ICOT Technical Report: TM-0073

TM-0073

# Syntactic Parsing with POPS
## —Its parsing time order and the comparison with other systems—

by
Hideki Hirakawa, Koichi Furukawa

September, 1984



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**

# Syntactic Parsing with POPS

## - Its parsing time order and the comparison with other systems -

Hideki Hirakawa, Koichi Furukawa

ICOT Research Center

Institute for New Generation Computer Technology,
Mita Kokusai Bldg. 21F, 4 - 28 Mita 1-chome,
Minato-ku, Tokyo 108

## Abstract

There have been developed many parsing systems based on logic programming languages such as DCG, BUP, XG and so on. Authors have developed OR-Parallel Optimizing Prolog System (POPS) and showed that POPS is very effective for the natural language syntactic parsing because it solved some qualitative problems in the above systems. In this paper the parsing efficiency of POPS-based parsing is discussed and compared with other context-free parsing algorithms such as Earley's algorithm and Chart parsing.

## 1. Introduction

We have implemented an OR-Parallel Optimizing Prolog System, POPS, based on an OR-parallel computational model for Pure Prolog [Hir 83]. POPS is especially effective for application fields where many sub-computations are duplicated to perform a total computation. It also effective for the fields where a main computation is some kind of searching, that is, the program has high-degree OR-parallel computation. One of the applications of POPS is a natural language syntactic parsing for which many Prolog-based systems are developed [Pereira 80], [Matsumoto 83]. POPS-based parsing has solved the following problems inherent in other Prolog-based parsing systems [Hirakawa 83]:

(1) Repetition of computation: Since backtracking repeatedly performs the same computation many times, the parsing time increases exponentially with the length of an input sentence, thereby resulting in poor parsing efficiency.

(2) Left recursive rules: Natural language have a recursive structure by nature. Reflecting this characteristic, their syntax rules tend to include left recursive rules. Interpreting these rules by Prolog's top-down, depth-first strategy causes the processing to enter an infinite loop; the computation never stop.

(3) Handling of epsilon-rules: In natural languages, word

omission frequently occurs. This correspond to the epsilon-rule in context-free grammars (CFG). The epsilon-rule should be handled as it is.

These are qualitative aspects of POPS-based parsing. This paper examines quantitative aspect of POPS-based parsing and compares it with conventional CFG parsing algorithms. Section 2 describes a POPS-based syntactic parsing, and Section 3 discusses POPS-based CFG parsing by calculating the order of parsing time. In Section 4, the POPS model is compared with Earley's parsing algorithm [Ear 68], Earley Deduction [Per 83], and Chart parsing [Kay 80].

## 2. POPS-based Syntactic Parsing

### 2.1 POPS

POPS is a Pure Prolog OR-parallel interpreter implemented in Concurrent Prolog [Sha 83]. POPS has the following features;

(1) Goals are executed serially and clauses are executed in parallel.
(2) Computational model is based on multiple processes and message transfer between processes.
(3) The graph reduction mechanism is introduced to prevent the same computations from being repeated.

The details of POPS is described in [Hir 83]. In this paper we will show the brief explanation of POPS and POPS-based syntactic parsing.

POPS consists of four components; processes, channels, a board and a horn data base. A process plays a key role in computations, that is, the computational behavior of POPS corresponds to the behavior of processes. The behavior of a process is defined later. A process corresponds to a clause being computed, such as H<--G1,G2. A process creates its child processes which computes the subgoal of the parent process. The parent process receives solutions from its child processes via a channel. A channel is a communication path between processes and is dynamically generated during computation. Data transferred through a channel is called a message. Each process has its output channel and input channel. The output channel connects to its parent processes and the input channel to its child processes.

The board is a storage area accessed by processes, and stores channel head pairs. A channel head pair is a pair of one subgoal and one channel through which the solutions (messages) for the subgoal are sent. A channel head pair is described in the following format:

Channel+Head

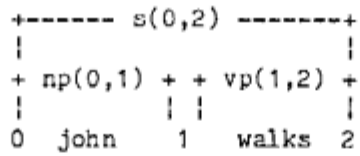
The operation on the board is called "board access" which registers a new channel head pair to the board if the head is not in the board otherwise the new channel is shared with the one of old channel head pair. This operation realizes the graph reduction mechanism which enables POPS to share the same subcomputations and to avoid the infinite rule application.

