The SCP Parsing Algorithm: Computational Framework and Formal Properties

José F. Quesada
josefran@cica.es
CICA (Centro de Informatica Cientifica de Andalucia)
Avda Reina Mercedes, s/n. 41012 Seville. Spain
jfq@babel.us.es
Grupo de Investigación Julietta. Universidad de Sevilla

Abstract

This paper presents a new parsing algorithm for unrestricted context-free grammars. Basically, it may be described as a bidirectional bottom-up parser that is driven by an event generation strategy and is based on a sophisticated Syntactic Constraint Propagation (SCP) technique which uses strong top-down predictions. First, we motivate the new algorithm by discussing the notion of overparsing. Next, we present the formal kernel of the algorithm (based on the relations of partial derivability and adjacency) and a detailed description of the algorithm. Finally, the paper presents some important results from three perspectives: linguistic, computational and formal.

1 Introduction

During the last years, the research in the field of context–free parsing has mostly focused its attention on the problems of efficiency (Tomita, 1991; Rayner & Carter, 1996; Pereira & Wright, 1996; Amores & Quesada 1997; Quesada 1997b), general scope (Nederhof & Sarbo, 1993; Rekers, 1992; Bunt & Tomita, 1996; Visser, 1997), formal models (Kay, 1980; Sikkel & Nijholt, 1997; Quesada 1997a) and integration with unification (Carroll, 1993; Maxwell & Kaplan, 1993; Carpenter & Penn, 1996).

Language engineering and new applications of parsing, such as real–time systems, multimodality or telecommunications, have brought efficiency to a central position (Tomita, 1991; Bunt & Tomita, 1996). This has motivated new proposals as well as improvements of classical techniques: stochastic methods (Magerman, 1995; Briscoe & Carroll, 1995; Collins, 1996), paralelism (Alblas et al., 1994), "reductionistic" techniques (Pereira & Wright, 1996; Rayner & Carter, 1996), improvements of tabular algorithms (Nederhof & Sarbo, 1993; Bunt & Tomita, 1996), etc.

The work described in this paper follows this line of research. We propose the Syntactic Constraint Propagation (SCP) parsing algorithm: a general, sound and very efficient algorithm, whose main characteristics are the following:

Scope: Any CFG, without any modification.

Environment and integration: SCP defines two protocols for its communication with the lexico–morphological and unification levels. The first one permits the manipula-

tion of multi–expression words, multi–word expressions and lexical ambiguity. The second one allows both on–line and off–line interfaces.

Formal kernel: Computational complexity and real efficiency are conditioned by overparsing (section 2). SCP avoids overparsing by means of a bidirectional bottom–up event–driven strategy, along with a strong top–down filtering mechanism based on the relations of partial derivability and adjacency (section 3).

Computational model: Formal robustness and efficiency are the two main goals of SCP, and, therefore, the implementation plays a crucial role. The computational layer of SCP includes several techniques: the *CaD* framework (section 4), which allows remote constraint propagation using uniquely local controls; the *Qmem* model for the compilation and representation of CFG according to the formal kernel; and the *RBmem* system, a low–level memory management layer.

Performance: Experimental results [1] show a performace of 10,000 to 20,000 words per second, with a complexity $O(n \log n)$, for common grammatical phenomena in natural languages, such as recursive constructions, local and non–local dependencies.

Robustness: Since the output of SCP is a multi–virtual tree, in case of non–grammatical inputs, SCP generates a map of the last state of the parser as well as a full list of all the partial analyses.

The following section describes the notion of overparsing, which motivates SCP. Section 3, then, presents its formal kernel: partial derivability, adjacency and coverage. Next, section 4 describes the algorithm, its main functions and data structures. Finally, section 5 deals with some of the main properties of SCP.

[1] With the implementation (in C) of SCP, using a medium–size workstation.

2 Overparsing

In Formal Language Theory a language is a set, and in (classical) Set Theory an element does or does not belong to a set. That is to say, a set (and hence a language) is an unambiguous structure. Thus, the notion of grammaticality corresponds to the relation of membership over a language (set).

A grammar may be considered as an intensive definition of a language. But a grammar incorporates more information than a simple report of the elements of the language: a grammar defines a structure. The distance between grammaticality and grammatical structure defines grammatical ambiguity.

A parser must be able to determine the relation of grammaticality and to obtain the grammatical structure, by means of a set of operations: the parsing structure. The distance between the grammatical structure and the parsing structure defines *overparsing* (temporal ambiguity) (Quesada 1998).

