Affix Grammars for Natural Languages

C.H.A. Koster*
Department of Informatics, University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract

Affix Grammars over a Finite Lattice (AGFLs), a simple form of two-level grammars admitting quite efficient implementations, are proposed as a formalism to express the syntax of natural languages. In this paper the concepts and notation of AGFLs are described. A brief example is given of their use in describing a fragment of the English language, followed by a discussion of issues like parsing and ambiguity.

1 Informatics and Linguistics

Informatics and Linguistics have a common interest in syntax. In the sixties of this century the emerging science of Informatics wholeheartedly embraced the theory of Formal Languages, which was already at that time available from Logics and Mathematical Linguistics.

In the next decades, informaticians have greatly enlarged and enriched Formal Language theory. They have devised efficient parsing algorithms and put the theory to useful work in the description of programming languages and the construction of compilers, software engineering environments, interactive user interfaces and other forms of syntax-directed software.

This whole body of theory and practice is available in its turn to modern Linguistics. Informatics can now pay back its historical debt to Linguistics — with interest.

1.1 On a division of labour

Linguists should not write programs. To begin with, they are ill-equipped and ill-educated for the profession of software developer. Furthermore, a fascination with programming tends to keep them from exercising their own speciality.

Linguists should write grammars rather than programs. Linguistic knowledge encapsulated in a program is like a corpse in a white-plastered grave. Even if the program works correctly and efficiently, it is doomed to disappear in the end, due to lack of maintenance. Nobody shall resuscitate the knowledge from the ashes of a C-program or LISP-program: we can be sure the programmer has reveled in the exploitation of clever tricks and could not be bothered to write documentation. On the other hand, a grammar is a "pure" description, without any operational considerations, which admits of many different implementations. A grammar can serve as the basis of many different pieces of

*Prepared while the author was visiting the Technical University of Budapest on a TEMPUS-grant.

software (analysers, spelling checkers, syntax- and style-checkers, natural language interfaces, translators). Since it captures only the essential, it may well be updated, improved and maintained by linguists over a long period of time, new versions of the software being generated as needed.

Turning a grammar for some language into a correct and efficient recognizer or parser is work for an informatician. Employing existing compiler construction methodology, also linguistic parsers can be generated automatically. Their correctness and efficiency is a worthy subject for Informatics research.

Conversely, informaticians should promise not to write grammars for natural languages, or at least not to pretend to any linguistic relevance of the grammars they do write — such as the examples in this paper.

1.2 Corpus Linguistics at Nijmegen University

The University of Nijmegen in The Netherlands (KUN) has an active group of corpus linguists, centered around the TOSCA-project [1, 8], with which I have the pleasure to cooperate.

Corpus Linguistics is that branch of Mathematical Linguistics that is concerned with the derivation of consistent and relatively complete grammars for natural languages, which are validated with respect to given corpora of text. Corpus linguists are the empirical scientists among grammarians. Rather than investigating isolated linguistic phenomena like the negation in Swahili or the expression of causality in Indo-European languages, they attempt to integrate the various theories regarding the structure of some particular language into a consistent whole. They are not afraid of formality and computers, and thus are excellent partners for collaboration with informaticians.

The group in Nijmegen has, after a large scale investigation of English using Context-Free Grammars, decided in the late seventies to adopt the formalism of Extended Affix Grammars (EAGs, see [15]) in projects describing a number of languages:

- English (J. Aarts and N. Oostdijk [17])
- Spanish (J. Hallebeek [5, 6])
- Modern Standard Arabic (E. Ditters [2]), and recently
- German (J. Aarts and J. Cloeren, KUN in collaboration with the University of Mainz).

Furthermore it should be mentioned that a group at Delft University [19] has been successfully using a related form of Affix Grammars in the description of Dutch.

In the course of the TOSCA project, a number of increasingly powerful parser generators have been used, which translate slightly restricted forms of EAG into equivalent nondeterministic parsers in machine code [15]. The EAG parser generators have been used rather successfully in these linguistic projects, in spite of the fact that they were primarily developed for applications in Informatics.

In applying the general EAG-system to linguistic applications, a number of problems were encountered:

- the large size of linguistic grammars

- the high degree of ambiguity found
- the need for a large and efficient lexicon, and
- the necessity to allow leftrecursion (which was until recently not supported by the EAG system).

