An Earley-type recognizer for dependency grammar

Vincenzo Lombardo and Leonardo Lesmo
Dipartimento di Informatica and Centro di Scienza Cognitiva
Università di Torino
c.so Svizzera 185, 10149 Torino, Italy
e-mail: {vincenzo, lesmo}@di.unito.it

Abstract

The paper is a first attempt to fill a gap in the dependency literature, by providing a mathematical result on the complexity of recognition with a dependency grammar. The paper describes an improved Earley-type recognizer with a complexity $O(|G|^2 n^3)$. The improvement is due to a precompilation of the dependency rules into parse tables, that determine the conditions of applicability of two primary actions, *predict* and *scan*, used in recognition.

1 Introduction

Dependency and constituency frameworks define different syntactic structures. Dependency grammars describe the structure of a sentence in terms of binary *head-modifier* (also called *dependency*) *relations* on the words of the sentence. A dependency relation is an asymmetric relation between a word called head (governor, parent), and a word called modifier (dependent, daughter). A word in the sentence can play the role of the head in several dependency relations, i.e. it can have several modifiers; but each word can play the role of the modifier exactly once. One special word does not play the role of the modifier in any relation, and it is named the *root*. The set of the dependency relations that can be defined on a sentence form a tree, called the *dependency tree* (fig. 1a).

Although born in the same years, dependency syntax (Tesniere 1959) and constituency, or phrase structure, syntax (Chomsky 1956) (see fig.1b), have had different impacts. The mainstream of formalisms consists almost exclusively of constituency approaches, but some of the original insights of the dependency tradition have found a role in the constituency formalisms: in particular, the concept of head of a phrase and the use of grammatical relations.

The identification of the head within a phrase has been a major point of all the recent frameworks in linguistics: the X-bar theory (Jackendoff 1977), defines phrases as projections of (pre)terminal symbols, i.e. word categories; in GPSG (Gazdar et al. 1985) and HPSG (Pollard, Sag 1987), each phrase structure rule identifies a head and a related subcategorization within its right-hand side; in HG (Pollard 1984) the head is involved in the so-called head-wrapping operations, which allow the formalism to go beyond the context-free power (Joshi et al. 1991).

Grammatical relations are the primitive entities of relational grammar (Perlmutter 1983) (classified as a dependency-based theory in (Mel'cuk 1988)):

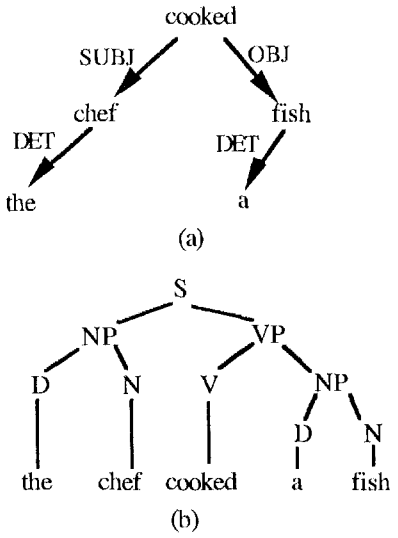

Figure 1. A dependency tree (a) and a p.s. tree (b) for the sentence "The chef cooked a fish". The leftward or rightward orientation of the arrows in the dependency tree represents the order constraints: the modifiers that precede the head stand on its left, the modifiers that follow the head stand on its right.

subject, *object*, *xcomplement*, ... label the dependency relations when the head is a verb. Grammatical relations gained much popularity within the unification formalisms in early 1980's. FUG (Kay 1979) and LFG (Kaplan, Bresnan 1982) exhibit mechanisms for producing a relational (or functional) structure of the sentence, based on the merging of feature representations.

