Parser Combinators for Extraction

Jan van Eijck[*]

Abstract
Dislocation phenomena in natural language can be, and often are, thought of as the effects of movement transformations. We propose to handle these phenomena in terms of parser combinators [3, 8] that transform recursive descent parsers for a 'deep structure language' into parsers for a 'surface structure language'. This combinator approach to extraction keeps close to the 'movement' intuition and gives a computational account of the well known island constraints on extraction first proposed in [7].

1 Introduction

Left extraction in natural language occurs when a subconstituent of some constituent is missing, and some other constituent to the left of the incomplete constituent represents that missing constituent in some way. In the generative tradition, such dislocations used to be accounted for by means of transformations that *move* a constituent while leaving a trace. Computational and logic-oriented approaches to NL processing and understanding replace the transformational account with an *in situ* analysis, through gap threading (lexical functional grammar, categorial grammar, GPSG, HPSG), through extension of the context free rule format with wrapping operations (extraposition grammars, tuple-based and tree-based extensions of context free grammars), or through extension of context free rules with stacks of indices (indexed grammars). We propose an account in terms of pushdown parser combinators for recursive descent parsing. Our account allows us to remain close to the spirit of the original movement analysis.

2 Parser Combinators

Parser combinators are functions that transform parsers for a language into parsers for a different language [3]. We can think of a recursive descent parser for a fragment of natural language as a function of type

$$[Cat] \to [(Cat, [Cat])].$$

[*] CWI and ILLC, Amsterdam and Uil-OTS, Utrecht

The parser transforms a list of categories — type $[Cat]$, with the square brackets indicating list formation — into a list of pairs $(Cat, [Cat])$, each consisting of a category and a list of remaining categories. Parsing with the rule $A \longrightarrow X_1 \cdots X_n$ gives the result:

$$[X_1, \dots, X_n, X_{n+1}, \dots, X_m] \Longrightarrow \underset{X_1 \quad \dots \quad X_n}{\mathrm{A}} \quad [X_{n+1}, \dots, X_m].$$

Parsing the input list $[X_1, \dots, X_m]$ with a set of rules may give a number of different results. Each successful parse of the input category-list will yield a pair consisting of (i) a recognized category for a prefix of the category list and (ii) the remainder of the category list. Parsing failures are indicated by return of the empty list, ambiguous parses by return of non-unit lists.

3 A Parser Combinator for Extraction

Relative clause formation in English is a simple example of left extraction. The structure of the relative clause in (1) is represented by the annotation that links a relative pronoun *that* to its trace t_i.

$$\textit{I hated the man that}_i \textit{ the woman sold the house to } t_i. \tag{1}$$

Abbreviating the type of parsers as *Parser*, the parser combinator that instructs a parser to expect a DP gap — represented by a trace t — has type *Parser* $\rightarrow$ *Parser*, and is given by the following definition by means of list comprehension:

```
expectDPgap  ::  Parser → Parser
expectDPgap  =   λ parser λ xs.
                 [(cat, zs) | ys ← randomInsert ”t” xs,
                              (cat, zs) ← parser ys,
                              hasDPgap cat              ]
```

What this says is that the parser combinator *expectDPgap* takes a parser as a first argument and yields a function from input category lists to lists of output pairs consisting of categories and remainder category lists, i.e., a parser. The function call *randomInsert ”t” xs* yields the list of all category lists that result from inserting trace t somewhere in the category list *xs*, so *ys* ranges over all such category lists, and (cat, zs) ranges over all pairs of categories and category lists that result from running the input parser on such *ys*. The function call *hasDPgap cat* is a Boolean check as to whether *cat* has an DP gap somewhere in it. Thus, if *expectDPgap* combines with a parser this yields a new parser that operates on an input category list *xs* by calling the input parser on category lists that differ from the input category list in the fact that the trace t occurs in it somewhere, and that yields as output those pairs (cat, zs) in the yield of the input parser with *cat* having a DP gap.

Suppose that the trace introduced by *expectDPgap* is parsed as a DP gap, and assume that *parseSent* is a parser for sentences. Then

$$expectDPgap\ parseSent$$

is a parser for relative clauses. The combinator account of movement naturally accommodates the well known island constraint on extraction [7] that rules out configurations of the form

$$\ldots \text{that}_i \ldots [_{\text{DP}} \ldots [_{\text{REL}} \text{ that}_j\ [_{\text{S}} \ldots t_j \ldots t_i \ldots]]].$$

The island constraint is imposed to explain the ungrammaticality of examples like (2).

**I admired the woman that$_i$ you liked the man that$_j$ [t_j sold it to t_i].* (2)

This island constraint is captured in the *hasDPgap* check.

