Formal Properties and Implementation of Bidirectional Charts

Giorgio Satta
University di Padova, via Belzoni 7
35131 Padova, Italy
and
Istituto per la Ricerca Scientifica e Tecnologica
38050 Povo, Tnento, Italy

Oliviero Stock
Istituto per la Ricerca Scientifica e Tecnologica
38050 Povo, Trento, Italy

Abstract

Several theories of grammar currently converge toward inserting subcategorization information within lexical entries. Such a tendency would benefit from a parsing algorithm able to work from "triggering positions" outward. In this paper a bidirectional extension of the chart algorithm is proposed and its most relevant formal properties are investigated using an Earley like formalism. The question as to whether the proposed approach guarantees computational feasibility is then addressed and is answered affirmatively. Also, it is shown that the method presented here maintains those characteristics that were so much appreciated in monodirectional charts. Finally, from this analysis indications arc derived for an efficient implementation of the parsing algorithm.

1 Motivation

It is superfluous to point out the relevance of charts [Kay, 1973, 1980, Kaplan, 1973] for natural language parsing. A didactic presentation of this technique is reported in [Thompson and Ritchie, 1984]. Yet, one constraining aspect of chart parsing is that it is directional, i.e. edges can be combined only in one fixed order (usually active left - inactive right). The fact that the edge extension operation is meant to be carried out in one direction, normally is not critical, given that most modern grammar theories use a monotonical approach. Of course, things are not exactly the same with fragmentary input: it may be impossible to start from one corner of a constituent, simply because the fragment may not include that corner. But even with well-formed input something somehow different may be desirable. There is a strong tendency in grammatical formalisms to privilege a particular element inside a given constituent, called the head of the constituent, that , in a number of cases, guides the acceptability of elements to its left as well as to its right. Should the parsing algorithm be able to work from the head outward, then it would benefit from a representation developed accordingly. More concretely, a number of possible partial interpretations would be pruned out earlier, on the basis of functional information attached to the head. This results in greater efficiency.

It may be worth recalling that X-bar theory [Jackendoff, 1977] establishes that the head of a constituent (phrase) individuates the categorial labels of the nodes that dominate it, up to two or three nodes above, and provides a general skeleton for the related branches. A number of grammar theories are influenced in one form or another by this idea. More recently, also, some grammar formalisms have explicitly emphasized the role of head-driven paradigm, e.g. HPSG [Sag and Pollard, 1987].

A proposal for bidirectional chart parsing was given in [Stock *et al.,* 1989], though mainly as an heuristic tool for controlling the recognition of spoken input. Another work on bidirectional charts [Steel and De Roeck, 1987] introduces heuristics that determine how a constituent is to be analyzed. Some other works in the formal languages literature [Bossi *et* a/., 1983] have dealt with some form of bidirectionality within a tabular approach, such as Earley's or Kasami-Cocke-Younger's.

Before proceeding any further, one should examine the computational feasibility of the proposed approach. Less fundamental, but still very relevant, is the question whether the method enjoys those characteristics that proved so useful in monodirectional charts. In Section 3 it will be shown that both these questions are answered affirmatively. A consequence of this analysis may also result in indications for a good implementation of the algorithm.

After having briefly stated, in the next section, the proposal for bidirectional chart parsing, coherently with the motivations stated above, the approach will be expressed in terms of a formalized extension of Earley's algorithm. Some formal definitions and proofs of basic theorems then will be given that guarantee the desired formal properties. Following that, aspects of the implementation are described, including a revision of the usual agenda-based execution. In the last part of the paper this approach is compared with other strategies for chart parsing.

2 A synopsis of bidirectional charts

In this section bidirectional charts are compared descriptively with monodirectional charts.

The main difference in data structures results in active edges that include two markers, say *Idot* and *rdot,* to indicate the borders of the recognized portion of the right-hand side of the specified production rule. The algorithm works outward from particular elements (triggering elements) that can occur in any position in a given constituent. Such positions within the right-hand side of a production rule are named triggering positions. The only requirement is that at least

one such position is indicated for each constituent (therefore the case of a grammar with one head per constituent is a particular case).

