A PARALLEL AUGMENTED CONTEXT-FREE PARSING SYSTEM FOR NATURAL LANGUAGE ANALYSIS

Hsin-Hsi Chen Jiunn-Liang Leu Yue-Shi Lee

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
hh_chen@nlg.csie.ntu.edu.tw

Abstract

Parsing efficiency is one of the important issues in building practical natural language processing systems. This paper proposes a design and an implementation of a parallel augmented context-free parsing system for natural language analysis. Natural language grammars are more than context-free, so that unification formalisms are adopted to enforce the linguistic constraints and to transfer the linguistic information. Lexical and structural ambiguities are the famous problems in parsing natural language sentences. Traditional LR approaches to deal with these problems are pseudo parallelism or blind parallelism. They fork many processes to take care of parsing. Apparently, it results in the scheduling problem in shared-memory model or the communication problem in distributed-memory model. This paper presents a merge mechanism to compose the same jobs into one. It can not only eliminate the duplications, but also reduce the number of forked processes to the great extent. The gapping problems are also treated in this parallel parsing system. Currently, it is implemented in Prolog and in Strand, and running on Sun-series workstations.

1. Introduction

There is a growing interest in applying parallel computation techniques to natural language processing (NLP) [1-2]. Two approaches may be adopted: massively parallel systems and non-massively parallel systems [3]. These papers [4-8] show the typical examples for the former systems. They try to map natural language grammars into connectionist networks. Because the functionality of the nodes in the network is very primitive, they are involved in the following problems: (1) Is the network independent of the length of input sentence? (2) Does the network accept recursive grammar rules? (3) What threshold values and weights are assigned to nodes and links in the network? On the other side, these papers [9-15] deal with the design of non-massively parallel parsing systems. Most of the papers touch on parallelizing CYK Parsing algorithm, LR Parsing algorithm, Chart Parsing algorithm, *etc.*, however, only few presented

methods to capture the specific linguistic phenomena. To evaluate these types of parallel parsing systems, besides the performance criteria, i.e., scheduling of the processes in the shared-memory model [16] or the communication cost among processes in the distributed-memory model [17], the expressive capability is also an important issue. This paper will propose a parallel augmented context-free parsing system for natural language analysis. The linguistic phenomena are considered in depth in our design, gapping problem in particular. From the comparisons among different parsing strategies [18], Tomita's extended LR parser [19] is a better selection in computational linguistics. This paper will also follow the concept to design the parallel parsing system.

2. LR Parsing

Shift and reduce are two basic operations in LR parsing. LR parser uses two tables (Action and Goto tables) and one stack to control the parsing procedure. The Action table shows when to shift, to reduce, to terminate successfully, or to signal a syntactic error. The Goto table defines the next state after a nonterminal is matched and shifted. The stack contains a sequence of parse states. The following is a sample grammar:

(1) S --> NP VP
(2) S --> S PP
(3) NP --> n
(4) NP --> det n
(5) NP --> NP PP
(6) VP --> t1 NP
(7) VP --> t2 S
(8) VP --> iv
(9) PP --> prep NP

Table 1 shows its corresponding Action and Goto tables.

Table 1. The Parsing Table for the Sample Grammar

	det	n	t1	t2	iv	prep	$	S	NP	VP	PP
0	s2	s1						3	4		
1			r3	r3	r3	r3	r3				
2		s5									
3						s6	acc				7
4			s10	s11	s12	s6				13	9
5			r4	r4	r4	r4	r4				
6	s2	s1							8		
7						r2	r2				
8			r9	r9	r9	s6/r9	r9				9
9			r5	r5	r5	r5	r5				
10	s2	s1							14		
11	s2	s1						15	4		
12						r8	r8				
13						r1	r1				
14						s6/r6	r6				9
15						s6/r7	r7				7

Table 2 demonstrates the parsing steps for the sentence "I saw her duck".

Table 2. The Detailed Steps for Parsing the Sentence "I saw her duck"