### 2.2 Parsing in POPS

POPS-based parsing is performed in the same way as the Definite Clause Grammar (DCG) incorporated into DEC10 Prolog. First the CFG grammar descriptions are translated into the Prolog clauses. Then the parsing is performed by executing the programs using POPS interpreter. To simplify the discussion, we assume the given grammar is a pure CFG. The following shows the example of grammar rules and the corresponding (Pure) Prolog clauses.

```
   CFG rule                    Prolog clause
s  --> np, vp.          s(X0, X) :- np(X0, X1),vp(X1, X).
np --> [john].          np([john|X],X).
vp --> [walks].         vp([walks|X],X).
```

The grammar notation and the translation results correspond to those of DCG. Two arguments of the Prolog clauses are D-lists which specify the position of the input sentence. To simplify the notation, in the remaning part of this paper we use figures rather than lists to indicate the word position in the input sentence as follows;




Generally, all literals appearing in the parsing stage have the format a(I,J) where I and J are figures.

## 3. Parsing Time Order with POPS

### 3.1 Procedure to Estimate Parsing Time Order

This subsection examines the order of the parsing time with the POPS-based parsing algorithm. Because POPS itself is based on a parallel parsing model, the actual computation time depends upon the number of processors in the system and other factors. Here we assume that the parsing system consists of a single processor and estimate how the total computation amount changes according to the length of an input string.

The syntactic parsing algorithms are classified into two different types according to their purposes: one type of algorithm determines whether an input string is acceptable with the given grammar, while the other obtains

all possible parsings (ie, parsing trees) for the input string. The two naturally differ in the computation amount required. Although original POPS adopted the latter algorithm, one change to the process operation permits the POPS to perform the former algorithm. In the following discussion, we refer to the former and latter algorithms as type1 and type2 POPS, respectively. We obtain the parsing time order for the two cases:

Case 1: Parsing using type1 POPS with an unambiguous grammar.
Case 2: Parsing using type2 POPS with an ambiguous grammar.

Here we call a grammar "unambiguous", if it assures the following relationship:

If fr(T1) = fr(T2) then T1 = T2

where T1 and T2 are derivation trees sharing the same root, and fr(T) is the concatenation of T's leaves.

In the subsequent subsections, we calculate the order of the POPS-based parsing time by the following procedure:

(1) Shows algorithms for type1 and type2 POPS processes to provide the order of time required for the operations included in each algorithm.
(2) Gives the order of the total computation cost for one process.
(3) Estimates the number of processes generated during a parse.
(4) Estimates the total parsing cost from (2) and (3).

### 3.2 Process Cost

This subsection first provides formal behavior of processes of type1 and type2 POPS, then estimates the process cost for each operation of processes.

Fig.1 shows the POPS process operation. This definition merges type1 and type2 POPS. The "if type1 then ... else" portion selects each type. Since the purpose of this paper is the estimate of parsing time order, the detailed explanation of the POPS process is omitted here.

The operations in Fig.1 have the following functions:

| | |
|---|---|
| terminated | : Determines whether a given clause has the "Head<--true" format. |
| send_message | : Sends "Head" to "OutChannel". |
| is_new_message | : Checks whether "Message" has already been sent to "OutChannel". |
| board_access | : Performs "board_access" operation (See Sec. 2). |
| == | : Checks whether the right side equals the left |

side.
clauses : Fetches a collection of expandable rules from the horn database.
create_new_process: Generates new processes for each element of "NewGoals" (a collection of clauses).
wait_receive : If the process receives a message, it becomes true as long as the message is not [] (end of message) else it becomes false.
create_new_process: Generates a new clause and the corresponding new process.

Suppose that a particular grammar has been fixed for parsing, we can estimate each computation time of the above operations. The parameter of the operations is the length (n) of an input string. Since the above operations except for "board_access" and "is_new_message" are obviously independent of the input string length, then they costs the constant order time provided that the grammar is fixed. On the other hand, "board_access" and "is_new_message" operation costs O(n) time.

Lemma 1: The "board_access" operation costs O(n) time.