In fact, these same problems are encountered when using PROLOG + DCGs in realistic linguistic applications.

On the other hand, sentences in a natural language tend to be much shorter than those in programming languages (programs!). Upon inspection of the grammars written, the use made by linguists of the EAG formalism turns out to be exceedingly simple, none of its costly features (like delayed affix evaluation and unification) being exploited.

All these experiences point to the fact that it is advantageous to implement a simpler formalism, sufficient for these applications, which can deal well with the problems mentioned.

2 Affix Grammars over a Finite Lattice

AGFLs are a formalization of a notion well known to linguists (see e.g. [11]): Context Free grammars are augmented with features for expressing agreement between parts of speech, where the features form a finite categorization. Such grammars have been used extensively in classical Linguistics (albeit in a non-formal form) for more than two thousand years.

Conjugation and declination rules based on feature distinctions can be modelled easily in an AGFL. Indeed the original motivation for Affix Grammars was linguistic, and their first application was a generative grammar for a small part of English [13], which was presented to the Euratom-colloquium organized by prof. E.W. Beth in 1962.

In this section, we give an informal description of AGFLs and introduce some notation and concepts.

An AGFL consists of meta-rules and rules. Their order is arbitrary.

2.1 The meta-rules

The *meta-rules* or *affix rules* are a collection of restricted Context Free rules, together forming the second level of the AGFL. Each affix rule defines the direct productions of a nonterminal affix. Such a direct production is either a terminal affix or a nonterminal affix, and recursion is not allowed. Consequently, a nonterminal affix has one or more terminal affixes as terminal productions. These must all be different.

Terminal affixes (or, rather, their representations) are written in small letters. Nonterminal affixes (or, rather, their names) are written with capital letters. Spaces may be used within nonterminal affixes to enhance readability.

Meta-rules can be recognized by the double colon separating their left- and right-hand-side. The meta-rule

```
NUMB :: singular; plural.
```

defines the nonterminal affix `NUMB` to have two direct productions. These two productions are terminal affixes and therefore also its terminal productions. Similarly,

```
NUMBER :: NUMB; dual.
```

defines NUMBER to have three terminal productions, of which it has two in common with NUMB.

When no confusion arises, we may use the word *affix* as a shorthand for *terminal affix* or *nonterminal affix*, as the context demands.

2.1.1 Domains

By the *domain* of a nonterminal affix we mean the set of its terminal productions. The meta-rules may be seen as a type-system, in which the nonterminal affixes, with their respective domains, are the types.

Since its domain is a finite enumeration of terminal affixes, any affix variable may be eliminated from a rule by systematically rewriting that rule into a number of rules, in each of which the affix variable is replaced by one of its terminal productions. By doing this for all affix variables, a CF grammar is obtained which is equivalent to the original AGFL.

Since an AGFL is therefore no more powerful than a CF grammar one might be tempted to use CF grammars instead. The CF grammar resulting from such an expansion may however be exceedingly large. The introduction of affixes in a CF grammar serves to shorten it considerably, making it possible to handle much more complicated grammars.

2.2 The rules

The *rules* of an AGFL are Context Free rules augmented with affixes. A rule consists of a left-hand-side, followed by a single colon, followed by a right-hand-side, e.g.

```
noun group (NUMB) :
  adjective, noun group (NUMB);
  subst (NUMB).
```

The left-hand-side of a rule consists of a *nonterminal symbol*, the *head*, optionally followed by a list of *affix expressions*, the *parameters*, enclosed between brackets.

The right-hand-side of a rule consists of one or more *alternatives*, separated from one another by semicolons. An alternative is a (possibly empty) list of *members*, separated by comma's. A member is either a *terminal symbol* or it is a *call*, which looks just like a left-hand-side. Nonterminal symbols can be written in small or large letters, spaces can be used to enhance readability. A terminal symbol is written as its representation enclosed between quotes.

A member that is optional is put between square brackets; given the rule

```
circumstance option :
  circumstance;  .
```

the member [circumstance] means the same as circumstance option.

An affix expression is either a nonterminal affix (which is then termed an *affix variable*), or it consists of one or more terminal affixes separated from one another by the *set union-operator* |.

All rules with a given nonterminal symbol as head together form the *definition* for that nonterminal. The rules that make up one definition may differ in *arity* (= number of parameters).