step	stack	comment	input string
1	[_,0]	initial state	{n}{t1,t2}{n,det}{n,iv}$
2	[_,0] [n,1]	action(0,n)=s1	{t1,t2}{n,det}{n,iv}$
3.1	[_,0] [NP,4] [t1,10]	action(1,t1)=r3, goto(0,NP)=4 action(4,t1)=s10	{n,det}{n,iv}$
3.2	[_,0] [NP,4] [t2,11]	action(1,t2)=r3, goto(0,NP)=4 action(4,t2)=s11	{n,det}{n,iv}$
4.1	[_,0] [NP,4] [t1,10] [n,1]	action(10,n)=s1	{n,iv}$
4.2	[_,0] [NP,4] [t1,10] [det,2]	action(10,det)=s2	{n,iv}$
4.3	[_,0] [NP,4] [t2,11] [n,1]	action(11,n)=s1	{n,iv}$
4.4	[_,0] [NP,4] [t2,11] [det,2]	action(11,det)=s2	{n,iv}$
5.1	[_,0] [NP,4] [t1,10] [n,1]	action(1,n)=fail	
5.2	[_,0] [NP,4] [t1,10] [NP,14]	action(1,iv)=r3, goto(10,NP)=14 action(14,iv)=fail	
5.3	[_,0] [NP,4] [t1,10] [det,2] [n,5]	action(2,n)=s5	$
5.4	[_,0] [NP,4] [t1,10] [det,2]	action(2,iv)=fail	
5.5	[_,0] [NP,4] [t2,11] [n,1]	action(1,n)=fail	
5.6	[_,0] [NP,4] [t2,11] [NP,4] [iv,12]	action(1,iv)=r3, goto(11,NP)=4 action(4,iv)=s12	$
5.7	[_,0] [NP,4] [t2,11] [det,2] [n,5]	action(2,n)=s5	$
5.8	[_,0] [NP,4] [t2,11] [det,2]	action(2,det)=fail	
6.1	[_,0] [NP,4] [t1,10] [NP,14]	action(5,$)=r4, goto(10,NP)=14	$
6.2	[_,0] [NP,4] [t2,11] [NP,4] [VP,13]	action(4,$)=r8, goto(4,VP)=13	$
6.3	[_,0] [NP,4] [t2,11] [NP,4]	action(5,$)=r4, goto(11,NP)=4	$
7.1	[_,0] [NP,4] [VP,13]	action(14,$)=r6, goto(4,VP)=13	$
7.2	[_,0] [NP,4] [t2,11] [S,15]	action(13,$)=r1, goto(11,s)=15	$
7.3	[_,0] [NP,4] [t2,11] [NP,4]	action(4,$)=fail	
8.1	[_,0] [S,3]	action(13,$)=r1, goto(0,S)=3	$
8.2	[_,0] [NP,4] [VP,13]	action(15,$)=r7, goto(4,VP)=13	$
9.1	[_,0] [S,3]	action(3,$)=acc	
9.2	[_,0] [S,3]	action(13,$)=r1, goto(0,S)=3	$
10	[_,0] [S,3]	action(3,$)=acc	

The words "I", "saw", "her" and "duck" have categories {n}, {t1,t2}, {det,n} and {n,iv} respectively. The last three have more than one category. The effect is multiplicative rather than additive. Four stacks are generated at steps (4.1) - (4.4). Besides multi-category problem, the conflict entry in the action table also introduces nondeterminism. For example, parse the sentence "I saw her duck with a telescope". When the preposition "with" is inspected, a conflict entry (shift 6/reduce 6) will be met. The knowledge to resolve PP-attachment problem is originated from diverse resources [20]. Even if the knowledge is encoded, which action is selected correctly must be deferred to the later stage(s). Conventional approach to deal with these problems is pseudo parallelism or blind parallelism. The former explores the alternatives in a special sequence, e.g. breadth-first in our example. The latter forks many processes to take care of the subsequent actions. These processes may spend much time doing the same jobs. That decreases the significance of the parallelism. Synchronizing by shift operation [12] or data availability [13] was proposed to avoid the duplications. They tried to merge the stacks generated by different processes into tree-structured stacks (TSSs). Subsequently, Tanaka and Suresh [21] took another view on the interpretation of the elements in the pushdown stacks. These elements are called *dot reverse items* (*drits*). A *drit* is a dotted rule [A -> X1 X2 ... Xk • X$k+1$ ... Xm, i], which is similar to the Earley's item. However, its meaning is reverse. In Earley parsing [22], we plan to construct a sequence of item lists, I1, I2, ..., In such that a dotted rule [A -> α • β, i] $\in$ Ij iff S $\Rightarrow^*$ γ A δ, γ $\Rightarrow^*$ ω1 ω2 ... ωi, and α $\Rightarrow^*$ $\omega i+1$ $\omega i+2$ ... ωj. The item *drit* means [A -> α • β, j] $\in$ Ii iff S $\Rightarrow^*$ γ A δ, β $\Rightarrow^*$ $\omega i+1$ $\omega i+2$ ... ωj, and δ $\Rightarrow^*$ $\omega j+1$ $\omega j+2$ ... ωn. A sentence is *recognizable* by a grammar iff [s -> • γ, n] $\in$ I0.

Such an interpretation matches the direction of reduction, so that the merge can be done to the most depth. Consider the following two stacks possessed by two processes respectively. Each element in the stacks has two arguments. The first denotes a set of position numbers and the second is a state.