Proof: All the literals in the grammar (DCG program) have the format, a(I,J), and any Head saved on the board has the first argument whose value falls within the range, O<=i<=n (n is the length of an input string). Therefore, the board can be structured as follows:

```
Board = { [0,C11+H11,C12+H12, ... ,C1p+H1p]
          [1,C21+H21,C22+H22, ... ,C2q+H2q]
                    :
          [n,Cn1+Hn1,Cn2+Hn2, ... ,Cnm+Hnm] }
```

The number of lists [...] is n+1 and the maximum number of channel head pairs in each list is equivalent to the number of the grammatical categories in the given grammar. Using this board structure, the board access related to one channel head pair is performed in the following

```
if terminated(Clause) then
      if type1 then
              send_message(Head,OutChannel);
      else if is_new_message(Message,Outchannel)
              send_message(Head,OutChannel);
else begin
      board_access(Head+Channel,Result);
        if Result == nltyet begin
            clauses(Body_top,Newgoals);
            create_new_processes(Newgoals,InChannel);
            while wait_receive(Message,InChannel) do
                  create_new_process(Clause,Message,OutChannel);
     end;
```

Fig.1 The definition of type1 and type2 POPS process operation

manner:

(1) Fetches a list from the board according to the first argument of a literal of the channel head pair.
(2) Searches the list for the head and, if found, identify both channels, otherwise, registers the channel head pair to the list.

Since (1) and (2) takes O(n) and O(c) respectively, the "board_access" can be done within time order O(n).

Lemma 2: "is_new_message" costs O(n) order time

Proof: "is_new_message" checks whether a message has already been sent to the output channel (OutChannel) which corresponds to the channel of the channel head pair in the board. A channel in the board contains a message in the a(i,j) format. Since 'i' equals the first argument of the head, it has a specific value falling within the range, 0<=i<=n. Therefore, as many as n+1 messages are passed to the channel, providing O(n) as the computation time for the operation.

The execution time order for the process operations are as follows:

O(n) : "board_access", "is_new_message"
O(c) : the other operations

### 3.3 Type1 POPS and Unambiguous Grammar

This subsection examines parsing time order using type1 POPS and an unambiguous grammar. Since the computation cost of each process operation has already been achieved, we obtain the process cost and the number of generated processes during parsing.

Generated process can be classified into two types according to the format of a clause in the process. One is the n-type (non-terminate) process that has a clause whose body is made up of one or more subgoals and the other is the t-type (terminate) process that has a clause whose body consists of "true".

```
n-type process      s(0,X)<-np(0,Y),vp(Y,X).
t-type process      n(2,3)<-true.
```

The n-type process performs one O(n) operation, "board_access" and several constant order operations. The t-type process performs only constant order operations, that is, "create_new_process" and "send_message". Then the each process cost is as follows:

n-type process cost: O(n)
t-type process cost: O(c)

Next, we estimate the number of n-type processes and t-type processes to be generated during the entire parsing.

Lemma 3: In parsing based on an unambiguous grammar and the type1 POPS, the number of the n-type processes which are generated during the parsing is O(n).

Proof: The n-type process corresponds to the following clause:

```
a(i,J)<-b(k,L), rests.    0 <= i,k <= n (n is input length)
                          J,L are variables
                          rests is the rest part of the body
```

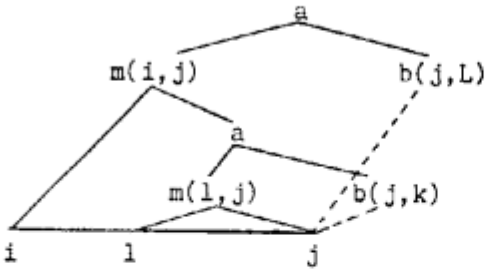
When a n-type process is generated, its 'i' and 'k' has been set to particular values. The process must be connected to one channel head pair in the board, because "board_access" operation for channel head pair, b(k,L)+Channel, should be executed. Conversely, a channel head pair, b(k,L)+Channel, can connect to at most constant number n-type processes which have certain head 'a' (Lemma 4). Since the board contains O(n) channel head pairs as shown in Section 3.2, the number of n-type processes to be generated during parsing is O(n).

Lemma 4: In parsing based on an unambiguous grammar and the type1 POPS, if there is any process which has the clause "a(i,K)<-b(j,L),n(L,K)" derived from a grammar rule "a->m,b,n" (b is not an empty production), there is no process which has "a(1,K)<-b(j,L),n(L,K)" (1 is not equal to i) as a clause.

Proof: Assume the parsing situation where one grammar rule "a->m,b,n" derives two processes which have the following clauses:

```
a(i,K)<-b(j,L),n(L,K)
a(1,K)<-b(j,L),n(L,K)    i < 1
```

Since the parsing for the nonterminal 'm' has been completed, both m(i,j) and m(1,j) simultaneously hold. The definition of the unambiguous grammar and the condition, i < 1, requires that the parsing tree associated with m(1,j) should form a subtree of the parsing tree for m(i,j).




