Parsing Dependency Grammar using ALE

Piet Mertens
Centre for Computational Linguistics - K.U.Leuven
Blijde-Inkomststraat 21, 3000 Leuven, Belgium
Piet.Mertens@arts.kuleuven.ac.be

Abstract

This paper describes a technique for parsing dependency grammars using a bottom-up chart parser originally designed for phrase-structure grammars, using typed feature structures as the only data structure. Each lexical item is represented as a tree where nodes indicate lexical elements (the anchor, its dependents and governor) and edges (branches) indicate dependency relations between these elements. Nodes may carry additional features, including one for node saturation. Trees combine into derived trees provided that node and edge features unify. The ALE system is used to implement an active chart parser where a chart edge represents a tree, and two adjacent edges are combined into a more saturated tree.

1 Dependency grammar

The term *dependency grammar* (DG) designates a class of grammars in which syntactic structure takes the form of a network of relations between words, such that each word is connected to another word. This should be contrasted with *phrase structure grammars* (PSG), which are based on constituency. This distinction is independent from that concerning the specification of syntactic elements in terms of (morphological, syntagmatic, or phrasal) categories or in terms of syntactic functions.

In a dependency tree there is exactly one node for each word in the sentence. Each word depends on exactly one other word. The former is called a *dependent*, the latter it's *governor*. The only exception is the root of the tree, which has no governor. A word may govern several dependents.

To illustrate this, figure 1 shows a dependency graph as proposed by (Tesnière, 1959), who calls it a *stemma*. An arc connects the dependent and its governor, with the dependent appearing below its governor. Arcs may be labelled to indicate the nature of the dependency relation.

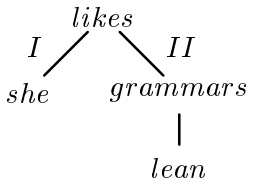

Figure 1: Stemma for "She likes lean grammars". I and II indicate subject and object, respectively.

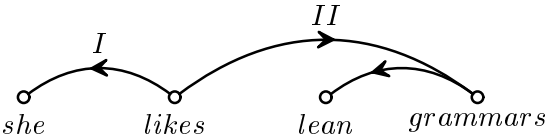

Figure 2: Ordered dependency graph

Word order is not necessarily preserved in a stemma. As a matter of fact, in DG word order is modelled independently of dependency. Figure 2 shows an alternative representation of the dependency network, but which preserves word order. The arcs have been replaced by arrows pointing from the governor to the dependent.

Dependency is related to valency, which provides a detailed characterization of the relations between a governor and its essential dependents. In "He was charged with murder. The soldiers charged the enemy. He charged me 1 dollar." the verb "to charge" appears in different argument configurations. We say this verb has multiple *predicators*, each of which selects its dependents and their properties.

In valency grammar one commonly distinguishes two major types of dependency relations and hence two types of dependents: *valency elements* and *adjuncts* (or *modifiers*). A valency element is a dependent the properties of which

are selected by its predicator. Modifiers, on the other hand, may be combined with all predicators of a certain type. The list of valency elements selected by a verbal predicator is sometimes called its *valency frame*.

Whereas verbal predicators are always considered to be governing their valency elements and modifiers, there is less agreement about the dependency status of determiners, prepositions, conjunctions, auxiliary verbs, and so on. Whatever position one takes in these issues has no implications for the parser described below. The parser merely provides general mechanisms to deal with types of objects such as valency elements and modifiers. It is up to the linguist to decide whether a particular lexical item will be represented by this or that object type. This decision will be reflected in the lexicon only.

Tree adjoining grammar (TAG (Joshi, 1987)), and Lexicalized TAG in particular, is related to dependency in that it uses trees (rather than categories) as its elementary units, i.e. the representation of a lexical element is a tree. Two types of trees are distinguished, with specialized combination procedures (substitution and insertion).

2 Elementary dependency trees and derivation

To illustrate the way elementary trees are combined into derived dependency trees, it is convenient to have a concise graphic representation of such trees.

Figure 3 shows some examples of *elementary trees*. Following TAG (including for terminology), two *types* of elementary trees are distinguished: initial and auxiliary trees, also called α and β-trees, respectively. A tree node may be labeled to indicate some of its relevant features. *Black* and *white* nodes (vertices) indicate saturated and unsaturated nodes, respectively. Arrows (edges) indicate dependency relations, where the arrow points from the head to the dependent. Elementary trees contain at least one *lexical anchor*, representing the lexical element that motivates the tree.