(a) ••• [{a},S1] [{b},S2] [{c},S3] [{e},S4]

(b) ••• [{a},S1] [{d},S2] [{c},S3] [{e},S4]

If the next action is a "reduce x" where x is "A -> B C D", we will get six *dirt*s shown as follows:

(1) [A -> B C • D, e] $\in$ I_c

(2) [A -> B C • D, e] $\in$ I_c

(3) [A -> B • C D, e] $\in$ I_b

(4) [A -> B • C D, e] $\in$ I_d

(5) [A -> • B C D, e] $\in$ I_a

(6) [A -> • B C D, e] $\in$ I_a

We can observe that (1) and (2), (5) and (6) have the same *drits*, i.e, the two processes do the same jobs. If we merge these two stacks into a TSS with the principle "merging the position numbers of those items with the same state from top of stacks", we can get a TSS like: "••• [{a},S1] [{b,d},S2] [{c},S3] [{e},S4]". Reducing the TSS by the same rule generates the same *drits* as before, however, it avoids the redundancy and decreases the number of processes to the great extent . Thus, the scheme can not only achieve the effects of chart parsing [23], but also is suitable to develop a new parallel parsing model.

3. A Parallel Parsing System

3.1 Parallel Recognizer

The fundamental concept of the recognizer is like the conventional LR algorithm except that the position numbers are used in the stacks. The following describes the basic recognizer:

(1) Initialize the stack to [{0},0], where the first 0 represents the word position and the next 0 denotes the initial state.
(2) Look up the first word of the remaining sentence in the dictionary, and return the feature structure(s)[1] of the word.
(3) Look up LR table by word category and the current state, and return a list of actions.
(4) Perform each action in the list.
    (a) accept: Terminate with success.
    (b) error: Terminate with failure.
    (c) shift: The position number is increased by 1 and go to step (5).
    (d) reduce: Do the reduce operation without changing the position number, and and try step (3) again.
(5) Merge the stacks with the same shift operation.
(6) Consume this word.
(7) If there are words left then go to step (2) else halt.

There are several places to enforce the parallelism:

(1) The table look-up at step (2) can be done in parallel.
(2) The actions in the action list can be performed in parallel.
(3) The merge operation at step (5) can be performed in parallel with the reduce operation at step (4.d).
(4) The new state created by the shift operation can be forwarded beforehand.
(5) The parse tree generation is overlapped with other actions (see next section).

Figure 1 demonstrates the architecture of the new parallel recognizer.

---

1 Unification is adopted. The unification-based formalism refers to [20].

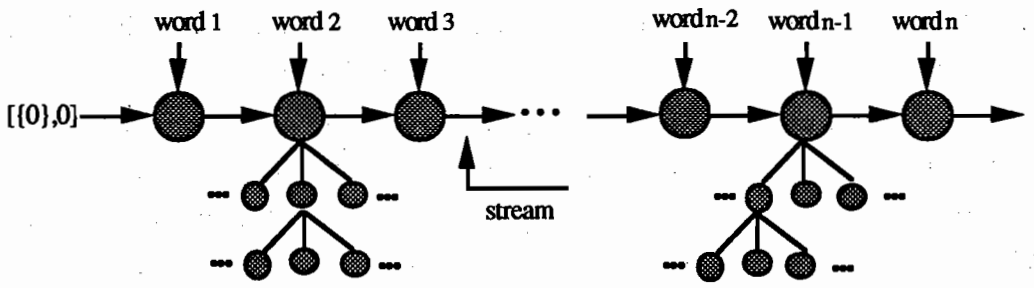

Figure 1. System Architecture of a Parallel Recognizer

A *word* process is initiated for each word to control all of its operations, i.e. shift and/or reduce operations. These processes are linked in a pipeline and communicate with each other by channels. The channel can transfer two kinds of information: merged stacks and a global table (see next section). When a word process receives the information (not necessary complete information) from its left neighbor word process, it begins its parsing steps immediately. Current system is developed by the language Strand88 [24], which is a parallel programming language. It can be run on parallel environments like transputers or be simulated on Unix systems. Strand provides a concise notation to describe process interactions. If a word process receives an incomplete information, it proceeds the parsing as possible as it can until it meets a variable. It is the characteristics of Strand language. The following shows the setup of the pipeline mechanism:

```
pglr(Sentence) :-
    pglr(0,Sentence,[[elt([0],0)]],_).
pglr(Pos,[],_,_).
pglr(Pos,[Word|Words],InStream,OutStream) :-
    dict:word(Word,WordStream),
    Pos1 is Pos + 1,
    goal(Pos1,Word,WordStream,InStream,MidStream),
    pglr(Pos1,Words,MidStream,OutStream).
```

A sequence of TSSs is transmitted from the left hand side to the right hand side via the special communication channel *stream*. The sequence is generated by the *word* process i ($1 \le i \le n$). We adopt the data structure for the stream: [TSS$i1$, TSS$i2$, ..., TSSim]. Each stack TSSij ($1 \le j \le m$) is represented as a list of elements of the form *elt(a list of position numbers,state number)*. Initially, the stream is: *[[elt([0],0)]]*. Because a word process may generate more than one TSS, a merger is used to merge m TSSs into a stream, and send them in sequence to its right neighbor. Figure 2 demonstrates a sample communication channel.

