

## Chapter 8

# Affix Grammars over a Finite Lattice

Affix grammars over a finite lattice (AGFLs) are a particularly simple form of two-level grammars, in which the second level consists of finite domains. The affixes are set-valued, and the operations (indicated by **guards**) are union and intersection.

In this chapter the concepts and notation of AGFLs are described. A brief example is given, describing a fragment of the English language, followed by a discussion of a number of linguistic issues concerning the use of AGFLs.

Affix grammars over a finite lattice (AGFLs) are a formalization of a notion well known to linguists (see e.g. [Krulec, 1991]): context-free grammars 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.

In this section, we give an informal description of AGFLs, introducing some notation and concepts. A more precise description can be found in the *GWB manual*.

An AGFL consists of affix rules and syntax rules. Their order is not important, although for the comprehension of the human reader it may be advisable to start with the affix rules. The first syntax rule is taken as the **root** of the grammar.

### 8.1 The affix rules

The **affix rules** are a collection of restricted context-free rules, together forming the second level of the AGFL. An affix rule defines a direct production of an affix nonterminal. Such a direct production is either a (single) affix terminal or another affix nonterminal, and recursion is not allowed. Consequently, an affix nonterminal has simply one or more affix terminals as terminal productions.

As is usual in affix grammars, affix rules can be recognized by the double colon separating their left-hand and right-hand sides. The affix rules

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

define the affix nonterminal **NUMB** to have two direct productions. Similarly, the rules

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

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

### 8.1.1 Domains

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

Since its domain is a finite enumeration of affix terminals, 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 CFG is obtained which is equivalent to the original AGFL.

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

## 8.2 The syntax rules

The **syntax rules** of an AGFL are context-free rules augmented with parameters. A syntax rule (or **rule** for short) 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).
```

The **left-hand side** of a rule consists of a **nonterminal symbol**, the **head**, optionally followed by a list of **parameters**, enclosed between brackets. Each parameter is either a **variable** (an affix nonterminal), or an **affix expression**, which consists of one or more affix terminals separated from one another by the union-operator |.

The **right-hand side** of a rule consists of a (possibly empty) list of **members**, separated by comma's. A member is either a **terminal symbol**, a **call**, a **group**, an **option**, or a **guard**.

A call has the same syntax as a left-hand side. A terminal symbol is written as its representation enclosed between quotes.

A nonterminal is identified by a nonterminal symbol (its **name**) and an **arity**, i.e. the number of parameters. All rules with a specific nonterminal symbol as head and with a specific arity together form the **definition** for a nonterminal. As usual, we may merge one or more rules with the same left-hand side (or, to put it differently, multi-rule definitions are allowed).

### 8.2.1 The union operator

The **union operator** serves to construct composed domains at parameter positions. An example of a multi-rule definition involving union-operators is

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

Instead of using the union operator, we may also use an affix with the required domain:

```
NONFIRST :: 1st; 3rd.  
to be (plur, NONFIRST) : "are".
```

### 8.2.2 Generic rules

Rules may have the same nonterminal but differ in arity; e.g. to avoid the passing of superfluous parameters. In programming language jargon, these are **generic rules**.

As an example, we can add to the previous rules

```
to be (inf) : "be".
```

since it would be silly to equip an infinitive with a number and person.

### 8.2.3 Options and groups

A member that is optional is put between curly brackets. E.g. given the definition

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

the member {circumstance} means the same as `circumstance option`.

A member of a rule may consist of a number of alternatives grouped together by means of round brackets, e.g.:

```
thousands:  
  singles, "thousand", (hundreds; tens; singles).
```

This construction, the *group*, provides a convenient notation for factorization.

### 8.2.4 Consistent substitution

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

```
PERS :: 1st; 2nd; 3rd.  
  
simple sentence :  
  pers pron (NUMB, PERS), to be (NUMB, PERS), adjective.  
  
pers pron (sing, 1st) : "I".  
pers pron (plur, 1st) : "we".  
pers pron (NUMB, 2nd) : "you".  
pers pron (sing, 3rd) : "he"; "she"; "it".  
pers pron (plur, 3rd) : "they".  
  
adjective : "great".
```

the sentence `you are great` has one parsing, with the variable `numb` assuming the value `{SING, PLUR}`.

Variables may be indexed with a number, to denote different instances of the same affix nonterminal; thus, `PERS1` is another instance of `PERS`, which is substituted for independently.

### 8.2.5 Guards

The guards provide a very weak operation on the affix domains, the **restrictions**. Guards are written between square brackets. A list of guards may be combined.

The first form of restriction is the confrontation of an affix variable with a domain, serving to restrict the value of the variable to that domain. As an example, the rule

