A left-branching grammar design for incremental parsing

Petter Haugereid[a] and Mathieu Morey*[a,b]

[a] Division of Linguistics and Multilingal Studies, Nanyang Technological University,
14 Nanyang Drive, Singapore 637332, Singapore
petterha@gmail.com
[b] Laboratoire Parole et Langage, Aix-Marseille Université,
5 avenue Pasteur, 13100 Aix-en-Provence, France
mathieu.morey@gmail.com

Summary. This paper presents a left-branching constructionalist grammar design where the phrase structure tree does not correspond to the conventional constituent structure. The constituent structure is rather reflected by embeddings on a feature STACK. The design is compatible with incremental processing, as words are combined from left to right, one by one, and it gives a simple account of long distance dependencies, where the extracted element is assumed to be dominated by the extraction site. It is motivated by psycholinguistic findings.

Keywords: HPSG, grammar engineering, incremental parsing, constructionalist grammars.

1 Introduction

Phillips (2003) shows that given a Government and Binding analysis involving Larsonian shells (Larson, 1988; Culicover, 1997), it is possible to parse a tree incrementally, from left-to-right, with a right-corner parser. The aim of this paper is to show that the same can be achieved with a bottom-up HPSG parser, given an analysis of long-distance dependencies where it is assumed that the fronted element is realized at the bottom left corner of the tree, rather than as the first daughter of the top node.

A grammar fragment for English will be introduced, which on the one hand makes comparable generalisations about syntactic structures as the Principles and Parameters theory, but which on the other hand is radically different in that it employs left-branching trees, rather than right-branching trees. The account does not assume verb movement.

The grammar referred to in this paper is a modified version of a grammar for Norwegian, Norsyg, (see Haugereid (2009)). It is implemented with the LKB system (Copestake, 2002), which is a grammar development environment mainly used to implement HPSG grammars. It is a bottom up parser that employs phrase structure rules. All grammatical objects are expressed as typed feature structures (Carpenter, 1992). The implemented grammar has much of the feature geometry in common with HPSG grammars, but some central assumptions are different. Most importantly, the grammar is a constructionalist grammar, and not a lexicalist grammar. This implies that open lexical items in principle do not constrain their syntactic context, and do not carry information about their argument structure. Instead, it is assumed that the syntactic structure is determined by functional signs like inflections, function words and phrase structure rules. The argument structure is determined by sub-constructions, which are syntactic realisations of Davidsonian sub-events.

* This research was supported in part by the Erasmus Mundus Action 2 program MULTI of the European Union, grant agreement number 2009-5259-5.

HPSG 2012 Conference/Ellipsis Workshop, July 18-21, 2012, Chungnam National University, Daejeon, Korea

The approach is motivated by psycholinguistic findings showing that arguments are incorporated into the syntactic structure before the verb is encountered (Aoshimia et al., 2009) and that ambiguous structures can be parsed more efficiently by humans than unambiguous structures (Swets et al., 2008).

2 Long Distance Dependencies

The use of a slash to account for long-distance dependencies in a mono-stratal account was introduced by Gerald Gazdar (1981). A trace of the extracted item was assumed in the extraction site, and the SLASH feature would establish a link between the trace and the filler. The SLASH feature would "percolate up" the tree with the information about the trace. The mechanism behind the more recent HPSG SLASH account in Bouma et al. (2001) involves entering all arguments and modifiers of a verb onto a separate DEPENDENTS list and retrieving the slash from this list lexically. One problem with the lexical approach, from a psycholinguistic perspective, is that since the modifiers of a verb cannot be listed in the verb's lexical entry, a DEPENDENTS list cannot be fixed before the parsing of the sentence is finished. So the SLASH mechanism ends up as a post-parsing process.

