Snelling
1973

A. Günther et al. (eds.), International Computing Symposium 1973
© North-Holland Publ. Co., 1974.

GENERAL CONTEXT-FREE PARSING IN TIME N^2.

Michel Snelling
MBLE Research Laboratory, Brussels, Belgium.
(Now at the University of Lille, France)

This paper presents a parser, which accepts any context-free grammar in BNF notation and works in a time proportional to n^2 in worst cases. Its general strategy is of a predictive type, like Earley's algorithm but a different organization and use of informations already obtained permits a better treatment of recursivities and ambiguities.

1. INTRODUCTION

Among the numerous general context-free parsers that have been described, Earley's algorithm seems to be the more efficient for time and space (Earley [1], [2]). But, in worst cases, it still works in a time proportionnel to n^3, if n is the length of the input string. This paper presents a parser, which accepts any context-free grammar in BNF notation and works in time n^2 in worst case. Its general strategy is of a predictive type, like Earley's algorithm, but a different organization and use of informations already obtained permits a better treatment of recursivities and ambiguities, in particular for grammars with unbounded direct ambiguity for which Earley's parsers runs in time n^3.

New definitions for "state" and especially "state set" are used. New types of state sets are introduced : constructed step by step as parsing progresses, they obtain all informations related to the recognition of a right recursive derivation and are such that their processing by different completer cannot generate twice the same state except for at most one of them. The other features of the algorithm are the way in which state sets are constructed and ordered, and the recursive action of completers processing state sets and cutting out redundant informations.

Although logically complete, this paper is easier to read if one is familiar with Earley's papers or at least with the informal explanation they present. In section 2 basic terminalogy is introduced. Section 3 describe the algorithm, which is proved correct in section 4. Section 5 is devoted to theorical results : time n^2 in general, time n for LR(k) grammars without using a "lookahead" in states. Section 6 gives several examples.

2. DEFINITIONS AND NOTATIONS

2.1. Context-free grammars.

A context-free grammar is a quadruple $G=(V,T,P,R)$ where V is a finite sets of symbols, the vocabulary ; $T\subset V$ is the set of terminal symbols, $N=V-T$ is the set of nonterminal symbols, $R\in N$ is a distinguihed nonterminal called the root of the grammar and P is a finite set of production rules written :

$$D_p \to C_p^1 \dots C_p^{\bar{p}} \qquad 1\leq p\leq d$$

with $D_p \in N$ and $C_p^j \in V$ for $1\leq j\leq \bar{p}$

If α and β are strings in V^* then $\alpha\Rightarrow\beta$ expresses that there exist $\gamma,\delta,\eta\in V^*$ and $A\in N$ such that $\alpha=\gamma A\delta$, $\beta=\gamma\eta\delta$ and $A\to\eta$ is a production in P. The symbol $\overset{*}{\Rightarrow}$ denotes the transitive closure of $\Rightarrow$.

A sentential form is a string $\alpha\in V^*$ such that $R\overset{*}{\Rightarrow}\alpha$. A sentence is a sentential form $\alpha\in T^*$. The language L(G) defined by a grammar G is the set of its sentences.

Λ denotes the empty string, $E=\{A\in N,\ A\overset{*}{\Rightarrow}\Lambda\}$. The input string will be written $X_1\dots X_n$. Let $\vdash$ be a new terminal ($\vdash\notin T$) and D_0 a new nonterminal ($D_0\notin N$) : the rule $D_0\to R\vdash$ is added to P and $X_1\dots X_n$ is transformed into $X_1\dots X_n\vdash$ (the modified input sentence for the modified grammar).

2.2. States

For the description of the different steps of the parse the algorthim builts states. A state is a triple $<p,j,f>$ where p, j and f are integers

$p(0 \leqslant p \leqslant d)$ is the number of a grammar rule

$j(0 \leqslant j \leqslant \bar{p})$ indicates a position in the rule's right-hand side

$f(0 \leqslant f \leqslant n)$ indicates a position in the input string.