α_2 is the elementary tree for the noun "bike", which appears as a black node at the center of the tree, with the feature N for 'noun'. As indicated by the arrows, this element governs one valency element while being itself governed by another. The element governed by the noun appears as the node below the anchor; it will have the function of determiner (noted as 'det'). The white node indicates the element has not been found (saturated) yet. The element governing the lexical anchor appears as the white node above it. A tree is unsaturated when one or more valency elements are unsaturated. A diamond next to the tree identification indicates a saturated tree. Note that initial trees don't contain nodes for modifiers, since the latter have no impact on the saturation of the local tree.

The initial tree α_4 illustrates a verbal predicator governing two valency elements: subject and object. Of course, the number and properties of the elements in the valency frame will depend on the identity of the predicator. These examples only specify syntactic function, but since valency slots will be represented by feature structures of arbitrary complexity, very fine-grained constraints can be expressed.

Both nodes and arrows may be labeled, but need not be. For α_1 the relation between the pronoun and its governor is labeled as 'subject', whereas this is not the case for α_2, which is *under-specified*. Whereas the pronoun 'he' can only be a subject, the noun (with its dependents) could also be the object, or (part of) an indirect object, and so on. So there is no need for specifying the noun's function. Indeed it is preferable not to do so, to avoid "duplicate" trees for cases where it has a function other than that of subject.

An auxiliary tree, such as β_1, is recognized by the square node for the lexical anchor. Its root node specifies the conditions that should be met (i.e. the features to be unified) in order to combine the β-tree with the anchor of a governing tree. This root node is called the *foot* of the β-tree. A β-tree may have valency elements (either optional or required), as for tree β_2, where the adjective "adequate" governs a prepositional head "for" (other analyses being possible, of course). β-trees may be combined with other β-trees, e.g. for degree adverbs such as "very" in "very new".

A β-tree is a dependent that contains (in its foot) the information about the kind of governor it combines with. For α-trees the situation is the other way round: here it is the governor that contains (in the valency slots) the infor-

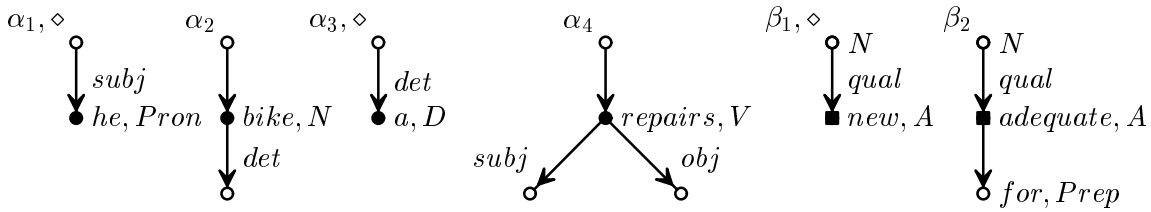

Figure 3: Some elementary trees, either initial (α) or auxiliary (β)

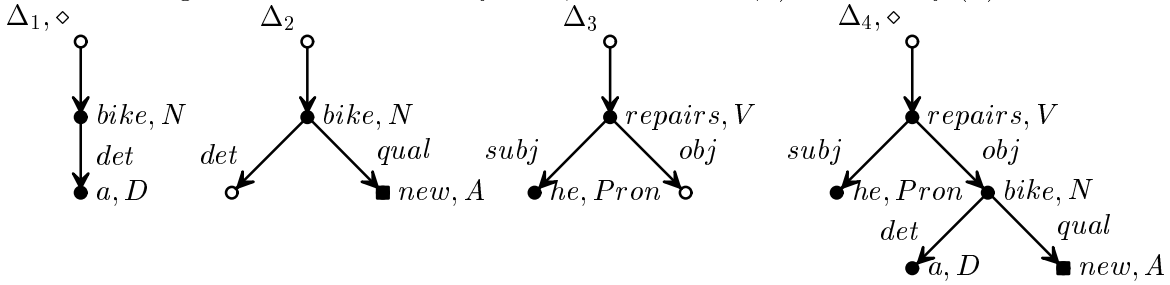

Figure 4: Derived trees for 'a bike', 'new bike', 'he repairs', 'he repairs a new bike'

mation about the dependents it combines with. The reason for this distinction is one of economy. If there were only α-trees, one would have to enumerate all possible modifiers as optional dependents, and do so for each elementary tree. This would be highly inefficient.

