A General, Sound and Efficient Natural Language Parsing Algorithm based on Syntactic Constraints Propagation*

José F. Quesada
CICA (Centro de Informática Científica de Andalucía)
Sevilla, Spain
e-mail: josefran@cica.es

Abstract

This paper presents a new context-free parsing algorithm based on a bidirectional strictly horizontal strategy which incorporates strong top–down predictions (derivations and adjacencies). From a functional point of view, the parser is able to propagate syntactic constraints reducing parsing ambiguity. From a computational perspective, the algorithm includes different techniques aimed at the improvement of the manipulation and representation of the structures used.

1 Parsing Ambiguity and Parsing Efficiency

In Formal Language Theory [Aho & Ullman 1972, Drobot 1989] a language is a set, and in Set Theory an element belongs or not to a set. That is to say, a set (and therefore a language) is an unambiguous structure. A grammar may be considered as an intensive definition of a language. Thus, the notion of grammaticality corresponds to the relation of membership over a language (set). But a grammar incorporates more information than the simple report of the elements of the language (the extensive specification). A grammar defines a structure: the parse tree or forest. The distance between grammaticality and grammatical structure is a first level of ambiguity: grammatical ambiguity.

The next notion to take into account is the process of analysis of a string of words with a grammar, that is, the parser [Kay 1980, Bolc 1987, Sikkel & Nijholt 1997]. A parser must be able to determine the relation of grammaticality and to obtain the grammatical structure, by mean of a set of operations, that we will call the parsing structure. The distance between the grammatical structure and the parsing structure defines a second level of ambiguity: parsing ambiguity, usually referred as temporal ambiguity.

Parsing ambiguity depends on two factors: the grammar and the parsing strategy. A very important design requirement of natural language parsers is to eliminate parsing

*Jose F. Quesada: A General, Sound and Efficient Natural Language Parsing Algorithm based on Syntactic Constraints Propagation. *Proceedings of CAEPIA'97*, M/'alaga, Spain. 775–786

ambiguity, that is, to reduce the work done by the parser to the amount of grammatical structures allowed by the grammar. The work presented here is a step more in this direction [Earley 1970, Kay 1980, Tomita 1987, Tomita 1991, Dowding et al. 1994, **?**].

The second goal of this paper is to present a computational model aimed at the improvement of the efficiency of the algorithm [Carroll 1994, Quesada & Amores Forthcoming]. In this sense, our proposal may be understood as the incorporation of strong top–down predictions (partial derivations and adjacencies) over a bottom–up framework.

And the two strategies (bottom–up and top–down) are mixed by a mechanism able to propagate syntactic constraints over a bidirectional model based on a strictly horizontal strategy [Quesada 1996].

Section 2 presents an informal introduction to the problem of parsing ambiguity with chart parsing [Kay 1980], but similar situations may be described for other strategies like Earley's algorithm [Earley 1970], DCG [Pereira & Warren 1980], GLR [Chapman 1987, Tomita 1991], etc. Section 3 defines formally the relations that support the mechanism of bottom–up bidirectional analysis, top–down predictions and constraints propagation. Section 4 presents in detail the parsing algorithm and finally Section 5 shows some experimental results.

2 An Informal Introduction

Let us consider the following grammar:

```
S  -> A1 b
S  -> A2 c
A1 -> a
A1 -> a A1
A2 -> a
A2 -> a A2
```

and the string of words `a a a b`. Figure 1 shows the arcs genereted by a bidirectional chart parser in a first stage where we have created only the arcs with at least one pre–terminal symbol. Each arc has been identified by a number, and indicates the symbol that the arc will generate and the expected symbol (only for active arcs).