Whenever it is not necessary to distinguish states $<p,j,f>$ with the same values for p and j, we shall also use another type of state : $<p,j,F>$ with the same meaning for p and j and where $F=\{f_i, i=1,2,\ldots,k \mid 0 \leqslant f_i \leqslant n\}$ is a set of positions in the input string.

2.3. State Sets

Although we could also consider that $<p,j,F>$ is a state set, we shall define as state sets :

S the set of all states used by the algorithm,

S_i, $0 \leqslant i \leqslant n+1$ an ordered subset of $S(S=\bigcup_{0 \leqslant i \leqslant n+1} S_i)$,

If $F=\{f_1,\ldots,f_k\}$, $S_F=S_{f_1} \cup \ldots \cup S_{f_k}$

$\forall D_p \in N$, $S_i^p=\{<p,j,f> \in S_i \mid C_p^{j+1}=D_p\}$,

$S_i^T=\{<p,j,f> \in S_i \mid C_p^{j+1} \in T\}$,

$S_i^X=\{<p,j,f> \in S_i \mid j=\bar{p}\}$, these sets being ordered by decreasing f, (and similarly for S_F^p, S_F^T, S_F^X),

$\bar{S}_F^p$ the set of all states that a completer called by a state $<p,\bar{p},F>$ will have to process recursively : let $S_1(F,p)=\bigcup_{i\in F} S_i^{D_p}$ and

$$S_2(F,p)=S_1(F,p)\cup\left(\bigcup_{<p',\bar{p}-1,F'>\in S_1(F,p)} S_2(F',p')\right).$$

Defined like this $S_2(F,p)$ is not a state set but a set of state sets : therefore one or more of its elements can contain the same states. $\bar{S}_F^p$ will be constructed step by step in the algorithm from $S_2(F,p)$ by cutting out all unnecessary states as soon as it finds them. This will be completed in 3.1.5.

3. ALGORITHM

3.1. The recognizer

3.1.1. Initialisation : start with the state $<0,0,0>$ in S_0, all nonterminals unmarked and $i=0$.

3.1.2.: E : to every state $<p,j,F>$ in S_i apply one of the following operations :

If $C_p^{j+1} \in N$ then PREDICTOR : if C_p^{j+1} is an unmarked nonterminal then for any r such that $D_r=C_p^{j+1}$ add $<r,0,i>$ to S_i and mark the nonterminal.

If $C_p^{j+1} \in T$ then SCANNER : if $C_p^{j+1}=X_{i+1}$ add $<p,j+1,F>$ to S_{i+1}.

If $j=\bar{p}$ then COMPLETER : for all states $<p',j',f'>$ in $\bar{S}_F^p$: there are five possibilities :

1) $j'+1=\bar{p}$ and a state $<p',j'+1,f''>$ with $f''>f'$ has been added to S_i by the same completer call : there is a right recursivity and the state $<p',j',f'>$ can be deleted from $\bar{S}_F^p$.

2) a state $<p',j'+1,f'>$ has been already added to S_i (there is an ambiguity) : if it is by the same completer call then the state $<p',j',f'>$ can be deleted from $\bar{S}_F^p$.

3) $j'+1=\bar{p}$ and there exist in S_i a state $<p'',\bar{p}'',f'>$ such that $D_{p'}=D_{p''}$: there is also an ambiguity.

4) a state $<p',j'+1,F''>$, such that $f' \notin F''$, is already in S_i : change it in $<p',j'+1,F''\cup\{f'\}>$ (it is in that way that the states of the type $<p,j,F>$ are constructed).

5) in any other case add $<p',j'+1,f'>$ to S_i.

When there is no more state to process in S_i, unmark all nonterminals, add one to i and if $i \leqslant n+1$ return to E.

3.1.3. Termination : if $S_{n+1}=\{<0,2,0>\}$ then $X_1\ldots X_n \in L(G)$. If there exist i, $0 \leqslant i \leqslant n+1$ such that $S_i=\emptyset$ then $X_1\ldots X_n \notin L(G)$.