Overparsing drastically decreases computational and real efficiency. To demonstrate this, let us consider the following grammar (G_{nm}):

```
(1:S ->N b) (2:S ->M c)
(3:N->N a)  (4:N->a)
(5:M->M M)  (6:M->a)
```

and the strings of words with the format a^+b. It is obvious that all input strings of this kind are not ambiguous with respect to G_{nm}. Nevertheless, parsing algorithms like Earley (1970),

chart (Kay, 1980) or GLR (Tomita, 1991) overgenerate the M structures. In contrast, SCP avoids overparsing. Table 1 compares the performance of SCP (n) with the performance (n^3) of Earley, chart and GLR, supposing all the algorithms have the same efficiency in the base case (length 1 : 0.0001 milliseconds), which is true for the implementation of the SCP algorithm.

Table 1: Parsing complexity: G_{nm}

Length n	SCP n	Earley chart GLR n^3
1	0.0001	0.0001
4	0.0004	0.0064
16	0.0016	0.4096
64	0.0064	26.2144
256	0.0256	1677.7200
		(27 minutes)
1024	0.1024	107374.0000
		(29 hours)

That is, for grammar G_{nm}, with strings of words of the kind a^+b, parsing algorithms like Earley, chart and GLR generate an overparsing on the order $n^3 - n$.

3 The Formal Kernel of SCP

3.1 Bottom-Up Derivation

Given $G =< G_T, G_N, G_P, G_R >$ where we have distinguished their terminal symbols G_T, non-terminal symbols G_N, vocabulary $G_V = G_T \cup G_N$, set of productions $G_P \subseteq G_N \times G_V^*$, and roots $G_R \subseteq G_N$, 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:

$$\Gamma\Delta\Omega \longrightarrow_G \Gamma\delta\Omega \text{ iff } (\delta \longrightarrow \Delta) \in G_P.$$

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

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

3.2 Partial Derivability and Adjacency

Let take $\alpha, \beta \in G_V$ and $\Delta \in G_V^*$:

Root Symbols. $R(\alpha)$ iff $\alpha \in G_R$

Epsilon Symbols. $E(\alpha)$ iff $\varepsilon \Longrightarrow_G \alpha$

String of Epsilon Symbols. $E(\Delta)$ iff $\forall \delta \in \Delta(E(\delta))$

Left Partial Derivability. β is a left partial derivation of α: $\alpha \longmapsto_l^* \beta$ iff $\exists \Gamma, \Delta, \Omega \in G_V^*$ such that $(\Gamma\alpha\Delta \Longrightarrow_G \Gamma\beta\Omega)$. $LPD(\alpha) = \{\beta \in G_V : \alpha \longmapsto_l^* \beta\} \cup \{\alpha\}$

Right Partial Derivability. β is a right partial derivation of α: $\alpha \longmapsto_r^* \beta$ iff $\exists \Gamma, \Delta, \Omega \in G_V^*$ such that $(\Gamma\alpha\Delta \Longrightarrow_G \Omega\beta\Delta)$. $RPD(\alpha) = \{\beta \in G_V : \alpha \longmapsto_r^* \beta\} \cup \{\alpha\}$

Primary Adjacency. β is a primary adjacent of α: $\alpha \Uparrow \beta$ iff $\exists \delta \in G_V$ and $\exists \Gamma, \Omega, \Delta \in G_V^*$ such that $(\delta \longrightarrow \Gamma\alpha\Delta\beta\Omega) \in G_P \wedge E(\Delta)$.

Left Adjacency. β is a left adjacent of α: $\alpha \Uparrow_l^* \beta$ iff $\exists \gamma \in LPD(\alpha)$ and $\exists \delta \in RPD(\beta)$ such that $(\delta \Uparrow \gamma)$. $LA(\alpha) = \{\beta \in G_V : \alpha \Uparrow_l^* \beta\}$.

Right Adjacency. β is a right adjacent of α: $\alpha \Uparrow_r^* \beta$ iff $\exists \gamma \in RPD(\alpha)$ and $\exists \delta \in LPD(\beta)$ such that $(\gamma \Uparrow \delta)$. $RA(\alpha) = \{\beta \in G_V : \alpha \Uparrow_r^* \beta\}$.