To accomplish that, two edge combination rules are introduced. The first rule is a bidirectional extension of the chart "fundamental" rule: it allows the combination of an active edge with an inactive edge occurring either to its right or to its left. The second rule allows the combination of two adjacent active edges, "working" on the same production rule and with the *rdot* of the leftmost edge coinciding with the *ldol* of the rightmost edge. This is a means of merging two attempts into a single, more advanced, attempt to analyze a constituent.

The application of any of the two above rules is subject to a check in the chart so that the edge combination operation is really extending in an innovative way the attempts present in the chart under the form of active edges. (This point will become clearer in the following sections). The general control strategy is bottom-up: whenever an inactive edge of a given category is introduced into the chart, for all the possible production rules that have occurrences of that category in a triggering position, an active edge is introduced, spanning just the given inactive edge.

In the following these concepts are formalized in terms of an extension of the Earley algorithm [Earley, 1970]. This allows use of formalizations and results that arc well established in the literature. Also, it is well known that chart parsing can be considered homomorph to Earley parsing, apart from the facts that 1) an explicit concept of active edge is introduced 2) the combination of a chart with an agenda separates the issue of control from the algorithm itself. Therefore the formal results described hold for this chart-based approach. It is returned to in Section 4, which deals with implementational issues, and includes a new organization of active edges and a new use of the agenda.

3 An extension of the Earley Algorithm

This section presents a formal extension of the Earley algorithm [Earley, 1970]. The formalism chosen here was in part inspired by the representation used in [Graham and Harrison, 1976], but it is rather different from the original one, given that now we are free to parse in any direction.

3.1 Preliminaries

Assume a context-free grammar G=(N, Z, P, S), where N is the finite set of all non terminal symbols, Z is the set of all terminal symbols, disjoint from N, P is the finite non empty set of production rules, and S is the so called start symbol. Let L(G) denote the language generated by the grammar G. Each rule in P is notated with the form D_D -> $C_{p.l}$ ...$C_{p,n(p)}$, where is a function from the set {1 .. IPI) to N^+ (in the remaining part of this section the integer *p* is used for the correspondent production rule in P). For simplicity, the grammar G is expressed in an e-production free form, without loss of generality [Aho and Ullman, 1972:148], though the algorithm can be easily reformulated for the general case. Let each production rule *p* in P have one or more triggering positions in its right-hand side, with the intended meaning that each symbol in such a position, once parsed, can trigger the production *p*. Let $w= a_1 \ldots a_n$, $a_i \in \Sigma$ where $1 \le i \le n$, be an input string; the notation $_iw_j$ will be also used for the substring $a_{i+1} \ldots a_j$.

A *state* is defined as a quintuple of the form [*p*, *ldot*, *rdot*, *m*, *h*], where $1 \le p \le |P|$, $0 \le ldot < rdot \le \pi$ (p), $m \in$ {-, *lm*, *rm*}, $h \in$ {-, +}. The *p* component stays for the correspondent production in P, *ldot* and *rdot* represent two different successive positions in the right-hand side of *p*. The components *m* and *h* serve as indices: *m=lm/rm* means that the value of *ldot/rdot* cannot be diminished/augmented any more,[1] *m*=- indicates the absence of the above limitations, *h* indicates whether the state is in or subsumes a triggering position. Each state is sometimes referred to as a "chunk" of analysis for the right-hand side of the correspondent rule *p*, going from constituent $C_{p,ldot+1}$ to constituent $C_{p,rdot}$. Let I_S be the set of all states. Assume the following definition for the function F:

$F: N \cup \Sigma \rightarrow I_s$

$F(X)=\{s=[p, ldot, ldot+1, -, h] \mid X=C_{p,ldot+1},$ *h*=+ iff $C_{p,ldot+1}$ is within a triggering position in the production rule *p*}

It is understood that the set F(X) consists of all the chunks of productions' right-hand sides that can be filled by the symbol X. An equivalence relation *Q* in $I_s \times I_s$ is defined so that for two generic states *s*=[*p*, *ldot*, *rdot*, *m*, *h*] and *s'*=[*p'*, *ldot'*, *rdot'*, *m'*, *h'*], *sQs'* holds if and only if *p=p'*, *ldot=ldot'* and *rdot=rdot'*. The algorithm will use an $(n+1)\times(n+1)$ matrix T as a program variable, where each $t_{i,j}$ element is an initial empty set of states.

