → VERB(NG*PNG*)*AUXH(NG*PNG*)*HSVK ? |
    (NG*PNG*)*AUXH(NG*PNG*)*HSVK ? |
    VERB(NG*PNG*)*? |
    (NG*PNG*)*?

PNG → PRAEP(DET|WDET|ε)(N|NPR) |
      PRAEP DATUMSZAHL

PNG → PRAEP DET N | PRAEP WDET N | PRAEP N |
      PRAEP DET NPR | PRAEP WDET NPR |
      PRAEP NPR | PRAEP DATUMSZAHL

NG → DET N | WDET N | N | DET NPR | WDET NPR |
     NPR | DATUMSZAHL
```

Finally those parts provided with * are removed introducing new symbols and productions.

```
S → VERB NG/PNG AUXH NG/PNG HSVK ? |
    NG/PNG AUXH NG/PNG HSVK ? |
    VERB NG/PNG ? | NG/PNG ?

NG/PNG → NG NG/PNG | PNG NG/PNG | NG | PNG | ε
```

If we form sets of productions out of the word classes, we get altogether:

```
S → VERB NG/PNG AUXH NG/PNG HSVK ? |
    NG/PNG AUXH NG/PNG HSVK ? |
    VERB NG/PNG ? | NG/PNG ?