Left–Most Symbol. α is a left–most symbol: $LM(\alpha)$ iff $\exists \delta \in G_R$ such that $(\alpha \longmapsto_l^* \delta)$

Right–Most Symbol. α is a right–most symbol: $RM(\alpha)$ iff $\exists \delta \in G_R$ such that $(\alpha \longmapsto_r^* \delta)$

3.3 Coverage Tables

For each $\alpha \in G_V$:

$LC1(\alpha) = \{(\delta \longrightarrow \alpha) \in G_P\}$

$LC2(\alpha) = \{(\delta \rightarrow \alpha\Omega) \in G_P : \Omega \in G_V^+\}$

$MC(\alpha) = \{(\delta \longrightarrow \Delta\alpha\Omega) \in G_P : \Omega, \Delta \in G_V^+\}$

$RC(\alpha) = \{(\delta \longrightarrow \Delta\alpha) \in G_P : \Delta \in G_V^+\}$

4 The Algorithm SCP

4.1 The Lexical Interface: Breaking Points

SCP expects that the le/-xi/-co–mor/-pho/-lo/-gi/-cal module breaks the input string into a set of pieces marked by means of breaking points. The input to SCP will be then a set of 4–tuples, containing the lexical input, the syntactic category and the interval of breaking points (first and last breaking points):

$$< lex\_unit, syn\_cat, fbp, lbp >$$

For instance, given the input sentence: "*Peter cannot see that man that is the doctor*"[2], the lexical module may divide the input in the following breaking points:

0 *Peter* 1 *can* 2 *not* 3 *see* 4 *that* 5 *man* 6 *that* 7 *is* 8 *the* 9 *doctor* 10

Table 2 contains the result of the lexical analysis. This model permits the representation of multi–word expressions (*that is*), multi–expression words (*cannot*) and lexical ambiguity (*that*). We will call this matrix input.

[2]Supposing that the sentence is the output of a speech recognizer and we don't have punctuation marks.

Table 2: Input to SCP

lex_unit	syn_cat	fbp	lbp
Peter	n	0	1
can	aux	1	2
not	neg	2	3
see	v	3	4
that	pron-rel	4	5
that	pron-dem	4	5
that	det-rel	4	5
man	n	5	6
that	pron-rel	6	7
that	pron-dem	6	7
that	det-rel	6	7
that is	loc-adv	6	8
is	v	7	8
the	det	8	9
doctor	n	9	10

4.2 *CaD* structures

SCP is divided in two main blocks: the initialization phase and the parsing cycle.

The initialization phase begins with the creation of the *CaD* structures.

`GenerateCaD` creates one *CaD* structure for each breaking point. An *CaD* structure contains the fields: `Lclosed`, `Rclosed`, `Ropen` and `Lopen` (lists of events: *LEvent*); `Rnodes` and `Lnodes` (lists of nodes: *LNode*).

4.3 *Node* structures

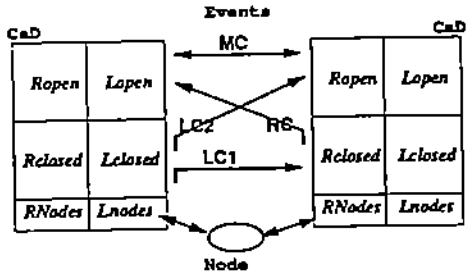
Once the *CaD* structures have been generated, each row of input will generate a node associated to the corresponding interval of *CaD*'s.

A *Node* structure stores the following information: `GrSymb` (grammatical symbol), `Lex_Unit` (the lexical unit associated to terminal nodes), `Mvt` (multi–virtual tree associated to non–terminal nodes), `Lcad` and `Rcad` (links to the *CaD*s that define the interval of the node).

4.4 *Event* structures

Each node will generate one event for each entry of the coverage tables of the grammatical symbol associated to the node. Figure 1 shows the criteria of connection between *Event*'s and *CaD*'s.

Figure 1: Linking events to CaDs

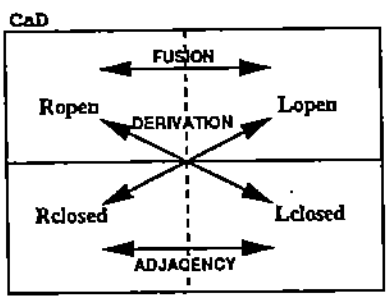


An *Event* contains the following fields: `GrRule` (rule), `Ldot` and `Rdot` (dots in the right–hand side of the rule), `Llinks` and `Rlinks` (links in its left and right extreme), `Ltype` and `Rtype` (types of connection to the *CaD*'s) and `Status` (logical status).