The algorithm presented in the next section satisfies the following invariant:

Invariant 1

(i) $s=[p, ldot, rdot, m, h] \in t_{i,j}$ *only if* $C_{p,ldot+1} \ldots C_{p,rdot} \overset{*}{\Rightarrow} a_{i+1} \ldots a_j$, $i<j$;

(ii) $C_{p,ldot+1} \ldots C_{p,rdot} \overset{*}{\Rightarrow} a_{i+1} \ldots a_j$, $i<j$ *and* some $C_{p,k}$, $ldot+1 \le k \le rdot$ is within a triggering position for production rule *p only if* a quadruple $u=[u_1, u_2, u_3, u_4]$, $u_q \ge 0$, $1 \le q \le 4$ exists such that $s=[p, ldot-u_1, rdot+u_2, m, h] \in t_{i-u3,j+u4}$;

(iii) $s=[p, ldot, rdot, lm, h] \in t_{i,j}$ *only if* $s'=[p, ldot, rdot+k, m', h'] \in t_{i,j'}$, $k>0$, $j'>j$;

(iv) $s=[p, ldot, rdot, rm, h] \in t_{i,j}$ *only if* $s'=[p, ldot-k, rdot, m', h'] \in t_{i',j}$, $k>0$, $i'<i$;

[1] Note that this has nothing to do with having reached the initial or final position in the right-hand side of the production rule.

3.2 Recognition Algorithm

A recognizer is a procedure that accepts a string we Σ^* iff *we* L(G); a parser is a procedure that, in addition, outputs one or more derivation trees for each accepted input Only a recognition algorithm will be presented here; the use of a simple algorithm able to reconstruct the derivation trees by interpreting the recognition matrix T (see for example [Graham and Harrison, 1976]) is sufficient to obtain a parser algorithm.

The resulting algorithm inserts in the recognition matrix T each chunk of analysis previously obtained, and tries to extend it via left and/or right expanding processing (which combines the chunk at hand with other chunks in the matrix). Each chunk, once completed, gives rise to some new production hypotheses, which can be subsequently expanded only if the triggering position within the production is covered. The algorithm presented employs a program variable A as if it were a set-valued variable; this is done in order to separate in a straightforward way the control problem from the algorithm itself.

Algorithm 1 **Let G=(N, Σ, P, S) be** a context free grammar without e-productions. Let **w=$a_1 \ldots a_n$, $n>0$,** be an input string. Form an (n+l)x(n+l) matrix T (with elements *t* indexed from 0 to *n* in both dimensions) as follows.

begin
  for i **in** {1 .. *n*} **do**
    for *s* **in** F(a_i) **do**
      insert the triple e=(s, i-1, i) in the program variable A;
      (* we are going to process all the states in A*)
  while A not empty do
    extract any element *e=(s,* i, *j)* from the set A and insert state *s* in *t* ; apply each of the following procedures, in any order, to the element *e:*
      left-expander(e)
      right-expander(e)
      completer(e);
    (* we do not want to specify any execution order *)
  if some $s=[p, 0, \pi(p), m, h]$ is in $t_{0,n}$, for some *p* in P such that $D_p=S$
    then *output*(true)
    else *output*(error)
  end.

The three processes mentioned above will now be described.

3.2.1 Left-expander

precondition **Apply iff** $e=(s, i, j)$ **with** $s=[p, ldot, rdot, m, h]$, $ldot>0$, $m \neq lm$.

description **For every** $s'=[p, ldot', ldot, m', h'] \in t_{i',i}$, $i'<i$, $m' \neq rm$, **if at least one of the two components** h, h' **equals +, create the state** $s''=[p, ldot', rdot, -, +]$, **and set** $m'=lm$ **in** s'. **If at least one such** s' **was found, set** $m=rm$ **in** s. **Add the triple** $e'=(s'', i', j)$ **to A only if** $s''Qs_q$ **does not hold true for any state** s_q **in** $t_{i',j}$ **or for any triple** $e_q=(s_q, i', j)$ **in A.**