An example of a multi-rule definition involving union-operators is

```
to be (sing, 1) : "am".
to be (plur, 1) : "are".
to be (NUMB, 2 | 3) : "are".
```

2.2.1 Consistent substitution

AGFLs use a modified form of the *consistent substitution* rule which states that, in rewriting a rule, all occurrences of one same affix variable obtain the same value. In AGFL that value is the set of all terminal affixes for which rewriting succeeds. Thus according to

```
PERS :: 1; 2; 3.

simple sentence :
  pers pron (NUMB, PERS), to be (NUMB, PERS), adjective.

pers pron (sing, 1) : "I".
pers pron (plur, 1) : "we".
pers pron (NUMB, 2) : "you".
pers pron (sing, 3) : "he"; "she"; "it".
pers pron (plur, 3) : "they".

adjective : "great".
```

the sentence *you are great* has one parsing, with the affix `numb` assuming the value {`sing`, `plur`}.

Affix varables may be indexed with a number, to denote different instance of the same affix nonterminal; thus, `PERSON1` is another instance of `PERSON`, which is substituted for independently.

2.2.2 The lattice structure

The possible values of an affix variable form a mathematical object called a lattice. The lattice for the affix defined by

```
PERSON :: first; second; third.
```

can be depicted as in fig. 1.

The *top*-element $\top$ of the lattice over three elements in fig. 1 can be seen as the union of all possibilities (the value may be `first` or `second` or `third` or any combination). As we obtain more information, the number of possibilities may be narrowed down to a particular value, or even further to the *bottom*-element $\bot$, which indicates inconsistency. We will denote the top-element by the affix *PERSON* itself. By the introduction of suitable affix rules, any point in the lattice can be given a nonterminal affix as name, e.g.

```
WE :: first; second.
```

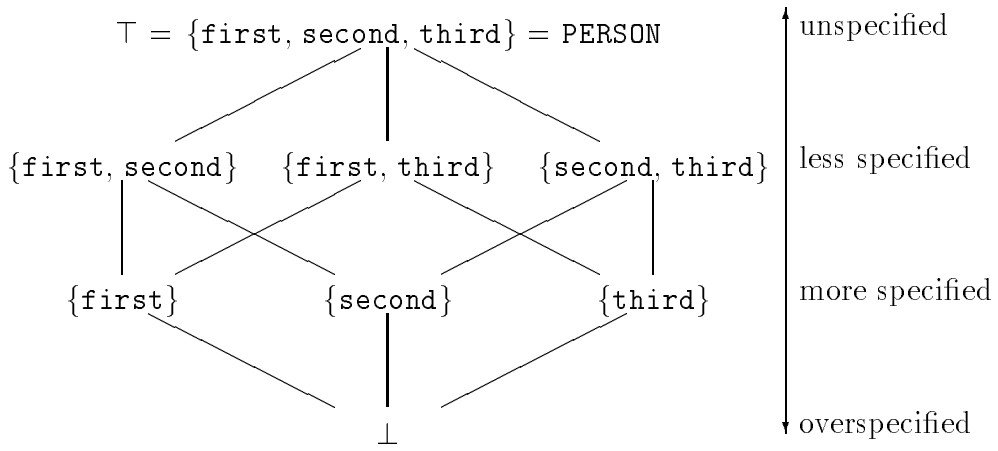

Figure 1: The lattice of values for the affix PERSON

can be used to give the name WE to the point marked {first, second}.

This particular kind of lattice over a set, with the set-union and set-intersection as operations, is called the *powerset lattice* of the set-valued affix.

Another form of lattice which we admit is the *flat lattice* of (bounded) integers, without any further operations. As an example, an enumeration of all verbs in a language, without any further operations, can be seen as a flat lattice, as in fig. 2. Apart from finite lattices,

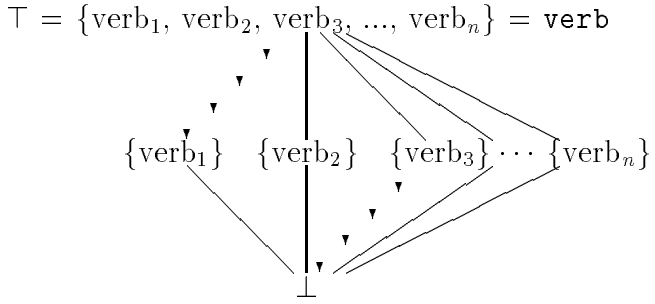

Figure 2: A flat lattice

with the union and intersection as operations, it is possible to introduce Affix Grammars over infinite lattices (see e.g. [16]). In AGFL we restrict ourselves to finite lattices, obtaining a formalism which admits of highly efficient implementation.