NG/PNG → NG NG/PNG | PNG NG/PNG | NG | PNG | ε

PNG → PRAEP DET N | PRAEP WDET N | PRAEP N |
      PRAEP DET NPR | PRAEP WDET NPR |
      PRAEP NPR | PRAEP DATUMSZAHL

NG → DET N | WDET N | N | DET NPR | WDET NPR |
     NPR | DATUMSZAHL

HSVK → VERB | ε

AUXH → hab

DET → der | die | das | ein | eine

N → Probe | Probenehmer | Jahr | Betrieb |
    Firma | Arsen

NPR → Stuttgart | Zimpel | Lauxmann

PRAEP → in | bei

VERB → enthalt | prüf | zieh

WDET → welch
```

The parsing of our question example by means of these productions yields the parse tree of fig. 4. The section of this tree corresponding to that of fig. 3 is represented in fig. 5.

Providing the productions with attributes

We will now give a list of attributes and semantic functions for the productions and augment the productions by them such that the evaluation of the semantic functions yields the translation of the sentence. We will do this only for those productions needed for the section of fig. 5.

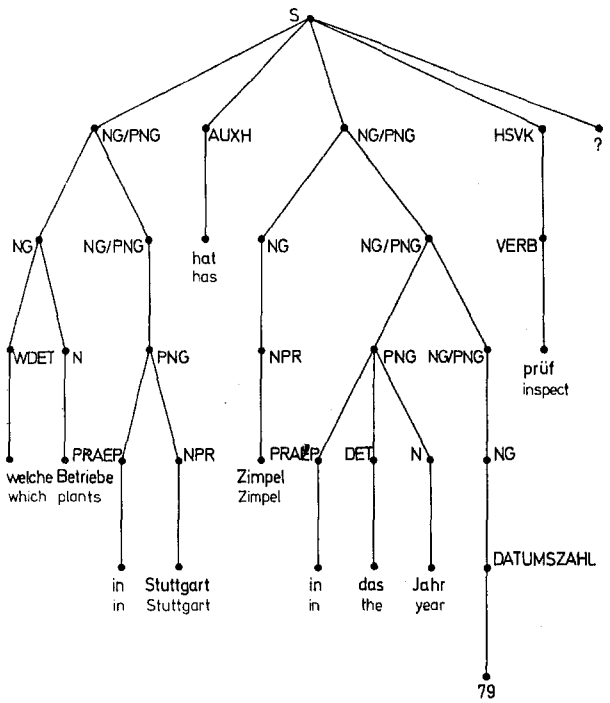


Figure 4

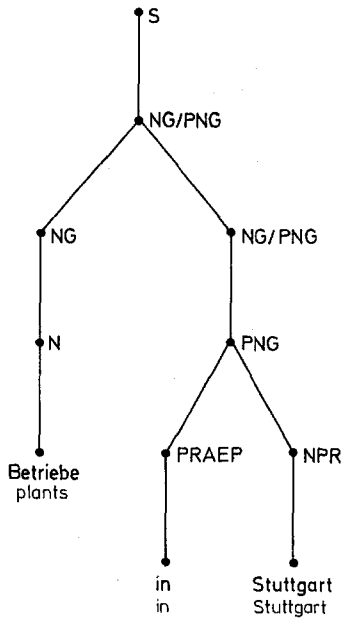


Figure 5

Attributes

name	class	domain
↑val	synthesized	n-tuple of KS-expressions (translations), in general n = 1
↑tree	synthesized	sets of triples, consisting of the position of a symbol in the parse tree, the symbol itself, and the value of the symbol
↓tree	inherited	same as with ↑tree
↓pos	inherited	finite sequences of positive integers, separated by dots

Semantic functions

name	mnemotechnic	argument	use
cat	category	translation t	yields the KS-syntactic category of t
sort	sort	translation t	yields the sort of t
symb	symbol	position p	yields from ↓tree the symbol of the node with position p
nth	n-th (=last) element	position p	yields the last integer of p
value	value	position p	yields from ↓tree the value of the node with position p
del	delete	position p	replaces in ↓tree the value of the node with position p by ω

↑val is the most important attribute, for it contains the translation of a node. The other attributes are auxiliary attributes. ↓tree contains in each node a relevant section of the parse tree with all necessary informations about the nodes of that section, namely their labels and their values. Already Knuth ([Kn 68]) has given a technique for representing the attribute values of all other nodes at each node of the tree. We adopt this technique here in a slightly modified way since it offers an elegant way to rewrite the conditions of the TR-rules as expressions containing semantic functions and attributes for appropriate productions. We will illustrate how this technique works, using the structure of fig. 5. The only information we are interested in for this example are the labels of the nodes. Let

$$X_0 \rightarrow X_1\ X_2\ \ldots\ X_n \quad (n \geq 0)$$

be a production, where the X_i $(i = 1,\ldots,n)$ are terminal or nonterminal symbols. If n=0, X_0 is terminal. Then:

$$\downarrow pos(X_i) = \begin{cases} \downarrow pos(X_o)\bullet i, & \text{if } X_o \neq S \\ i, & \text{if } X_o = S \end{cases}$$

$$\downarrow tree(X_i) = \begin{cases} \downarrow tree(X_o), & \text{if } X_o \neq S \\ \uparrow tree(X_i), & \text{if } X_o = S \end{cases}$$

$$\uparrow tree(X_o) = \begin{cases} \{(\downarrow pos(X_o), X_o)\} \cup \bigcup_{i=1}^{n} \uparrow tree(X_i), & \text{if } X_o \neq S \\ \bigcup_{i=1}^{n} \uparrow tree(X_i), & \text{if } X_o = S \end{cases}$$

We can easily obtain the ↓pos-value for each node of fig. 5 beginning with the NG/PNG-node which is dominated by the S-node and for which we assume: ↓pos(NG/PNG) = k. The ↓pos-values are given in fig. 6. Using these values, we obtain e.g.

↑tree(PNG) = {(k•2•1,PNG),(k•2•1•1,PRAEP),
(k•2•1•1•1,in),(k•2•1•2,NPR),
(k•2•1•2•1,Stuttgart)}

The ↓tree-value for NG/PNG (and thus for all other nodes) is

↓tree(NG/PNG) = {(k,NG/PNG),(k•1,NG),(k•1•1,N),
(k•1•1•1,Betriebe),(k•2,NG/PNG),
(k•2•1,PNG),(k•2•1•1,PRAEP),
(k•2•1•1•1,in),(k•2•1•2,NPR),
(k•2•1•2•1,Stuttgart)}

In order to obtain the values of attributes defined for the productions it is often necessary to determine a new position starting from a given one. For this purpose some of the integers at the end of the position must be omitted or others must be appended. If the last integer of the position belonging to symbol X shall be omitted we write

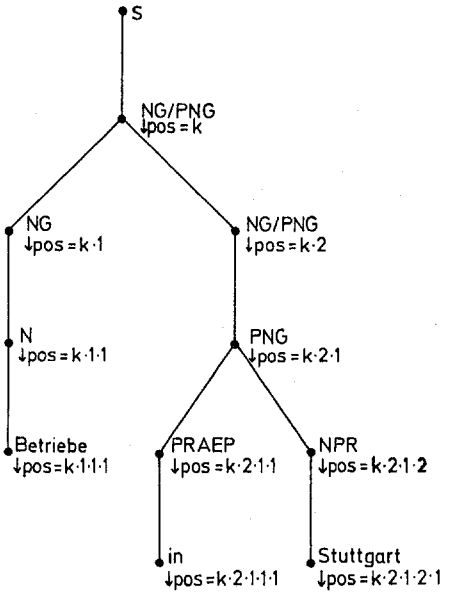


Figure 6

↓pos(X)-1

If the last two integers shall be omitted, we write

↓pos(X)-2

etc. If an integer, say k, shall be appended j times, we write

↓pos(X)•k^j

Now we are ready to give the productions needed for the structure of fig. 5, provided with attributes and semantic functions.

N → Betrieb