Tree derivation. Larger trees are derived from elementary trees in two ways. First, by unifying an unsaturated valency slot with a dependent tree of type α. This supposes that the properties of the corresponding nodes and their dependency relations (as indicated by the labels on the adjacent arrows) match.[1] As an additional condition it may be required that the dependent is saturated also. The combination of trees α_2 and α_3 produces the derived tree Δ_1 (figure 4). The determiner node of the noun appears in black since it is now saturated. The combination of α_1 and α_4 results in Δ_3.

The second type of combination concerns β-trees being added as a dependent of the anchor node of its governor, as in Δ_2. The analysis of the sentence "he repears a new bike" results in the dependency tree Δ_4.

3 Word order

The general mechanism of tree derivation outlined in the previous section does not prevent sentences like "repairs he new a bike" from being analyzed as correct: they indeed respect the constraints on valency and dependency captured in the elementary trees. An additional criterion is required to verify word order.

In many languages it is possible to specify word order *locally* as the position of dependents relative to their head as well as to other dependents. Some of the factors involved in the relative order of the dependents for a given governor are syntactic function, part-of-speech of the dependent, the clitic nature of pronouns, the nature of the dependent's dependents (e.g. presence of a preposition), morphological features (person), and so on.

The position (side and distance) of a dependent relative to its governor may be captured in rules in which the nature of both the governor and the dependent is identified by one or more features. As an illustration, consider the feature combinations and the corresponding distances in table 1, for the direct an indirect object to the right of the verb. For any two dependents A and B of governor G, if B follows A to the right of G, then B is assigned a distance greater than A. If A and B appear in both orders, they are assigned the same distance.

4 Trees, feature structures and unification

A governor's valency frame is most easily represented as a list where each element corresponds to a FS stating the features that are appropriate

[1] Note that we are unifying entire trees (rather than isolated nodes or relations), i.e. the properties of the lexical anchor node, its relation to the governor, and all of the anchor's dependency (nodes and relations) when present.

example	it	him	book	to him
POS	clit pron	pron	noun	pron, noun
function	obj	iobj	obj	iobj
preposition	-	no	-	yes
distance	5	10	20	20

Table 1: Relative order of valency elements following the verb

for that valency element.

An adequate representation of the valency frame should allow for the verification of the following properties. *Appropriateness*: a slot can be filled only by an element that is appropriate for it. This requirement is easily achieved through unification. *Uniqueness*: a slot may be filled only once (in certain constructions). *Completeness (saturation)*: it should be clear when the valency frame is saturated, i.e. when all obligatory slots are filled. *Optionality*: it should be clear whether a slot is optional or required.

In order to verify uniqueness in the context of unification, a feature (SATUR) is be reserved the value of which indicates whether or not the slot is saturated. This feature's value will be toggled when the slot is filled. It should be disregarded when verifying appropriateness to avoid interference between the saturation of the candidate and that of the slot. Similarly, the OPTION attribute indicates whether the slot is optional or required. A tree is saturated when all of its obligatory valency slots are saturated.

In the parser, elementary and derived trees are represented as FS. A simplified FS for the initial tree α_1 for the pronoun "he" is shown in (1). It subsumes the complete FS used in the actual system. The path TREE groups features that concern the tree as a whole and that may be modified in the derivation process. The feature DEPS is reserved for the dependents, with subfeatures for the valency frame and the list of adjuncts (β-trees). The latter is initially empty. Since this pronoun has no valency elements, its valency frame is empty. An α-tree doesn't have a foot node.

(1)
$$\begin{bmatrix} \text{TREE} & \begin{bmatrix} \text{TYPE} & \text{ALPHA} \\ \text{SATUR} & \text{TRUE} \\ \text{FUNC} & \text{SUBJ} \end{bmatrix} \\ \text{ANCHOR} & \begin{bmatrix} \text{CAT} & \text{PRONOUN} \\ \text{LEMMA} & \textbf{he} \end{bmatrix} \\ \text{DEPS} & \begin{bmatrix} \text{VALENCY} & \langle\rangle \\ \text{ADJUNCTS} & \langle\rangle \end{bmatrix} \end{bmatrix}$$

In the case of a β-tree, the FEET feature will specify as the first element the features of the governing node with which it may combine (2).