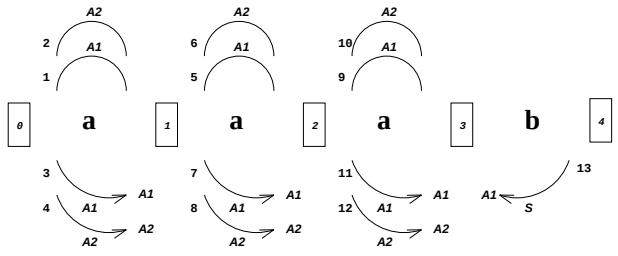

Figure 1

Let us consider what happens at position 3. There exists an obvious relation between arcs `13` and `9`, but arcs `10`, `11` and `12` don't have a correspondent link at this position. Figure 2 shows the parsing state once we have deleted these three arcs.

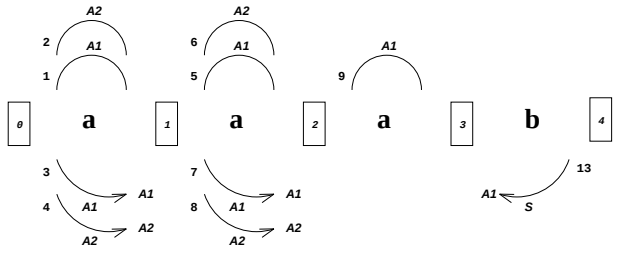

Figure 2

At position 2 there exists a relation between arcs 7 and 9, and arcs 5, 6 and 8 may be deleted. Once we have deleted these three arcs, if we analyze position 1 we can delete now arcs 1, 2 and 4 obtaining Figure 3.

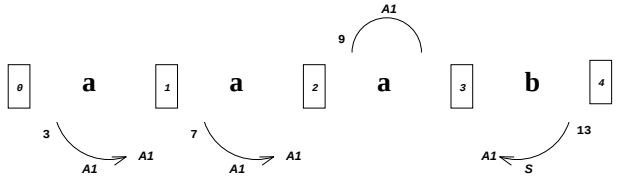

Figure 3

Therefore, our next goal will be to define formally the relations between arcs that guarantee their success during parsing.

3 The Mathematical Kernel

3.1 Bottom-Up Derivation

Given a context-free grammar $G =< G_T, G_N, G_P, G_R >$ where we have distinguished their set of productions G_P, roots G_R, terminal symbols G_T, non-terminal symbols G_N and vocabulary $G_V = G_T \cup G_N$, we will define the bottom-up derivation as follows. Let be $\delta \in G_V$ and $\Delta, \Gamma, \Omega \in G_V^*$. The direct bottom–up derivation in G, $\longrightarrow_G$, is defined as:

$$\Gamma\Delta\Omega \longrightarrow_G \Gamma\delta\Omega \quad \text{iff} \quad \delta \longrightarrow \Delta \in G_P$$

The bottom–up derivation in G, $\Longrightarrow_G$, will be defined as the reflexive and transitive closure of the direct bottom–up derivation:

$$\Gamma \Longrightarrow_G \Omega \quad \text{iff} \quad \exists \Delta_1, \ldots, \Delta_n \in G_V^* \text{ such that } \forall i_{(1 \leq i < n)} \Delta_i \longrightarrow_G \Delta_{i+1}$$
$$\text{where } \Delta_1 \equiv \Gamma \text{ and } \Delta_n \equiv \Omega$$

3.2 Partial Derivability and Adjacency

Root Symbols. α is a root symbol: $R(\alpha)$ iff $\alpha \in G_R$

Epsilon Symbols. α is an epsilon symbol: $E(\alpha)$ iff $\varepsilon \Longrightarrow_G \alpha$ [1]

[1] ε is the empty string.