The procedure is applied if and only if it is possible to expand to the left the chunk at hand (*ldot*>0), and it has not already been expanded to its right (*m*≠*lm*): this is to say that its hypothesis is the rightward largest one. Then a search is made for every chunk of analysis regarding the production *p* that matches to the left with the chunk at hand, and that subsumes a suffix of the string $_0w_i$; a similar test is applied to the index *m.* Before combining two chunks, a check is made on the subsumed triggering positions: it is undesirable to expand two chunks, both of which are out of any triggering position. In all those cases of accepted combinations, the index *m* is marked in both the combined chunks, because they are now subsumed by the combination chunk. Note that, under the definition given above for the equivalence relation Q, the procedure never duplicates the triples in A, nor the states belonging to the same component of recognition matrix T.

3.2.2 Right-expander

precondition **Apply iff** $e=(s, i, j)$ **with** $s=[p, ldot, rdot, m, h]$, $rdot<\pi(p)$, $m \neq rm$.

description **For every** $s'=[p, rdot, rdot', m', h'] \in t_{j,j'}$, $j'>j$, $m' \neq lm$, **if at least one of the two components** h, h' **equals +, create the state** $s''=[p, ldot, rdot', -, +]$, **and set** $m'=rm$ **on** s'. **If at least one such** s' **was found, set** $m=lm$ **in** s. **Add the triple** $e'=(s'', i, j')$ **to A only if** $s''Qs_q$ **does not hold true for any state** s_q **in** $t_{i,j'}$ **or for any triple** $e_q=(s_q, i, j')$ **in A.**

This procedure is symmetric to the left-expander procedure, so the explanation is omitted.

3.2.3 Completer

precondition **Apply iff** $e=(s, i, j)$, **with** $s=[p, 0, \pi(p), m, h]$.

description **For every** s **in** $F(D_p)$, **update the program variable A with all the triples** $e=(s, i, j)$.

Every time the parsing of a constituent is completed, preparation begins for the processing of all the hypotheses in the set obtained from the application of the function F to that constituent symbol.

3.3 Formal Properties

Only the most interesting formal properties of Algorithm 1 are presented here; for a formal proof of Invariant 1 refer to [Satta, 1988]. The soundness and completeness of the algorithm are corollaries of the above invariant.

Note that there is a fundamental difference between Algorithm 1 and Earley's algorithm: Earley's algorithm extends each state from left to right, hence it suffices to avoid state duplication to save analysis duplication. Working within a bidirectional framework things are more complex. In fact if a state is expanded independently along two opposite sides, the two resulting states will be mutually overlapping and then, once completed, will unnecessarily duplicate analyses. Similar undesired duplications may arise from different states that stand for the same production and compete (from opposite sides) for the same constituent. Therefore a formal proof of the absence of

partial overlapping states is the basis for a proof of the absence of (partial) analysis duplication in Algorithm 1. More formally the overlap relation $\mathcal{D}$ is defined as a subset of $I_s \times I_s$, in the following way:

Definition 1 Partial overlap relation. Two states $s=[p, ldot, rdot, m, h] \in t_{i,j}$, $s'=[p, ldot', rdot', m', h'] \in t_{i',j'}$ are in partial overlap relation ($s \mathcal{D} s'$) if $i<i'<j<j'$, $ldot<ldot'<rdot<rdot'$ and s subsumes the *same* constituents $C_{p,ldot'+1} \ldots C_{p,rdot}$ subsumed by s'.

A formal proof for the following result is given below.

Theorem 1 Algorithm 1 never generates two states s and s' such that $s \mathcal{D} s'$.

Proof It suffices to show that the existence of a state $s=[p, ldot, rdot, m, h]$ such that $s \mathcal{D} s'$ leads to a contradiction, where s' is any other state generated by the algorithm. The proof will be by induction on the function $L(s)=rdot-ldot$ (note that $L(s) \geq 1$).

Basis For all s such that $L(s)=1$, the condition holds vacuously.