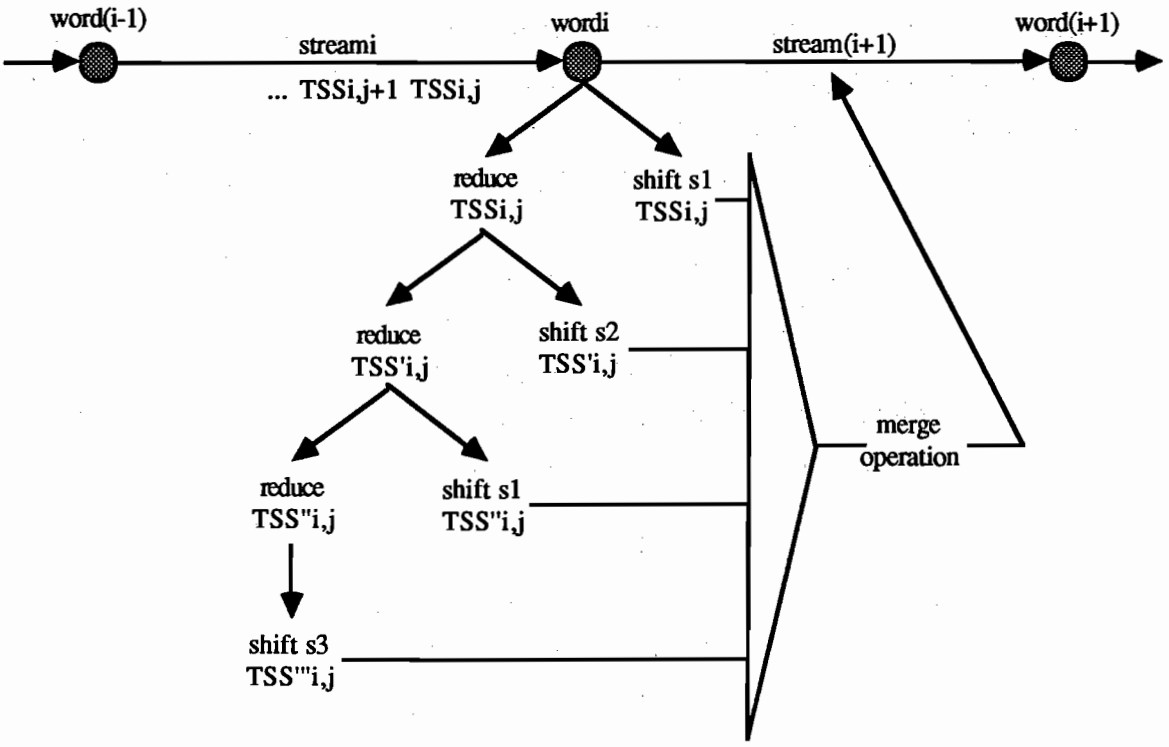

Figure 2. Communication Channel

The following shows the detailed step:

(a) s1: Keep {<s1, stackij, Tail(i+1)1>} and Stream(i+1)1, and
send out [[elt([i],s1)|Tail(i+1)1]|Stream(i+1)1].
That is, Stream(i+1) := [[elt([i],s1)|Tail(i+1)1]|Stream(i+1)1].

(b) s2: Keep {<s1, stackij, Tail(i+1)1>, <s2, stack'ij, Tail(i+1)2>} and Stream(i+1)2 and send out [[elt([i],s2)|Tail(i+1)2]|Stream(i+1)2].
That is, Stream(i+1)1 := [[elt([i],s2)|Tail(i+1)2]|Stream(i+1)2].

(c) s1: Merge stackij and stack"ij into nstack. Keep {<s1, nstack, Tail(i+1)1>, <s2, stack'ij, Tail(i+1)2>} and Stream(i+1)2.

(d) s3: Keep {<s1, nstack, Tail(i+1)1>, <s2, stack'ij, Tail(i+1)2>, <s3, stack'"ij, Tail(i+1)3>} and Stream(i+1)3, and
send out [[elt([i],s3)|Tail(i+1)3]|Stream(i+1)3].
That is, Stream(i+1)2 := [[elt([i],s3)|Tail(i+1)3]|Stream(i+1)3].

If the left stream is exhausted, let

Tail(i+1)1 := nstack, Tail(i+1)2 := stack'ij, Tail(i+1)3 := stack'"ij and
Stream(i+1)3 := [].