String of Epsilon Symbols. Δ is a string of epsilon symbols: $E(\Delta)$ iff $\forall \delta \in \Delta(E(\delta))$

Left Partial Derivability. β is a left partial derivation of α:

$$\alpha \longmapsto_l^* \beta \quad \text{iff} \quad \exists \Gamma, \Delta, \Omega \in G_V^* \text{ such that } (\Gamma\alpha\Delta \Longrightarrow_G \Gamma\beta\Omega).$$

We define $LPD(\alpha) = \{\beta \in G_V : \alpha \longmapsto_l^* \beta\} \cup \{\alpha\}$ [2]

Right Partial Derivability. β is a right partial derivation of α:

$$\alpha \longmapsto_r^* \beta \quad \text{iff} \quad \exists \Gamma, \Delta, \Omega \in G_V^* \text{ such that } (\Gamma\alpha\Delta \Longrightarrow_G \Omega\beta\Delta)$$

We define $RPD(\alpha) = \{\beta \in G_V : \alpha \longmapsto_r^* \beta\} \cup \{\alpha\}$ [3]

Primary Adjacency. β is a primary adjacent of α:

$$\alpha \Uparrow \beta \quad \text{iff} \quad \exists \delta \in G_V \text{ and } \exists \Gamma, \Omega, \Delta \in G_V^* \text{ such that } (\delta \longrightarrow \Gamma\alpha\Delta\beta\Omega \in G_P \wedge E(\Delta))$$

Left Adjacency. β is a left adjacent of α:

$$\alpha \Uparrow_l^* \beta \quad \text{iff} \quad \exists \gamma \in LPD(\alpha) \text{ and } \exists \delta \in RPD(\beta) \text{ such that } (\delta \Uparrow \gamma)$$

We define $LA(\alpha) = \{\beta \in G_V : \alpha \Uparrow_l^* \beta\}$.

Right Adjacency. β is a right adjacent of α:

$$\alpha \Uparrow_r^* \beta \quad \text{iff} \quad \exists \gamma \in RPD(\alpha) \text{ and } \exists \delta \in LPD(\beta) \text{ such that } (\gamma \Uparrow \delta)$$

We define $RA(\alpha) = \{\beta \in G_V : \alpha \Uparrow_r^* \beta\}$.

Left–Most Symbol. α is a left–most symbol:

$$LM(\alpha) \quad \text{iff} \quad \exists \delta \in G_V \text{ such that } (\alpha \longmapsto_l^* \delta \wedge R(\delta))$$

Right–Most Symbol. α is a right–most symbol:

$$RM(\alpha) \quad \text{iff} \quad \exists \delta \in G_V \text{ such that } (\alpha \longmapsto_r^* \delta \wedge R(\delta))$$

3.3 Coverage Tables

Finally we present the formal definition of the coverage tables which are in charge of triggering the events of the bidirectional parser.

For each symbol of a grammar, $\alpha \in G_V$, their left, $LC1(\alpha)$ and $LC2(\alpha)$, medium, $MC(\alpha)$, and right, $RC(\alpha)$, coverages are defined as sets of productions in the following way:

$$LC1(\alpha) = \{(\delta \longrightarrow \alpha \in G_P) : \delta \in G_N\}$$
$$LC2(\alpha) = \{(\delta \longrightarrow \alpha\Omega \in G_P) : \delta \in G_N \wedge \Omega \in G_V^* \wedge \neg E(\Omega)\}$$
$$MC(\alpha) = \{(\delta \longrightarrow \Delta\alpha\Omega \in G_P) : \delta \in G_N \wedge \Omega, \Delta \in G_V^* \wedge \neg E(\Delta) \wedge \neg E(\Omega)\}$$
$$RC(\alpha) = \{(\delta \longrightarrow \Delta\alpha \in G_P) : \delta \in G_N \wedge \Delta \in G_V^* \wedge \neg E(\Delta)\}$$

[2] We will consider that a symbol is a left partial derivation of itself.

[3] We will consider that a symbol is a right partial derivation of itself.

4 The Parsing Algorithm

4.1 Parsing Input.

The main task of the lexical analyzer is to separate the input string in a set of items, each one associated with one or more (lexical ambiguity) pre-terminal symbols (syntactic categories). Our parsing algorithm is also able to deal with "multi-word expressions" and "multi-expression words"[4].