4 Pushdown Parsers

The parsing-as-deduction metaphor [6] assimilates parsing with CF rules to logical deduction. The goal is to prove the sentence symbol from a list of premisses corresponding to the categories of the input word list, with the CF rule $A \longrightarrow X_1 \cdots X_n$ read as $X_1 \cdots X_n \vdash A$.

In this perspective, parsing with a dislocated constituent can be seen as parsing with a hypothesis to be discharged at the point where the corresponding gap is encountered. Parsing with CF rules relates to parsing with CF rules allowing hypothetical reasoning in roughly the same way as basic categorial grammar relates to Lambek style categorial grammar, but for the fact that in the case of categorial grammar hypothetical reasoning does not increase (weak) expressive power [5], while in the case of CF grammar it does (see below).

The appropriate function for 'parsing with hypotheses' is a pushdown parser that collects the list of undischarged hypotheses on a stack. A *PdParser* is a function of the following type:

$$[Cat] \rightarrow [Cat] \rightarrow [(Cat, [Cat], [Cat])].$$

Such a function takes a list of undischarged hypotheses and a list of unparsed categories, and it produces a list of triples consisting of a category, a list of remaining hypotheses, and a list of remaining categories.

If a displaced constituent is encountered, a gap category is pushed onto the stack of hypotheses. If, during the parse, a corresponding category is expected but not found in the input category list, a hypothesis may be discharged.

Suppose the parser expects a DP. If a gap of type DP is on top of the stack, then it is possible to parse the DP as this gap, and it is also possible to parse the DP using a DP rule, and carry the gap along. Discharging the gap is done by using the *pop* parser combinator:

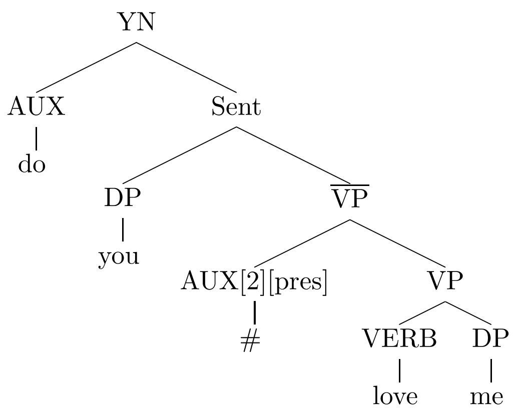
$$\begin{aligned} \mathit{pop} \quad &:: \quad \mathit{CatLabel} \to \mathit{PdParser} \\ \mathit{pop}\ l\ (h : \mathit{hs})\ \mathit{xs} \quad &= \quad [\ (h, \mathit{hs}, \mathit{xs}) \mid \mathit{catLabel}\ h = l\ ] \end{aligned}$$

The following function propagates a hypothesis through a pushdown parser:

$$\begin{aligned} \mathit{propagate} \quad &:: \quad \mathit{CatLabel} \to \mathit{Cat} \to \mathit{PdParser} \to \mathit{PdParser} \\ \mathit{propagate}\ l\ h\ p \quad &= \quad \lambda \mathit{hs}\ \lambda \mathit{xs}. \\ &\qquad [\ (\mathit{cat}, h : \mathit{is}, \mathit{ys}) \mid (\mathit{cat}, \mathit{is}, \mathit{ys}) \leftarrow p\ \mathit{hs}\ \mathit{xs}, \mathit{catLabel}\ h = l\ ] \end{aligned}$$

This choice between pop and propagation ensures that the island constraint is met: the pending hypothesis cannot be discharged inside a category with the same label. (In the other case, i.e., if the label of the pending hypothesis is different from the label of the expected category, the hypothesis can be used inside.)

Yes/no questions can be thought of as the result of extracting an auxiliary from a sentence, e.g.:



A Wh-question is the result of extracting a Wh-phrase (either a DP or a PP) from a YN-question, so parsing a Wh-question is just a matter of first finding a Wh-phrase and next letting a parser for YN-questions look for a matching Wh-phrase gap. Figure 1 gives a structure tree for *What did they break it with?* Similarly, we get a parse for *With what did they break it?* by pushing a PP gap onto the hypotheses stack.

