

The family of Affix Grammars

Cornelis H. A. Koster

Sept 1995

Abstract

This note describes the rationale behind Affix Grammars. Affix Grammars are a class of two-level grammars, Context-Free grammars extended with features and operations. Originally invented for the description of natural languages, various forms of Affix Grammars have found numerous applications in Computer Science, and efficient implementations were made. In the last years Affix Grammars with set-valued attributes (AGFL) have found successful applications as a “lean formalism” for describing the surface syntax of natural languages. Readers who have some experience in the use of Affix Grammars may find in this note the necessary background to place this experience into perspective.

Introduction

This note describes the rationale behind Affix Grammars. It gives a brief account of their genesis in Linguistics and subsequent development in Computer Science, and discusses the degrees of freedom that characterize the family of Affix Grammars.

It focusses on those forms of Affix Grammars that have found applications in Linguistics, AGFL and EAG, and discusses their merits. Finally, the relationship between Affix Grammars and other two-level grammars is discussed.

It is not our intention to give here a rigorous description of Affix Grammars — for a formal definition of Affix Grammars the reader is referred to [12]. Nor is this an easy and comprehensive introduction to the subject — linguistic readers are referred to [14] and Computer Scientists to [15], or other publications cited in the bibliography. Instead, we try to provide the reader who has some experience in the use of Affix Grammars with the necessary background to place his experience into perspective.

1 Genesis of Affix Grammars

“aurea prima sata est”
Vergilius, Aeneis.

In 1962 prof. E.W. Beth organized an Euratom Colloquium at the University of Amsterdam, which brought together people interested in far-out subjects like Chomsky and modern linguistics. In the next decade this colloquium proved to have been the cradle of much of the research in Formal Languages, Artificial Intelligence and Logics in the Netherlands.

Amongst the participants were two second year students in Mathematics and Physics, Lambert Meertens and Kees Koster. We had nothing to seek there and were purely driven

by curiosity. Like all participants we were assigned the review and discussion of a recent paper (on Immediate Constituent grammars) which we voluntarily complemented with the demonstration of a generative grammar.

We first made a CF grammar for a fragment of Dutch (sentences like “een kikvorsch is een grote kikker”) and figured out how to program it in Assembly Language for the Electrologica X1 computer. Then we embarked on the construction of a grammar for a large fragment of English. We found CF grammar lacking in means for the expression of agreement, and it was Lambert Meertens who suggested the extension with affixes. In two days, working with hardly any sleep, we wrote the grammar, transcribed it to Machine language and presented our contribution to the Colloquium [16]. Over the next few years I found a way to construct parsers for Affix Grammars and wrote Affix Transducers for the transformation active-to-passive in Dutch, and the translation from Dutch to German [11]. Lambert Meertens applied Affix Grammars to the description and composition of Music, and obtained a special prize from the jury at the 1968 IFIP Congress in Edinburgh for his computer-generated string quartet, based on the first non-Context-Free affix grammar. Together with Leo Geurts, we worked on the generation of speech from written text, involving the transduction of text to phonemes.

This idyllic period of early Natural Language processing came to an end in February 1966, when Lambert Meertens and I started working as full time Computer Scientists. Now our noses were kept firmly to the grindstone.

2 The intuition behind affixes

Affix grammars arose quite naturally as extensions of CF grammars in describing natural languages. As an example, consider the following CF grammar:

```
sentence → subject verb object

subject → "I" | "you" | "the devil"

verb → "like" | "likes"

object → "me" | "you" | "the devil"
```

Apart from a few reasonable english sentences, this grammar also generates rubbish like `you likes me`. The **agreement** in person between subject and verb is not assured. Now it is quite possible to correct the grammar, introducing a distinction between `subject+first` (a subject in the first person) and `subject+third` (in the third), and similarly distinguishing between `verb+first` and `verb+third`, but in constructing a full-scale grammar this is a losing battle, considering the complexity of the agreements to be described. What is needed is a generalisation of CF grammar.