```
to be (plur, 1st | 3rd) : "are".
```

could also have been written as

```
to be (NUMB, PERS): [NUMB :: plur, PERS :: 1st | 3rd] , "are".
```

In case the value of the variable has no element in common with the domain of the restriction, the guard fails (e.g. if `PERS` has the value `{2nd}`) and therefore in this case the rule fails.

A second form of restriction is the confrontation between two variables, serving to restrict both to the intersection of their values. An example: `[Q=R]`.

## 8.3 The lattice structure

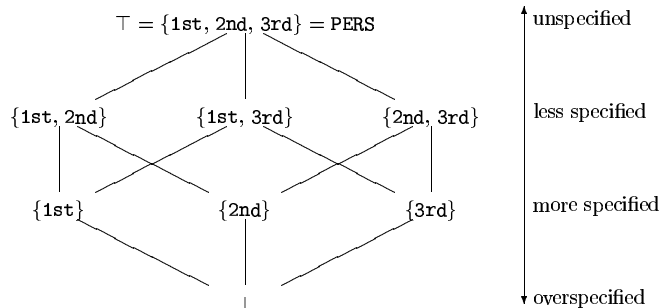


Figure 8.1: The lattice of values for the affix `pers`

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

```
PERS :: 1st; 2nd; 3rd.
```

can be depicted as in Figure 8.1.

The top-element  $\top$  of the lattice over three elements in Figure 8.1 can be seen as the union of all possibilities (“the value may be `1st` or `2nd` or `3rd` 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  $\perp$ , which indicates inconsistency. We will denote the top-element by the affix nonterminal `PERS` itself.

By the introduction of suitable affix rules, any point in the lattice can be given an affix nonterminal as name, e.g.