In any case, the parsing input will be a list of breaking points and a set of pre-terminal symbols, each one associated with a lexical unit (a portion of the input string) and two breaking points. For instance, we can consider the input string `a a a b`. This string will be lexically analyzed obtaining 5 breaking points and 4 pre-terminal symbols:

$$\boxed{0}\ a\ \boxed{1}\ a\ \boxed{2}\ a\ \boxed{3}\ b\ \boxed{4}$$

4.2 Step 1: CaD creation.

For each breaking point we will generate a CaD (collection and diffusion of information) structure, which has 6 fields: the first four fields are lists of $Events$ and the two last ones are lists of $Nodes$: *tole* (events arriving at the CaD from the right side), *frle* (events going to the left from the CaD), *tori* (events arriving at the CaD from the left side), *frri* (events going to the right from the CaD), *ndle* (nodes at the left of the CaD) and *ndri* (nodes at the right of the CaD).

If the lexical analyzer has obtained n breaking points, then we will store the CaD structures as a matrix of n pointers to CaD structures. We will call this matrix CaD_{root}.

4.3 Step 2: $Node$ creation.

For each element `<lexical_unit,pre-terminal_symbol,fbp`[5] `,lbp`[6] `>` we will generate a $Node$ structure, which has the following fields: *grsymbol* (grammar symbol) and *cmanalysis* (complex analysis, a list of *Analysis* structures). The new node $newNode$ will be associated with the corresponding CaD structures:

$$CaD_{root}[fbp]\texttt{->}ndri = AddNode(newNode)$$
$$CaD_{root}[lbp]\texttt{->}ndle = AddNode(newNode)$$

4.4 Step 3: $Event$ creation.

For each node created at step 2, we will generate their correspondent events using the coverage tables. An *Event* has the following fields: *grprod* (production or grammar rule), *leftdot* (left dot), *rightdot* (right dot), *leftlinks* (list of *Link* structures associated with the left extreme), *rightlinks* (list of *Link* structures associated with the right extreme) and *status* (logical status). Let us suppose that the node created ($newNode$) has been associated with the grammar symbol α. Then:

For each production $p \in LC1(\alpha)$ we will create the appropriate new event ($newEvent$) and:

[4]Words that contain more than one lexical unit, such as clitics in Spanish or compounds in German.
[5]The first or left breaking point of the lexical unit.
[6]The last or right breaking point of the lexical unit.

$$CaD_{root}[fbp]\texttt{->}frri = AddEvent(newEvent)$$
$$CaD_{root}[lbp]\texttt{->}frle = AddEvent(newEvent)$$

For each production $p \in LC2(\alpha)$ we will create the appropriate new event ($newEvent$) and:

$$CaD_{root}[fbp]\texttt{->}frri = AddEvent(newEvent)$$
$$CaD_{root}[lbp]\texttt{->}tole = AddEvent(newEvent)$$

For each production $p \in MC(\alpha)$ we will create the appropriate new event ($newEvent$) and:

$$CaD_{root}[fbp]\texttt{->}tori = AddEvent(newEvent)$$
$$CaD_{root}[lbp]\texttt{->}tole = AddEvent(newEvent)$$

For each production $p \in RC(\alpha)$ we will create the appropriate new event ($newEvent$) and:

$$CaD_{root}[fbp]\texttt{->}tori = AddEvent(newEvent)$$
$$CaD_{root}[lbp]\texttt{->}frle = AddEvent(newEvent)$$

4.5 Step 4: *Link* creation.

For each event created we have to analyze their possible links with other events. This operation is internal to the CaD structure according to the following criteria:

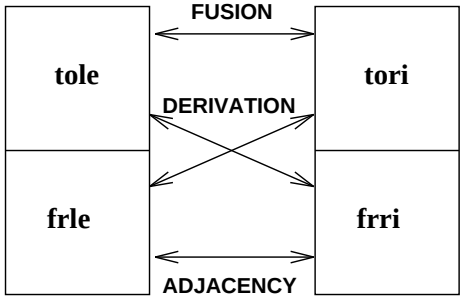

Figure 4. Links inside a CaD structure.