Otherwise, do the same job again. Strand language provides a predefined process *merger*. It allows many processes to write on a single stream. This approach has an advantage: the different shift message can be forwarded to next process before the reduce operation is terminated. It results in a better performance. The definition of

goal for a word process is given below. The process *pathval* retrieves the category information from a feature structure. The process *subgoal* transforms the TSSs in *InStream* into a merge list according to the LR parsing table. The list *MergeList* is a communication channel between processes *subgoal* and *dispatcher*. The process *dispatcher* sends the merged TSS into the right hand side neighbor one at a time via the channel *OutStream*.

```
goal(Pos,Word,[H],InStream,OutStream) :-
    pathval(H,[cat],Cat),
    subgoal(Pos,Cat,InStream,MergeList),
    dispatcher(OutStream,MergeList).
goal(Pos,Word,[H|T],InStream,OutStream) :-
    T =\= [] |
        pathval(H,[cat],Cat),
        subgoal(Pos,Cat,InStream,MergeList),
        dispatcher(OutStream1,MergeList),
        goal(Pos,Word,T,InStream,OutStream2),
        merger([merge(OutStream1),merge(OutStream2)],OutStream).
```

In natural languages, a word may have more than one category. This problem can be treated easily in the parallel parsing. Assume a word has N categories. The system can fork N processes and copy TSSs to each process to deal with those N categories. Theory 1 tell us: "Given any two stacks, no matter what states of their top of stacks are, if they receive different categories, they will not shift to the same state." That is, the new stacks cannot be merged. Based on the theory, the TSSs generated by any process can be sent to the next word process immediately without waiting for the generation of other TSSs. This can reduce the merge time. Table 3 lists the TSSs generated by word processes for the sentence "I saw her duck."

Table 3. TSSs Produced by Word Processes for the Sentence "I saw her duck"

node	tree-structured stacks (TSSs)	input string
1	[_,0] [n,1]	{n}
2	[_,0] [NP,4] [t1,10] [_,0] [NP,4] [t2,11]	{t1,t2}
3	[_,0] [NP,4] [t1,10]---[n,1] [_,0] [NP,4] [t2,11]┘ [_,0] [NP,4] [t1,10]---[det,2] [_,0] [NP,4] [t2,11]┘	{n,det}
4	[_,0] [NP,4] [t2,11] [NP,4] [iv,12] [_,0] [NP,4] [t1,10]---[det,2] [n,5] [_,0] [NP,4] [t2,11]┘	{n,iv}

Theorem 1. Given two stacks in a word process, no matter what states of their top of stacks are, they will not shift to the same state if they receive different categories.

Proof:

When the LR parser reads a word, it may execute reduce actions successively. At the end, it will meet a shift action, an accept action or an unacceptable signal. Assume the states of the top of two stacks are S1 and S2 respectively. Two cases are shown as follows.

(1) S1 = S2. It is trivial that the two stacks will not go into the same state.

(2) S1 ≠ S2. Assume the state S1 receives category c1 and the state S2 receives category c2. The configuration set of S1 can be divided into two groups: those configurations have the form "Pn -> Rn • c1 Un" and those do not. The configuration set of S2 can also be divided into "Pm -> Rm • c2 Um" and those do not. According to S1, the next state which receives c1 can be divided into two groups: "Pn -> Rn c1 • Un" and the prediction set of Un. According to S2, the next state which receives c2 can be divided into two groups: "Pm -> Rm c2 • Um" and prediction set of Um. Because {Pn -> Rn c1 • Un} ≠ {Pm -> Rm c2 • U2} and {Pn ->Rn c1 • Un} ≠ the prediction set of Um, the next states after receiving c1 and c2 cannot be the same. ■

3.2 Parsing Tree Generator

The conventional LR parsing algorithm keeps partial parse trees in the stacks. In the current implementation, only position numbers are recorded. This is because the interpretation of *drits* is from right to left, and it avoids the overheads during the merge and split operations. Under such a situation, if there does not exist an efficient parsing tree generation algorithm, the benefits from merge operation are lost. This section presents a parsing tree generator. It is active when any reduce action is performed. It will lookup tables, extract the Rhs of the production rule, apply the unification formulas and produce Lhs. There are two tables used: one is a global table received from the previous word process and the other is a delta table produced from its parent action process. The global table is the union of delta tables produced by the left-hand side word processes. Given a sentence "I saw her duck", Table 4 lists the delta tables generated by the word processes.

Table 4. The Delta Tables Produced by Word Processes for "I saw her duck"

node	word	partial trees produced by the node	global table
1	I	Δ1={1.<n,0,1>}	[]
2	saw	Δ2={2.<t1,1,2>,3.<t2,1,2>,4.<NP(1),0,1>}	Δ1
3	her	Δ3={5.<n,2,3>,6.<det,2,3>}	Δ1UΔ2
4	duck	Δ4={7.<n,3,4>,8.<iv,3,4>,9.<NP(5),2,3>}	Δ1UΔ2UΔ3
5	$	Δ5={10.<NP(6,7),2,4>,11.<VP(8),3,4>,12.<VP(2,10),1,4>, 13.<S(9,11),2,4>,14.<S(4,12),0,4>,15.<VP(3,13),1,4>, 16.<S(4,15),0,4>}	Δ1UΔ2UΔ3UΔ4