```
WE :: 1st; 2nd.
```

can be used to give the name `WE` to the point marked `{1st, 2nd}`.

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

Another form of lattice which we admit is the **flat lattice** of (bounded) integers, without any further operations. This lattice is specified by the predefined affix `INT`. Similarly, the predefined affix `TEXT` denotes the domain of the strings over the ASCII alphabet. These domains do not allow the use of the union operator, nor are any other operations available, apart from the guards. In particular, it is impossible to count in this formalism.

As an example, an enumeration of all verbs in a language, without any further operations, can be depicted as the flat lattice in Figure 8.2.

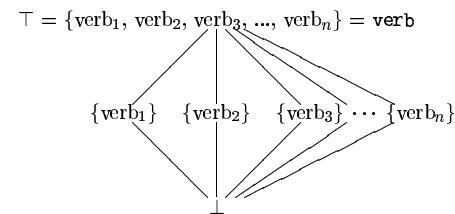


Figure 8.2: A flat lattice

Apart from finite lattices with the union and intersection as operations, it would be possible to introduce Affix Grammars over infinite lattices. In AGFL we restrict ourselves to finite lattices, obtaining a formalism which admits of highly efficient implementation.

## 8.4 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 fragment from a toy grammar of the English language (the **MEKO grammar**). 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 collection of affix rules, which is small and quite conventional, not to say classical.

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

PERS :: 1st; 2nd; 3rd.

As good Latinists we shall distinguish four cases:

CASE :: nom; gen; dat; acc.

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*, using the term *complement* for the prepositional object found with many verbs. The following rule is the root of the grammar.

SVC0 sentence :  
{circumstance}, subject (NUMB, PERS), VCO phrase (NUMB, PERS).

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

subject (NUMB, PERS):  
noun phrase (NUMB, PERS, nom).

noun phrase (NUMB, PERS, CASE):  
noun part (NUMB, PERS, CASE), rel phrase.  
noun phrase (plur, PERS, CASE):  
noun part (NUMB, PERS, CASE), coordinator,  
noun phrase (NUMB1, PERS1, CASE).

These two rules have the head *noun phrase* in common. Notice the adventurous treatment of coordinated noun phrases like *Tricky Dicky* and *the Cool Cats* in the second.

noun part (NUMB, 3rd, CASE):  
art (NUMB), noun group (NUMB);  
det (NUMB), noun group (NUMB);  
noun (NUMB1,gen), noun group (NUMB);  
poss pron, noun group (NUMB);  
noun group (NUMB), modifier.  
noun part (NUMB, PERS, CASE):  
pers pron (NUMB, PERS, CASE).

noun group (NUMB):  
adjective phrase, noun group (NUMB);  
noun (NUMB, nom).

The first alternative expresses (through recursion) the fact that a noun may have any number of adjectives.

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

rel phrase:  
rel pron (nom), VCO phrase (NUMB, PERS);  
rel pron (gen), oSVC phrase;  
prep, rel pron (dat), SVC phrase;  
{rel pron (dat)}, SVC phrase;  
{rel pron (acc)}, SVCn0 phrase.

Somehow this looks more like Latin than like English, but it is not totally unfounded. We shall leave the loose ends of the grammar to the imagination of the reader. One example for each production:

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

## 8.5 Parsing with AGFLS

Parsing a sentence according to an AGFL can be accomplished either by first parsing the sentence according to the underlying CFG and later computing the parameters, or by computing the parameters *on-the-fly* during the parsing process. The second approach is more desirable, because it allows to weed out at an early stage those CF parsings that are impossible according to the parameters and context dependencies. One technique which is suitable for parsing AGFL's will be described in chapter 10.

### 8.5.1 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 syntactic ambiguity (and we shall not even discuss semantical 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 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. We must therefore employ an economic representation for a collection of ambiguous parse trees (a **parse forests**) [Dekkers et al, 1992].

### 8.5.2 Computing affix values

In scanning a sentence from left to right, there is a gradually increasing knowledge of the values of the variables. 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 variable can be seen as a process in which a particular instance of the variable may initially have any value in its domain (represented by the set of all values in that domain). At each application of the variable, 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 value for a variable becomes empty, no consistent valuation is possible and the corresponding parse can be rejected.

### 8.5.3 Checking

It is as easy to make errors in writing a grammar as it is in writing a program: spelling errors, forgotten definitions, missing parameters, wrong or interchanged parameters, 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 affix rules of an AGFL permit a quite thorough consistency and completeness check, which turns out to be very valuable in developing a grammar with the AGFL system. The Grammar Work Bench for AGFL (GWB, see [Nederhof and Koster, 1992]) also provides useful information about the underlying CFG, such as (hidden) left recursion, common starters of alternatives, and many more. As a further diagnostic aid, it can be used generatively to produce (random) examples of terminal productions of any nonterminal.

## 8.6 Some linguistic issues

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

### 8.6.1 Underspecification

In parsing a sentence, the value of some variable 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 AGFLs, any set of affix terminals 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 in the parse tree. This is preferable to the introduction of a special affix terminal to express underspecification, as in

NUMB :: sing; plur; both.

### 8.6.2 Directions

In many text books 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.

### 8.6.3 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 an affix rule (in a notation inspired by [Schoorl and Belder, 1990]) 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 instead to allow one same nonterminal symbol to be used with different arities.

#### 8.6.4 Inheritance and defaults

Conventional linguistic syntactic notations, such as augmented transition networks (ATN, see [Woods, 1970]) and various forms of attribute grammars, have rather elaborate mechanisms for describing the inheritance of features and the default values for features.

In HPSG 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 AGFLs), the result is very hard to describe, comprehend or verify.

In AGFLs all inheritance relations are completely explicit. 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, 3rd):
  noun part (NUMB).
subject (plur, 3rd):
  subject (NUMB, PERS), coordinator, subject (NUMB1, PERS1).
```

shows that quite complicated relationships can be succinctly expressed.

#### 8.6.5 Free word order

It is claimed for many natural languages that certain parts can occur in any order. Since all phrase-structure grammars (like AGFLs) impose a strict order of parts, such a free word order may be hard to describe.

For example in [Károly, 1972] the claim that “the Hungarian word order is free” is illustrated 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. Notational extensions like the one proposed by Gazdar in Generalized Phrase Structure Grammars [Gazdar et al, 1985] may alleviate the problem.

Let me in passing mention the phenomena of ellipsis (in a suitable context, various parts of speech may be left out) and extraposition (some words may wander out of their part of speech 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. Computer scientists will use their ingenuity to achieve acceptable parsing speed, in spite of all handicaps.

#### 8.6.6 The AGFL system

The AGFL formalism is accompanied by a grammar development environment. This AGFL system consists at the moment of the following components:

- The Grammar Work Bench GWB [Nederhof and Koster, 1992] is a lingware engineering environment which supports the development of grammars and their validation, by determining their properties and generating examples.
- The parser generator GEN generates efficient parsers, suitable not only for linguistic research but also for embedding into applications like full-text Information Retrieval systems, Grammar Checkers and the like.
- the lexicon generator LEXGEN allows the efficient interfacing of parsers to large compressed lexica, even on PCs.
- The Linguist's Data Base LDB [van Halteren and van den Heuvel, 1990] allows the inspection, disambiguation and further linguistic and statistical analysis of parsed corpora.

The AGFL system is distributed free of charge by WWW via  
<http://www.cs.kun.nl/agfl/>