4.5 *Link* structures

For each event, and in its two extremes, it is necessary to analyze the possible links with other events in the same *CaD*. There are 3 kinds of links (figure 2).

Figure 2: Links inside a CaD



The `CheckLinks` function will analyze the links in the left and right extremes of the event (using the information obtained during the compilation phase: left and right partial derivability, and adjacency), and according to the result of this analisys, the function will evaluate the logical status of the event.

4.6 Event's Logical Status.

There are 4 logical states:

RUN (Running): Events closed in both extremes and with links in both extremes.

DER (Derivation): Events with at least one open extreme, with both extremes linked and with at least one Partial Derivation link.

EPS (Epsilon): Events with at least one open extreme, with both extremes linked and with all the links of one open extreme of the kind Epsilon.

DEL (Delete): The rest, that is, events with no links in at least one of the extremes.

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.

4.6.1 Step 6: Parsing Cycle.

```
function SCPcycle(cad)
  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.

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.

Delete Event. To delete an event implies to delete it and their links. For each link deleted, the logical status of the second event in the link will be analized. So this mechanism permits a remote constraint propagation using uniquely local controls (inside a *CaD*).

Run Event. To run a closed–closed event involves the application of a grammar rule, incorporating a new node. 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 *Mvt* 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. The control of cyclic phenomena and the on–line interface to unification are also controlled during this phase.

Fusion Events. There will be distinguished 4 cases depending on whether there are or not other links in the open extremes that will be fusioned.

5 Formal, Computational and Linguistic Properties

From the formal point of view, it is important to highlight two properties. First, SCP is complete and sound. Using a new framework for the formalization of bottom–up parsing techniques[3], based on the notions of π-equivalence, partition, cover, locally–connected cover, lc derivation and connected form, we have obtained a proof of the theorems of completeness and soundness for the SCP parsing algorithm: for a given CFG G, every grammatical form generated by G is a SCP–analyzed form (completeness) and viceversa (soundness). Second, we have tested SCP with 17 natural and artificial language phenomena described in the literature[4], and for all the phenomena SCP avoids overparsing, behaving equally or better than Earley, chart and GLR.

From the linguistic perspective, the previous results mean that SCP can parse any CFG without any modification.

But perhaps it is at a computational (real efficiency) level of comparison where SCP obtains its best rates. The first experiment (G_{rec}) uses recursive constructions and compares an implementation of chart (Quesada & Amores, Forthcoming) and SCP (table

[3] Algorithm schemata of Kay (1980) and parsing schemata of Sikkel & Nijholt (1997) are mainly oriented to top–down strategies.

[4] Including cyclic grammars, epsilon rules, hidden–left recursion, exponential ambiguity, recursive constructions, local and non–local dependencies, garden path sentences, etc.

3).