All the recent constituency formalisms acknowledge the importance of the lexicon, and reduce the amount of information brought by the phrasal categories. The "lexicalization" of context-free grammars (Schabes, Waters 1993) points out many similarities between the two paradigms (Rambow, Joshi 1992). Dependency syntax is an extremely lexicalized framework, because the phrase structure component is totally absent. Like the other lexicalized frameworks, the dependency approach does not produce spurious grammars, and this facility is of a practical interest, especially in writing realistic grammars. For instance, there are no heavily ambiguous, infinitely ambiguous or cyclic dependency grammars (such as $S \rightarrow SS$; $S \rightarrow a$; $S \rightarrow \varepsilon$; see (Tomita 1985), pp. 72-73).

Dependency syntax is attractive because of the immediate mapping of dependency structures on the predicate-arguments structure (accessible by the semantic interpreter), and because of the treatment of free-word order constructs (Sgall et al. 1986) (Mel'cuk 1988) (Hudson 1990). A number of parsers have been developed for some dependency frameworks (Fraser 1989) (Covington 1990) (Kwon, Yoon 1991) (Sleator, Temperley 1993) (Hahn et al. 1994) (Lai, Huang 1995): however, no result of algorithmic efficiency has been published as far as we know. The theoretical worst-case analysis of $O(n^3)$ descends from the (weak) equivalence between projective dependency grammars (a restricted of dependency grammars) and context-free grammars (Gaifman 1965), and not from an actual parsing algorithm.

This paper is a first attempt to fill a gap in the literature between the linguistic merits of the dependency approach (widely debated) and the mathematical properties of such formalisms (quite negleted). We describe an improved Earley-type recognizer for a projective dependency formalism. As a starting point we have adopted a restricted dependency formalism with context-free power, that, for the sake of clearness, is described in the notation introduced by Gaifman (1965). The dependency grammar is translated into a set of parse tables that determine the conditions of applicability of the primary parser operations. Then the recognition algorithm consults the parse tables to build the sets of items as in Earley's algorithm for context-free grammars.

2 A dependency formalism

In this section we introduce a dependency formalism. We express the dependency relations in terms of rules that are very similar to their constituency counterpart, i.e. context-free grammars. The formalism has been adapted from (Gaifman 1965). Less constrained dependency formalisms exist in the literature (Mel'cuk 1988) (Fraser, Hudson 1992), but no mathematical studies on their expressive power exist.

A *dependency grammar* is a quintuple <S, C, W, L, T>, where

W is a finite set of symbols (vocabulary of words of a natural language),

C is a set of syntactic categories (preterminals, in constituency terms),

S is a non-empty set of root categories ($C \supseteq S$),

L is a set of category assignment rules of the form X: x, where $X \in C$, $x \in W$, and

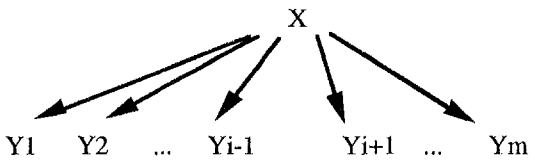

Figure 2 - A dependency rule: X is the governor, and $Y_1, ..., Y_m$ are the dependent of X in the given order (X is in # position).

T is a set of *dependency rules* of the form $X(Y_1\ Y_2\ ...\ Y_{i-1}\ \#\ Y_{i+1}\ ...\ Y_m)$, where $X \in C$, $Y_1 \in C$, $...,Y_m \in C$, and # is a special symbol that does not belong to C. (see fig. 2).

The modifier symbols Y_j can take the form Y_j*: as usual, this means that an indefinite number of Y_j's (zero or more) may appear in an application of the rule[1]. In the sample grammar below, this extension allows for several prepositional modifiers under a single verbal or nominal head without introducing intermediate symbols; the predicate-arguments structure is immediately represented by a one-level (flat) dependency structure.