3.1.4. Case of empty rules : if the grammar contains empty rules, then if a state $<p,j,F>$ such that $C_p^{j+1} \in E$ is added to S_i, then the state $<p,j+1,F>$ must be added to S_i at the same time. (The set E can be constructed once for all when the grammar is given).

3.1.5. Remarks

1) We can now complete the definition of $\bar{S}_F^p$: it is the set (which can sometimes contain twice or more the same element) of all states in the elements of $S_2(F,p)$ less those that are deleted in cases 1 and 2 of the completer.

2) Case 2 of the completer permits to delete from $\bar{S}_F^p$ all the states which, when completed, could create more than once the same state.

3) Case 1 of the completer treats right recursivities : in that case each time the last state in the recursivity (the one with the higher f) is completed, all the others are automatically completed too. They are not usefull and the corresponding states can be deleted from $\bar{S}^p_F$, which keeps only the "wayouts" of the recursivity.

4) Ordering of S_i ensure that the states in a right recursivity are processed in order, from the last one.

5) Bouckaert et al. [4], [5] studies the more efficient way to use the context in this type of algorithm. The method given can be applied straighforward here.

3.2. The analyser

To transform the recognizer of last paragraph, into a parser, one just needs to add a procedure which construct the tree in the same time. Earley [1] gives a simple way to obtain this. It can be easily adapted to this algorithm. However we must say that in case of very ambiguous grammars, we get only a "factorized tree" from which all the different trees can be obtained. This is quite suffisant in practice, so much more as some grammars can give an infinite number of derivations, even for empty strings.

3.3. Implementation

A detailed description of the implementation would be tedious, so we shall give only a sketch of it. Based mainly upon lists, it is similar to the one given in Earley [1], but with a few differences :

1) the sets S^p_i $(0 \leq p \leq d)$, S^T_i and S^X_i are organized in different lists (instead of one list for S_i).

2) the pointer i created by the predictor processing $\langle p,j,F \rangle \in S_i$ is in fact a pointer towards the list S^r_i. In that way we get very simply the structure of the list $\bar{S}^p_F$ with a list of these pointers for the different elements of F.

3) to delete a state we just remove one element from a list.

4) the case of empty rules is simplified by the use of E.

5) although its definition is more complicated, the completer operation does not take much more time : it is a recursive function organized as a case instruction.

4. PROOF OF CORRECTNESS

4.1. Lemma 1 : *If* $\langle p,j,F \rangle \in S_i$ *then* $\forall f \in F$:

$$f \leq i \text{ and } C^1_p \ldots C^j_p \overset{*}{\Rightarrow} X_{f+1} \ldots X_i .$$

4.2. Lemma 2 : *If* $\langle p,j,F \rangle \in S_i$ *and* $C^{j+1}_p \ldots C^{j+x}_p \overset{*}{\Rightarrow} X_{i+1} \ldots X_g$ $(j+x \leq \bar{p}, g \leq n)$ *then the state* $\langle p,j+x,F \rangle$ *will be added to* S_g *if it is not an element of a right recursivity.*

4.3. Theorem 1 : $R \overset{*}{\Rightarrow} X_1 \ldots X_n$ *if and only if* $S_{n+1} = \{\langle 0,2,0 \rangle\}$.

The proofs are very simple and similar to the ones of Bouckaert et al. [4]. However we must also use the rather obvious property that the deletions does not modified the states created : they only prevents the creation of more than once the same state in case 2 and of useless states in case 1.

5. RECOGNITION TIME

It is natural to take for unit of time an operation which is independant of the grammar and of the string to parse. This choice is implementation dependant : we shall not discuss the problem here. As our implementation is basically similar to Earley's, his discussion holds unchanged and we can take as "basic step" the generation of a state, i.e. the action of adding a state to S or attempting to add one which is not necessary (for example, in a completer call the number of steps is the number of elements of $\bar{S}^p_F$).