```
(1:S -> A BC D) (2:A -> a) (3:A -> A a)
(4:BC-> b c)    (5:BC-> b BC c)
(6:D -> d)      (7:D -> d D)
```

Table 3: Parsing efficiency: G_{rec}

L[5]	chart			SCP		
	T[6]	N[7]	E[8]	T	N	E
4	0.0	8	21	0.000	8	10
8	0.0	20	49	0.000	15	20
16	0.0	56	123	0.001	29	40
32	0.0	176	343	0.002	57	80
64	0.2	608	1071	0.004	113	160
128	2.8	2240	3679	0.008	225	320
256	68.2	8576	13K	0.018	449	640
512	2K	51K	33K	0.041	897	1280
1024				0.099	1793	2560

Table 4: Parsing efficiency: G_{xx}

L	chart			SCP		
	T	N	E	T	N	E
2	0.000	3	8	0.000	3	6
3	0.000	7	17	0.000	6	12
4	0.000	16	36	0.000	10	21
5	0.000	39	83	0.000	15	34
6	0.016	104	214	0.000	21	52
7	0.066	301	609	0.000	28	76
8	0.866	927	1862	0.000	36	107
9	9.666	2983	5975	0.000	45	146
10	160.679	19K	10K	0.000	55	194
20				0.233	210	1389
30				1.416	465	4584

The second experiment analyzes the computational improvements of SCP. Table 4 compares diferent strings of words (x^L) using the grammar G_{xx} (`X -> X X`).

6 Conclusion and Future Work

The syntactic analysis of Context–Free Grammars is an important field of research, so much as for its application to Computer Science as to Computational Linguistics. The aim of this work is the introduction of the SCP parsing algorithm, which is based on a multidisciplinary approach to this problem. This new algorithm entails original research on the logic, computational and linguistic levels involved in the problem.

From the point of view of its strategy, the SCP algorithm can be described as a bidirectional bottom–up parser, event driven and based on multi–virtual trees, which incorporates a very elaborated mechanism of constraint propagation.

The formal and computational models show a high level of efficiency, which multiplies by 10 to 1000 times the results described in the specialized literature. It is necessary to make emphasis on the fact that this level of efficiency (between 2,000 and 20,000 words per second) is achieved without diminishing the general applicability of the algorithm.

Among the current lines of research in the SCP framework we can mention incremental compilation, parallelization and the adaptation of the algorithm for syntax definition and prototyping of programming languages.

[5]Length of input string (number of words).
[6]Time, in seconds.
[7]Number of nodes generated.
[8]Number of events generated.

References

Alblas, H., R. den Akker, P. O Luttighuis & K. Sikkel. 1994. A bibliography on parallel parsing. *SIGPLAN Notices*, **29(1)**, 54–65.

Amores, J. G. & J. F. Quesada. 1997. Efficiency and Elegance in NLP: the EPISTEME Approach. In Mitkov, R., Nicolov, N. & Nikolov, N. eds. *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, Bulgaria. 82–91.

Briscoe, T. & J. Carroll. 1995. Developping and Evaluating a Probabilistic LR Parser of Part–of–Speech and Punctuation Labels. *Proceedings of the ACL/SIGPARSE 4th International Workshop on Parsing Technologies*, 48–58.

Bunt, H. & M. Tomita. eds. 1996. *Recent Advances in Parsing Technology*. London: Kluwer.

Carpenter, B. & G. Penn. 1996. Efficient Parsing of Compiled Typed Attribute–Value Logic Grammars. In (Bunt & Tomita, 1996), 145–168.

Carroll, J. A. 1993. *Practical Unification–based Parsing of Natural Language*. PhD dissertation. University of Cambridge.

Collins, M. J. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. Proceedings of ACL'96. Also in *cmp-lg/9605012*.

Earley, J. 1970. An Efficient Context–Free Parsing Algorithm. *Comm. of the ACM*, **13(2)**, 94-102.

Kay, M. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. *Report CSL-80-12*, Xerox Palo Alto Research Center, Palo Alto, CA.

Magerman, D. 1995. Statistical Decision–Tree Models for Parsing. *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283.

Maxwell III, J. T. & R. M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, **19(4)**, 571–590.

Nederhof, M.-J. & J. J. Sarbo. 1993. Increasing the applicability of LR parsing. *Third International Workshop on Parsing Technologies*, 187–201.

Pereira, F. C. N. & R. Wright. 1996. Finite–State Approximation of Phrase–Structure Grammars. *cmp-lg/9603002*.

Quesada, J. F. 1997. *El algoritmo SCP de análisis sintáctico mediante propagación de restricciones. [The SCP parsing algorithm based on syntactic constraints propagation]*. PhD Thesis. July 1997. University of Seville.

Quesada, J. F. 1997. A General, Sound and Efficient Natural Language Parsing Algorithm based on Syntactic Constraints Propagation. *Proceedings of the VII Conference AEPIA'97*, 775–786.

Quesada, J. F. 1998. Overparsing. *Workshop on Mathematical Linguistics*, Pennsylvania State University, State College, 17 April 1998.

Quesada, J. F. & J. G. Amores. (Forthcoming). *C for Natural Language Processing*. London: UCL Press.

Rayner, M. & D. Carter. 1996. Fast Parsing using Pruning and Grammar Specialization. Proceedings of ACL'96. Also in *cmp-lg/9604017*.

Rekers, J. 1992. *Parser Generation for Interactive Environments*. PhD dissertation. University of Amsterdam.

Sikkel, K. & A. Nijholt. 1997. Parsing of Context–Free Languages. In Rozenberg, R. & A. Salomaa. eds. 1997. *The Handbook of Formal Languages. Vol. II*. Berlin: Springer Verlag.

Tomita, M. ed. 1991. *Current Issues in Parsing Technology*. London: Kluwer.

Visser, E. 1997. *Syntax Definition for Language Prototyping*. PhD dissertation. University of Amsterdam.