One possibility is to go over to Context Sensitive grammars from the Chomsky hierarchy, but this introduces its own problems. CS grammars are hard to write or read and there is no general way to derive parsers from them. Another possibility is to formalise the attachment of affixes like `-first` and `-third` to the names of nonterminals. This is the intuition that leads to Affix Grammars.

Introducing an affix `person` to distinguish between first, second and third person

person = first | secnd | third

we can express the desired agreement as follows, using the original notation from 1962:

sentence → subject + person verb + person object.

Actually this is a **rule schema**, in which one of three values has to be substituted for **person** in order to obtain a CF rule. The **consistent substitution** rule requires the same value to be substituted for every occurrence of a specific affix in a rule.

We can complete this rudimentary grammar by adding some rules defining the **lexicon**:

subject (first) → "I"

subject (secnd) → "you"

subject (third) → "the devil"

verb (first) → "like"

verb (secnd) → "like"

verb (third) → "likes"

object → "me" | "you" | "the devil"

The original notation emphasized the intuition of the affixes as suffixes to the nonterminal symbol. It has been superseded by a notation emphasizing another intuition: that of the affixes as parameters to the nonterminal.

3 Affix Grammars

Affix Grammars were formalized as an extension of CF-grammars with affixes and operations. In fact, Affix Grammars form a family of two-level grammars

$$\left\{ \begin{array}{l} \text{affixes} + \text{domains} \\ \text{CFgrammar} + \text{operations} \end{array} \right\}$$

where the first or lower level consists of *CF rule-schemata*, CF rules extended with metavariables (the *affixes*) and the second level defines the domains of these affixes by means of CF *metarules*. Included among the nonterminals of the first level are some symbols standing for *operations*. The left column in the diagram is syntactic and the right column algebraic in nature. In this sense, Affix Grammars are an early example of a unification between grammars and functional languages.

In choosing a particular member of the family, a number of degrees of freedom are available:

- (domain and operations) The domains of the affixes may be chosen to be (restricted) Context-Free languages, or a more machine-oriented domain may be chosen, such as the integers of some computer, or the datatypes of some programming language. In each case, the domain is accompanied by a collection of operations applied to it on the first level of the AG. Still other choices of domains are possible, e.g. functional domains or lattices [19].

- (power of the underlying CF grammar) The underlying CF grammar may be chosen to be deterministic (e.g. LL(1)) for easy parseability, or it may be nondeterministic (in which case a more powerful computational model will be needed). For applications in natural languages, nondeterminism is essential.
- (wellformedness rules) It is customary to subsume, under this name, any restrictions concerning the consistency and directionality of the semantic flow (cf. Knuth's wellformedness rules in [10]). The semantic flow may be strictly left-to-right, so that all affixes can be evaluated during parsing, or it can be quite unrestricted, implying some form of delayed evaluation. Restricting the semantic flow leads to a formalism with a functional flavour and efficient implementation. Unrestricted semantic flow leads to a specification formalism which it is hard to implement efficiently.

Affix Grammars are, because of this freedom, a large family, in which many different positions can be chosen. We shall show that this form of grammars may be quite powerful, and then discuss a number of particular forms, comparing their descriptive power.

4 Context sensitivity

Affix Grammars can not only describe CF languages in a compact way but they may also describe Context-Sensitive languages [23]. The following grammar (in a more modern notation than the previous, treating affixes as parameters) describes the Context-Sensitive language

$$a^n b^n c^n$$

We make use of a recursive metarule

`N :: one; one N.`

which describes the natural numbers in unary notation.

`sentence : a (N), b (N), c (N).`

`a (one) : "a" .`

`a (one N) : "a", a (N).`

`b (one) : "b" .`

`b (one N) : "b", b (N).`

`c (one) : "c" .`

`c (one N) : "c", c (N).`

Notice that this description is purely generative and does not suggest any technique for parsing the language (other than trying all numbers one by one). On the other hand, the following (much) more operational description of the same language is rather asymmetrical because it describes a computation from left to right.

The root of the grammar stays the same.

