TR-IT-0071

On the automatic transformation of a set of EBMT Constituent Boundary Patterns into a Context-Free Grammar, and associated bottom-up algorithms

Christian Boitet

1994.9.7

EBMT as pursued at ATR uses a set of "Constituent Boundary Patterns" to describe the input language. We show how to automatically convert such a set P into an equivalent CFG G=*T*(P), which is not only weakly (generatively) equivalent, but quasi-structurally equivalent: there is a very simple homomorphism transforming a G-tree into the corresponding P-tree.

As a result, any classical CFG-based analysis algorithm may be used with G=*T*(P), and the parse trees converted to P-trees. In particular, all bottom-up algorithms are applicable. However, due to the layered character of the set of non-terminals of any such G, it is possible to propose a simple and efficient (non-deterministic) bottom-up analysis algorithm.

Because special "constituent boundary" tags are introduced by a pre-processing phase into the strings described by P (and hence by G), it is also possible to adapt the idea of "operator precedence parsing" to G, and perhaps even directly to P.

Interpreting Telecommunications Research Laboratories
2–2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

On the automatic transformation of a set of EBMT Constituent Boundary Patterns into a Context-Free Grammar, and associated bottom-up algorithms

Christian BOITET*
GETA, IMAG (UJF & CNRS)
BP 53, 38041 Grenoble Cedex 9, France
Christian.Boitet@imag.fr

Abstract

EBMT as pursued at ATR uses a set of "Constituent Boundary Patterns" to describe the input language. We show how to automatically convert such a set P into an equivalent CFG G=*T*(P), which is not only weakly (generatively) equivalent, but quasi-structurally equivalent: there is a very simple homomorphism transforming a G-tree into the corresponding P-tree.

As a result, any classical CFG-based analysis algorithm may be used with G=*T*(P), and the parse trees converted to P-trees. In particular, all bottom-up algorithms are applicable. However, due to the layered character of the set of non-terminals of any such G, it is possible to propose a simple and efficient (non-deterministic) bottom-up analysis algorithm.

Because special "constituent boundary" tags are introduced by a pre-processing phase into the strings described by P (and hence by G), it is also possible to adapt the idea of "operator precedence parsing" to G, and perhaps even directly to P.

1. The ATR-EBMT formalism of "constituent boundary patterns"

In their COLING-94 paper [4], Furuse and Iida present a grammatical formalism using patterns and levels, which they use for top-down parsing in their EBMT system. Here are typical pattern examples. The formalism is explained in more detail after the table below.

Level	Name	Pattern (proper)	"Next-Priority" (rewriting levels)
1	CONF-I_SUPPOSE	(?X I <PRO-V> SUPPOSE)	
1	CONF-RIGHT	(?X RIGHT)	(:X 2)
2	SCONJ2-AND	(?X AND ?Y)	(:Y 4)
...			
4	VP-INV-FOR	(FOR ?X <N-DET> ?Y)	(:X 6 :Y 5)
4	REQ-LET_US	(LET US ?X)	(:X 5.3)
5	...		
5.1	...		
...			
5.4	BE_ING	(?X ING+ ?Y)	(:Y 5.3)
...			
5.7	...		
...			
6	...		
...			
7	...		
...			
8	...		
...			
9	X_NUMBER	(?X <N-N> ?Y)	(:X 10)
9	<NUM-NUM>	(?X <NUM-NUM> ?Y)	(:Y 10)

* Visiting researcher at ATR Interpreting Telecommunication Laboratories

In this example, the level goes from 1 to 9, with some fine-grained levels between 5 and 6 (5.1–5.7). There is also an implied level of 10, as shown by the "Next-Priority" (rewriting levels) column.

The patterns contain 4 types of elements:

- Variables such as `?X, ?Y, ?Z;`
- Class symbols such as `NUMBER`, which stand for classes of words;
- Constituent Boundary symbols, such as `<N-DET>`, which are inserted by a pre-processing (tagging) phase prior to analysis;
- Words, such as "I", "suppose", etc.

Class symbols, Constituent Boundary symbols, and words form the terminal vocabulary, which we will denote by `V`.

The variables `?X, ?Y,...` are analogous, but not identical, to auxiliary or non-terminal symbols in classical context-free grammars.

The language generated by a set of such patterns is obtained by rewriting from the pattern `(?X)` of level 1 until a string without variables is obtained. A variable `?X` in a pattern of level `j` may be rewritten [4]:

- using any pattern of level `k≥j` if `?X` does not appear in the Next-Priority field of the pattern;
- or using any pattern of level `k≥h` if `:X h` appears in the Next-Priority field of the pattern.

For short, we will call such an arrangement a "set of constituent-boundary patterns", or CBP-set.