```
if symb(↓pos(N)-1)∈{NG,PNG} ∧
   symb((↓pos(N)-2)•2•1)=NG ∧
   ((symb(↓pos(N)•1)∈{Monat,Jahr)} ∧
     ∃j>0: cat(value((↓pos(N)-2)•2•1•j))=KONST ∧
           sort(value((↓pos(N)-2)•2•1•j))=INT) ∨
    (∃j>0: symb((↓pos(N)-2)•2•1•j)∈{N,NPR} ∧
           cat(value((↓pos(N)-2)•2•1•j))=TERM ∧
           cat(value(↓pos(N)•1))=LTERM ∧
           sort(value(↓pos(N)•1))=
                sort(value((↓pos(N)-2)•2•1•j))))
then ↑val(N) = ω
else if sort(value((↓pos(N)-1)•2•1))=INDUSTRIE
     then if ∃j>0: symb((↓pos(N)-1)•2^j•1)=PNG
                   sort(value((↓pos(N)-1)•2^j•1))
                                        =ORT
          then ↑val(N) = [LAMBDA X.BETR[BETRIEB
                          value((↓pos(N)-1)•2•1)
                          value((↓pos(N)-1)•2^j•1)
                          X.BETR]]
               del((↓pos(N)-1)•2•1)
               del((↓pos(N)-1)•2^j•1)
          else ↑val(N) = [LAMBDA X.BETR[BETRIEB
                          value((↓pos(N)-1)•2•1)
                          X.ORT X.BETR]]
               del((↓pos(N)-1)•2•1)
     else if ∃j>0: symb((↓pos(N)-1)•2^j•1)=PNG
                   sort(value((↓pos(N)-1)•2^j•1))
                                        =ORT
          then ↑val(N) = [LAMBDA X.BETR[BETRIEB
                          X.INDUSTRIE
                          value((↓pos(N)-1)•2^j•1)
                          X.BETR]]
               del((↓pos(N)-1)•2^j•1)
          else ↑val(N) = [LAMBDA X.BETR[BETRIEB
                          X.INDUSTRIE X.ORT
                          X.BETR]]
↑tree(N) = {(↓pos(N),N,↑val(N)),
            (↓pos(Betrieb),Betrieb,ω)}
↓pos(Betrieb) = ↓pos(N)•1
```

The first part of the expression needed to determine the value of ↑val(N) comes from the TR-rule for N, the second part from the TR-rule for Betrieb (plant).

For the symbol NG there is a TR-rule too. We omit it here, because it does not apply to our example. Therefore we deal with the production NG → N in such a way, as if there were no TR-rule for NG.