Inductive step Assume the hypothesis true for all s such that $L(s)<n$, $n>1$. Suppose that there exists a state $s=[p, ldot, rdot, m, h] \in t_{i,j}$ with $L(s)=n$, such that $s \mathcal{D} s_d$ for every s_d in some *non empty* set S_d. The state set $S_{min} \subseteq S_d$ is defined such that every state s_{min} in S_{min} is not subsumed by any other state in S_{min}. In other words the "shortest" states in S_d are used to form the state set S_{min}, which is not empty, from the hypothesis on S_d. Take a generic element $s_{min}=[p, ldot', rdot', m', h']$ in S_{min}, and suppose it was chosen from $t_{i',j'}$. From the inductive hypothesis it results that the state s cannot have been generated from some combination involving a state belonging to $t_{i'',j}$ with $i''<i'$ or $i''>i'$; hence, by virtue of the partial overlap relation, a state $s_x=[p, ldot', rdot, m_x, h_x] \in t_{i',j}$, $ldot'>ldot$, was involved in the combination, via the application of one of the two expander procedures. Note how, after that was done, m_x is set to rm in s_x. Consider now the construction of the set S_{min}: the state s_{min} above cannot have been generated from a combination extending a state in $t_{i',j''}$ with $j''<j$ or $j''>j$; therefore state s_{min} was generated from the application of an expander procedure extending exactly the same state s_x above, by virtue of the partial overlap relation and of the fact that the algorithm never duplicates states (i.e. s_x is unique). This results in a contradiction, because the expander procedure never applies to states with $m=rm$. Note that an application of an expander procedure to state s_x before its lm-marking would have blocked s state formation in a symmetrical way (s_x would have been rm-marked). Hence, the set S_d is empty.

The above result states the absence of partial overlapping analyses in the recognition matrix T. Such a property is obtained for Algorithm 1 by changing the value of the m component in some state previously inserted in T. This is done in dependence of the particular strategy used in expanding each state on both sides. But note also that general properties of monotonicity still hold for the algorithm with respect to the equivalence relation Q,

Complexity analysis follows for Algorithm 1. Only a sketch of the formal proof is given here.

Theorem 2 Algorithm 1 uses a total amount of space $O(n^2)$ and it lakes an amount of time $O(n^3)$, where n is the length of the input string.

Sketch of the proof There are only a finite number of different states, and this number is not dependent upon *n:* in fact every component of the state defining quintuple has a finite range independent from *n.* Algorithm 1 uses a number of elements in the recognition matrix T proportional to n^2 (to be precise, the algorithm uses only those elements t such that j>i and it carefully avoids duplication of any state in the generic element t . Considering that the states inserted in T are the all and only states inserted in A, it is concluded that Algorithm 1 uses a number of states proportional to n^2, hence the first part of the statement holds true. The processing of a state consists of the applications of the three procedures *completer, left-expander, right-expander.* The *completer* procedure, whenever it can apply, takes only a finite amount of time independent from *n.* The *left-expander* procedure, whenever it can apply, looks for all states in a column of the matrix T, spending a constant amount of time for each state. Note in fact that the check to avoid duplication of states within the same element in the recognition matrix can be done in constant (with respect to *n)* time; this is also true for the same check on the program variable A (organize A as an (n+l)x(n+l) matrix). The number of states in a column of the recognition matrix T is proportional to n, so the *left-expander* takes an amount of time proportional to *n.* By following similar reasoning for the *right-expander* procedure and combining the result with the above observations about the number of states created by the algorithm, it can be concluded that Algorithm 1 takes an amount of time $O(n^3)$.

4 How to Implement a Bidirectional Algorithm within the Chart Data Structure

The formalization used in the previous section was chosen because of its expository simplicity and its ease of use in proving relevant formal properties. Some of the concepts expressed in Section 2 will now be presented in more detail, suggesting an efficient way of implementing Algorithm 1 using a chart framework, along with some improvements in data structures and bookkeeping.

4.1 Implementation-oriented Representation

To implement Algorithm 1 using the chart, it is sufficient to represent each vertex with the usual two fields: leftward and rightward. The leftward field of the j-th vertex can access all the states in the (upper) j-th column of the previously defined recognition matrix T, and the rightward field can access all the states in the (right) *j-lh* row of the same matrix. Each partially completed state is represented with an active edge, and each completed state with an inactive edge.