2. Transformation of a CBP-set P into an equivalent CFG G=*T*(P)

According to the remark above, we first normalize the notation by representing all strings of terminals in the same manner. More specifically, let V be the set of terminals. Each pattern may be represented as:

$$[l,\ n,\ u_0\ X_1\ u_1 \ldots\ u_{p-1}\ X_p\ u_p,\ (X_1\ l1) \ldots\ (Xp\ lp)]$$

where
- l is the level of the pattern.
- n is its name.
- $X_1, \ldots X_p$ are the non-terminals ?X, ?Y, ?Z,... ($p \leq 3$ in the examples we have seen),
- $u_0, u_p \in V^*$ and $u_1 \ldots u_{p-1} \in V^+$. This simply means that two variables are always separated by at least one terminal, while the leftmost and rightmost elements may or may not be terminal strings.
- A pair $(X_i\ li)$ corresponds to a pair $:X_i\ li$ in the Next-Priority list.

Algorithm *T*

Input: a set P of patterns, in the notation above.
Output: an equivalent CFG G.
Method:

1) Create the set of non-terminals $E_0, E_1, \ldots E_L$.
   -- recall that L is the maximal level.
   -- E_0 is the axiom of G.
2) Create the rules:
   $E_0 \rightarrow E_1 \mid E_2 \ldots \mid E_L$
   $E_1 \rightarrow E_2 \ldots \mid E_L$
   ...
   $E_{L-1} \rightarrow E_L$
3) For each pattern $[l, n, u_0\ X_1\ u_1 \ldots\ u_{p-1}\ X_p\ u_p,\ (X_1\ l1) \ldots\ (X_p\ lp)]$ add the rule:

n: E_1 -> u_0 E_{l1} u_1… u_{p-1} E_{lp} u_p

3. Equivalence

It should be clear from the explanation of the derivation process in a CBP-set that the grammar G produced generates the same strings, that is, is weakly equivalent to P.

The parse trees produced by G are not identical with those associated with P. Hence, the two formalisms are not strongly equivalent. But there is a simple homomorphism *H* transforming a G-tree into a P-tree.

Algorithm *H*

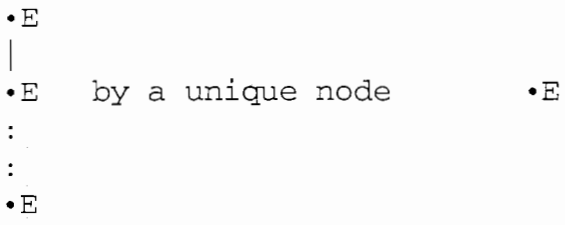
Input: a parse tree for a terminal string w associated with G=*T*(P).

Output: the associated P-tree.

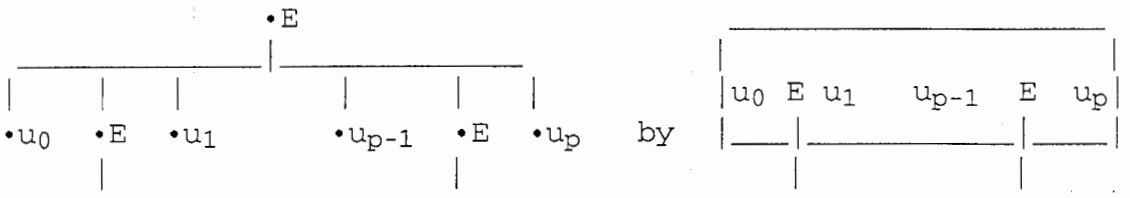
Method:

1) In the G-tree, rename all non-terminals E_0, E_1,… E_L as E.

2) Replace all unary chains of the form



3) Replace each subtree of the form



4) Rename the symbols E as ?X, ?Y,… as in the original pattern (which name may be kept in the G-tree as the name of the rule derived from it).

4. Parsing methods

Any CFG-based parsing method can of course be used with G=*T*(P). In particular, any bottom-up method can be used.

However, due to the layered character of the set of non-terminals of any such G, the levels i of non-terminals E_i never decrease while traversing a parse tree from the root to a leaf[1]. It is thus possible to propose a simple non-deterministic analysis algorithm, *A*.

Algorithm *A:* non-deterministic bottom-up "layered parsing"

Input: a terminal string w associated with G=*T*(P).

Output: a set of parse trees for w associated with G=*T*(P).

Method:

1) For each level j from L down to 0 do
reduce by using only the rules rewriting E_j

[1] Note that, in the example above, one can go back from level 5.4 to level 5.3, which contradicts what is said in [4]. If this is the case, we can still define adequate subgrammars, by using the rules rewriting E_j and the rules rewriting all the E_i such that $k \le i \le j$ and E_k appears in a right hand side of a rule rewriting E_j.

```
    2)  Output all (or the n "best") trees with root E0 covering w.
            -- "best" according to a distance, or a score, etc.
```

Still another possibility may be suggested by the fact that special "constituent boundary" tags are introduced into the strings to be analyzed, so that two non-terminals are always separated by at least one terminal in any right hand side of a rule of G.