```
NG → N

if value(↓pos(N)) = ω
then ↑val(NG) = ω
else ↑val(NG) = ↑val(N)
↑tree(NG) = {(↓pos(NG),NG,↑val(NG))} ∪ ↑tree(N)
↓tree(N) = ↓tree(NG)
↓pos(N) = ↓pos(NG)•1

PRAEP → in

if symb(↓pos(PRAEP)-1) = PNG
then if ∃j≠nth(↓pos(PRAEP)):
          symb((↓pos(PRAEP)-1)•j)∈{N,NPR} ∧
          sort(value((↓pos(PRAEP)-1)•j))=ORT
     then ↑val(PRAEP) = [LAMBDA X.ORT[IN
                         value((↓pos(PRAEP)-1)•j)
                                        X.ORT]]
          del((↓pos(PRAEP)-1)•j)
     else if ∃j≠nth(↓pos(PRAEP)):
               sort(value((↓pos(PRAEP)-1)•j))
                                        ≤ INT
          then ↑val(PRAEP) = [LAMBDA X.INT
                 [INTEMP
                    value((↓pos(PRAEP)-1)•j)
                    X.INT]]
               del((↓pos(PRAEP)-1)•j)
          else if ∃j>0: symb((↓pos(PRAEP)-2)
                                   •2^j•1) = NG
                        sort(value((↓pos(PRAEP)
                        -2)•2^j•1)) ≤ INT
               then ↑val(PRAEP) = [LAMBDA X.INT
                      [INTEMP value((↓pos(PRAEP)
                        -2)•2^j•1)  X.INT]]
                    del((↓pos(PRAEP)-2)•2^j•1)
               else ↑val(PRAEP) = ω
else ↑val(PRAEP) = ω
↑tree(PRAEP) = {(↓pos(PRAEP),PRAEP,↑val(PRAEP)),
                (↓pos(in),in,ω)}
↓pos(in) = ↓pos(PRAEP)•1

NPR → Stuttgart

↑val(NPR) = STUTTGART
↑tree(NPR) = {(↓pos(NPR),NPR,↑val(NPR)),
              (↓pos(Stuttgart),Stuttgart,ω)}
↓pos(Stuttgart) = ↓pos(NPR)•1

PNG → PRAEP NPR

if ∃j>0: symb(↓pos(PNG)•j) = N
         cat(value(↓pos(PNG)•j)) ≠ KONST
then if ∃i>0: symb(↓pos(PNG)•i)∈{DET,QDET,WDET,
                                 NEGDET,ZAHL}
              cat(value(↓pos(PNG)•i)) = QUANT
     then if cat(↑val(PRAEP)) = LTERM
          then ↑val(PNG) = [value(↓pos(PNG)•i)
                            ↑val(PRAEP)]
               del(↓pos(PNG)•i)
               del(↓pos(PRAEP))
          else ↑val(PNG) = [value(↓pos(PNG)•i)
                            value(↓pos(PNG)•j)]
               del(↓pos(PNG)•i)
               del(↓pos(PNG)•j)
     else if cat(↑val(PRAEP)) = LTERM
          then ↑val(PNG) = [EIN ↑val(PRAEP)]
               del(↓pos(PRAEP))
          else ↑val(PNG) = [EIN value(↓pos(PNG)
                                        •j)]
               del(↓pos(PNG)•j)
else if ↑val(PRAEP) = ω
     then ↑val(PNG) = ↑val(NPR)
     else ↑val(PNG) = ↑val(PRAEP)
↑tree(PNG) = {(↓pos(PNG),PNG,↑val(PNG))}
             ∪ ↑tree(PRAEP) ∪ ↑tree(NPR)
↓tree(PRAEP) = ↓tree(PNG)
↓tree(NPR) = ↓tree(PNG)
↓pos(PRAEP) = ↓pos(PNG)•1
↓pos(NPR) = ↓pos(PNG)•2