The following notations will be used :

$m = \max \bar{p}$, $0 \leq p \leq d$

C is a constant independant of the length of the string

$\sim i$ means proportionally to i, that is to say a quantity which can grow with i.

5.1. Theorem 2 : *the time required to recognize any sentence of length n as the member of the language generated by any given C-F grammar is bounded by* Cn^2.

Proof Let us prove that the number of states generated by the algorithm is bounded by Cn^2.

There are n state sets, less if the sentence does not belong to the language.

In each state set the predictor generates at most d states, one for each rule, since, at step i, the predictor adds states of the form $<p,0,\{i\}>$ and only once for each non-terminal.

In each state set the scanner generates at most dm states since, at step i, it processes only the states of S_i and, if $C_p^{j+1}=X_{i+1}$, just modifies j, without taking care of F.

In these two cases, the presence of empty rules can involve the generation of at most dm other states.

In each state set there are at most d completer main calls : by the states $<p,j,F>$ created at step i-1 by the scanner and for which $j=\bar{p}$. Two cases must be considered :

1) $F=\{f_1,\dots,f_{k1}\}$ with $k1 \sim i$ (as for UBDA 2) : the completer must process all states in $S^p_{f_{k1}},\dots,S^p_{f_1}$.

2) k1 is not $\sim i$ (for example 1 as for UBDA1) but at least one of the states $<p',j',F'>\in S^p_{f1}$ is such that $j'+1=\bar{p}'$ (or $C_{p'}^{j'+1}\dots C_{p'}^{\bar{p}} \overset{*}{\Rightarrow} \Lambda$) : then it calls immediately the completer, and so on. In that case the completer can also process $\sim i$ state sets : it adds recursively to S_i $<p,\bar{p}, f_{k1}\}>,\dots,<p,\bar{p}, f_1>$. Case 1 and 2 can occur in the same time. Anyway a state set is only processed once : S^p_f is processed by the completer called by the first state of the form $<p,\bar{p}, f>$ in S_i and the other states of this form in S_i just denote ambiguities.

Therefore a completer main call can process, at most once, k1 ($\sim i$) state sets. In these state sets, and even in each of them, there can exist states of the form $<p',j',F'>$ with $F'= f'_1,\dots,f'_{k2}\}$ and $k2 \sim i$(Sf contains at most dmf states).

In that case the completer main call can generate k1 k2 states, i^2 if k1 and k2 are i. In any other case the number of states generated is bounded by dmi and the theorem holds. The problem is therefore to prove that the number of states really generated is in fact $\sim i$, because $\sim i$ state sets have already been completed together, or that the number of such calls is bounded independantly of i.

$<p,\bar{p},F>\in S_i$ with $F= f_1,\dots,f_{k1}$ implies by lemma1

$$D_p \overset{*}{\Rightarrow} X_{f_1+1}\dots X_i$$
$$\vdots$$
$$D_p \overset{*}{\Rightarrow} X_{f_{k1}+1}\dots X_i$$

As the number of rules and non terminals is bounded but not k1, this implies a recursivity in the derivation of D_p : there must exist $D_q \in N$ such that $D_p \overset{*}{\Rightarrow} \alpha_1 D_q \alpha_2$ and $D_q \overset{*}{\Rightarrow} \alpha_3 D_q \alpha_4$. In the same way $<p',j',F'>\in S_\ell, \ell=f_1,\dots,f_{k1}$ with $F'=\{f'_1,\dots,f'_{k2}\}$ implies by lemma 1

$$C^1_{p'}\dots C^{j'}_{p'} \overset{*}{\Rightarrow} X_{f'_1+1}\dots X_\ell$$
$$\vdots \qquad \ell=f_1,\dots,f_{k1}.$$
$$C^1_{p'}\dots C^{j'}_{p'} \overset{*}{\Rightarrow} X_{f'_{k2}+1}\dots X_\ell$$

and there is a recursivity in the derivation of $C^1_{p'}\dots C^{j'}_{p'}$.

