Partitioning Grammars and Composing Parsers

Fuliang Weng and Andreas Stolcke[1]

Speech Technology and Research Laboratory
SRI international, 333 Ravenswood Ave., Menlo Park, CA 94025
{fuliang,stolcke}@speech.sri.com

Formal and natural languages may not be homogeneous in the sense that no single style parser can do well for any languages or sublanguages. Natural languages as a whole do not belong to any of the subclasses of CFG with efficient and compact parsers. On the other hand, natural language exhibits distinct sublanguages for various phrase types, idioms, etc., which can conveniently be described by specialized sub-grammars in various formalisms.

From a theoretical point of view, there are grammars G that lead to LR(k) parsers with size exponential in the size of G, given by Earley, 1968 and Ukkonen, 1982.

$S \rightarrow A_i \quad (1 \le i \le n)$
$A_i \rightarrow a_j A_i \, (1 \le i \ne j \le n)$
$A_i \rightarrow a_i B_i | b_i \quad (1 \le i \le n)$
$B_i \rightarrow a_j B_i | b_i \, (1 \le i, j \le n)$

If we divide this grammar into n sub-grammars based on i and compile them separately, however, we can get LR(k) parsers with a polynomial size, though its recognition time complexity increases to $\sqrt{|G|}$ times of the original one.

From a practical point of view, in a large parser, sublanguages may be described by pre-existing sub-grammars, possibly using specialized parsing algorithms. Combining sub-grammars may also be needed to extend existing grammars in the future. Therefore, a framework for partitioning grammars and combining parsers would prove useful.

The main points in this paper are: a general schema for partitioning a grammar into sub-grammars, and the combination of parsers for sub-grammars into an overall parser that yields the same parses as one for the original full grammar. Formal definitions for grammar partitioning will be presented, along with a correctness theorem, and a parser composition in the context of GLR parsing framework is presented.

Given $G = (\Sigma, NT, P, S)$ is a CFG, $P = \cup P_i$ where $P_i \cap P_j = \phi, i \ne j$ and $p_S \in P_0$. Without loss of generality, we assume in the rest of the paper that there is only one production rule with S as its LHS symbol.

Definition 1: $\{G_i = (\Sigma_i, NT_i, P_i, \nabla)\}_{i=0}^{n}$ is called a partition of G, where $NT_i = \{LHS(p) | p \in P_i\}$.

Definition 2: Let $P_A = \{p | p \in P \wedge LHS(p) = A\}$, $\{A | A \in NT, A \in RHS(p), p \in P_i$ and $P_A - P_i \ne \phi\}$ is called the *input* of G_i, $INPUT_{G_i}$, and $\{A | A \in LHS(p), p \in P_i, (\exists j) A \in INPUT_{G_j}\}$ is called the *output* of G_i, $OUTPUT_{G_i}$; for G_0, $OUTPUT_{G_0}$ has an additional element S. The *INPUT* is those NTs used by a sub-grammar that were previously parsed by other sub-grammars. The *OUTPUT* is those NTs that are the result of parsing by a sub-grammar and are handed on to other sub-grammars for further parsing.

Definition 3: Let $\{G_i = (\Sigma_i, NT_i, P_i, \nabla)\}_{i=0}^{n}$ be a partition of G, $\{G_i^X = (\Sigma_i^o, NT_i^o, P_i^o, X)\}_{i=0}^{n}$ is called the sub-grammar set of G with respect to partition $\{G_i\}_{i=0}^{n}$, where $\Sigma_i^o = \Sigma_i \cup \{vtm_A | A \in INPUT_{G_i}\}$, $NT_i^o = NT_i \cup \{A | A \in INPUT_{G_i}\}$, $P_i^o = P_i \cup \{A \rightarrow vtm_A | A \in INPUT_{G_i}\}$[2], $X \in OUTPUT_{G_i}$. In particular, $G_0^S = (\Sigma_0^o, NT_0^o, P_0^o, S)$ is called the master sub-grammar. A directed calling graph for the sub-grammar set of G is defined as (V, E), where V is the sub-grammar set and $E = (A, B)$, where the start symbol of sub-grammar B is in the *INPUT* of sub-grammar A.