For the two expander procedures, all the edges (states) included in each vertex field are partitioned by production rule, along with a position within the given rule: this renders the edge search more efficient. Each set within the partition is called an *edge-selection.* With such a representation it is possible to omit the somewhat inefficient splitting of each completed state done by the function F within the *completer* procedure. Such a splitting will be necessary only when the completed states correspond to a triggering position within some production rule. In all other cases it suffices to insert the completed state (edge) in the edge-selection of the incident vertices, with the proviso of using an appropriate representation for the marking field m, as is done in the next section.

4.2 The Agenda and the Concept of Delayed Evaluation

Implementations of the chart algorithm often make use of a data structure, called the *agenda,* to retain all the established tasks. Generally a task is a proposed combination of two edges: the use of an agenda separates the control problem from the algorithm schemata in a straightforward way.

The algorithm presented in the previous section combines two edges only if they have not already been subsumed by other edges at the moment of the combination. This is done to avoid analysis duplication. Working with an agenda is less straightforward in such cases. In fact, the condition about the absence of subsuming edges, on the basis of which the combinations to be inserted in the agenda are established, may no longer be true at the moment combinations are picked up from the agenda. The problem is therefore one of a "delayed evaluation" of which edge combinations are to actually be performed. One obvious but inefficient solution would be to erase some tasks from the agenda when the involved edges are subsumed by other edges. The following, more efficient implementation, is proposed here. Instead of treating the subsumption marking by means of a specific index for each state (as was done in the previous section), it may be more convenient to use a partial ordering relation (<) among edges exiting from a given vertex. Given two edges *e* and *a* (the latter being necessarily active) belonging to the same edge-selection, *e<a* iff *e* is subsumed by *a.* The above partial ordering relation gives rise to tree forest structures, each one associated with an edge-selection and each one composed of edges. In this way, each edge belonging to the yield of a tree in some edge-selection *s,* within a direction field, say *d,* of some vertex v, will subsume all its ancestors (in that tree), that are edges leaving from vertex v in direction *d* with regard to edge-selection *s.* The algorithm takes care only of those edges that are nodes in the yields of some forest; hence it is worth representing each edge-selection in such a way that it contains exactly those edges that belong to the yields of the associated forest Every time a new (active or inactive) edge is inserted into the chart, the relevant edge-selections at both the involved vertices are updated. The problem of the "delayed evaluation" within the agenda is then solved in the following way. Each task is represented as an ordered pair of two adjacent edge-selections belonging to the same vertex. The two edge-selections are *adjacent* in the sense that they refer to the same production rule and to the same position within that production, but are opposed in their directions. Each vertex also contains a field, called *active-e-sels,* in which the algorithm takes note of those edge-selections involved in a task present in the agenda. Every time a task is picked up from the agenda, cross combinations between the two yields are executed. In doing this a difference must be drawn between edge pairs that had been already combined before the insertion of the current task in the agenda and edge pairs that had not. The former case may have occured if an equivalent task (i.e. one involving the same adjacent edge-selections) had been previously extended at a time when the above edges were already in the respective edge-selections.[2] The active-e-sels field of the involved vertex is updated accordingly. The new edges that resulted from the execution of the task are inserted in the chart; edge-selections are then updated and new tasks will be inserted into the agenda only if they were not already there. Figures 1 and 2 report a brief example. In Figure 1 an active edge for a production rule p: X—> ABC spans from vertex 2 to vertex 3, with a subsumed inactive edge of category B, and consequently two tasks have been established and inserted in the agenda. For each vertex field (leftward, rightward and active-e-sels) the relevant edge-selection is also depicted. After the first task is picked up from the agenda (i.e. 2: A.BC) and the combination edge *e* is inserted in the chart, no new task is inserted in the agenda (the active-e-sels field at vertex 3 is active for the relevant edge-selection) but the relevant edge-selection at vertex 3 is updated (Figure 2). When the last task in the agenda (i.e. 3:AB.C) is executed, the subsumed edge e_4 is no longer present in the edge-selection; its descendant edge *e* is then combined, and partial overlapping is avoided.