At each step of this last recursivity a state $<p,0,f>$ is generated. In full generality it is for a subset of F, but a subset containing $\sim i$ elements, so we can suppose here it is F itself. In that way, as the derivation of D_p is recursive and if the input string matches, we can obtain in S a state of the form $<p,\bar{p},F>$, each time D_p is recognized. If this happens more than once, the first completion of $\bar{S}^p_F$ can generate i^2 states, but the other ones only $\sim i$ by construction of the completer (In the cases of UBDA1 and UBDA2 where F gets one more element at each step, the problem remains the same since the set F of step i is the set of step i-1, which has already been completed, plus one element).

The problem is thus to show that there cannot exist $\sim i$ different such recursivities of $\sim i$ steps in the derivation of a string of length i (which could lead to $\sim i$ F's differing by $\sim i$ elements for the different steps).

As there is at most d non-terminals we can res-

trict ourselves to one : suppose $D_p \overset{*}{\Rightarrow} \alpha D_p \alpha'$ with $\alpha \in V^*$. We can suppose $\alpha \neq \Lambda$: in the case of empty rules there should exist nonterminals derivable as empty rules in as many different ways as we want : it is possible (for example $A \to AA$, $A \to \Lambda$) but generates at most dm states since they are all generated at the same step with the same f. Therefore to have $\sim i$ different recursivities the string generated by α must have a length growing with i and there cannot exist $\sim i$ steps of recursivity in a string of length i.
This achieves the proof.

5.2. Theorem 3 : *the time required to recognize any sentence of length n as the member of the language generated by an LR grammar is bounded by Cn .*

Proof : as in Lewis and Stearns [3] we say that G is LR(k) if it is unambiguous and if for all $w_1, w_2, w_3, w_3' \in T^*$, $A \in N$,
$R \overset{*}{\Rightarrow} w_1 A w_2$, $A \overset{*}{\Rightarrow} w_2$, $R \overset{*}{\Rightarrow} w_1 w_2 w_3'$ and $k{:}w1 = k{:}w_3'$ imply $R \overset{*}{\Rightarrow} w_1 A w_3'$. To generate i^2 states we must have completed $\sim i$ $\bar{S}_F^p$ containing $\sim i$ elements during the recognition of a substring of length i, and, among them, there must exist at least a state <p,j,F> with $F = \{f_1, \ldots, f_{kl}\}$, $kl \sim i$.
This implies $R \overset{*}{\Rightarrow} X_1 \ldots X_f \, C_p^1 \ldots C_p^j \, \beta_f$ and $C_p^1 \ldots C_p^j \overset{*}{\Rightarrow} X_{f+1} \ldots X_i$ for $f = f_1, \ldots f_{kl}$ with $\beta_f \in V^*$ and this would imply ambiguities in the derivation. The only solution is that $\beta_1, \ldots, \beta_{kl}$ are such that one only can be matched with an input string, all other possibilities being wrong. Moreover if G is LR(k) this detection must be possible with only the first k terminals generable by the β_s'. Therefore the only possibility is a right recursivity but then we are in case 1 of the completer and only the possible "way outs" of the recursivity are generated : at most $\sim i$ states for the whole recursivity (otherwise the grammar would be ambiguous).
This achieves the proof.

6. EXAMPLES

We shall use a representation similar to Earley's [2] for states. States generated but not added are between ( ), followed by a * if this causes deletion. States deleted are between [ ] followed by the number of the step where this occur. We suppose that <0,1,0> cannot be added before step n.

6.1. Grammar RR : $A \to aA$; $A \to a$.
It is an example of grammar for which Earley's algorithm is in time n^2 without lookahead. For our algorithm it is in time n because only one state remains in $\bar{S}_F^p$, the other ones being deleted at each step by case 1 of the completer. As it is very simple we shall not detail it here.

6.2. $A \to aAa$; $A \to a$.
It is an example of nonambiguous grammar for which our algorithm, just like Earley's needs time and space n^2. In this case we have $\beta_f = a^f$ which is not bounded. For this case both algorithms are similar, so we shall not detail it.