The account of long distance dependencies in this paper is similar to the "trace" account, apart from the fact that the SLASH feature "percolates down" the tree, rather than "up". The tree in Figure 1 is an analysis of the Wh-question *Who does John admire?*[1] [2]

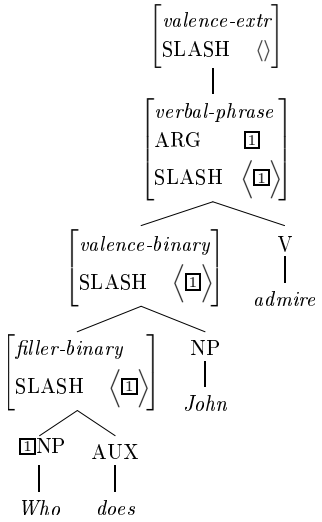

Figure 1: The SLASH feature: Fronted object.

At the bottom of the tree, the head filler rule combines the fronted element (the NP *Who*) with the auxiliary (*does*). The NP is entered onto the SLASH list. The binary filler rule is illustrated in (1). The next two rules, the binary valence rule and the verbal predicate rule, combine the NP *John* and the verb *admire* with the head projection. (Both these rules are head-initial.) The SLASH feature of the daughter is reentered in the mother in both rules. And finally, at the top of the tree, the valence extraction rule unifies the element on the SLASH list of its daughter with the extracted argument. This rule is illustrated in (2).

[1] The feature geometry in the implemented grammar is richer and more embedded than the one shown here. For expository reasons, features that are not relevant for the present discussion have been omitted. There has also been some overgeneralization with regard to what information is reentered in the SLASH list in the filler and extraction rules. In reality, only the HEAD, VAL(ENCE), CONT(ENT), and CASE features are copied across. Finally, the *force* rules that come on top of all parsed sentences in the implemented grammar have not been included.

[2] The function of the feature ARG(UMENT) is to allow a word or phrase to constrain what kind of argument it can combine with.

(1) $\begin{bmatrix} \textit{filler-binary} \\ \text{SLASH} \quad \langle \boxed{1} \rangle \\ \text{ARGS} \quad \left\langle \boxed{1}, \begin{bmatrix} \text{SLASH} & \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix}$

(2) $\begin{bmatrix} \textit{valence-extr} \\ \text{SLASH} \quad \langle\rangle \\ \text{ARGS} \quad \left\langle \begin{bmatrix} \text{ARG} & \boxed{2} \\ \text{SLASH} & \langle \boxed{2} \rangle \end{bmatrix} \right\rangle \end{bmatrix}$

It is assumed that also subjects undergo the SLASH mechanism when they appear as the first constituent in the clause. The sentence *John admires Mary* is given the analysis in Figure 2. Here, the subject, *John*, is filled in by the unary head-filler rule, and subsequently entered onto the SLASH list by the unary extraction rule. The unary filler rule is shown in (3). The rule can be seen as the combination of the filled-in constituent and an empty auxiliary.

(3) $\begin{bmatrix} \textit{filler-unary} \\ \text{HEAD} \quad \textit{aux} \\ \text{SLASH} \quad \langle \boxed{1} \rangle \\ \text{ARGS} \quad \langle \boxed{1} \rangle \end{bmatrix}$

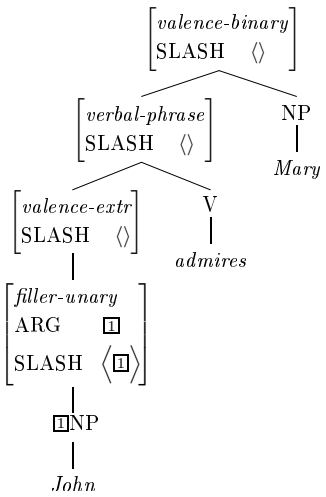

Figure 2: The SLASH feature: Fronted subject.

3 Parsing with the left-branching grammar design

The left-branching grammar design does not represent constituents in the syntactic tree, as is common in most other frameworks. In this section, it will be shown how the constituent structure of an utterance is reflected, and then how the design opens for incremental processing in a way which is compatible with psycholinguistic findings.