Each entry with a unique index has three arguments: the first is a partial tree, and the last two denote the left and the right positions respectively. For the performance issue, all the two tables are sorted and packed. The system uses the merge sort to arrange the partial trees. The right position is regarded as a primary key, and the left is a secondary key. The primary key is in descendant order and the secondary key is in ascendant order. This is the most efficient arrangement. Observe the delta tables in Table 4. The delta table created by each word process has a very interesting feature: "The right positions of all partial trees equal to the node number minus one except the leaf node." This is because the action process performs the reduce action until it meets a shift action, and each reduction will promote a partial tree up one level. Under the arrangement, we just append the global table to the delta table without applying the merge sort to these two tables. There is an important result in sorting the delta table: to keep the table entries unredundant. Because the system records the partial trees by a global table, it cannot distinguish which partial trees were produced by which processes when reduce action occurs. In fact, the distinction is not necessary. If the system cannot manage the tables efficiently, it will take a lot of time to search tables and will also have redundant solutions. Thus, we put an ambiguous forest [19] in the same table item and keep only one copy of subtrees that have the same structure and range. Figure 3 summarizes the flow of table constructions.

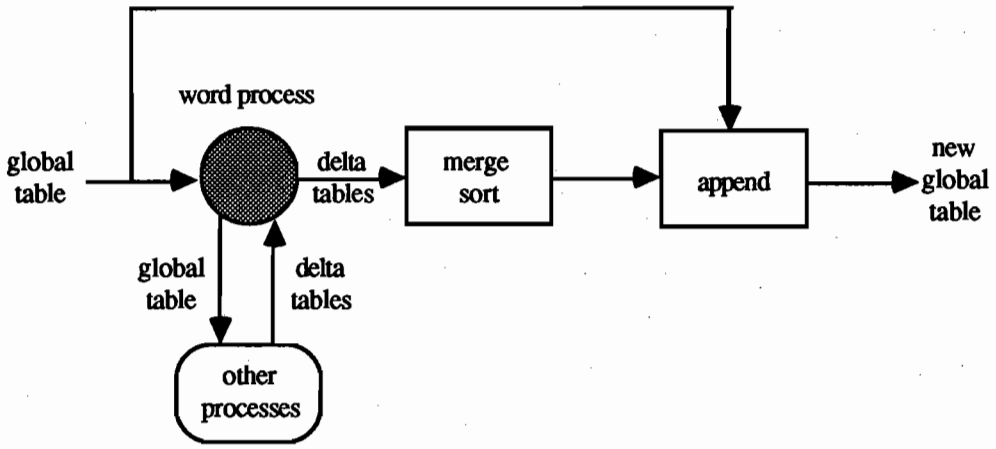

Figure 3. Table Management

4. Resolving Gapping Problem

Gapping is a common phenomenon in natural language sentences. Topicalization and relativization are two famous examples. In the sentence "The apples, I like", the constituent "The apples" is displaced from the object position to the topic position. These phenomena are regarded as movement transformations. To capture them, these papers [25-26] extended the conventional augmented context-free grammar formalism with two extra symbols ">>>" and "<<<" shown as follows:

(1) C --> C1, C2, ..., C*(i-1)*, C*i* <<< trace, C*(i+1)*, ..., C*n*.
This rule can be interpreted as "C is composed of C1, C2, ..., C*i*, ..., C*n*, where C*i* is moved from the position dominated by C*(i+1)*, ..., or C*n* and a trace is left at that position". The position of C*i* is called a *landing site.*

(2) C --> C1, C2, ..., C*(i-1)*, trace >>> C*i* , C*(i+1)*, ..., C*n*.
The interpretation of this rule is similar to the above except that the constituent C*i* is moved from its left hand side.

(3) C --> C1, C2, ..., C*(i-1)*, trace, C*(i+1)*, ..., C*n*.
This rule can be read as "C is composed of C1, C2, ..., C*(i-1)*, C*(i+1)*, ..., C*n*, and an empty constituent is left between C*(i-1)* and C*(i+1)*". The position of trace is called an *empty site.*

Under this grammar formalism, only the landing site and the empty site are specified. It is different from the slash technique in that no explicit slash feature is specified in the grammar. Consider a sample grammar shown below:

```
(1) syn_rule S1Bar --> TOPIC <<< TRACE, S:
        [TOPIC,head] === [TRACE,head].
(2) syn_rule S1Bar --> S.
(3) syn_rule S --> NP, VP:
        [NP,head] === [VP,subj].
(4) syn_rule NP --> *Det, *N:
        [NP,head] === [N,head].
(5) syn_rule NP --> *N:
        [NP,head] === [N,head].
(6) syn_rule VP --> *TV, NP:
        [VP,subj] === [TV,subj],
        [TV,obj] === [NP,head].
(7) syn_rule VP --> *TV, TRACE:
        [VP,subj] === [TV,subj],
        [TV,obj] === [TRACE,head].
```

Rule (1) deals with the topicalization and the others are the normal grammar rules. An empty constituent appears in rule (7).