3 An example

Before discussing in the next sections some finer points of the application of AGFLs to natural languages, we shall first, by way of example, introduce a grammar for a fragment of the English language. It should be stressed that this grammar is not meant to have any linguistic merit. It merely serves to clarify the notation and concepts described and to provide some material for the following discussion.

We start by choosing a metasyntax, which is small and quite conventional, not to say classical.

```
NUMB :: sing; plur.

PERS :: 1; 2; 3.
```

As good latinists we shall distinguish four cases:

```
CASE :: nom; gen; dat; acc.
```

Verbs can be either transitive or intransitive:

```
TRAN :: trav; intv.
```

We intend to deal with the fact that some verbs have one or more associated preferred prepositions (*to look* at, *to look* for, *to shout* at, ...) and therefore introduce an affix enumerating such prepositions.

```
PREP :: none; to; from; at; in.
```

Now we come to the syntax. We shall describe only one type of English sentence, which displays the basic order `subject-verb-object` or `subject-verb-complement-object`, abusing the term `complement` for the prepositional object found with many verbs.

```
SVCO phrase :
  [circumstance], subject (NUMB, PERS), VCO phrase (NUMB, PERS).
```

The first member, enclosed between square brackets, is an optional `circumstance` (an indication of time or place). The other two members have to agree in number and person.

```
subject (NUMB, PERS):
  pers pron (NUMB, PERS, nom);
  noun phrase (NUMB, PERS, nom).

noun phrase (NUMB, PERS, CASE):
  noun phrase (NUMB, PERS, CASE), rel phrase (NUMB, PERS, CASE1).
noun phrase (NUMB, 3, CASE):
  noun part (NUMB), [modifier].
noun phrase (plur, 3, CASE):
  noun phrase (NUMB, PERS, CASE),
     coordinator, noun phrase (NUMB1, PERS1, CASE).
```

These three rules have the head `noun phrase` in common. The first one indicates that a subject may have (through recursion) any number of relative phrases. Notice the treatment of coordinated noun phrases like *Tricky Dicky and the Cool Cats*.

```
noun part (NUMB):
  noun group (NUMB);
  determiner (NUMB), noun group (NUMB);
  poss pron, noun group (NUMB).

noun group (NUMB):
  adjective, noun group (NUMB);
  subst (NUMB).
```

Now we come to the pièce de résistance of this fragmentary grammar.

```
rel phrase (NUMB, PERS, nom):
  rel pron (nom), VCO phrase (NUMB, PERS).
rel phrase (NUMB, PERS, gen):
  rel pron (gen), oSVC phrase.
rel phrase (NUMB, PERS, dat):
  preposition (prep), rel pron (dat), SVO phrase (prep);
  rel pron (dat), SVO phrase (prep);
  SVO phrase (prep).
rel phrase (NUMB, PERS, acc):
  rel pron (acc), SVC phrase (trav);
  SVC phrase (trav).
```

Somehow this looks more like Latin than like English, but it is not totally unfounded. Let me give one example for each production:

the man who owes me a dollar
the man whose dollar I stole
the man to whom I must give the money
the man whom I must give the money
the man I must give the money
the man whom I hit with my hammer
the man I hit with my hammer

The rest of this grammar is left to the imagination of the reader.

4 Parsing AGFLs

Parsing a sentence according to an AGFL can be accomplished either by first parsing the sentence according to the underlying CF grammar and later computing the affix values, or by computing the affix values on-the-fly during the parsing process. The second approach is more desireable, because it allows to weed out at an early stage those CF parsings that are impossible according to the affix values and context dependencies.

4.1 Ambiguity

Ambiguity is the bane of Computational Linguistics. Any nontrivial grammar for a natural language will attribute more than one structure to some sentences.

There are many legitimate sources of ambiguity:

- (lexical ambiguity) At the lexical level, many word forms may possess multiple interpretations. In English, for instance, the fact that "any noun can be verbed" leads to an interesting confusion of plural noun forms and verb forms ending in *s*.