Let $x=a_1 a_2 ... a_p \in W^*$ be a sentence. A *dependency tree* of x is a tree such that:

1) the nodes are the symbols $a_i \in W$ $(1 \le i \le p)$;

2) a node $a_{k,j}$ has left daughters $a_{k,1}, ..., a_{k,j-1}$ occurring in this order and right daughters $a_{k,j+1}, ..., a_{k,q}$ in this order if and only if there exist the rules $A_{k,1}$: $a_{k,1}, ..., A_{k,j}$: $a_{k,j}, ..., A_{k,q}$: $a_{k,q}$ in L and the rule $A_{k,j}(A_{k,1} ... A_{k,j-1}\ \#\ A_{k,j+1} ... A_{k,q})$ in T. We say that $a_{k,1}, ..., a_{k,j-1}, a_{k,j+1}, ..., a_{k,q}$ *directly depend* on $a_{k,j}$, or equivalently that $a_{k,j}$ *directly governs* $a_{k,1}, ..., a_{k,j-1}, a_{k,j+1}, ..., a_{k,q}$. $a_{k,j}$ and $a_{k,h}$ $(h = 1, ..., j-1, j+1, ..., q)$ are said to be in a *dependency relation*, where $a_{k,j}$ is the *head* and $a_{k,h}$ is the *modifier*; if there exists a sequence of nodes $a_i, a_{i+1}, ..., a_{j-1}, a_j$ such that a_k directly depends on a_{k-1} for each k such that $i+1 \le k \le j$, then we say that a_i *depends* on a_j;

3) it satisfies the *condition of projectivity* with respect to the order in x, that is, if a_i depends directly on a_j and a_k intervenes between them ($i<k<j$ or $j<k<i$), then either a_k depends on a_i or a_k depends on a_j (see fig. 3);

4) the root is a unique symbol a_s such that A_s: $a_s \in L$ and $A_s \in S$.

The condition of projectivity limits the expressive power of the formalism to be equivalent to the context-free power. Intuitively, this principle states

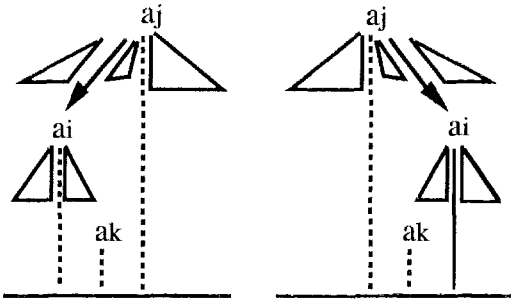

Figure 3. The condition of projectivity.

[1] The use of the Kleene star is a notational change with respect to Gaifman: however, it is not uncommon to allow the symbols on the right hand side of a rule to be regular expressions in order to augment the perspicuity of the syntactic representation, but not the expressive power of the grammar (a similar extension appears in the context-free part of the LFG formalism (Kaplan, Bresnan 1982)).

that a dependent is never separated from its governor by anything other than another dependent, together with its subtree, or by a dependent of its own.

As an example, consider the grammar

G_1= <{V},
{V, N, P, A, D},
{I, saw, a, tall, old, man, in, the, park, with, telescope},
{N: I, V: saw, D: a, A: tall, A: old, N: man, P: in, D: the,
N: park, P: with, N: telescope}
T1>,

where T1 is the following set of dependency rules:

1. V(N # P*); 2. V(N # N P*);
3. N(A*#P*); 4. N(DA* # P*);
5. P(# N); 6. A(#); 7. D(#).

For instance, the two rules for the root category V(erb) specify that a verb (V) can dominate one or two nouns and some prepositions (*).

3 Recognition with a dependency grammar

The recognizer is an improved Earley-type algorithm, where the predictive component has been compiled in a set of parse tables. We use two primary actions: *predict*, that corresponds to the top-down guessing of a category, and *scan*, that corresponds to the scanning of the current input word. In the subsection 3.1 we describe the data structures and the algorithms for translating the dependency rules into the *parse tables*: the dependency rules for a category are first translated into a *transition graph*, and then the transition graph is mapped onto a parse table. In the subsection 3.2 we present the Earley-type recognizer, that equals the most efficient recognizers for context-free grammar.

