ICOT Technical Memorandum: TM-0215

TM-0215

A Parallel Parsing Algorithm and its Complexity

by

Y. Matsumoto

August, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Parallel Parsing Algorithm and its Complexity

Yuji Matsumoto
ICOT Research Center
Institute for New Generation Computer Technology
1–4–28, Mita, Minato-ku, Tokyo 108 Japan

(8th August 1986)

Extended Abstract

The aim of this paper is to describe the basic algorithm that our parallel parsing system [Matsumoto 86a] is based upon, and to estimate its time complexity. Although the system itself is intended as a general parsing system for Definite Clause Grammars [Pereira 80], this paper describes the essential part of the parallel parsing algorithm for pure context-free grammars. We show that the time complexity of the algorithm is proportional to the length of the input sentence when it is described by W-PRAM (Write-enabled Parallel Random Access Machine), which allows both read and write conflicts, provided that simultaneous write must be attempted with the same value. We, then, show that the algorithm is directly translated into parallel logic programming languages such as Guarded Horn Clauses [Ueda 85].

A context-free grammar G is defined by three-tuple $G = (V, P, S)$, where V is the set of nonterminal and terminal symbols, P is the set of grammar rules and S is the starting nonterminal symbol. Our algorithm does not distinguish terminal symbols and nonterminal symbols. Therefore, the input string $a_1, a_2, ..., a_n$ with length of n can be any sequence consisting of terminal and nonterminal symbols. Greek letters stand for finite sequences of nonterminal and terminal symbols. For the purpose of describing our algorithm, we uniquely put a distinct identifier to every place between two consecutive symbols in the righthand side of a grammar rule. For example, for the rule (1), we put identifiers as shown in (2):

```
(1) A → B C D
(2) A → B 1 C 2 D
```

We use natural numbers for identifiers though they can be anything. Let K be the set of such identifiers. We define the following sets:

$$H = \{(A, i, j) \mid A \in V, 0 \leq i < j \leq n\}$$

and

$$T_j = \{(k, i) \mid k \in K, 0 \leq i < j\} \qquad (1 \leq j < n)$$

In our parsing algorithm, these sets are regarded as tables, the elements of which are entries of them, whose values are 'true' or 'false'. Each entry has 'false' as its initial value. If the value of (A, i, j) in H is 'true', it means that there is an analysis tree whose root is A and whose leaves are $a_{i+1}, a_{i+2}, ..., a_j$. If the value of (k, i) in T_j is 'true' and the identifier k is contained in the rule $A \rightarrow \alpha\ k\ \beta$, it means that there exits a derivation of $a_{i+1}, ..., a_j$ from α.

The Algorithm

```
begin
1:      for each  $0 \leq i < n, A \in V$ such that $A = a_i$
                  parallel do $(A, i, i+1) := true$ ;
        repeat $N$ times
2:      for each  $(B, i, j) = true$ and $A \rightarrow B\ k\ \alpha$
                  parallel do $(k, i) := true$ in $T_j$ ;
2':     for each  $(B, i, j) = true$ and $A \rightarrow B$
                  parallel do $(A, i, j) := true$ ;
3:      for each  $(B, j, m) = true$ and $(k, i) = true$ in $T_j$ such that $A \rightarrow \alpha\ k\ B$
                  parallel do $(A, i, m) := true$ ;
3':     for each  $(B, j, m) = true$ and $(k, i) = true$ in $T_j$
                  such that $A \rightarrow \alpha\ k\ B\ k+1\ \beta$
                  parallel do $(k+1, i) := true$ in $T_m$
        end repeat ;
        if $(S, 0, n) = true$ then ACCEPT
end ;
```

Suppose that the maximum length of the righthand sides of the grammar rules is c, the number of the repetition of the algorithm, N, can be less than $(c-1) * n$. This shows that the time complexity of our parallel parsing algorithm is $o(n)$. Moreover, once an analysis tree is completed, it is never constructed repeatedly.

This parallel parsing algorithm is easily translated into parallel logic programming languages such as Guarded Horn Clauses. To do this, we regard terms in H as processes of the parallel logic programming language, and the tables T_i's as streams. Making an element in H to be 'true' is identified as creating the corresponding process. Tables are not considered as sets but as first-in first-out queues. Suppose $a_1, a_2, ..., a_n$ is the input string, the initial goals for the program is as follows:

```
a1(T0,T1),a2(T1,T2),....,an(Tn-1,Tn).
```