4.5.1 Analyses of Partial Derivations

These analyses are applied over the open extremes of an event. Basically, we have to check if the symbol needed is partially derivable from the real symbol.

We will distinguish two types of analyses of derivations depending on the direction of the open extreme of the event.

Left–Derivation Analysis: $TOLE - FRRI$**.** Let us suppose that an event $evtole$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\delta_h \dots \delta_i\gamma_1 \dots \gamma_j\Omega$ where $1 \leq h \leq i < n$ and $1 \leq j \leq m$. In fact, this event is making a prediction of the (required) symbol δ_{i+1} over the (real) symbol γ_1. The right extreme of this event will be associated with the component $tole$ of a CaD structure. In this case, we have to check if there exists a second event, $evfrri$, which left extreme belongs to the $frri$ field of the same CaD, such that $\delta_{i+1} \in LPD(\gamma)$, where γ is the left-hand side of the production associated with $evfrri$: $\gamma \longrightarrow \gamma_1 \dots \gamma_m$.

Right–Derivation Analysis: $TORI - FRLE$**.** Let us suppose that an event $evtori$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\gamma_j \dots \gamma_m\delta_h \dots \delta_i\Omega$ where $1 < h \leq i \leq n$ and $1 \leq j \leq m$. In fact, this event is making a prediction of the (required) symbol δ_{h-1} over the (real) symbol γ_m. The left extreme of this event will be associated with the component $tori$ of a CaD structure. In this case, we have to check if there exists a second event, $evfrle$, which right extreme belongs to the $frle$ field of the same CaD, such that $\delta_{h-1} \in RPD(\gamma)$, where γ is the left-hand side of the production associated with $evfrle$: $\gamma \longrightarrow \gamma_1 \dots \gamma_m$.

4.5.2 Analyses of Adjacencies

These analyses are applied over the closed extremes of an event. Basically, we have to check the adjacency relation between the symbol that the event will generate (the left–hand side of the production) and the symbol that appears next to the closed extreme of the event.

We will distinguish two types of analyses of adjacencies depending on the direction of the closed extreme of the event.

Left–Adjacency Analysis: $FRRI - FRLE$**.** Let us suppose that an event $evfrri$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\gamma_j \dots \gamma_m\delta_1 \dots \delta_i\Omega$ where $1 \leq i \leq n$ and $1 \leq j \leq m$. The left extreme of this event belongs to the $frri$ field of a CaD structure. Then we have to analyze if there exists an $evfrle$ event which right extreme belongs to the $frle$ field of the same CaD such that $\gamma \in LA(\delta)$, where γ is the lhs of the production of $evfrle$: $\gamma \longrightarrow \gamma_1 \dots \gamma_m$.

If $\Gamma\gamma_j \dots \gamma_m$ is empty, that is, the CaD associated with the left extreme of $evfrri$ is the first one, then we have to check if $LM(\delta)$.

Right–Adjacency Analysis: $FRLE - FRRI$**.** Let us suppose that an event $evfrle$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\delta_i \dots \delta_n\gamma_1 \dots \gamma_j\Delta$ where $1 \leq i \leq n$ and $1 \leq j \leq m$. The right extreme of this event belongs to the $frle$ field of a CaD structure. Then we have to analyze if there exists an $evfrri$ event which left extreme belongs to the $frri$ field of the same CaD such that $\gamma \in RA(\delta)$, where γ is the lhs of the production of $evfrri$: $\gamma \longrightarrow \gamma_1 \dots \gamma_m$.

If $\gamma_1 \dots \gamma_j\Delta$ is empty, that is, the CaD associated with the left extreme of $evfrle$ is the last one, then we have to check if $RM(\delta)$.