3.1 Transition graphs and parse tables

A *transition graph* is a pair (V, E), where V is a set of vertices called *states*, and E is a set of directed edges labelled with a syntactic category or the symbol #. Given a grammar G=<S, C, W, L, T>, a state of the transition graph for a category Cat ∈ C is a set of dotted strings of the form ". β", where Cat(αβ) ∈ T and α, β ∈ (C ∪ {#})*; an edge is a triple <s_i, s_j, Y>, where s_i, s_j ∈ V and Y ∈ C ∪ {#}. A state that contains the dotted string "." is called *final*; a final state signals that the recognition of one or more dependency rules has been completed. The following algorithm constructs the transition graph for the category Cat:

```
function graph (Cat, G):
initialization
s0 := ∅;
for each rule in G of the form Cat(α) do
        s0 := s0 ∪ {star ( .α )}
endfor;
V := {s0};
E := ∅;
expansion
repeat
   take a non-marked state s from V;
   mark s;
   for each category Y ∈ C ∪ {#} do
      s' := ∅;
      for each dotted string r = .Yβ in s do
         if Y is starred
            then s' := s' ∪ star(.Yβ)
            else s' := s' ∪ {.β};
      endfor each dotted string;
      V := V ∪ {s'};
      E := E ∪ {<s, s', Y>}
   endfor each category
until all states in V are marked;
graph := <V,E>.

function star (dotted-string):
set-of-strings:= {dotted-string};
repeat
   take a non-marked dotted string ds from set-of-strings;
   mark ds;
   if ds has the form ".Yβ" and Y is starred then
      set-of-strings := set-of-strings ∪ {".β"}
until all dotted strings in set-of-strings are marked
star:= set-of-strings.
```

The initial set of states consists of a single state s_0, that contains all the possible strings ".α", such that Cat(α) is a dependency rule. Each string is prefixed with a dot. The marked states are the states that were expanded in a previous step. The expansion of a state s takes into account each symbol Y that immediately follows a dot (Y ∈ C ∪ {#}). Y is a possible continuation to a new state s', that contains the dotted string ".β", where ".Yβ" is a dotted string in s. s' is added to the set of states, and a new edge from s to s' labelled with Y is added to the set of edges. A dotted string of the form .Y*β is treated as a pair of dotted strings {.Y*β, .β}, so to allow a number of iterations (one or more Y's follow) or no iteration (the first symbol in β follows) in the next step. The function "star" takes into account these cases; the *repeat* loop accounts for the case when the first symbol of β is starred too.

The transition graphs obtained for the five categories of G_1 are in fig. 4. Conventionally, we indicate the non-final states as h and the final states as $k, where h and k are integers.

The total number of states of all the transition graphs for a grammar G is at most O(|G|), where |G| is the sum of the lengths of the dependency rules. The length of a dependency rule Cat(α) is the length of α. Starting from the transition graph for a category Cat, we can build the *parse table* for Cat, i.e. PT_{Cat}. PT_{Cat} is an array h × k, where h is the number of states of the transition graph and k is the number of syntactic categories in C. Each row is identified by a pair <Cat, State>, where State is the label of a state of the corresponding transition graph; each column is associated with a syntactic category. In order to improve the top-down algorithm we introduce the concept of "first" of a category. The first of the category Cat is the set of categories that appear as leftmost node of a subtree headed by Cat. The first of a category X is computed by a simple procedure that we omit here. The function *parse_table* computes the parse tables of the various categories. E(t-graph$_{Cat}$) returns the set of the edges of the graph t-graph$_{Cat}$. The contents of the entries in the parse tables are sets (possibly empty) of *predict* and *scan*. The initialization step consists in setting all entries of the table to the empty set.