The idea is to adapt the idea of "operator precedence parsing" to G, and perhaps even directly to P. For details about operator precedence parsing, see [1]. The good thing about this sort of parsing method is that it does not distinguish among different non-terminals. This lack of distinction is actually what we want, at least for each level.

Let us call G_i the subgrammar of G rewriting E_i, that is, containing the rules with left hand side E_i.

Algorithm *B*: less non-deterministic, bottom-up, "layered" parsing

```
Input:  a terminal string w associated with G=T(P).
Output: a set of parse trees for w associated with G=T(P).
Method:
    1)  For each level i from L down to 0 do
            compute the precedence table for Gi
            -- if it has no conflict, this step is deterministic
            use it to reduce the current string
                -- (made of terminals and of dummy non-terminals,
                --  normally pointing to partial parse trees).
    2)  Output all (or the n "best") trees with root E0 covering w.
            -- "best" according to a distance, or a score, etc.
```

In normal precedence parsing, a successful pass empties the input and leaves exactly one (dummy) non-terminal on the stack, pointing to the final parse tree. Here, each pass should also empty the input string, but the resulting stack may contain terminals as well as (dummy) non-terminals, and is in fact the input to the next pass.

To adapt this idea to handle P directly simply means that, instead of producing a G-subtree when performing a reduction, we would produce a P-subtree (see step 3 of algorithm *H* above). Note that operator precedence parsing never produces "unary chains", so that there is no need for something like step 2 of algorithm *H* above.

Discussion and conclusion

What we have shown is that any CBP-set P can be transformed into an equivalent CFG G=*T*(P), and that the P-trees can be easily recovered from the G-trees. While any CFG-based parsing algorithm may be applied to G, the layered character of P suggests a layered bottom-up parsing algorithm, which should be more efficient, essentially because it partitions G into as many subgrammars as there are layers, and uses a different subgrammar at each pass.

As a matter of fact, CF-recognition time complexity is at least quadratic in the size of the grammar (for the Earley algorithm, it is $O(|G|^2.|w|^3)$ [3]). Hence, dividing a grammar of size 10.S into 10 subgrammars of size S should decrease the recognition time from $100.S^2\ |w|^k$ to $10.S^2\ |w|^k$, a factor of 10, whatever the exact value of k. In the example CBP-set considered above, there are about 16 levels, so that the gain could be a factor of 16, and so forth.

Still a more efficient class of bottom-up algorithms may be derived from the technique of "operator precedence parsing". Due to the use of statistical techniques to assign exactly one morphosyntactic class to each input word, and to insert non-ambiguous context boundary markers, it can be hoped that the subgrammars of G will in practice be "operator precedence grammars", that is, that their precedence tables will show no conflict.

Even if some (or all) of the subgrammars are not "operator precedence grammars", this technique should be more efficient, because it does not use the rules directly, but only consults the precedence table. The drawback – there always is one! – is that this table is quadratic in the size of the terminal vocabulary |V|. But |V| is necessarily a fraction of the size of the initial CBP-set P itself, because of the definition of V.

Acknowledgments

Thanks should go to M. Seligman, who prompted me to study this question and revised the successive drafts of this note, to H. Okuma, who communicated an example CBP-set, and to Y. Sobashima, who kindly explained the bottom-up method he uses to parse according to a different kind of pattern sets [5].

-o-o-o-o-o-o-o-o-o-o-

References

[1] **Aho A. & Ullman J., ed. (1972)** *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, Englewood-Cliffs, New-Jersey, 700 p.

[2] **Aho A. & Ullman J., ed. (1977)** *Principles of Compiler Design.* Addison-Wesley, Reading, Massachussetts, 250 p.

[3] **Barton G. E. J., Berwick R. C. & Ristad E. S., ed. (1987)** *Computational Complexity and Natural Language.* MIT Press, Cambridge, Massachussetts, 335 p.

[4] **Furuse O. & Iida H. (1994)** *Constituent Boundary Parsing for Example-Based Machine Translation.* Proc. COLING-94, Kyoto, 5-9 Aug. 1994, Kyoto Univ., vol. 1/2, 105–111.

[5] **Sobashima Y. & Seligman M. (1994)** *A Context-Dependent Parsing Method Using Bracketed Corpora for Spoken Dialogues.* Internal report, ATR-ITL, July 1994, 6 p. (submitted to NewLap-94, UMIST, Manchester, UK).

-o-o-o-o-o-o-o-o-o-o-