`sentence : a (N), b (N), c (N).`

The affixes, however, are now considered as parameters with a **direction**: the sign $>$ is written before an input-parameter (**inherited affix**) and after an output-parameter (**derived affix**). Also we follow the convention that different instances of an affix may be distinguished by a numerical suffix (thus avoiding consistent substitution).

```
a (N>) :
  "a",
  (a (N1), incr (N1, N);
   assign (one, N)).
```

The **operations** `assign` and `incr` are introduced as quasi-nonterminals and the affixes `N` and `N1` serve as variables, transporting information from left to right. Notice the use of brackets to group two alternatives together; this allows manual left-factorization of the rule.

```
b (>N) :
  equal (N, 0);
  "b", decr (N, N1), b (N1).
```

We need another operation `decr` and the test `equal`.

```
c (>N) :
  equal (N, 0);
  "c", decr (N, N1), c (N1).
```

The operations have the following semantics:

$$\begin{aligned} \text{assign } (>x, y>) &=_{def} \lambda_{x,y}.x = y \\ \text{incr } (>x, y>) &=_{def} \lambda_{x,y}.y = x + 1 \\ \text{decr } (>x, y>) &=_{def} \lambda_{x,y}.y = x - 1 \\ \text{equal } (>x, >y) &=_{def} \lambda_{x,y}.x = y \end{aligned}$$

It should be obvious that this extremely operational version is a description of a parser rather than a grammar.

One of the most important issues in the development of the Affix Grammar formalism over the years has been the quest for a formalism which is applicative and descriptive in its use, while still guaranteeing the automatic construction of efficient parsers. We wish to avoid that the human users (writers of readers) of a grammar should have to think operationally in order to understand it. In this development they have been strongly influenced by **W grammars** [25]. This formalism was introduced by Aad van Wijngaarden in 1965 for the formal description of programming languages like ALGOL68 [26]. It is a formalism of great elegance and exemplary simplicity. W-grammars are a “pure” form of two-level grammars, without any admixture of functions, a mechanism for specification without any concessions to implementability. An informal introduction to W-grammars can be found in [3].

5 Extended Affix Grammars

Extended Affix Grammars (EAGs) were introduced by David Watt [24] to form a bridge between Affix Grammars and W-grammars. His aim was to achieve the expressivity of W-grammars without sacrificing the implementability of Affix Grammars.

In EAG the domains of all affixes are the same: strings over the given alphabet. Therefore all affixes have the same type. Affix rules merely serve to express a restriction, which has to be enforced dynamically in a costly fashion, and are therefore hardly used in EAG. The only operations in EAG are

- the **join**, composing two affix values by concatenation
- the **nondeterministic split** of an affix value.

The crucial extension of EAG with respect to AG is that these operations are “sugared away” by allowing composed affix expressions at formal parameter positions.

As an example, the previous grammar can be formulated in EAG as follows:

`sentence` : `a (N)`, `b (N)`, `c (N)`.

The following rules are split according to different parametrizations.

`a (one>)` : `"a"`.

`a (N one>)` : `"a"`, `a (N)`.

Notice the implicit join-operation in the last rule.

`b (>one)` : `"b"`.

`b (>N one)` : `"b"`, `b (N)`.

The last rule contains an implicit split-operation.

`c (>one)` : `"c"`.

`c (>N one)` : `"c"`, `c (N)`.

Apart from the split and join, EAG does not possess any operations. It can be proved that in spite of this simplicity EAG has Turing power.

EAG’s proved surprisingly hard to implement. The first implementations restricted (like Watt himself) the underlying CF grammar to be deterministic, which is fine for describing programming languages [9] but which severely limited the applicability of EAG in Linguistics. In his Thesis Hans Meijer first showed how to implement virtually unrestricted EAGs [17]. In the simple case of affixes with a finite domain (like the grammar in section 2) the EAG can be “exploded” into an equivalent CF grammar, which can then be parsed by a (nondeterministic) CF parser.