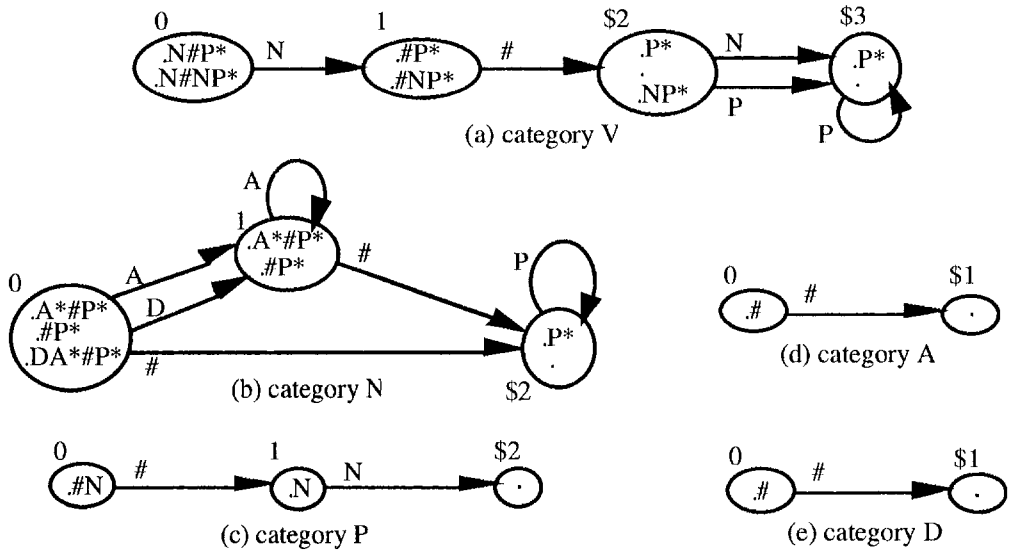

Fig. 4 - The transition graphs obtained for the grammar G_1.

<cat,state> \ input cat	V	N	A	P	D
<V,0>		<predict(N),1>	<predict(N),1>		<predict(N),1>
<V,1>	<scan,$2>				
<V,$2>		<predict(N),$3>	<predict(N),$3>	<predict(P),$3>	<predict(N),$3>
<V,$3>				<predict(P),$3>	
<N,0>		<scan,$2>	<predict(A),1>		<predict(D),1>
<N,1>		<scan,$2>	<predict(A),1>		
<N,$2>				<predict(P),$2>	
<P,0>				<scan,1>	
<P,1>		<predict(N),$2>	<predict(N),$2>		<predict(N),$2>
<A,0>			<scan,$1>		
<D,0>					<scan,$1>

Figure 5 - The parse tables for the grammar G_1

```
function parse-table (Cat, t-graph_Cat):
initialize PT_Cat;
for each edge <s, s', Y> in E(t-graph_Cat) do
  if Y ∈ C then
    for each category Z ∈ first(Y) do
      PT_Cat(<Cat, s>, Z) := PT_Cat(<Cat, s>, Z) ∪
                             PT_Cat(<Cat, s>, Z) ∪
                             {<predict(Y), s'>}
  elseif Y = # then
    PT_Cat(<Cat, s>, Cat) := PT_Cat(<Cat, s>, Cat) ∪
                             {<scan, s'>}
  endif;
endfor
parse-table := PT_Cat.
```

The parse tables for the grammar G_1 are reported in fig. 5. The firsts are: first(V)=first(N)={N, A, D}; first(P)={P}; first(A)={A}; first(D)={D}. Note that an entry of a table can contain more than one action, although this does not happen for our simple grammar G_1.

3.2 A dependency recognizer

The dependency recognizer exhibits the same data structures of Earley's recognizer (Earley 1970), but improves the performance of that algorithm because of the precompilation of the predictive component into the parse tables.