- (subordination) Parts of a phrase, like preposition clauses, may be subordinated to others in more than one way. Sequences of such parts lead to a combinatorial explosion of trivial ambiguities. For each subordination structure, examples can be found and none may be ruled out on syntactic grounds alone. In our daily use of language, we are hardly aware of this phenomenon, since we tend to exploit semantic

clues and intonation to disambiguate our communications, but to a simple syntactic parser these are not available.

- (apposition) The fact that whole constructs may serve to modify other constructs by apposition (e.g. of noun phrases) is a dependable source of ambiguity.

We shall give just a few examples of subordination ambiguities that need some semantic knowledge for their resolution.

eat the food on the table

eat the food on the couch

(subordination of preposition phrase to nounpart or verbpart), and

there is a man in this house with one leg

there is food on the table with one leg.

The famous sentence *time flies like an arrow* displays lexical ambiguity as well as subordination ambiguity. Appositions of many nouns, like

software quality assurance conference

are ambiguous as soon as they consist of more than two parts.

Ambiguity is a fact of life, with which we have to deal in a responsible fashion. Certainly it does not go away by ignoring it. We must therefore employ parsers that can find all correct analyses of a sentence.

A secondary but not negligible problem is that multiple parsing trees, richly decorated with affixes, tend to demand an excessive amount of storage and i/o. We must therefore employ economic representations for ambiguous parse trees, like the condensed trees described in [20].

4.2 Context-Free parsing

In Informatics, many forms of deterministic top-down and bottom-up parsers for CF grammars have been designed (LL(k), LR(k), SLR(k), LALR(k), ...), which are highly efficient in parsing even very long sentences (e.g. programs) provided their grammar is not ambiguous. In Informatics this is the usual situation.

For Linguistics the work on nondeterministic parsers is more relevant:

- Earley's algorithm [3] and related algorithms (for an overview see Harrison [7]).
- Tomita's algorithm [20].
- The Recursive Backup parsing algorithm [10] and its Left Corner variant [14].

The first two algorithms are extremely efficient (recognizing any CF language in $O(n^3)$ time, where n is the number of words in the sentence). They achieve this efficiency, however, by combining parser states as much as possible (a kind of maximal leftfactoring). This makes it difficult to compute affix values on-the-fly. There is no opportunity to use agreement information and semantic clues to steer the parsing process. Furthermore the construction of the parser from the grammar is a relatively time-consuming process, making it hard to experiment with evolving grammars. Lastly, the number of parse trees to be generated need not be polynomially bounded.

That is why the use of bactracking parsing algorithms like Left Corner Recursive Backup may be attractive in spite of their exponentially bounded behaviour [12]. In

particular, it is quite simple to extend this algorithm with on-the-fly affix evaluation. The same holds for PROLOG with DCGs, but its implementation can unfortunately not in general deal with leftrecursion.

4.3 Computing affix values

In scanning a sentence from left to right, we have a gradually increasing knowledge of the values of the affixes. We may consider as an example the French sentence:

je travaillais comme serveuse dans un restaurant

where it transpires half-way that the subject is feminine. Similarly, in parsing a sentence according to

```
sentence : noun phrase (NUMB, PERS), verb phrase (NUMB, PERS).
```

upon recognizing the noun phrase we will have acquired at least partial information about the value of `numb`, but recognizing the verb phrase may give us further information about its value, in sentences like

you are a very beautiful person

The gradual acquisition of knowledge about an affix can be seen as a process in which a particular instance of the affix may initially have any value in its domain (represented by the set of all values in that domain). At each application of the affix, we obtain some information about its value, which may serve to restrict the set of values it possesses (a simple form of *unification*).

If at any stage in the parsing process the set of values for an affix becomes empty, no consistent valuation of the affix is possible and the corresponding parse can be rejected.