(2)
$$\begin{bmatrix} \text{TREE} & \begin{bmatrix} \text{TYPE} & \text{BETA} \\ \text{SATUR} & \text{TRUE} \\ \text{FUNC} & \text{QUAL} \end{bmatrix} \\ \text{ANCHOR} & \begin{bmatrix} \text{CAT} & \text{ADJECTIVE} \\ \text{LEMMA} & \textbf{new} \end{bmatrix} \\ \text{DEPS} & \begin{bmatrix} \text{VALENCY} & \langle\rangle \\ \text{ADJUNCTS} & \langle\rangle \end{bmatrix} \\ \text{FEET} & \left\langle \begin{bmatrix} \text{ANCHOR} & \begin{bmatrix} \text{CAT} & \text{NOUN} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

For an α-tree with a non-empty valency frame, the frame slots at the path DEPS|VALENCY contain FSs constraining the dependents that may fill the slots (figure 5).

When the trees for the elements "he" and "repairs" are combined, a derived tree results in which the subject slot of the verb is unified with the FS of the pronoun (figure 6). The unification between the FSs of the candidate for a slot and the slot itself should disregard the feature SATUR (something which can be achieved in several ways). After unification the feature SATUR is set to TRUE.

5 Implementing DG in ALE

(Kahane, 2001) includes a detailed account of approaches used in parsing DG. Some are designed specifically for DG (e.g. (Courtin and Genthial, 1998), (Hudson, 2000), (Nasr, 1995), (Nasr, to appear)); but in others (e.g. (Lombardo and Lesmo, 1996)) algorithms originally proposed for PS grammars (CKY parser, Earley-type parser) are applied to DG. This section describes how a unification-based chart parser designed for PSG can be used as an active chart parser performing dependency tree unification. By using an existing unification parser

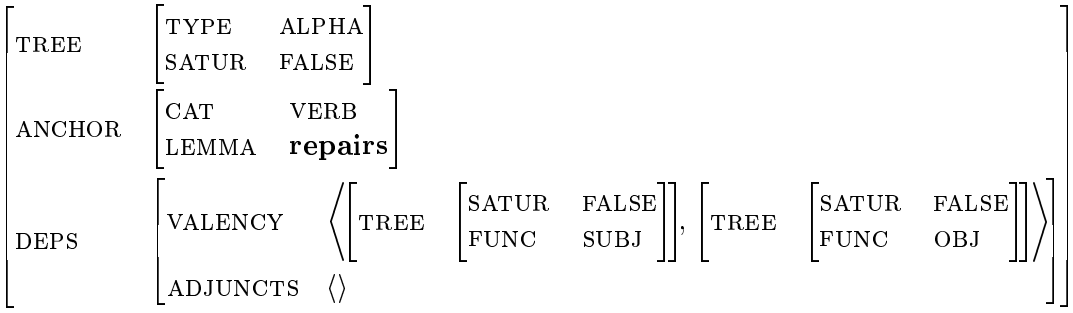

Figure 5: FS of a lexical element with a non-empty valency frame.