This structure, however, is not possible unless k = j or b is an empty production. Therefore, the above two processes cannot be simultaneously present.

Lemma 5: In parsing based on an unambiguous grammar and the type1 POPS, the number of t-type processes which are generated during the parsing is O(n2).

Proof: A t-type process has a clause in the format, a(k,L)<-true, and is generated in these two cases;

(a) When an active process that has "x(i,J)<-a(k,L),rests" as a clause is derived
(b) When a waiting process that has a clause in the "a(i,J)->x(k,L)" format receives a message

First we assume k is fixed. Since the grammar is fixed, the number of 'a'-related unit clauses is constant and the number of grammar rules whose body contains 'a' is also constant. According to the condition 0<=i<=n, the process with the clause "x(i,J)<-a(k,L),rests" produces as many as n+1 t-type processes. Thus in case (a), O(n) processes are generated. Since, in (b), the number of grammar rules that have 'a' as their heads is on the constant order, and i for each rule falls within the range, 0<=i<=n, also O(n) processes are generated. Therefore the number of processes generated for a(k,L) is O(n).

Because k satisfies 0<=k<=n, a total of O(n2) t-type processes are generated.

Theorem 1: In parsing based on an unambiguous grammar and the type1 POPS, the order of the total parsing time is O(n2) where n is input sentence length.

Proof: Using Lemma 3 to 5, the order of total parsing time is estimated as follows:

(t-type process cost) x (number of t-type processes) +
(n-type process cost) x (number of n-type processes)
= O(n) x O(n) + O(c) x O(n2)
= O(n2)

### 3.4 Type2 POPS and ambiguous Grammar

This subsection examines parsing time order using type2 POPS and a ambiguous grammar. The estimating strategy is same as that of previous section.

Lemma 6: The n-type and t-type process cost of type2 POPS are as follows;

n-type process cost: O(n)
t-type process cost: O(n)

Proof: The difference between type1 and type2 POPS definition is that type2 POPS executes "is_new_message" operation which costs O(n). Then the n-type process cost is same as that of type1 POPS and t-type process costs is O(n) because of "is_new_message" operation.

Next, we estimate the number of n-type processes and t-type processes to be generated during the entire parsing.

Lemma 7: In the parsing with ambiguous grammar and type2 POPS, the number of n-type processes which are generated during the parsing is O(n2).

Proof: The proof of Lemma 3 can be modified to provide the n-type process number in this case. The modification is that the number of the processes which can be associated with one channel head pair is O(n). This is because the given grammar is ambiguous one. Then the order of n-type process number is given as follows:

(number of channel head pair) x (n-type process number) =
O(n) x O(n) = O(n2).

Lemma 8: In the parsing with ambiguous grammar and type2 POPS, the number of t-type processes which are generated during the parsing is O(n2).

Proof: The proof of Lemma 5 can also be applied to the parsing with type2 POPS and an ambiguous grammar. Therefore, the number of t-type processes is O(n2).

Theorem 2: In parsing based on an ambiguous grammar and the type2 POPS, the order of the total parsing time is O(n3) where n is input sentence length.

Proof: From Lemma 6 to 8, the order of the total parsing time is given by:

(t-type process cost) x (number of t-type processes) +
(n-type process cost) x (number of n-type processes)
= O(n) x O(n2) + O(n) x O(n2)

= O(n3)

## 4. Earley's Algorithm, Earley Deduction, and Chart Parsing vs. POPS

This section compares POPS-based parsing (execution of a DCG program) with analyses by other parsing algorithms.

### 4.1 Earley's Algorithm vs. POPS

Earley's algorithm is a "Good practical algorithm" of a CFG. It generates an (n+1) x (n+1) recognition matrix for an n-long input string. As shown in Fig.2, the recognition matrix has elements consisting of grammar rules and a meta-symbol, and includes-all parsing trees. Note, unlike the POPS-based parsing, all parsing trees are generated from the recognition matrix by the algorithm. Fig.3 shows the Earley's algorithm in the bit vector version.