4.3.1 Left-recursion

Linguists prefer to describe some constructs by left-recursion, e.g.:

```
noungroup:
    noun; noungroup, postmodifier.
```

which leads to a left-going parse tree:

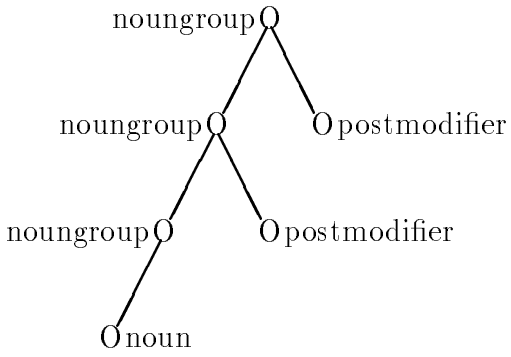

Figure 3: A left-going parse tree

Removing left-recursion in the obvious way

```
noungroup: noun, poms.
poms: postmodifier, poms; .
```

leads to a right-going parse tree instead, with different node labels:

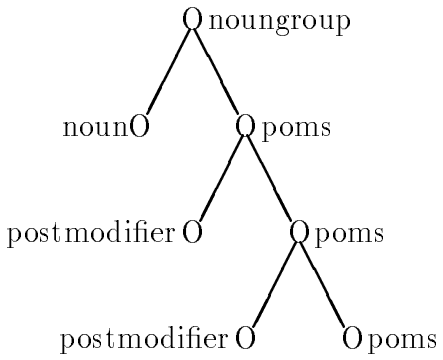

Figure 4: A right-going parse tree

For many applications, the difference in parse tree may not matter, but for some linguistic applications it is essential that the parse trees should follow the original (left-recursive) syntax.

In the AGFL-system, all left-recursions are transformed away automatically by the LC-heuristic, but the resulting parse trees are given in terms of the original syntax. The net effect is that a left-recursive formulation is just as acceptable as a right-recursive one, and that it may even be somewhat more efficient to parse.

4.4 Checking

It is as easy to make errors in writing a grammar as it is in writing a program: spelling errors, forgotten definitions, missing affixes, wrong or interchanged affixes, etc. Testing a large nondeterministic program (like a grammar) is not a simple business, no matter what formalism it is written in.

The type-restrictions imposed by the metarules of the AGFL permit a quite thorough consistency and completeness check, which turns out to be very valuable in developing a grammar with the AGFL-system. This system also provides useful information about the underlying CF grammar (leftrecursion, emptyness, hidden leftrecursion, common starters of alternatives). As a further diagnostic aid, it can be used generatively to produce (random) examples of terminal productions of any nonterminal.

5 Some linguistic issues

In this section we shall briefly discuss some important issues that arise in describing natural languages by AGFL.

5.1 Underspecification

In parsing a sentence, the value of some affix may not be specified uniquely, as in:

you are not satisfied

in which the `number` of the subject may be either singular or plural. Apart from this, the two analyses have the same parse tree.

It is useful to distinguish this form of ambiguity (*affix-ambiguity*) from the structural ambiguity in the famous sentence

they are flying planes

where the various analyses have different parse trees.

In AGFL, any set of terminal affixes can be denoted by an affix nonterminal with that set as domain. This makes it easy to express any degree of knowledge about possible values by a nonterminal affix. In the example given, the underspecified value for `numb` is denoted by `numb` itself.

This is preferable to the introduction of a special affix terminal to express underspecification, as in

```
NUMB :: sing; plur; both.
```

5.2 Directions

Often the agreement between various parts of the sentence is considered as a (directed) dependency, one part prescribing an affix value for another. Upon closer inspection, the directionality of these dependencies may be quite vague.

In particular, dependencies need not always be from left to right, since parts of speech may often occur in different orders, as is the case for the subject and the verb form in the Dutch sentences:

ik ga naar school
in de winter ga ik naar school

In AGFLs there is no notion of directionality, the consistent substitution rule guides all agreements. In this respect they are preferable over Attribute Grammars, which tend to rely strongly on dependency information for the efficient computation of attributes.

5.3 Dependent affixes

Some parts possess affixes that may or may not be present, depending on the value of some other affix. As an example, consider verb forms in English: a verb form has a tense, which indicates whether it is an infinitive or a participle or a finite verb, and only in the last case does it also possess a number and person. It is tedious to have to attribute some dummy number and person to, say, an infinitive.

This situation might be expressed in a meta-rule (in a notation inspired by [19]) by attaching those latter affixes as parameters to the terminal affix to which they belong:

```
TENSE :: infinitive; particle (TIME); finite (TIME, NUMB, PERS).
```

thus opening the door to a form of polymorphy.

In AGFL, we have chosen rather to allow one same nonterminal to be used with different arities.

5.4 Inheritance and defaults

Conventional linguistic syntactic notations, such as ATN and various forms of Attribute Grammars, have rather elaborate mechanisms for describing the inheritance of features and the default values for features.