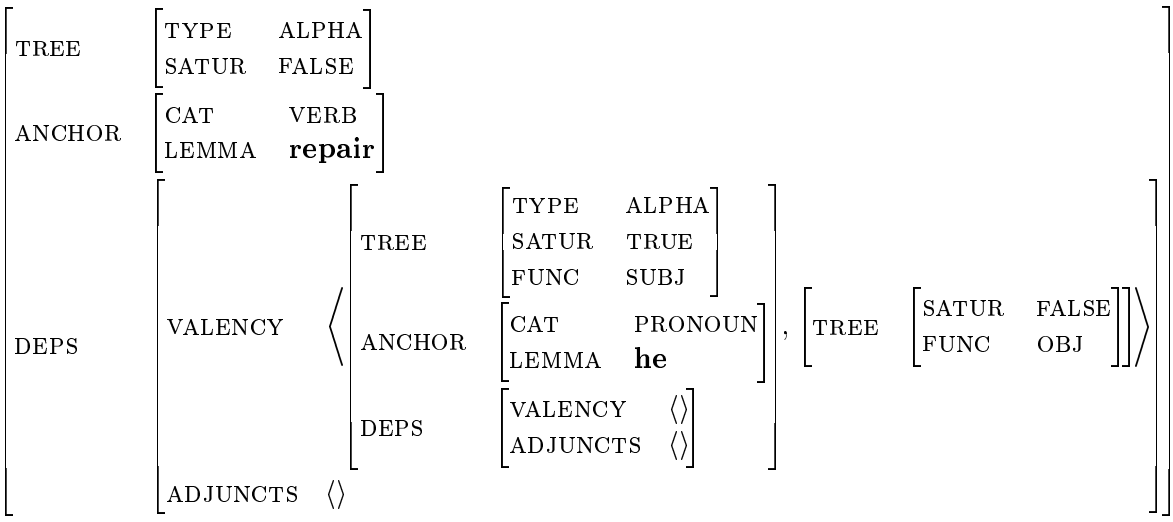

Figure 6: FS (i.e derived tree) obtained by (selective) unification of the FS at the subject slot of the verb with the FS of the dependent

one takes full advantage of the benefits of unification in general (extendability, declarativeness) and of long-standing efforts on efficiency optimization.

The *Attribute Logic Engine* (Carpenter and Penn, 1999) is a PS parser in which terms are typed feature structures, rather than categories. ALE also integrates a definite clause logic programming system. The latter is reminiscent of Definite Clause Grammar (DCG) (Pereira and Warren, 1980); but whereas in DCG definite clauses are Prolog goals using Prolog terms as their arguments, in ALE the arguments are typed feature structures. The ALE formalism defines the notation of the signature (i.e. the declaration of the typed feature hierarchy), lexicon entries, grammar rules, macros, definite clauses, as well as other components which will not be touched upon here.

ALE uses a bottom-up chart parser. When a word is read from the input, an edge with the FS for that word (or multiple edges, for an ambiguous word) is placed in the chart. This new edge is combined with adjacent edges on the basis of the grammar rules, before the next word is processed.

A CFG rule A $\rightarrow$ B C would be written as follows, where A, B, and C should be replaced by one or more constraints on the FS of the those elements.

```
name_of_rule rule
( A )
===>
cat> ( B ),
cat> ( C ).
```

For instance, if the category information is

stored at the path CAT, B would be replaced by `(cat:b)`, which corresponds to [CAT B].

If the chart contains an edge covering (i,j) and matching the constraints in B, and a second one covering (j,k) and matching the constraints in C, then a new edge covering (i,k) is created the FS of which will unify the constraints in A. The reserved word `cat>` indicates that its argument (B) is unified with the FS of an edge e_i in the chart. The edge consumed by the next `cat>` in the rule, will be to the right side of edge e_i and adjacent to it.

When dependency trees are represented as FS, the above procedure can be used to derive a larger tree from two smaller trees. Below is the simplified rule to combine an α-tree dependent with the (elementary or derived) tree to its right.

```
alpha_left rule
( DerivedTree )
===>
cat>  ( Dependent,
        @type(alpha) ),
cat>  ( Governor,
        @unsaturated ),
goal> ( unify_slot(Governor,Dependent,Frame),
        word_order(left,Governor,Dependent),
        derived_tree(alpha,left,Governor,
            Frame,DerivedTree) ).

beta_left rule
( DerivedTree )
===>
cat>  ( Dependent,
        @type(beta),
        @foot(Governor) ),
cat>  ( Governor ),
goal> ( word_order(left,Governor,Dependent),
        derived_tree(beta,left,Governor,_,
            DerivedTree) ).
```

`Dependent`, `Governor` and `DerivedTree` are variables (as indicated by the initial capital) and are unified with the FS of the corresponding edges. The macro `@type(alpha)` selects the appropriate path (TREE|TYPE) in the FS of `Dependent` and verifies that it is of the appropriate tree type.

The keyword `goal>` starts a definite clause section. The use of definite clauses stems from (1) the representation of the valency frame as a list, and (2) the fact that for some features values will be modified to reflect the properties of the derived tree. First, `unify_slot` looks for an open slot in the valency frame (of the governor) that matches (i.e. unifies with) the properties of the dependent. This is similar to substitution in TAG. If the constraints on word order are satisfied, a FS is constructed representing the derived tree on the basis of the FS of governor and dependent, and reflecting modifications to saturation as well as other features.

The second rule (for β dependents) differs only slightly from the previous one. When combining a β-tree with another tree, the foot features of the dependent (indicating the required properties of the head) is unified (shared variable `Governor`) with the tree of the governor. This is similar to insertion in TAG.

The two rules above have twins for the case where the dependent is to the right of the tree it combines with.