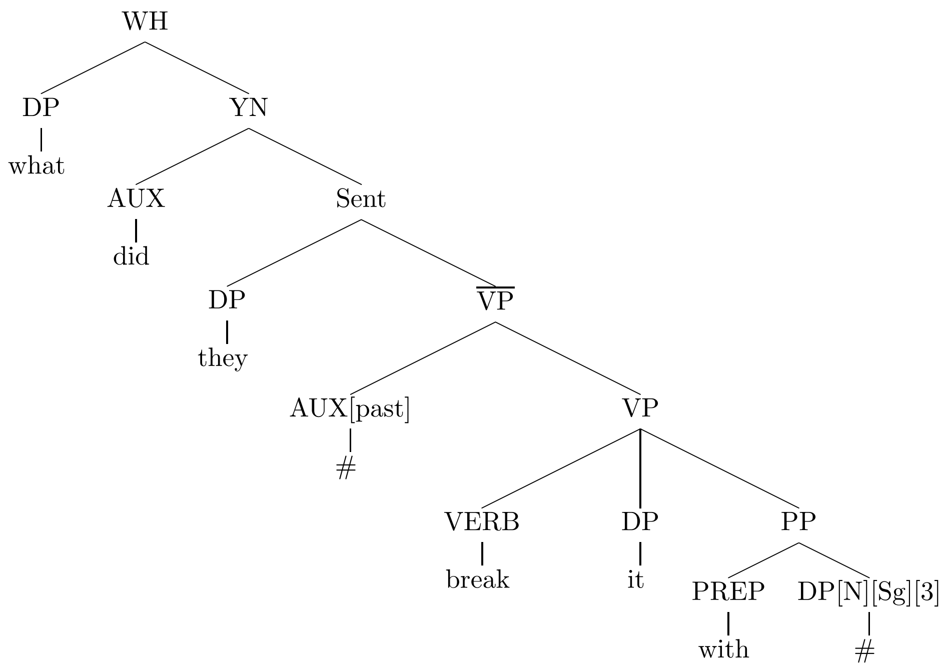


Figure 1: Parse tree for *What did they break it with?*

5 Semantics

For generation of logical forms, replace each category by a pair of the type (*Cat, LF*), and modify the push function as follows:

$$\begin{array}{rcl} \textit{push} & :: & \textit{(Cat,LF)} \rightarrow \textit{PdParser} \\ \textit{push (gap,v) f hs xs} & = & [\ ((c, \lambda v.\textit{lf}), \textit{is}, \textit{ys}) \mid ((c, \textit{lf}), \textit{is}, \textit{ys}) \leftarrow \textit{f (gap,v):hs xs}\ ] \end{array}$$

Suppose the gap variable has type α and the LF component of the output of parser f has type β. Then the LF component of the output of the parser

push (gap, v) f

has type $\alpha \rightarrow \beta$. This makes storage of a gap category (introduction of a hypothesis) correspond to lambda abstraction over a variable that interprets the gap, as it should.

6 Recognizing Power

Starting out from a set of combinators for CF parsing, the addition of the pushdown stack of hypotheses allows for the parsing of non-CF languages, as the following example of a parser for the language $a^n b^n c^n$ illustrates:

$$\begin{array}{rcl} \mathit{parseS}, \mathit{parseZ} & :: & \mathit{PdParser} \\ \mathit{parseS} & = & \mathit{parsesAs}\ \ 'S'\ \ [\mathit{symb}\ \ 'a', \mathit{push}\ 'X'\ \mathit{parseS}] \\ & \oplus & \mathit{parseZ} \\ \mathit{parseZ} & = & \mathit{parsesAs}\ \ 'Z'\ \ [\mathit{symb}\ \ 'b', \mathit{pop}\ 'X'\ \mathit{parseZ}, \mathit{symb}\ \ 'c'] \\ & \oplus & \mathit{eps} \end{array}$$

This uses only the CF combinators (*symb* for recognition of individual symbols, $\oplus$ for choice, *parseAs* for sequential composition of a list of parsers under a label, and *eps* for recognizing the empty string), plus the *push* and *pop* combinators.

A pushdown parser for recognizing a category A comes with a local stack of undischarged hypotheses that can be used either in recognizing A or in recognizing categories further on in the parse process. This is similar to the nested stack automata from [2], the machine model that matches the class of indexed languages [1]. We conjecture that parsing with recursive descent pushdown parsers, using only parser combinators for the context free combinators, plus the combinators for storage (push) and retrieval (pop) of hypotheses, allows for the recognition of all indexed languages.

Pushdown parser combinators have been implemented in the lazy functional programming language Haskell [4], yielding promising parsers for interesting NL fragments.

References

[1] Alfred V. Aho. Indexed grammars — an extension of context-free grammars. *Journal of the ACM*, 15(4):647– 671, 1968.

[2] Alfred V. Aho. Nested stack automata. *Journal of the ACM*, 16(3):383–406, 1969.

[3] G. Hutton. Higher-order functions for parsing. *Journal of Functional Programming*, 2(3):323–343, 1992.

[4] S. Peyton Jones, editor. *Haskell 98 Language and Libraries; The Revised Report*. Cambridge University Press, 2003.

[5] M. Pentus. Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Amalitos, California, 1993.

[6] F.C.N. Pereira and H.D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the ACL*, pages 137–111. MIT, Cambridge, Mass., 1983.

[7] J.R. Ross. *Constraints on Variables in Syntax*. PhD thesis, MIT, 1967.

[8] P. Wadler. How to replace failure by a list of successes. In Jean-Pierre Jouannaud, editor, *FPCA*, volume 201, pages 113–128, 1985.