6.3. Grammar UBDA1 $A \to AA$, $A \to a$.
In this example cases 1 and 2 occur both to permit time n^2 instead of n^3 as for Earley's.

S0 :	R → .A	0
	A → .AA	0
	A → .a	0
S1 :	A → a.	0
	[A → A.A	0] 3
	A → .AA	1
	A → .a	1
S2 :	A → a.	1
	[A → A.A	1(0)] 4
	A → AA.	0
	(A → A.A	1)
	A → .AA	2
	A → .a	2
S3 :	A → a.	2
	A → A.A	2 (1) (0)
	A → AA.	1,0
	(A → A.A	1)
	(A → AA.	0)
	(A → A.A	0)
	A → .AA	3
	A → .a	3
S4 :	A → a.	3
	A → A.A	3 (2) (1) (0)
	A → AA.	2,1,0
	(A → A.A	2)
	(A → AA.	1,0)
	(A → A.A	1)
	(A → A.A	0)
	A → .AA	4
	A → .a	4

and so on ...

Rem case 2 is always suffisant to get time n^2 as it results from proof of theorem 2.

6.4. Grammar UBDA2 : A → AAa ; A → a

In this case only case 2 permits deletions.

S0 :	R → .A	0	
	A → .AAa	0	
	A → .a	0	
S1 :	A → a.	0	
	[A → A.Aa	0]	6
	A → .AAa	1	
	A → .a	1	
S2 :	A → a.	1	
	A → A.Aa	1	
	A → AA.a	0	
	A → .AAa	2	
	A → .a	2	
S3 :	A → a.	2	
	A → A.Aa	2 (0)	
	A → AA.a	1	
	A → AAa.	0	
	(A → A.Aa	0)	
	A → .AAa	3	
	A → .a	3	
S4 :	A → a.	3	
	A → A.Aa	3 (1)	
	A → AA.a	2,0	
	A → AAa.	1	
	(A → A.Aa	1)	
	(A → AA.a	0)	
	A → .AAa	4	
	A → .a	4	
S5 :	A → a.	4	
	A → A.Aa	4 (2) (0)	
	A → AA.a	3,1	
	A → AAa.	2,0	
	(A → A.Aa	2)	
	(A → AA.a	1)	
	(A → A.Aa	0)	
	A → .AAa	5	
	A → .a	5	
S6 :	A → a.	5	
	A → A.Aa	5 (3) (1)	
	A → AA.a	4 (2) (0)	
	A → AAa.	3,1	
	(A → A.Aa	3)	
	(A → AA.a	2,0)	
	(A → A.Aa	1)	
	(A → AA.a	0)	

and so on ...

7. CONCLUSION

On the practical side the parser described in this paper is not very efficient when compared to specialized ones. But it has the advantage of processing any context-free grammars without transformation, which is very interesting in applications like syntactic macro-processors. If we use the context to reduce the number of generated states, and a garbage collector for useless states we obtain very good time and space performances which is of crucial importance in these applications, especially for the space.

On the theoretical side it seems very difficult to obtain a time n parser, but one could try to enlarge the class of grammars processed in time and space n, or at least to give a good definition of these classes.

BIBLIOGRAPHY

[1] J. EARLEY. An efficient Context-Free Parsing Algorithm.Thesis, dept. of Computer Science, Carnegie-Mellon Univ. (1968).

[2] J. EARLEY. An efficient Context-Free Parsing Algorithm.C.A.C.M. 13, 2, pp. 94-102 (1970).

[3] P. M. LEWIS II, R. E. STEARNS. Syntax directed transduction. J.A.C.M. 15, 3, pp. 465-488 (1968).

[4] M. BOUCKAERT, A. PIROTTE, M. SNELLING. More efficient general context-free top-down parsers. M.B.L.E. Researc Lab. Brussels (Belgium) Report R173 (1971).

[5] M. BOUCKAERT, A. PIROTTE, M. SNELLING. Efficient parsing algorithms for general context-free grammars. To appear in Information Sciences.