Figure 4 shows one of the relationships between the displaced constituent and its corresponding empty constituent, which is a leftward movement. Rightward movement is symmetric, so their treatments are the same.

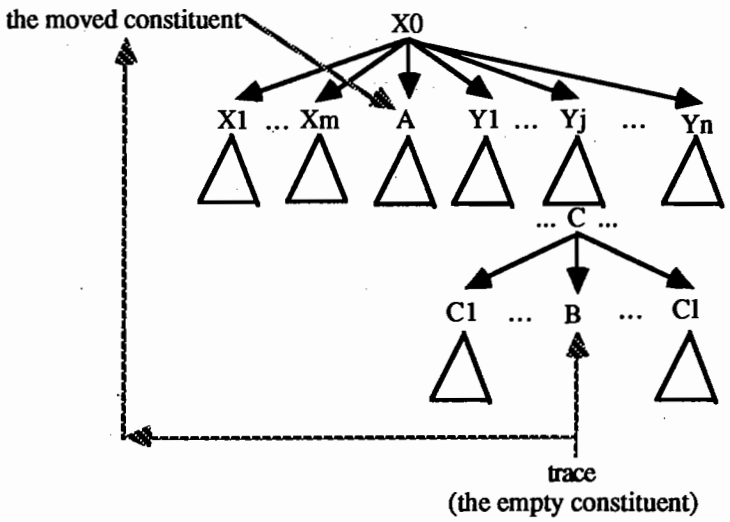

Figure 4. Leftward Movement

Before the grammar is translated, a preprocessing procedure computes the domination path, e.g. Y*j* dominates C and C dominates B. The empty constituent is raised up to the level dominated by X0 via C and Y*j*. For example, the above grammar is preprocessed like:

(1') syn_rule S1Bar --> TOPIC, S:
[TOPIC,head] === [S,trace,head].

(2') syn_rule S1Bar --> S:
[S1Bar,trace] === [S,trace].

(3') syn_rule S --> NP, VP:
[NP,head] === [VP,subj],
[S,trace] === [VP,trace].

(4') syn_rule NP --> *Det, *N:
[NP,head] === [N,head],
[NP,trace] === none.

(5') syn_rule NP --> *N:
[NP,head] === [N,head],
[NP,trace] === none.

(6') syn_rule VP --> *TV, NP:
[VP,subj] === [TV,subj],
[TV,obj] === [NP,head],
[VP,trace] === none.

(7') syn_rule VP --> *TV:
[VP,subj] === [TV,subj],
[TV,obj] === [VP,trace,head].

Rule (7') specifies that all the information about the empty constituent in the original rule is inherited by the mother category, i.e., VP. Because S dominates VP, rule (3') shows this information is also passed to S. Rule (1') depicts that the information is unified to the moved constituent. Rules (4'), (5') and (6') do not dominate any trace category, so the formulas "[FS,trace] === none" are added. The preprocessor automatically generates the *trace* feature for rules. It not only avoids the burden of grammar writing, but also detects the grammar errors beforehand. Finally, consider two general cases for preprocessing.

(a) For a rule C --> C1, C2, ..., C*(i-1)*, C*i* <<< trace, C*(i+1)*, ..., C*n*, if there exists more than one C*k* ((i+1) ≤ k ≤ n) that dominates *trace*, *trace* may be transferred up from different paths. During preprocessing, we split such a rule into several rules with the same Lhs and Rhs, and different sets of unification formulas. Because our parsing system can handle the conflict condition, these rules can be tried in parallel.

(b) For a rule C --> C1, C2, ..., C*n*, if there exists more than one C*k* (1 ≤ k ≤ n) that dominates *trace*, *trace* may be transferred up through the mother category. In this way, trace is considered as a disjunction feature to transfer all the possible information up.

5. Concluding Remarks

This paper proposes a design and an implementation of a parallel parsing system for natural language analysis based on LR parsing algorithm. It adopts dot reverse items instead of the conventional Earley items. This interpretation can not only achieve the same effect as Chart parsing, but also reduce the number of processes to the great extent. An efficient table management algorithm is also presented to construct the parsing trees. A global table for parse tree generation is set up incrementally. It is transferred from the leftmost word process to the rightmost process. Because the delta table generated by an intermediate word process is mutual exclusive of the global table sent from its left hand side word process, and the former is much smaller than the latter, it is easy to keep tables sorted and packed. For the well-treatment of the gapping phenomena, the formalism to specify the landing site and the empty site is introduced. A grammar translator adds disjunctive trace features to unification formulas automatically. Currently, it can capture the relationship of serial binding. The parallel parsing system is implemented with Prolog and with Strand, and running on Sun-series workstations.

Acknowledgements

Research on this paper was partially supported by National Science Council grant NSC-81-0408-E002-514, Taipei, Taiwan, R.O.C.