For details of the algorithm, see [Harrison 78]. Here we describe only some key features of the algorithm. The basic operations in the algorithm have the following functions:

predict  Expands a collection of nonterminal symbols according to a grammar rule. (Top-down prediction).
ex. For S->A,B, A->a, A ->D,E,
Predict({S}) = { S->*A B, A->*a, A->*D E }

```
begin
  t<0> =  predict({s})
               0
               :
               0
  for j := 1 to n do
   begin
     scanner :
       t<j> := t<j-1> * {a<j>};
     computer:
       for k:=j-1 down to 0 do
         t<j> := t<j> U (t<k> * t<k,j>);
     predictor:
       t<j,j> := predict(U t<i,j>)
                          where 0<i<j
   end
end
```

Fig.3 Earley's parsing algorithm

Fig.2 Recognition Matrix for the input "a+a*a" with the grammar E-> a | E+E | E*E.

scanner Collates a top-down-predicted terminal symbol with an input symbol.
ex. If, in the previous example, the input string contains "a" for the prediction, A->*a, this operation adds "A->a*" to the recognition matrix.

computer When "scanner" recognizes a nonterminal symbol, this operation generates a new element from the recognition matrix element that has already predicted the symbol recognized by the scanner. The new element is added to the matrix. This computation is applied repeatedly as many times as possible (depending upon the growth of the bottom-up tree).
ex. S->A*C is generated from A->a* and S->*A C

These operations relates to the POPS operations as follows:

| Earley | POPS |
|---|---|
| predict | Generation of a child process by "create_new_processes" operation. |
| scanner | Processing of a terminated clause, that is, "terminate" and "send_message" operation. |
| computer | Transfer of a message from a child process and the generation of a new process, that is, "send_message","wait_receive" and "create_new_process" operation. |

Recognition matrix elements of Earley's algorithm can be directly associated with POPS processes. Thus, Earley's algorithm is equivalent with the POPS in parsing operation. A basic difference is that while Earley's algorithm functions to generate a recognition matrix, POPS operates to obtain all parsing trees. Therefore, the generation of a recognition matrix by Earley's algorithm that requires O(n3) time corresponds to the parsing with type2 POPS and an ambiguous grammar which requires O(n3) time.

## 4.2 Earley Deduction vs. POPS

The Earley Deduction proof procedure schema based on Earley's algorithm is applied to two sets of definite clauses: program and state. A "program" is a set of input clauses corresponding to an input string and remains unchanged during deduction. A "state" is a collection of derived clauses and expands during deduction. Each derived state has a selected negative literal. The derivation rule maps the current state to new state by adding a new derived clause to the current state, and consists of "instantiation" and "reduction". "instantiation", an equivalent of the operation "predict" of Earley's algorithm, tries to unify a selected literal of a clause with a positive literal of non-unit clause, and if succeeds, adds the unification result to the current state. By contrast, "reduction" unifies a selected literal of a clause in the current state with a unit clause in the

program and current state, and adds the derivation result to the current state. When the derivation result has already been subsumed by a clause in the current state, however, the addition is not performed. "reduction" can be related to the "scanner" and "computer" operations of Earley's algorithm. The operations of Earley Deduction can be associated with the POPS operations as follows:

| Earley Deduction | POPS |
|---|---|
| instantiation | "clauses" and "create_new_processes" operations. |
| reduction | "send_mess","wait_receive" and "create_new_process" operations. |
| subsume check | "board_access" operation. |

This relationship permits POPS to be considered an Earley Deduction processing system with these features:

(1) The POPS uses the idea of multiple processes and message passing as the base for computation to explicitly express parallelism.

(2) POPS introduces channel connection to optimize the state search operation in reduction mode of Earley Deduction.

(3) POPS processes correspond to Earley Deduction's state elements. Note POPS processes disappear, eventually leaving the board alone.

(4) While Earley Deduction requires clauses to be subsumed, the POPS requires nothing but the checking of the equivalence of a selected literal.

(5) While a selected literal in Earley Deduction is any of the subgoals, in POPS, it is associated with the head subgoal (AND-serial left-to-right interpretation).

Thus, the POPS can basically be considered an equivalent to the Earley Deduction. However, with an additional channel concept included, POPS can perform processing which can't be performed with Earley Deduction. This feature will be described in Section 5.

### 4.3 Chart Parsing vs. POPS

Chart parsing is a general framework for CFG parsing algorithms featuring the use of a data structure, Chart, for bookkeeping.