Creating these goals as the initial goals actually correspond to the first step of the algorithm. Ti's are streams shared by pairs of consecutive input symbols, provided that T0 and Tn are special terms and are recognized only by the process corresponding to the starting symbol. Each process has two arguments. The first of them is the input stream and the second is the output stream. The followings are a direct translation of the steps in the algorithm into GHC clauses.

```
2:   b(Ti,Tj) :- true | Tj=[(k,Ti)].  (for each A → B k α ∈ P)
2':  b(Ti,Tj) :- true | a(Ti,Tj).  (for each A → B ∈ P)
3:   b([(k,Ti)|Tj],Tm) :- true | a(Ti,Tm1),b(Tj,Tm2),merge(Tm1,Tm2,Tm).
                                   (for each T → α k B ∈ P)
3':  b([(k,Ti)|Tj],Tm) :- true | Tm=[(k+1,Ti)|Tm1],b(Tj,Tm1).
                                         (for each A → α k B k+1 β ∈ P)
```

Since tables T_i's are now represented by streams, their contents are processed sequentially. This is implemented by 3 and 3'. Note that all of these processes can be and should be executed in parallel. Moreover, outputs of all the process sharing the same output stream should be merged into a single stream. Therefore, the further translation is necessary. In our parallel parsing system, clause bodies of the processes of 2 and 2' having the same head predicate are bundled up into a single clause. Clauses such as 3 and 3' are defined as OR clause alternatives since no identifier is recognized by more than one process. The further translation is illustrated by the following example. The nonterminal symbol 'NOUN' in the sample grammar (3) is first translated into the clauses in (4). Each occurrence of 'NOUN' in the body of grammar rules produces one clause in (4). Clauses of (5) are the final translation of these clauses.

```
(3) NP → DET 1 NOUN
    NP → DET 2 NOUN 3 REL_CLAUSE
    NP → NOUN
    NOUN → NOUN 4 PP
    NOUN → NOUN 5 PRED
(4) noun(X,Y) :- true | Y=[(4,X)].
    noun(X,Y) :- true | Y=[(5,X)].
    noun(X,Y) :- true | np(X,Y).
    noun([(1,X)|X1],Y) :- true | np(X,Y1),noun(X1,Y2),merge(Y1,Y2,Y).
    noun([(2,X)|X1],Y) :- true | Y=[(3,X)|Y1],noun(X1,Y1).
(5) noun(X,Y) :- true | noun1(X,Y1),noun2(X,Y2),merge(Y1,Y2,Y).
    noun1(X,Y) :- true | Y=[(4,X),(5,X)|Y1],np(X,Y1).
    noun2([],Y) :- true | Y=[].
    noun2([(1,X)|X1],Y) :- true | np(X,Y1),noun2(X1,Y2),merge(Y1,Y2,Y).
    noun2([(2,X)|X1],Y) :- true | Y=[(3,X)|Y1],noun2(X1,Y1).
    noun2([_|X],Y) :- otherwise | noun2(X,Y).
```

In (5), processes for 'noun' are classified into two types of clauses, 'noun1' and 'noun2'. The former deals with the clauses that work without regard to the content of the input stream (i.e. processes whose first argument is a variable). The first three clauses in (4) are bundled up into the single process 'noun1' in (5). The work of the other clauses in (4) depends on the first content of the input stream and they are defined as OR processes. The first clause for 'noun2' deals with the case where the input stream is empty, and the last clause is for discarding the first element of the input stream if the identifier in it is different from any identifiers that are useful for the process.

In our implementation, T0 is defined as a special term, which is recognized only by the starting symbol. When the starting symbol receives the term, it produces another reserved term that indicates the end of the analysis. In order to recognize the termination of the analysis, we define a process that receives the stream Tn, the one produced by the last symbol in the input string. This process reports the end of the analysis when it receives the specially reserved term.

Considerations

First of all, we have to note that the derived parallel parsing program written in GHC is not a linear algorithm since the random access tables are realized as streams. It may be possible to implement the tables by processes which actually manage random access memory structure. However, we still have the problem of resolving simultaneous write requests.

In the parallel logic programming implementation, processes are independent each other. Therefore, processes that have the same root name and cover the same portion of the input string might be duplicated. Although this does not affect the time complexity if the resource of processes is abundant, this problem should not be neglected since the assumption is not realistic.

The Prolog implementation of our algorithm has been already finished. The streams are realized by difference lists in this implementation. This system shows that our algorithm is also practical as a sequential analysis of natural languages based on Definite Clause Grammars [Matsumoto 86b].

References

[Matsumoto 86a] Matsumoto, Y., "A Parallel Parsing System for Natural Language Analysis," Proc. International Conference on Logic Programming, pp.396-409, Imperial College, London, 1986.

[Matsumoto 86b] Matsumoto, Y. and Sugimura, R., "SAX: A Parsing System based on Logic Programming Languages," Computer Softwarc, Vol.3, No.4, 1986. (in Japanese)

[Pereira 80] Pereira, F.C.N. and D.H.D.Warren, "Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks," Artificial Intelligence, 13, pp.231-278, 1980.

[Ueda 85] Ueda, K., "Guarded Horn Clauses," Proc. The Logic Programming Conference, ICOT, 1985.