In order to recognize a sentence of n words, n+1 sets S_i of items are built. An item is a quadruple

<Category, State, Position, Depcat>

where the first two elements (Category and State) correspond to a row of the parse table $PT_{Category}$, the third element (Position) gives the index i of the set S_i where the recognition of a substructure began, and the fourth one (Depcat) is used to request the completion of a substructure headed by Depcat,

before continuing in the recognition of the larger structure headed by Category (Depcat = "_" means that the item is not waiting for any completion).

```
Sentence: w0 w1 ... wn-1
initialization
for each root category V do
    INSERT <V, 0, 0, _> into S0
endfor
body
for i from 0 to n do
  for each item P=<Cat, State, j, _> in Si do
    completer:
    if final(State) then
      for each item <Cat', k, j', Cat> in Sj do
        INSERT <Cat', k, j', _>into Si
      endfor
    endif
    predictor:
    if <predict(Cat'), State'>∈ PTCat(<Cat, State> ×
                                        Inputcat) then
      INSERT <Cat', 0, i, _> into Si;
      INSERT <Cat, State', j, Cat'>into Si
    endif
    scanner:
    if <scan, State'> ∈ PTCat(<Cat, State> ×
                                    Inputcat) then
      INSERT <Cat, State', j, _> into Si+1
    endif
  endfor each item
endfor;
termination
if <V,$k, 0, _> is in Sn then accept else reject endif.
```

The external loop of the algorithm cycles on the sets S_i ($0 \le i \le n$); the inner loop cycles on the items of the set S_i of the form <Cat, State, j, _>. At each step of the inner loop, the action(s) given by the entry "<Cat, State> × Inputcat" in the parse table PT_{Cat} is(are) executed (where Inputcat is one of the categories of the current word). Like in Earley's parser there are three phases: *completer*, *predictor* and *scanner*.

completer: When an item is in a final state (of the form $h), the algorithm looks for the items which represent the beginning of the input portion just analyzed: they are the four-element items contained in the set referred by j. These items are inserted into S_i after having set to "null" the fourth element (_).

predictor: "<predict(Cat'), State'>" corresponds to a prediction of the category Cat' as a modifier of the category Cat and to the transition to State', in case a substructure headed by Cat' is actually found. This is modeled by introducing two new items in the set:

a) <Cat', 0, i, _>, which represents the initial state of the transition graph of the category Cat' which will span a portion of the input starting at i. In Earley's terms, this item corresponds to all the dotted rules of the form Cat'(. α).

b) <Cat, State', j, Cat'>, which represents the arc of the transition graph of the category Cat, entering the state State' and labelled Cat'. In Earley's terms, this item corresponds to a dotted rule of the form Cat(α . Cat' β). The items including a non-null Depcat are just passive receptors waiting to be re-activated later when (and if) the recognition of the hypothesized substructure has successfully completed.

scanner: "<scan, State'>" results in inserting a new item <Cat, State', i, _> into the set S_{i+1}.

Let us trace the recognition of the sentence "I saw a tall old man in the park with a telescope". The first set S_0 (fig. 6) includes three items: the first one, <V, 0, 0, _>, is produced by the initialization; the next two, <V, 1, 0, N> and <N, 0, 0, _> are produced by the predictor (a N-headed subtree beginning in position 0 must be recognized and, in case such a recognition occurs, the governing V can pass to state 1).

In S_1 the first item <N, $2, 0, _> is produced by the scanner: it is the result of advancing on the input string according to the item <N, 0, 0, _> in S_0 with an input noun "I" (the entry in the parse table PT_N <N, 0> × N contains <scan,$2>). The next item, <V, 1, 0, _> is produced by applying the completer to the item in S_0 <V, 1, 0, N>.

S_2 contains the item <V, $2, 0, _>, obtained by the scanner, that advances on the verb "saw". The other four items are the result of a double application of the predictor, which, in a sense, builds a "chain" that consists of a noun governed by the root verb and of a determiner governed by that noun; this is the only way, according to the grammar, to accomodate an incoming determiner when a verb is under analysis. The subsequent steps can easily be traced by the reader. The input sentence is accepted because of the appearance in the last set of the item <V, $3, 0, _>, encoding that a structure headed by a verb (i.e. a root category), ending in a final state ($3), and covering all the words from the beginning of the sentence has been successfully recognized.