3.1 Constituency

The left-branching grammar design represents constituents by means of a stacking/popping mechanism. This mechanism allows the parser to enter embedded structures by entering selected syntactic and semantic features of the matrix constituent on a stack while taking on features of the

embedded structure. When the embedded structure has been processed, the matrix features are popped from the stack, and the processing of the matrix constituent proceeds. Examples of constituents where this mechanism is employed are NPs, PPs, CPs, and IPs. The mechanism allows for multiple embeddings.

The STACK mechanism is motivated by the fact that gaps can appear inside embedded constituents. The SLASH feature is not affected by the STACK mechanism, in the sense that while the syntactic HEAD and VAL features and the semantic HOOK features are entered onto the stack, the SLASH feature is passed up from the (first) daughter to the mother.[3] Since the SLASH feature in this way is passed on to the embedded structure, rather than the stack, the mechanism allows us to keep the assumption that the extraction rule dominates the filler rule, also when the extraction site is in an embedded structure.[4]

The STACK mechanism consists of two types of rules: i) the embedding rules, which enter selected features of the matrix constituent on the STACK list, and ii) the popping rule, which pops the features of the matrix constituent from the stack and takes them on. The stacking/popping mechanism is illustrated for the CP *that he slept* in (4) in Figure 3.[5]

(4) John says that he slept.

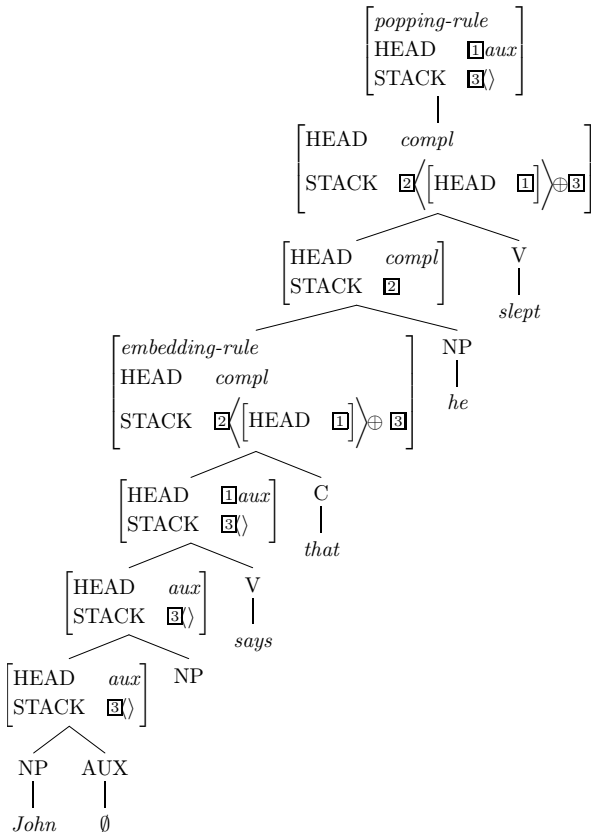

Figure 3: STACK mechanism in embedded clause

[3] An exception to this principle is when the embedded constituent is an NP. (See discussion in Section 4.)

[4] The percolation of the SLASH feature from mother to (initial) daughter in the left-branching structures makes the presence of a gap accessible to all constituents appearing between the filler and the gap, and hence offers a straightforward account of the registering of the extraction path that is reflected on verbs and complementizers in languages like Chamorro (Chung, 1998) and Irish (McCloskey, 1979).

[5] Only the reentrancies of the HEAD feature is displayed in this analysis. As mentioned, also the VAL features and the semantic HOOK are entered into the STACK.

The use of the stack reflects the constituent structure of a parsed string. In (4), there is one embedding, the subordinate clause. The embedding rule and the popping rule marks the beginning and the end of the embedded constituent. The constituent structure of this clause is given in Figure 4.