The rules above aren't rules of syntax, but merely implement a tree unification procedure. All syntactic information resides in the trees that are combined; the grammar is fully lexicalized.

The first rule (for α dependents) is to tree unification what the fundamental rule is to chart parsing. The saturation features perform the task which in chart parsing is performed by dotted rules. But whereas the latter impose a fixed order on the combination of elements, here valency terms can integrate the governing tree (so to speak) in any order, reducing the number of required elementary trees to a minimum.

The procedure implements bottom-up head-corner (bidirectional) active chart parsing on dependency trees. Analysis proceeds *bottom-up* because it amounts to the combination of elementary and derived trees, each of which originated from a lexical element. At no point predictions about the properties of an element are made. As a result there is no need for *filtering* techniques. It is *active* chart parsing because derived trees (i.e. edges) may or may not be saturated. The use of a chart also enables non-deterministic parsing, generating all parses of ambiguous sentences. It is *head-corner* parsing, since a dependent tree can only be combined with a tree (elementary or derived) containing its governor as the lexical anchor. With the ex-

ception of optional valency elements, optional elements appear as adjuncts (β-trees), and combinations are performed only when the optional element is encountered in the input, increasing efficiency. The procedure produces the same results irrespective of the direction (left-to-right or right-to-left) in which elementary trees are processed. To obtain incremental parsing it would be necessary to slightly modify the algorithm used in ALE.

If two adjacent words can not be combined in a dependency network, this will not halt parsing; there only will be multiple unconnected dependency networks in the chart. Parsing can work locally. Tree combination may be restricted to saturated dependents, but it need not be. By lifting this requirement, incomplete trees can be parsed as well. However, this considerably increases the number of edges in the chart and hence parsing time.

6 Conclusion

This paper briefly describes an approach to parsing for dependency grammar which builds on ideas and insights from various research areas. From TAG we adopt the representation of lexical elements as trees, tree derivation, node substitution, the distinction between α and β trees. However, the internal structure of trees is simplified, node properties are specified in functional rather than phrasal terms, and word order is treated independently. By applying the fundamental rule of chart parsing to dependency trees, tree derivation can be seen as a process that constructs saturated trees step by step, by combining adjacent parts into larger structures which get more saturated at each step. From unification grammar we inherit FS, the representation of the parse tree as a FS, and, of course, unification itself. Finally, ALE provides a powerful tool to test these ideas.

The parser has been tested on a small set of lexical elements consisting of nouns, adjectives, determiners, tensed verbs, pronouns, prepositions, adverbs, and relative pronouns. Our aim was to verify the possibility of parsing DG using standard unification and chart parsing. This research provides a powerful technique for tree unification (as opposed to tree adjoining and insertion). In the process of this work two additional tree types (other than α and β trees) were defined; they are not described here, but offer interesting properties both for parsing and linguistic analysis, and require only minimal modifications.

References

Bob Carpenter and Gerald Penn. 1999. *The Attribute Logic Engine. User's Guide (version 3.2 Beta).*

Jacques Courtin and Damien Genthial. 1998. Parsing with dependency relations and robust parsing. In Alain Kahane, Sylvain; Polguère, editor, *Processing of Dependency-based Grammars. Proceedings of the Workshop, COLING-ACL'98, Montreal*, pages 95–101.

Richard Hudson. 2000. Discontinuity. *Traitement automatique des langues*, 41:16–56.

A. Joshi. 1987. Introduction to tree adjoining grammar. In A. Manaster Ramer, editor, *The Mathematics of Language*, pages 87–114. Benjamins, Amsterdam.

Sylvain Kahane. 2001. Grammaires de dépendance formelles et théorie sens-texte. In *Proc. TALN2001*, volume 2, pages 17–76.

Vincenzo Lombardo and Leonardo Lesmo. 1996. An earley-type recognizer for dependency grammar. In *Proceedings COLING '96.*

Alexis Nasr. 1995. A formalism and a parser for lexicalised dependency grammars. In *4th International Workshop on Parsing Technologies, Prague 1995*, pages 186–195.

Alexis Nasr. to appear. A generative approach to parsing in the framework of the meaning-text theory. In Alain Polguére and Leo Wanner, editors, *Selected Aspects of Dependency Theory.* Benjamins Academic Publishers, Amsterdam, Philadelphia.

F. C. N. Pereira and 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*, 13:231–278.

Lucien Tesnière. 1959. *Eléments de syntaxe structurale.* Klincksieck, Paris.