The space complexity of the recognizer is O(|G| n^2). Each item is a quadruple <Cat, State, Position, Depcat>: Depcat is a constant of the grammar; the pairs of Cat and State are bounded by O(|G|);

S_0 [I]
<V, 0, 0, _>
<V, 1, 0, N>
<N, 0, 0, _>

S_1 [saw]
<N, $2, 0, _>
<V, 1, 0, _>

S_2 [a]
<V, $2, 0, _>
<V, $3, 0, N>
<N, 0, 2 _>
<N, 1, 2, D>
<D, 0, 2, _>

S_3 [tall]
<D, $1, 2, _>
<N, 1, 2, _>
<A, 0, 3, _>
<N, 1, 2, A>

S_4 [old]
<A, $1, 3, _>
<N, 1, 2, _>
<A, 0, 4, _>
<N, 1, 2, A>

S_5 [man]
<A, $1, 4, _>
<N, 1, 2, _>

S_6 [in]
<N, $2, 2, _>
<V, $3, 2, _>
<P, 0, 6, _>
<V, $3, 0, P>
<N, $2, 2, P>

S_7 [the]
<P, 1, 6, _>
<N, 0, 7, _>
<P, $2, 6, N>
<D, 0, 7, _>
<N, 1, 7, D>

S_8 [park]
<D, $1, 7, _>
<N, 1, 7, _>

S_9 [with]
<N, $2, 7, _>
<P, $2, 6, _>
<N, $2, 2, _>
<V, $3, 0, _>
<P, 0, 9, _>
<N, $2, 7, P>
<N, $2, 2, P>
<V, $3, 0, P>

S_{10} [a]
<P, 1, 9, _>
<N, 0, 10, _>
<P, $2, 9, N>
<D, 0, 10, _>
<N, 1, 10, D>

S_{11} [telescope]
<D, $1, 10, _>
<N, 1, 10, _>

S_{12}
<N, $2, 10, _>
<P, $2, 9, _>
<N, $2, 7, _>
<N, $2, 2, _>
<V, $3, 0, _>
<P, $2, 6, _>

Figure 6. Sets of items generated in recognizing "I saw a tall old man in the park with a telescope".

Position is bounded by O(n). The number of such quadruples in a set of items is bounded by O(|G| n) and there are n sets of items.

The time complexity of the recognizer is $O(|G|^2 n^3)$. The phases *scanner* and *predictor* execute at most O(|G|) actions per item; the items are at most $O(|G| n^2)$ and the cost of these two phases for the whole algorithm is $O(|G|^2 n^2)$. The phase *completer* executes at most one action per pair of items. The variables of such a pair of items are the two states ($O(|G|^2)$), the two sets that contain them ($O(n^2)$), and the two positions ($O(n^2)$). But the pairs considered are not all the possible pairs: one of the sets has the index which is the same of one of the positions, and the complexity of the *completer* is $O(|G|^2 n^3)$. The phase *completer* prevails on the other two phases and the total complexity of the algorithm is $O(|G|^2 n^3)$. Even if the O-analysis is equivalent to Earley's, the phase of precompilation into the parse tables allows to save a lot of computation time needed by the predictor. All the possible predictions are precomputed in the transition to a new state. A similar device is presented in (Schabes 1990) for context-free grammars.

4 Conclusion

The paper has described a recognition algorithm for dependency grammar. The dependency formalism is translated into parse tables, that determine the conditions of applicability of the parser actions. The recognizer is an improved Earley-type algorithm, whose performances are comparable to the best recognizers for the context-free grammars, the formalism which is equivalent to the dependency formalism described in this paper. The algorithm has been implemented in Common Lisp and runs under the Unix operating system. The next step in our research will be to relax the condition of projectivity in order to improve the expressive power and to deal with phenomena that go beyond the context-free power. These changes imply the restructuring of some parts of the recognizer, with a plausible increment of the complexity.

References

Chomsky N., Three models for the description of language, *IRE Transactions on Information Theory, IT-2*, 1956, 113-124.