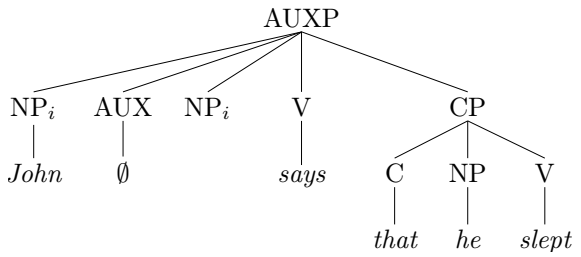

Figure 4: Constituent structure of sentence with subordinate clause

The sentence in (5) has three levels of embedding (CP, PP, and DP). The constituent structure of the sentence is given in Figure 5.

(5) John says that he slept in the garden.

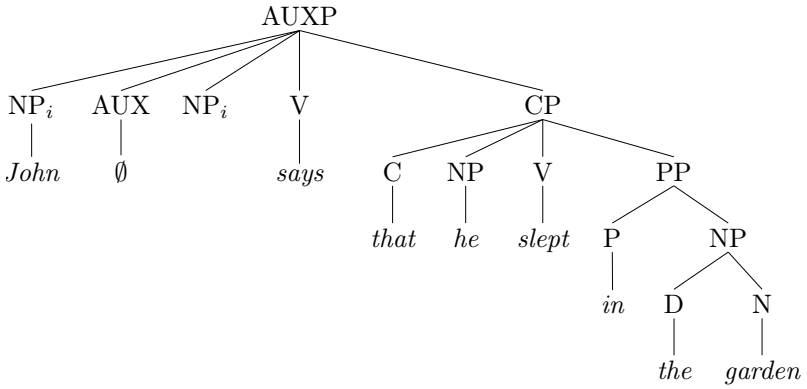

Figure 5: Constituent structure of sentence with subordinate clause

The fact that the left-branching grammar design operates with a stack, should normally make it non-incremental. It is however not so that constituents are put on a stack for later processing. It is rather a way to keep track of what level of embedding the parser is operating on, and only a few selected features of the matrix structure are entered. It can be compared to the use of SLASH in HPSG, which function is to make sure that the values of certain features are reentered in another part of the structure in order to account for long-distance dependencies.

3.2 Efficient processing of ambiguous structures

Even though the left-branching grammar design is incremental, it expresses the same ambiguities as other constraint-based grammars. Since ambiguities always add complexity, the more ambiguous an utterance is, the bigger is the processing effort for the parser. This contrasts with a psycholinguistic study by Swets et al. (2008) which shows that processing of ambiguous syntactic structures actually can be more efficient than corresponding unambiguous structures.

The left-branching grammar design could however open for a parsing strategy which would yield results compatible with the findings reported by Swets et al. (2008). Instead of conducting a full analysis of all possible readings of an ambiguous utterance and perform a parse ranking after all the analyses are finished, one could let a parser for each processed word commit itself to a

particular analysis given the information available at that stage, that is, the structure that is built so far and the word that is added to the structure. Assuming that at each step, a default analysis would be available, parsing an ambiguous structure could in fact turn out to be more efficient, since an unambiguity could conflict with the default analysis and hence create a greater processing effort.

One expected effect of such an incremental parsing strategy would be that the parser would run into garden paths where it has to backtrack and do parts of the analysis over. This would also be in line with psycholinguistic findings. Such a technique is proposed for HPSG parsers in Ninomiya et al. (2009) and Ytrestøl (2011), who use shift-reduce parsers with backtracking mechanisms instead of chart parsers.

4 Discussion

In the presentation of long distance dependencies in Section 2, the SLASH feature is "detached" from the constituent tree. This makes it possible to give a very simple account of long distance dependencies, namely one where the gap dominates the filler. The dependency between the gap and the filler is accounted for by the SLASH feature, which goes from mother to the first daughter.