NG/PNG → PNG

if value(↓pos(PNG)) = ω
then ↑val(NG/PNG) = ω
else ↑val(NG/PNG) = ↑val(PNG)
↑tree(NG/PNG) = {(↓pos(NG/PNG),NG/PNG,
                  ↑val(NG/PNG))} ∪ ↑tree(PNG)
↓tree(PNG) = ↓tree(NG/PNG)
↓pos(PNG) = ↓pos(NG/PNG)•1

NG/PNG_1 → NG NG/PNG_2

if value(↓pos(NG)) = ω
then ↑val(NG/PNG_1) = ↑val(NG/PNG_2)
else if value(↓pos(NG/PNG_2)) = ω
     then ↑val(NG/PNG_1) = ↑val(NG)
     else ↑val(NG/PNG_1) = (↑val(NG),
                            ↑val(NG/PNG_2))
↑tree(NG/PNG_1) = {(↓pos(NG/PNG_1),NG/PNG,
                    ↑val(NG/PNG_1))} ∪ ↑tree(NG)
                       ∪ ↑tree(NG/PNG_2)
↓tree(NG) = ↓tree(NG/PNG_1)
↓tree(NG/PNG_2) = ↓tree(NG/PNG_1)
↓pos(NG) = ↓pos(NG/PNG_1)•1
↓pos(NG/PNG_2) = ↓pos(NG/PNG_1)•2
```

With these productions we obtain immediately the value of the ↓tree-attribute for each node of the structure of fig. 5 or 6, when we postpone the evaluation of the ↑val-attribute. The value is

```
{(k,NG/PNG,↑val(NG/PNG)),(k•1,NG,↑val(NG)),
 (k•1•1,N,↑val(N)),(k•1•1•1,Betriebe,ω),(k•2,
 NG/PNG,↑val(NG/PNG)),(k•2•1,PNG,↑val(PNG)),
 (k•2•1•1,PRAEP,↑val(PRAEP)),(k•2•1•1•1,in,ω),
 (k•2•1•2,NPR,↑val(NPR)),(k•2•1•2•1,Stuttgart,ω)}
```

The production NPR → Stuttgart yields

```
↑val(NPR) = STUTTGART
```

We can substitute this value in ↓tree or regard "↑val(NPR)" as a pointer to this value. Now we try to determine ↑val(PRAEP) from the production PRAEP → in. First we have

```
symb(↓pos(PRAEP)-1) = symb(k•2•1•1 - 1)
                    = symb(k•2•1) = PNG
```

That is, the first condition holds. Next