EAG was the basis for a unique collaboration between Linguists and Computer Scientists at the University of Nijmegen, one side providing the technology [18] and the other applying EAG to the description of natural languages [8]. This resulted in EAGs for English [20], Spanish [7] and Modern Standard Arabic [5]. The development of the EAG technology was supported by the Dutch National Organization for Scientific Research NWO.

6 Affix Grammars over a Finite Lattice

In studying the various EAGs constructed by linguists I was struck by the fact that their authors did not seem to need the full (Turing) power of the formalism: they were quite happy using only affixes with finite domains. This observation was strengthened by the remark of Pullum and Gazdar [22] that

“every published argument purporting that one or another natural language is not a Context-Free language is invalid, either formally, or empirically, or both”

Therefore I defined the class of Affix Grammars over a Finite Lattice (AGFL), which was implemented in 1991 by Arend van Zwol using the Recursive Backup parsing technique [13]. This development was again supported by NWO.

In AGFL, metarules define the terminal productions of an affix as a choice between single terminal affixes. Since in natural languages this choice is perforce often ambiguous (or rather, under-specified) the domain of an affix may be any nonempty subset of the (finite) set of terminal productions of the affix. The operations **set union** and **set difference** on these domains, which induce a lattice structure, can conveniently be sugared away at parameter positions (as in EAG) or they can explicitly be expressed by **guards**.

Basing the computational model on set-unification rather than function computation, a compact and powerful notation for CFG is obtained, which has two advantages over EAG:

1. due to the strong typing of affixes and the finiteness of the domains, the consistency of a grammar can be checked statically to a remarkable degree
2. treating the affixes as parameters to a Recursive Backup parser, evaluated on-the-fly during parsing, in conjunction with a number of optimizations implemented by Arjan Knijff (supported by NWO), has led to very efficient parsers.

Because of these advantages, the AGFL technology was successfully accepted by our linguistic colleagues, in spite of the hardship of converting all grammars from EAG to AGFL. In this conversion, a surprising number of inconsistencies were eliminated, which were very hard to find in the original EAGs. Furthermore, the performance of the parsers improved dramatically.

The new features provided at the request of users by version 1.5 of the AGFL system (lexicon system, cut-operator, penalties, regular expressions as wildcards) should assure an improved usefulness of the system for linguistic applications.

For users demanding more computational power than CFG (e.g. for dealing with semantics) I am presently considering an extension of AGFL with term-valued affixes (AGTL) – but this will need some time, as well as support by users and NWO. And certainly we will extend AGFL with transduction and probabilities – but that is another story.

7 Some related formalisms

For linguistic purposes, a large number of formalisms resembling Affix Grammars (and AGFL) to a great extent have been invented. To mention a few:

- Attribute grammars (AtG) were invented by Don Knuth, hand-in-hand with LR(1)-parsing, in the middle sixties [10]. A recent overview of the state-of-the-art in Attribute Grammars can be found in [1]. Attribute grammars have found a limited use in Linguistics because hardly any nondeterministic implementations were available. They have however influenced many formalisms specifically

developed for linguistic applications. A number of syntactic formalisms widely used in linguistics are closely related to attribute grammars: various kinds of feature-based and unification-based grammars (e.g. GPSG [6] and HPSG).

- Metamorphosis grammars, defined by Colmerauer in the late sixties for linguistic purposes [4] and strongly influenced by W-grammars, have been the direct precursor of PROLOG.

PROLOG with DCGs [21] is remarkably like AGFL, but lacks an explicit second level and has much larger computational power. The former makes the parsers slow, the latter invites the writing of operational parsers rather than grammars. PROLOG is very convenient for prototyping but parsers in PROLOG are hard to scale up to realistic grammars and lexica.