4.5.3 Analyses of Fusions

Left–Fusion Analysis: $TORI - TOLE$**.** Let us suppose that an event $evtori$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\gamma\delta_i \dots \delta_j\Delta$ where $1 < i \leq j \leq n$. The left extreme of this event belongs to the $tori$ field of a CaD structure. Then we have to analyze if there exists an $evtole$ event in the $tole$ field of the same CaD such that $evtole$ is applying the same production that $evtori$ over the surface $\delta_h \dots \delta_{i-1}$ where $1 \leq h$.

Right–Fusion Analysis: $TOLE - TORI$**.** Let us suppose that an event $evtole$ is applying the production $\delta \longrightarrow \delta_1 \dots \delta_n$ over the surface $\Gamma\delta_i \dots \delta_j\gamma\Delta$ where $1 \leq i \leq j < n$. The right extreme of this event belongs to the $tole$ field of a CaD structure. Then we have to analyze if there exists an $evtori$ event in the $tori$ field of the same

CaD such that *evtori* is applying the same production that *evtole* over the surface $\delta_{j+1} \dots \delta_k$ where $k \leq n$.

Link **creation.** Each time an analysis is successfull, we will generate a *Link* structure between the two events involved. For LM and RM analysis the *Link* will have only one event.

4.6 Step 5: Event's Logical Status.

Each time a *Link* is created we have to study the logical status of the events involved. Also, at the end of the analysis of the links of an event (step 4) we will analyze its logical status.

Let be e an event applying the production $\delta \longrightarrow \delta_1 \dots \delta_{i-1} \bullet \delta_i \dots \delta_j \bullet \delta_{j+1} \dots \delta_n$

4.6.1 Closed-Closed events ($FRRI + FRLE$): $i = 1$ and $j = n$.

```
if ((!e->leftlinks) || (!e->rightlinks))
    nstatus = DELETE
else
    nstatus = RUN
```

4.6.2 Closed-Open events ($FRRI + TORI$): $i = 1$ and $j < n$.

```
if (!e->leftlinks)
    nstatus = DELETE
else if (e->rightlinks)
    nstatus = DERIVATION
else if (E(δ_{j+1}))
    nstatus = EPSILON
else
    nstatus = DELETE
```

4.6.3 Open-Closed events ($TOLE + FRLE$): $i > 1$ and $j = n$.

```
if (!e->rightlinks)
    nstatus = DELETE
else if (e->leftlinks)
    nstatus = DERIVATION
else if (E(δ_{i-1}))
    nstatus = EPSILON
else
    nstatus = DELETE
```

4.6.4 Open-Open events ($TOLE + TORI$): $i > 1$ and $j < n$.

```
if ((e->leftlinks) && (e->rightlinks))
    nstatus = DERIVATION
else if ((e->leftlinks) && (E(δ_{j+1})))
    nstatus = EPSILON
else if ((e->rightlinks) && (E(δ_{i-1})))
```

```
        nstatus = EPSILON
    else
        nstatus = DELETE
```

If `nstatus` is different that `e->status` we will change the logical status of the event. To improve the efficiency it is possible to maintain four lists of events (DERIVATION, RUN, DELETE and EPSILON). To change the status of an event implies to move the event from one list to another, but this may be done in constant time.

4.6.5 Step 6: Parsing Cycle.

This is the kernel of the algorithm:

```
cycle = 1
while (cycle)
    cycle = 0
    if (event = GetEpsilonEvent())
        cycle = 1
        EpsilonExpansion(event)
    else if (event = GetDeleteEvent())
        cycle = 1
        DeleteEvent(event)
    else if (event = GetRunEvent())
        cycle = 1
        RunEvent(event)
    else if (link = GetFusionLink())
        cycle = 1
        FusionLink(link)
```

The functions `Get*` return the first element of the correspondent list and change the head of the list to the following element, which are constant operations.

6.1.- Epsilon Expansion. This operation moves the left dot one position to the left or the right dot one position to the right, depending on the open extreme marked as EPSILON.

6.2.- Delete Event. To delete an event implies to delete it and their links.