In GPSG [4] for instance, a construct may inherit feature values from the context, or from its own constituents. More defined values may override less defined values, and the inheritance process is mixed with the agreement rules. Since most of this activity takes place behind the scenes (the features being implicit in the grammar, rather than explicit as they are in AGFL), the result is very hard to describe, comprehend or verify.

In AGFL all inheritance relations are completely explicit in the grammar. The consistent substitution rule is the one mechanism expressing both agreement and inheritance. The nonterminal affixes may act as default values, defaults being seen as a form of underspecification. The rule

```
subject (NUMB, PERS):
  pers pron (NUMB, PERS, nom);
  subject (NUMB, PERS), rel phrase (NUMB, PERS, CASE).
subject (NUMB, 3):
  noun part (NUMB).
subject (plur, 3):
  subject (NUMB, PERS), coordinator, subject (NUMB1, PERS1).
```

shows that quite complicated relationships can be succinctly expressed.

5.5 Word order

It is claimed for many natural languages that certain parts can occur in any order. Since all Phrase Structure grammars (like AGFL) impose a strict order of parts, such a free word order may be hard or even impossible to describe.

For example in [9] the claim that "the Hungarian word order is free" is adstructed by the simple sentence, consisting of seven elements

a fiam elküldte a könyvet a barátjának
(my son sent the book to his friend)

for which a total of 25 word order permutations is given.

In spite of appearances, 25 is much smaller than 7*!*, so only a small fraction of all possible permutations is realized. Furthermore, upon closer inspection only seven forms turn out to be pure permutations of the original sentence, from which they differ only in topicalization. The others employ a different form of the verb (*küldte el*), in which a preposition has been split off.

In fact any language employs some mixture of constituent ordering and agreement as clues to the syntactic structure of a sentence and to the functions of its parts. The fact that English relies mostly on constituent order may have given rise to the impression that Context-Free structure is paramount, but formalisms like AGFL can exploit agreement clues just as well.

Still, there is no denying that a relatively free word order leads to tediously enumerative syntactic descriptions. Some notational extension like the one proposed by Gazdar in Generalized Phrase Structure Grammars [4] may alleviate the problem.

Let me in passing mention the phenomena of ellipsis (in a suitable context, various parts of speach may be left out) and extraposition (some words may wander out of their part of speach to other positions in the sentence). These may again be hard to describe in a Phrase Structure grammar without special extensions.

The fact that these phenomena complicate a syntactic description enormously is a cause for serious concern. The fact that they also complicate parsing should however be no concern for linguists. Informaticians will use their ingenuity to achieve acceptable parsing speed, in spite of all handicaps.

5.6 Adequacy

Are AGFLs adequate to describe the syntax of natural languages? This question can be answered both yes and no.

There is no such thing as the syntax of a natural language. Some linguistic phenomena can be captured easily and neatly in this formalism, others strain the notation and some may continue to elude formalization. Therefore it is not hard to say no. The question is rather, whether the use of AGFL can bring us further in the development of reliable, comprehensive, elegant, communicable and utilizable syntactic descriptions. And that remains to be seen.

There is, however, a respectable school in Linguistic that holds that CFGs can offer an adequate description of the structure of natural languages, or at least that "every published argument purporting that one or another natural language is not a Contex-Free language is invalid, either formally, or empirically, or both" [18]. Hiding behind Gazdar's broad back, I dare to think the answer to the second question is yes.

Every CF grammar is of course an AGFL, but the use of affixes may lead to an appreciable reduction in the size of the grammar. More importantly, in using a two-level grammar the questions of agreement can be factored out of the questions of sentence structure. This makes it feasible to construct large and complicated grammars with more confidence.

6 Afterword

The early work on formal languages and parsing methods in Informatics has benefited greatly from the syntactic theories and techniques already developed in Linguistics. Chomsky can be seen as one of the founding fathers of Informatics. It is my conviction that Linguistics can now benefit substantially from the syntactic technology developed in Informatics.

The same holds for Artificial Intelligence: It is interesting to note that the first implementation of PROLOG by Colmerauer was based on the work he did in 1969-1970 on Metamorphosis grammars, another form of two-level grammars. This is the reason for the famous adage that PROLOG was designed by Kowalski in 1972 and implemented by Colmerauer in 1971.