References

[1] A. Nijholt, "Parallel Parsing Strategies in Natural Language Processing," *Proceedings of International Parsing Workshop on Parsing Technologies*, 1989, pp. 240-253.

[2] R. Akker, H. Alblas, A. Nijholt and P.O. Luttighuis, "An Annotated Bibliography on Parallel Parsing," *Memoranda Informatica 89-67*, Department of Computer Science, University of Twente, the Netherlands, 1989.

[3] H. Schnelle, "Panel Discussion on Parallel Processing in Computational Linguistics," *Proceedings of 12th International Conference on Computational Linguistics*, 1988, pp. 595-598.

[4] D.L. Waltz and J.B. Pollack, "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation," *Cognitive Science*, Vol. 9, 1985, pp. 51-74.

[5] T. Li and H.-W. Chun, "A Massively Parallel Network-Based Natural Language Parsing System," *Proceedings of the Second International Conference on Computers and Applications*, 1987, pp. 401-408.

[6] B. Selman and G. Hirst, "Parsing as an Energy Minimization Problem," in *Genetic Algorithms and Simulated Annealing*, Lawrence Davis (Editor), Morgan Kaufmann Publishers, 1987, pp. 141-154.

[7] H. Nakagawa and M. Tatsunori, "A Parser based on Connectionist Model," *Proceedings of 12th International Conference on Computational Linguistics*, 1988, pp. 454-458.

[8] H. Schnelle and R. Wilkens, "The Translation of Constituent Structure Grammars into Connectionist Networks," *Proceedings of 13th International Conference on Computational Linguistics*, 1990, pp. 53-55.

[9] X. Huang and G. Louise, "Parsing in Parallel," *Proceedings of 11th International Conference on Computational Linguistics*, 1986, pp. 140-145.

[10] A. Haas, "Parallel Parsing for Unification Grammars," *Proceedings of International Joint Conference on Artificial Intelligence*, 1987, pp. 615-618.

[11] E.L. Lozinskii and S. Nirenburg, "Parsing in Parallel," *Computer Language*, Vol. 11, No. 1, 1986, pp. 39-51.

[12] H. Tanaka and H. Numazaki, "Parallel Generalized LR Parsing Based on Logic Programming," *Proceedings of International Workshop on Parsing Technologies*, 1989, pp. 329-338.

[13] H. Numazaki and H. Tanaka, "A New Parallel Algorithm for Generalized LR Parsing," *Proceedings of 13th International Conference on Computational Linguistics*, 1990, pp. 305-310.

[14] Y. Matsumoto, "Handling Coordination in a Logic-Based Concurrent Parser," *Natural Language Understanding and Logic Programming*, 1991, pp. 1-12.

[15] A. Kouji, "Parallel Parsing System Based on Dependency Grammar," *Natural Language Understanding and Logic Programming*, 1991, pp. 147-157.

[16] R. Grishman and M. Chitrao, "Evaluation of a Parallel Chart Parser," *Proceedings of the 2nd Conference on Applied Natural Language Processing*, 1988, pp. 71-76.

[17] H.S. Thompson, "Chart Parsing for Loosely Coupled Parallel Systems," *Proceedings of International Workshop on Parsing Technologies*, 1989, pp. 320-328.

[18] P. Shann, "The Selection of a Parsing Strategy for an On-Line Machine Translation System in a Sublanguage Domain," *Proceedings of International Workshop on Parsing Technologies*, 1989, pp. 264-276.

[19] M. Tomita, "An Efficient Augmented-Context-Free Parsing Algorithm," *Computational Linguistics*, Vol. 13, No. 1-2, 1987, pp. 31-46.

[20] K.-H. Chen and H.-H. Chen, "Attachment and Transfer of Prepositional Phrases with Constraint Propagation," Submitted to *Computer Processing of Chinese and Oriental Languages* (first revision).

[21] H. Tanaka and K.G. Suresh, "YAGLR: Yet Another Generalized LR Parser," *Proceedings of ROCLING IV*, Taiwan, 1991, pp. 21-31.

[22] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Vol. 1: Parsing, Prentice-Hall, 1973.

[23] M. Kay, "Algorithm Schemata and Data Structures in Syntactic Processing," in *Readings in Natural Language Processing*, B.J. Grosz (Editor), Morgan Kaufmann, 1986, pp. 35-70.

[24] I. Foster and S. Taylor, *Strand: New Concept in Parallel Programming*, Prentice Hall, 1989.

[25] H.-H. Chen, I.-P. Lin and C.-P. Wu, "A New Design of Prolog-Based Bottom-Up Parsing System with Government-Binding Theory," *Proceedings of the 12th International Conference on Computational Linguistics*, 1988, pp. 112-116.

[26] H.-H. Chen, "A Logic-Based Government-Binding Parser for Mandarin Chinese," *Proceedings of the 13th International Conference on Computational Linguistics*, 1990, pp. 48-53.