6.3.- Run Event. To run a closed-closed event involves the application of a grammar rule, incorporating a new node (step 2). But if this node has been previously created between the same *CaD* structures, we can obtain a representation model based on subtree-sharing and local ambiguity packing, associating the analysis correspondent to the last one with the previously created node. This way, a node will have a list of *Analysis* structures, and this structure is defined as a list of *Node* structures. The result of this mechanism is a representation based on *virtual* relations between the skeleton of the parse forest and the nodes included in it.

6.4.- Fusion Events. Let us consider the production $\delta \longrightarrow \delta_1 \ldots \delta_h \ldots \delta_i \delta_{i+1} \ldots \delta_j \ldots \delta_n$ and two events $e_1 : \delta \longrightarrow \delta_1 \ldots \bullet \delta_h \ldots \delta_i \bullet \delta_{i+1} \ldots \delta_n$ and $e_2 : \delta \longrightarrow \delta_1 \ldots \delta_i \bullet \delta_{i+1} \ldots \delta_j \bullet \ldots \delta_n$.

If there exists a fusion link ($e_1 e_2 link$) between e_1 (rightlinks) and e_2 (leftlinks) in the context $\delta_i \delta_{i+1}$ the application of their fusion will generate the following actions:

Case 6.4.1: Fusion with Double Derivation:

`if ((`e_1`->rightlinks) && (`e_2`->leftlinks))`

`Create a new event` $e_n : \delta \longrightarrow \delta_1 \ldots \bullet \delta_h \ldots \delta_i \delta_{i+1} \ldots \delta_j \bullet \ldots \delta_n$

Case 6.4.2: Fusion with Single Right Derivation:

`else if (`e_1`->rightlinks)`

`Modify` $e_2 : \delta \longrightarrow \delta_1 \ldots \bullet \delta_h \ldots \delta_i \delta_{i+1} \ldots \delta_j \bullet \ldots \delta_n$

Case 6.4.3: Fusion with Single Left Derivation:

`else if (`e_2`->leftlinks)`

`Modify` $e_1 : \delta \longrightarrow \delta_1 \ldots \bullet \delta_h \ldots \delta_i \delta_{i+1} \ldots \delta_j \bullet \ldots \delta_n$

Case 6.4.4: Fusion without Derivation:

`else`

`Modify` $e_1 : \delta \longrightarrow \delta_1 \ldots \bullet \delta_h \ldots \delta_i \delta_{i+1} \ldots \delta_j \bullet \ldots \delta_n$

`Delete` e_2

5 Implementation and Experimental Results

This algorithm has been implemented in C including a specific layer for the memory management that improves the classical operations of **malloc** and **free**.

Our experimental results show that this algorithm fully eliminates parsing ambiguity for recursive, local and non-local dependency constructions. For this kind of phenomena, the experimental results show a real complexity of the order $O(nlog(n))$ where n is the length of the input string. [7]

Next, we show the predicted model obtained for each type of grammar. The dependent variable T is the time used for the complete analysis (in seconds) and the factor used, W, has been the length of the input string (number of words).

We show the results obtained with a Simple Lineal Regression Test for two cases. The first one uses T as the response and *W log(W)* as the factor. The second one uses T/W as the response and the same factor *W log(W)*. In addition, we have included Pearson Correlation Coefficients for both cases.

- Recursive Constructions:

$$T = -5.183 + 219E - 7 * (Wlg(W)); PCC(T, Wlg(W)) = 0.999$$
$$T/W = 0.0001 + 46E - 13 * (Wlg(W)); PCC(T/W, Wlg(W)) = 0.974$$

- Local Dependencies

$$T = -17.82 + 352E - 7 * (Wlg(W)); PCC(T, Wlg(W)) = 0.993$$
$$T/W = 0.0002 + 38E - 12 * (Wlg(W)); PCC(T/W, Wlg(W)) = 0.998$$

- Non-local Dependencies

[7] [Quesada 1997] contains a full description of the algorithm as well as a more detailed analysis of the experiments, including the grammars, string of words and results.

$$T = -1.031 + 851 - 8 * (Wlg(W)); PCC(T, Wlg(W)) = 0.998$$
$$T/W = 0.0002 + 14E - 12 * (Wlg(W)); PCC(T/W, Wlg(W)) = 0.998$$

6 Conclusion

The problem of parsing natural languages must be studied from three perspectives: computational, linguistic and formal. In this paper we have presented a general, sound and efficient natural language parsing algorithm which accomplishes the main requirements of the three levels.