The status of the parser is represented by a chart consisting of directed graphs. A node on a chart shows the position of an input and an edge corresponds to an application of a grammar rule. For further details, see [Kay 80], [Winograd 83], and [Hirakawa 83]. According to

[Pereira 83], Earley's parsing algorithm can be considered an example of Chart parsing. Also according to the report in which the parsing is regarded as deduction (ex. Earley Deduction), the following holds:

(1) Chart parsing is a proof procedure using chart nodes as special arguments.
(2) A proof procedure provides a framework of an expanded CFG ("gap" and "dependency" processing).

Because POPS basically is an equivalent of Earley Deduction as described in the previous subsection, the above discussion by Pereira can also be directly applied to the relationship between Chart parsing and POPS. In conclusion, Fig.4 gives the interrelationship among the parsing algorithms described in this section.

## 5. Discussion

Using channels as message-transferring paths, POPS permits processes to always send a message. A channel forms an 'unbounded buffer' [Takeuchi 83]. The characteristic of the POPS channel determine the overall computation behavior. If we introduces an 'bounded buffer' communication to the POPS, the system operation can be controlled. That is, the unbounded buffer enables the computation to be executed in an eager manner, while the bounded buffer in a lazy fashion. While the POPS-based parsing is basically equivalent to Earley Deduction in parsing operation, POPS can perform Earley's algorithm in this flexible manner. Eager and lazy executions will be reported on another paper [Hir 84].

## 6. Conclusion

This paper described the natural language parsing as an application field of the POPS and estimated the computation time required for CFG parsing, then compared the POPS-based parsing with analyses by other parsing algorithms. As a result, we found that POPS-based parsing can

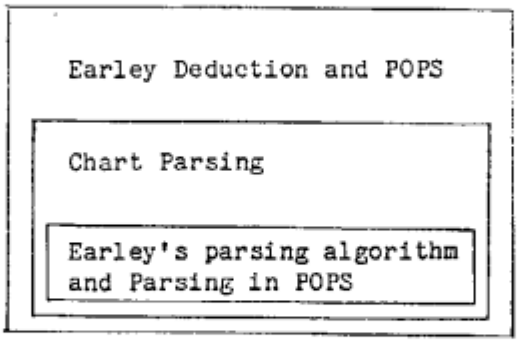



Fig.4 The relation between POPS and other systems

analyze an input string on the same time order as Earley's parsing algorithm when the parsing time estimate is done with the input string length as a parameter. Also this paper shows POPS has a close relation with Earley Deduction system which is a generalization of Earley's parsing algorithm.

## Acknowledgement

We would like to thank Kondo for his suggestions on the algorithm analysis, and Shibayama, Sakai and Yokomori (at Fujitsu Kokusai Laboratory) for their discussions on the parsing time order.

## References

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

[Pereira 83] Pereira, F. and Warren, D. H.: "Parsing as Deduction", SRI Technical Note 295, 1983.

[Matsumoto 83] Matsumoto, Y., Tanaka, H. et. al.: "BUP: A Bottom Up Parser embedded in Prolog", New Generation Computing vol. 2, 1983.

[Harrison 78] Harrison,M.A.: "Introduction to formal Language Theory", Addison Wesley pub., 1978

[Kay 80] Kay,M.: "Algorithm Schemata and Data Structures in Syntactic Processing", Xerox Technical Report, 1980.

[Shapiro 83] Shapiro,E.Y.: "A Subset of Concurrent Prolog and Its Interpreter", ICOT Technical Report TR-003, 1983.

[Takeuchi 83] Takeuchi,A and Furukawa,K: "Interprocess Communication in Concurrent Prolog", Proc. of Logic Programming Workshop, 1983.

[Hirakawa 83] Hirakawa,H.: "Chart Parsing in Concurrent Prolog", ICOT Technical Report TR-008, 1983.

[Hirakawa 83] Hirakawa,H. Onai,R. and Furukawa,K.: "Implementing POPS in Concurrent Prolog", ICOT Technical Report TR-020, 1983.

[Hirakawa 84] Hirakawa,H. Chikayama,T. and Furukawa,K.: "Eager and Lazy Enumerations in Concurrent Prolog", ICOT Technical Memo TM-036, 1984.

[Winograd 83] Winograd,T.: "Language as a Cognitive Process", Addison

Wesley pub., 1978.

[Aho 72] Aho,A.V. and Ullman,J.D.: "The Theory of Parsing, Translation, and Compiling, vol.1 Parsing", Prentice-Hall pub., 1972.

[Earley 68] Earley, j., "An Efficient Context-free Parsing Algorithm," Ph.D. Thesis, Carnegie-Mellon University, 1968.