---

1. The authors would like to thank P. Price and M. Cohen of STAR lab/SRI for their support.

2. It is a modified version of the virtual terminal technique used by Korenjak, 1969 and Weng, 1993.

Definition 4: A derivation of the sub-grammar set $\{G_i^S\}_{i=0}^n$ of G is one of the following:

1. $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if $\exists i$ such that $A \rightarrow \gamma \in P_i^o$.
2. $\alpha vtm_A \beta \Rightarrow \alpha A \beta$ if $\exists i$ such that $A \in OUTPUT_{G_i}$.

Theorem 1: A string of terminals α is derived from G iff it can be derived from the sub-grammar set $\{G_i^S\}_{i=0}^n$, starting from the master sub-grammar G_0^S.

For simplicity, we assume that G is partitioned into two subsets, a master sub-grammar G_S with the start symbol S and a slave sub-grammar G_A with the start symbol A. Any number of slave grammars can be accommodated with trivial modifications as long as the calling graph between sub-grammars is a DAG. Let $\{A\}$ be the output of G_A and the input of G_S, and S the output of G_S. Compiling G_A and G_S, by using a GLR(0) parser generator, we obtain four tables: a pair of Action and Goto tables for each of G_A and G_S. Based on (Tomita, 1986), we modify the GLR parsing process as follows:

Starting $parser_{G_S}$ from its initial state and initial position of 1 in an input string of length n:

1. In $parser_{G_S}$ create register(i) for the i-th word in the current input; initialize register(*) to ϕ at the beginning of parsing, that is, in $PARSE(G_S, a_1, \ldots, a_n)$ of Tomita's algorithm.

2. In the $PARSEWORD(i)$ process of Tomita's algorithm, instead of initializing Q (a place to hold shift action) to ϕ, assign to Q the value of register(i), that possibly stores node-state pairs $\langle v, s \rangle$ given by sub-parser(s) $parser_{G_A}$, where v is the root node of a forest returned by $parser_{G_A}$ and s is a state that $parser_{G_S}$ will go to after shifting in node v. In other words, $parser_{G_S}$ delayed this shift action until position i when its sub-parser gets there.

3. When $parser_{G_S}$ reaches state s and looks at terminal $t \in \Sigma$ at position j of the current input, do normal GLR parsing using G_S.

4. When $parser_{G_S}$ reaches its state s and scans position j of the input, $Action_{G_S}(s, vtm_A) \neq \phi$:

(a) If $\langle reduce, i \rangle \in Action_{G_S}(s, vtm_A)$, do it as in the normal GLR parsing by using G_S.

(b) If $\langle shift, i \rangle \in Action_{G_S}(s, vtm_A)$, switch the control to $parser_{G_A}$ and start the parsing process of G_A from the initial state of G_A and position j until *accept* is reached at position $k \leq n$ in the input. Hang the whole forest under A, which has vtm_A as its son in the partially parsed forest created by G_S, and eliminate this vtm_A. Put $\langle shift, i \rangle$ in register(k) and continue this process until $parser_{G_A}$ reaches the end of the input, then return the control to $parser_{G_S}$.

With a synchronization mechanism, the DAG restriction on the calling graph for the algorithm can be removed.

Compared with other work, the grammar partitioning presented here gives more freedom as to how the grammar is divided, and can be generalized to combine parsers of different styles. This is different from Korenjak's partitioning grammars and constructing LR(k) processors, and also from Abney's chunking parser.

References

S. Abney, *Parsing by Chunks*, In *Principle-Based Parsing: Computation and Psycholinguistics*, R. C. Berwick et al. (eds.), Kluwer Academic Publishers, 1991

J. Earley, *An Efficient Context-free Parsing Algorithm*, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1968.

A. Korenjak, *A Practical Method for Constructing LR(k) Processors*, *CACM* 12 (11), 1969.

M. Tomita, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, 1986.

E. Ukkonen, *Lower Bounds on the Size of Deterministic Parsers*, *J. Computer and System Sciences* 26, 153-170, 1983.

F. Weng, *Handling Syntactic Extra-grammaticality*, In *Proceedings of the 3rd International Workshop on Parsing Technologies*, Tilburg, the Netherlands, 1993.