Covington M. A., Parsing Discontinuous Constituents in Dependency Grammar, *Computational Linguistics 16*, 1990, 234-236.

Covington M. A., An Empirically Motivated Reinterpretation of Dependency Grammar, Res. Rep. AI-1994-01, Univ. of Georgia (also on CompLing Server), 1994.

Earley J., An Efficient Context-free Parsing Algorithm. *Comm. of the ACM 13* , 1970, 94-102.

Fraser N.M., Parsing and Dependency Grammar, UCL Working Papers in Linguistics, 1989, 296-319.

Fraser N.M., Hudson R. A., Inheritance in Word Grammar, *Computational Linguistics 18*, 1992, 133-158.

Gaifman H., Dependency Systems and Phrase Structure Systems, *Information and Control 8*, 1965, 304-337.

Gazdar G., Klein E., Pullum G., Sag I., *Generalized Phrase Structure Grammars*, Basil Blackwell, Oxford, 1985.

Graham S. L., Harrison M. A., Ruzzo W. L., An improved Context-Free Recognizer, *ACM Trans. on Programming Languages and Systems 2*, 1980, 415-462.

Hahn U., Schacht S., Broker N., Concurrent, Object-Oriented Natural Language Parsing: The ParseTalk Model, CLIF Report 9/94, Albert-Ludwigs-Universitat, Freiburg, Germany.

Hudson R., *English Word Grammar*, Basil Blackwell, Oxford, 1990.

Jackendoff R., *X-bar Syntax: A Study of Phrase Structure*, MIT Press, 1977.

Jacobs P.S., Rau L. F., Innovations in Text Interpretation, *Artificial Intelligence Journal 63/1-2*, 1993, 143-191.

Joshi A.K., Vijay-Shanker K., Weir D., The Convergence of Mildly Context-sensitive grammatical formalisms, in Sells P., Shieber S., Wasow T. (eds.), *Foundational Issues in Natural Language Processing*, MIT Press, 1991.

Kaplan R., Bresnan J., Lexical-Functional Grammar: A Formal System for Grammatical Representation, in Bresnan J. (ed.), *The mental representation of grammatical relations*, MIT Press, 1982.

Kay M., Functional Grammar, *Proc. 5th Meeting of the Berkeley Linguistic Society*, 1979, 142-158.

Kwon H., Yoon A., Unification-Based Dependency Parsing of Governor-Final Languages, *Proc. IWPT 91*, 1991, 182-192.

Lai B.Y.T., Huang C., Dependency Grammar and the Parsing of Chinese Sentences, Unpublished Manuscript on CompLing Server, 1995.

Mel'cuk I., *Dependency Syntax: Theory and Practice*, SUNY Press, Albany, 1988.

Perlmutter 1983, *Studies in Relational Grammar 1*, Univ. of Chicago Press, Chicago, 1983.

Pollard C.J., *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*, Ph.D. Thesis, Stanford Univ., 1984.

Pollard C.J., Sag I., *An Information Based Syntax and Semantics, vol.1, Fundamentals*, CSLI Lecture Note 13. CSLI, Stanford, 1987.

Rambow O., Joshi A., A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena, *Proc. of the Int. Workshop on the Meaning-Text Theory*, Darmstadt, 1992.

Schabes Y., Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-Free Grammars, *Proc. ACL 90*, Pittsburgh (PA), 1990, 106-113.

Schabes Y., Waters R. C., Lexicalized Context-Free Grammars, *Proc. ACL 93*, 121-129.

Sgall P., Haijcova E., Panevova J., *The Meaning of Sentence in its Semantic and Pragmatic Aspects*, D.Reidel Publ. Co., Dordrecht, 1986.

Sleator D. D., Temperley D., Parsing English with a Link Grammar, *Proc. of IWPT 93*, 1993, .277-291.

Tesniere L., *Elements de Syntaxe Structurale*, Kliensieck, Paris, 1959.

Tomita M., *Efficient Parsing for Natural Language*, Kluwer Acad. Publ., 1985.