In Informatics the interest in formal grammars for natural languages is growing, owing to the present interest in expert systems. An *expert system* is a programmed system that, in important respects, behaves like an expert on some specific subject. It needs a fund of factual knowledge (in the fifth-generation jargon a *knowledge-base*), inference rules for making logical deductions from facts, and also a linguistic component for input and output in natural language form, or in a form near to it.

It is in this area that Informatics and Linguistics will have increasing contact and, I hope, collaboration. In the coming years, linguists will have to provide a collection af relatively complete and linguistically sound formal grammars of natural languages, suitable as a framework for attaching semantics (there is no sense in dealing with semantics without an adequate syntax: syntax is the backbone of semantics). The task can be best performed in direct professional collaboration between linguists (who provide the grammars) and informaticians (who provide the technology and the programming). The field of expert systems is too important to leave it to hobbyists.

References

[1] AARTS, J. and VAN DEN HEUVEL, Th. (1985), *Computational tools for the syntactic analysis of corpora.* In: Linguistics 23, 303-335.

[2] DITTERS, E (1988), *A Formal Grammar for Automatic Syntactic Analysis and other Applications*, In: *Proceedings of the Regional Conference on Informatics and Arabization*, IRSIT, Tunis, Vol.1, 128-45.

[3] EARLEY, J. (1970), *An efficient context-free parsing algorithm.* Communications of the ACM 13, 94-102.

[4] GAZDAR, G. et al. (1985), *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge Mass.

[5] HALLEBEEK, J. (1987), *Hacia un sistema de análisis sintáctico automatizado: el proyecto ASATE.* In: Martin Vide, C. (Ed), *Actas del II Congreso de Lenguajes Naturales y Lenguajes Formales.* Universidad de Barcelona, 545-558.

[6] HALLEBEEK, J. (1990), *Een grammatica voor automatische analyse van het Spaans.* Diss. University of Nijmegen. In Dutch, French translation to appear.

[7] HARRISON, M.A. (1987), *Introduction to Formal Language Theory.* Addison-Wesley.

[8] HEUVEL, Th. van de, et al. (1983), *Extended Affix Grammars in Linguistics. A Manual.* English Department, University of Nijmegen.

[9] KÁROLY, S. (1972), *The Grammatical System of Hungarian*, In: Loránd Benkö and Samu Imre (Eds), *The Hungarian Language*, Akadémiai Kiadó, Budapest.

[10] KOSTER, C.H.A. (1975), *A technique for parsing ambiguous grammars*, In: *GI 4. Jahrestagung*, D. Siefkes (Ed.), Lecture Notes in Computer Science 26, Springer Verlag.

[11] KRULEE, G.K. (1991), *Computer Processing of Natural Language*, Prentice Hall.

[12] LEO, J. (1986), *On the Complexity of Top-Down Backtrack Parsers; A Reexamination*, In: Proceedings Annual NGI Conference, Utrecht.

[13] MEERTENS, L.G.L.Th. & C.H.A. KOSTER (1962), *An affix grammar for a part of the English language*, presented at Euratom Colloquium, University of Amsterdam, 1962. Not published. Reprints may be obtained from the author.

[14] MEIJER, H. (1986), *PROGRAMMAR a Translator Generator.* Diss. University of Nijmegen.

[15] MEIJER H. (1990), *The project on Extended Affix Grammars at Nijmegen*, In: *Attribute Grammars and their Applications*, SLNC 461, 130-142.

[16] MORITZ, M.P.G. (1989), *Description and Analysis of Static Semantics by Fixed Point Equations.* Diss. University of Nijmegen.

[17] OOSTDIJK, N. (1984), *An Extended Affix Grammar for the English Noun Phrase*, In: J. Aarts and W. Meijs (eds), *Corpus Linguistics. Recent Developments in the Use of Computer Corpora in English Language Research*, Amsterdam: Rodopi.

[18] PULLUM, G. and GAZDAR, G. (1982) *Natural Languages and Context Free Languages*, Linguistics and Philosophy 4.

[19] SCHOORL, J.J and BELDER, S. (1990), *Computational Linguistics at Delft, a Status Report*, Report WTM/TT 90-09, Delft University of Technology.

[20] TOMITA, M (1988), *Efficient Parsing for Natural Language, a Fast Algorithm for Practical Systems.* Kluwer.