The computational layer includes a specific memory management model and a strategy for grammar compilation. This module has been designed with the goal of efficiency. The linguistic layer is in charge of general applicability, and includes basically a mecanism for the integration of the algorithm with unification grammar. Finally, at the formal level, the mathematical kernel proposed permits the demostration of the correctness and soundness of the algorithm [Quesada 1997].

This paper has concentrated on the description of the algorithm itself, describing the data model and the parsing strategy.

References

[Aho & Ullman 1972] Alfred V. Aho & Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling. Vol. I: Parsing.* Englewood Cliffs, N.J.: Prentice Hall.

[Bolc 1987] Leonard Bolc. ed. 1987. *Natural Language Parsing Systems.* Heidelberg: Springer–Verlag.

[Bunt & Tomita 1996] Harry Bunt & Masaru Tomita. eds. 1996. *Recent Advances in Parsing Technology.* Kluwer Academic Publishers.

[Carroll 1994] J. Carroll. 1994. Relating Complexity to Practical Performance to Natural–Language Processing. In *32rd Annual Meeting of the Association for Computational Linguistics*, pages 287–94.

[Chapman 1987] Nigel P. Chapman. 1987. *LR Parsing. Theory and Practice.* Cambridge: Cambridge University Press.

[Dowding et al. 1994] J. Dowding, R. Moore, F. Andry & D. Moran. 1994. Interleaving Syntax and Semantics in an Efficient Bottom–Up Parser. In *32rd Annual Meeting of the Association for Computational Linguistics*, pages 110–16.

[Drobot 1989] Vladimir Drobot. 1989. *Formal Languages and Automata Theory.* Rockville, MD: Computer Science Press.

[Earley 1970] Jay Earley. 1970. An Efficient Context–Free Parsing Algorithm. *Communications of the ACM*, **13(2)**, 94-102.

[Kay 1980] Martin Kay. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. *CSL-80-12 Xerox Palo Alto Research Center.*

[Pereira & Warren 1980] Fernando C. N. Pereira & David H. D. Warren. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, **13**, 231–78.

[Quesada 1996] José F. Quesada. 1996. Bidirectional and Event–Driven Parsing with Multi Virtual Trees. *II International Conference on Mathematical Linguistics*, Tarragona, May 1996.

[Quesada 1997] José F. Quesada. 1997. *El algoritmo SCP de Anlisis Sintctico mediante Propagacin de Restricciones [The SCP parsing algorithm based on Syntactic Constraints Propagation]*. PhD dissertation. Universidad de Sevilla, Junio 1997.

[Quesada & Amores Forthcoming] José F. Quesada & J. Gabriel Amores. (Forthcoming). *C for Natural Language Processing*. London: UCL Press.

[Rozenberg & Salomaa 1997] G. Rozenberg & A. Salomaa. eds. 1997. *The Handbook of Formal Languages. Vol. II.* Berlin: Springer Verlag.

[Sikkel & Nijholt 1997] Klaas Sikkel & Anton Nijholt. 1997. Parsing of Context–Free Languages. In [Rozenberg & Salomaa 1997].

[Tomita 1987] Masaru Tomita. 1987. An Efficient Augmented Context–Free Parsing Algorithm. *Computational Linguistics* **13 (1-2)**, 31-46.

[Tomita 1991] Masaru Tomita. 1991. *Generalized LR Parsing.* London: Kluwer Academic Publishers.