References

- [1] H. Alblas & B. Melichar (1991), Attribute Grammars, Applications and Systems, Springer Lecture Notes in Computer Science 545.
- [2] J. G. Beney, J. F. Boulicaut, *StarLet: an affix-based Compiler compiler designed as a logic programming system*, Third International Workshop on Compiler Compilers, CC'90, Schwerin, October 1990, In Lecture Notes in Computer Science n 477, pp 71-85, Springer-Verlag, 1990.
- [3] J.C. Cleaveland & R.C. Uzgalis (1977), Grammars for Programming Languages. *Elsevier/North Holland*.
- [4] A. Colmerauer (1978), Metamorphosis Grammars. In: L. Bolc (ed.), *Natural Language Communication with computers*, Springer-Verlag, Berlin, 133-189.
- [5] E. Ditters (1992), *A Formal Approach to Arabic Syntax*, Diss. University of Nijmegen.
- [6] G. Gazdar, E. Klein, G. Pullum & I. Sag (1985), Generalized Phrase Structure Grammar. Harvard University Press.
- [7] J. Hallebeek (1990), *Een grammatica voor automatische analyse van het Spaans*. Diss. University of Nijmegen. In Dutch, French translation to appear.
- [8] Th. van de Heuvel et al. (1983), Extended Affix Grammars in linguistics. A Manual. *English Department, University of Nijmegen*.
- [9] B. Hoffmann (1983), Compiler generation: From Language Definitions to Abstract Compilers. *PhD thesis, TU Berlin*; Bericht 5/83 Fachbereich Mathematik/Informatik, Universität Bremen.

- [10] D.E. Knuth (1968), Semantics of context-free languages. In: *Mathematical Systems Theory 2(2)*, 127-145.
- [11] C.H.A. Koster (1965), On the Construction of ALGOL-Procedures for Generating, Analyzing and Translating Sentences in Natural Languages. *Report MR72, Mathematisch Centrum*.
- [12] C.H.A. Koster (1971), *Affix Grammars*. In: J.E.L. Peck (ed.), *algol 68 Implementation*, pp 95-109. North-Holland Publishing Company.
- [13] C.H.A. Koster (1975), A technique for parsing ambiguous grammars, In: *GI 4. Jahrestagung*, D. Siefkes (Ed.), Springer Lecture notes in Computer Science 26.
- [14] C.H.A. Koster (1991), Affix Grammars for Natural Languages. In [1].
- [15] C.H.A. Koster (1991), Affix Grammars for Programming Languages. In [1].
- [16] L.G.L.Th. Meertens & C.H.A. Koster (1962), An affix grammar for an part of the English language, presented at Euratom Colloquium, University of Amsterdam, 1962, *not published* (reprints may be obtained from the authors).
- [17] H. Meijer (1986), PROGRAMMAR a Translator Generator. *PhD thesis, University of Nijmegen*.
- [18] H. Meijer (1990), The project on Extended Affix Grammars at Nijmegen, In: *Attribute Grammars and their Applications, SLNC 461, 130-142*.
- [19] M.P.G. Moritz (1989), Description and Analysis of Static Semantics by Fixed Point Equations. *PhD thesis, University of Nijmegen*.
- [20] N. Oostdijk (1991), *Corpus linguistics and the automatic analysis of English*, Dissertation University of Nijmegen.
- [21] F.C.N. Pereira & D.H.D. Warren (1980), Definite clause grammars for language analysis – A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence 13, 3, 231-278*.
- [22] G. Pullum and R. Gazdar (1982), *Natural Languages and Context Free Languages*, Linguistics and Philosophy 4.
- [23] M. Sintzoff (1967), Existence of Van Wijngaarden syntax for every recursively enumerable set. In: *Annales de la Société Scientifique de Bruxelles 81*, 115-118.
- [24] D.A. Watt (1977), The parsing problem for affix-grammars. In: *Acta Informatica 8(1)*, 111-20.
- [25] A. van Wijngaarden (1965), *Orthogonal Design and Description of a Formal Language*. Report MR76, Mathematisch Centrum, Amsterdam.
- [26] A. van Wijngaarden, B.J.M. Mailloux, J.E.L. Peck & C.H.A. Koster (1969), Report on the algorithmic language ALGOL 68. In: *Numerische Mathematik 14*, 79-218.