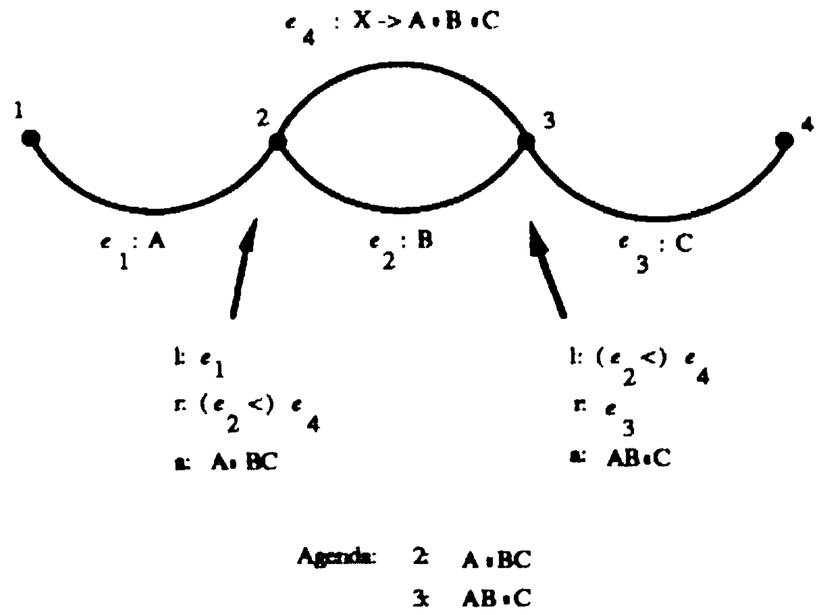

Fig. 1

[2] In the actual implementation, the (dynamic) distinction between these two classes of edges is carried out in the following way. Each yield is represented by means of a list of edges, *l*. Every time an edge *e* gets subsumed by some set of edges E, *e* is deleted from *l* (if it was still there) and the new edges in E are inserted at the top of *l*. To obtain the distinction mentioned in the text, it sufficies to update a pointer to the current value of *l* whenever a task involving the edge-selection represented by *l* is inserted in the agenda. When a task is executed, each pair of edges that are both in the respective "old" list, will not be combined.

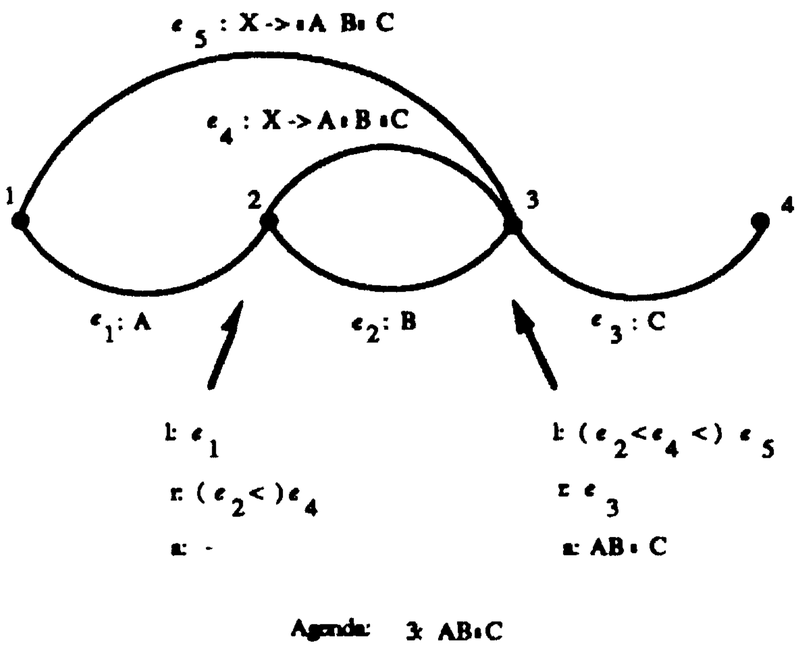

Fig. 2

This implementation solves the problem of "delayed evaluation" within the agenda, and avoids dealing with sterile tasks. Note also that the yield updating operation and the active-e-sels field updating operation take only a constant amount of time, so the complexity results of Theorem 2 still is obtained for the proposed implementation.

5 Conclusions

This paper has described some relevant formal properties of bidirectional charts, a concept that may prove to be a step forward in nondeterministic natural language parsing. Some relevant theorems have been proved, adopting an Earley style formalism. Some aspects of the implementation of bidirectional charts have been also shown, together with the notion of "delayed evaluation" of tasks to be executed. We shall conclude with a brief discussion of the possible application of this technique compared with traditional bottom-up parsing.