```
nth(↓pos(PRAEP)) = 1
```

therefore j>1. Assume j=2. Then

```
symb((↓pos(PRAEP)-1)•j) = symb((k•2•1•1 - 1)•2)
                        = symb(k•2•1•2) = NPR
```

Further

sort(value((↓pos(PRAEP)-1)•2))
= sort(value(k•2•1•2)) = sort(STUTTGART)
= ORT

The second condition holds too, thus we get

↑val(PRAEP) = [LAMBDA X.ORT[IN STUTTGART X.ORT]]

Within the production PNG → PRAEP NPR the first condition needed to determine ↑val(PNG) does not hold, so we get

↑val(PNG) = ↑val(PRAEP)

If we assume these values to be substituted in ↑tree, we now have the intermediate result

{(k,NG/PNG,↑val(NG/PNG)),(k•1,NG,↑val(NG)),
(k•1•1,N,↑val(N)),(k•1•1•1,Betriebe,ω),
(k•2,NG/PNG,↑val(NG/PNG)),(k•2•1,PNG,
[LAMBDA X.ORT[IN STUTTGART X.ORT]]),(k•2•1•1,
PRAEP,[LAMBDA X.ORT[IN STUTTGART X.ORT]]),
(k•2•1•1•1,in,ω),(k•2•1•2,NPR,ω),(k•2•1•2•1,
Stuttgart,ω)}

It is left to the raeder to compute the final result applying the remainding productions.

Conclusion

We have illustrated how an attribute grammar can be produced from the networks and TR-rules used within PLIDIS, which has the same expressive power as the underlying networks and rules. The advantages of the ATN-grammars for the parsing of natural language sentences are well known. Above all they are an elegant tool to write grammars especially suited for linguists. The TR-rules have advantages similar to these. Surely they are easier to write than the somewhat cumbersome expressions for determining attribute values, particularly those for the ↑val-attribute. In the TR-rules, however, attempt is made to describe and deal with all possible occurrences of a symbol. With the attributed productions this is not necessary, since for a single production some of the cases which stem from the possible occurrences of one or more symbols can be omitted a priori. For example, in the production PNG → PRAEP NPR the whole first part for determining ↑val(PNG) can be omitted, because the first condition does not hold for this production. In a similar way we can omit some part in the production N → Betrieb. Further improvements can be made by changing the productions themselves, e. g. by partly eliminating those symbols which denote word classes. Performing all possible improvements certainly leads to an attribute grammar which yields translations of sentences in a rather efficient way. On the basis of this grammar we can detect circularities which can occur in the translation process by means of well defined algorithms (cf. [Bo 76]).

References

[Ba 78] M. Bates, The theory and practice of augmented transition network grammars
in: L. Bolc (ed.), Natural language communication with computers
Springer Lecture Notes in Computer Science, 63, 191-259, Berlin 1978

[BW 78] G.L. Berry-Rogghe/H. Wulz, An overview of PLIDIS, a problem solving information system with German as query language
in: L. Bolc (ed.), Natural language communication with computers
Springer Lecture Notes in Computer Science, 63, 87-132, Berlin 1978

[Bo 76] G.V. Bochmann, Semantic evaluation from left to right
in: CACM 19(2), 1976, 55-62

[DZ 78] W. Dilger/G. Zifonun, The predicate calculus-language KS as query language
in: H. Gallaire/J. Minker (eds.), Logic and data bases
Plenum Press New York, 1978, 377-408

[Kn 68] D.E. Knuth, Semantics of context-free languages
in: Math. Systems Th. 2, 1968, 127-145
and Math. Systems Th. 5, 1971, 95-96

[Kn 71] D.E. Knuth, Examples of formal semantics
in: E. Engeler (ed.), Symposium on semantics of algorithmic languages
Springer Lecture Notes in Mathematics, 188, 212-235, Berlin 1971

[KL 79] M. Kolvenbach/A. Lötscher/H-D. Lutz (eds.) Künstliche Intelligenz und natürliche Sprache
Forschungsberichte des Instituts für deutsche Sprache, 42,
G. Narr-Verlag Tübingen, 1979

[Wo 70] W.A. Woods, Transition network grammars for natural language analysis
in: CACM 13, 1970, 591-606

[Wo 73] W.A. Woods, An experimental parsing system for transition network grammars
in: R. Rustin (ed.), Natural language processing
Algorithmic Press New York, 1973, 112-154

[Wu 79] H. Wulz, Formalismen einer Übersetzungsgrammatik
Forschungsberichte des Instituts für deutsche Sprache, 46
G. Narr-Verlag Tübingen, 1979

[Zi 77] G. Zifonun, Die Konstruktsprache KS
in: K. Heger/J. Petöfi (eds.), Kasustheorie, Klassifikation, semantische Interpretation
Papiere zur Textlinguistik 11, Hamburg 1977