Bottom-up parsing, in its many implementations [Wir6n, 1987] can be seen as a particular, restricted application of our algorithm, in which triggering positions are assigned to the first constituent position of the right-hand side of each production rule; in such a case analysis direction is forced to go from left to right. Restricting the triggering position to the left is inefficient for two reasons. First of all, such a position is often occupied by elements that can be modifiers at the beginning of different constituents, and therefore the early identification of the "current" constituent is fuzzy. In the second place, the left position is often occupied by optional phrases and this fact brings in the well known parsing problem of right common factors within production rules [Wang, 1985]. The algorithm here discussed is more flexible in the sense that triggering positions can be chosen for the most convenient positions within the right-hand side of the production rules. Finally, provided a representation for production rules that makes use of regular expressions (see for example LFG [Kaplan and Bresnan, 1982]), the presented algorithm can take advantage of right common factors as much as of left common factors.

References

[Bossi *et al.,* 1983] A. Bossi, N. Cocco and L. Colussi. A divide-and-conquer approach to general context-free parsing. *Information Processing Letters,* 16 - pp. 203-208, 1983.

[Earley, 1970] Jay Earley. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM,* 13 - pp. 94-102, 1970.

[Graham and Harrison, 1976] Susan L. Graham and Michael A. Harrison. Parsing of General Context Free Languages. *Advances in Computers* - pp. 77-185 - , Academic Press, New York, 1976.

[Jackendoff, 1977] Ray Jackendoff. X-bar Syntax: A Study of Phrase Structure. The MIT Press, Cambridge, Massachusetts, 1977.

[Kaplan, 1973] Ronald M. Kaplan. 1973. A General Syntactic Processor. In: (R. Rustin, ed) *Natural Language Processing* - pp. 193-241 - , Algorithmics Press, New York, New York, 1973.

[Kaplan and Bresnan 1982] Ronald M. Kaplan and Joan Bresnan. Lexical Functional Grammar: A Formal System for Grammatical Representation. In: (J. Bresnan, ed) *The Mental Representation of Grammatical Relations* - pp. 173-281 -, The MIT Press, 1982.

[Kay, 1973] Martin Kay. The MIND System. In: (R. Rustin, ed) *Natural Language Processing* - pp. 155-188 - , Algorithmics Press, New York, New York, 1973.

[Kay, 1980] Martin Kay. Algorithm Schemata and Data Structures in Syntactic Processing. *Tecnical Report CSL-80* Xerox-PARC, Palo Alto, California, 1980.

[Sag and Pollard, 1987] Ivan Sag and Carl Pollard. Head Driven Phrase Structure Grammar: An Informal Synopsis.. Report No. CSLI 87-79, CSLI, Stanford University, Stanford, 1987.

[Satta, 1988] Giorgio Satta. Some Results for a Bidirectional Extension of Earley's Algorithm. *Draft,* IRST, Trento, 1988.

[Steel and De Roeck, 1987] Sam Steel and Anne De Roeck. Bidirectional Chart Parsing. *Proceedings of AISB-87,* Edinburgh, Scotland, 1987.

[Stock *et al.,* 1989] Oliviero Stock, Rino Falcone and Patrizia Insinnamo. Bidirectional Charts: a Potential Technique for Parsing Spoken Natural Language. Computer Speech and Language, 3, 1989.

[Thompson and Ritchie, 1984] Henry Thompson and Graeme Ritchie. Implementing Natural Language Parsers. In: (T. O'Shea, M. Eisenstadt eds) *Artificial Intelligence* - pp. 245-300 -,Harper & Row, New York, 1985.

[Wang, 1985] Weigno Wang. Computational Linguistics Technical Notes No. 2.. Technical Report 85/013, Computer Science Department, Boston University, Boston, Massachusetts, 1985.

[Wir6n, 1987] Mats Wir6n. A Comparison of Rule-Invocation Strategies in Context-Free Parsing. *Proceedings of the 3rd Conference of the European Chapter of the Association for Computational Linguistics,* Copenhagen, Denmark, 1987.