The presentation did not include the treatment of NP constituents. Like the subordinate clause constituents and the PP constituents, the NP constituents should also be analyzed as embedded structures, but in contrast to the other embedded structures mentioned, the SLASH value will here be transferred to the STACK, rather than directly to the mother (and hence the embedded constituent). This accounts for island effects of complex NPs, where elements cannot be extracted from complex NPs (Ross, 1967, 118–158).

All elements that are represented as constituents in the constituent trees in (4) and (5) can be coordinated. Coordination can be accounted for by means of coordination rules, which, when one conjunct is parsed, will initiate another conjunct, which will be coordinated with the first.[6] Each conjunct will get the same SLASH list from the matrix constituent, and so coordination island effects are accounted for.

As in other HPSG grammars, the semantics is composed in parallel with the syntax. This means that there will be a (partial) semantic representation for each word added to the structure. The constructionalist design of the grammar allows arguments to be linked as they appear. So even if the language is verb final, like Japanese, the arguments will be linked instantly. With a lexicalist design on the other hand, the arguments of a verb cannot be linked before the verb itself has been parsed. So given a verb-final sentence, the whole sentence must be parsed before the arguments can be linked (given that the parsing is done from left to right).

5 Conclusion

The grammar design that has been presented is radically different from standard HPSG. The most striking diference is probaly the fact that the syntactic structure is not reflecting the constituent structure, but rather the parsing strategy. This is a result both of providing a simple account of long distance dependencies as well as making the grammar compatible with incremental processing in line with psycholinguistic findings.

References

Sachiko Aoshimia, Masaya Yoshida, and Colin Phillips. 2009. Incremental processing of coreference and binding in japanese. *Syntax*, 12(2):93–134.

Gosse Bouma, Rob Malouf, and Ivan A. Sag. 2001. Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory*, 1(19):1–65. URL `http://ftp-linguistics.stanford.edu/sag/bms-nllt.pdf`.

[6] For the moment, the grammar has special rules to account for coordination ov VPs which in the analysis presented does not have a designated constituent.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures with Applications to Unification-based Grammars, Logic Programming and Constraint Resolution*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, New York.

Sandra Chung. 1998. *The Design of Agreement: Evidence from Chamorro*. Folktales of the World Series. University of Chicago Press. URL http://books.google.com.sg/books?id=UwOdArsREBEC.

Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI publications.

Peter W. Culicover. 1997. *Principles and Parameters. An Introduction to Syntactic Theory*. Oxford University Press.

Gerald Gazdar. 1981. Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, 12:155–184.

Petter Haugereid. 2009. *A constructionalist grammar design, exemplified with Norwegian and English*. Ph.D. thesis, NTNU, Norwegian University of Science and Technology. URL http://urn.kb.se/resolve?urn=urn:nbn:no:ntnu:diva-5755.

Richard K. Larson. 1988. On the double object construction. *Linguistic Inquiry*, 19:335–391.

James McCloskey. 1979. *Transformational Syntax and Model-Theoretic Semantics*. Dordrecht: Reidel.

Takashi Ninomiya, Takuya Matsuzaki, Nobuyuki Shimizu, and Hiroshi Nakagawa. 2009. Deterministic shift-reduce parsing for unification-based grammars by using default unification. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 603–611. Association for Computational Linguistics, Athens, Greece. URL http://www.aclweb.org/anthology/E09-1069.

Colin Phillips. 2003. Linear order and constituency. *Linguistic Inquiry*, 34:37–90.

John Robert Ross. 1967. *Constraints on variables in syntax*. Ph.D. thesis, MIT.

Benjamin Swets, Timothy Desmet, Charles Clifton Jr., and Fernanda Ferreira. 2008. Underspecification of syntactic ambiguities: Evidence from self-paced reading. *Memory & Cognition*, 36(1):201–216.

Gisle Ytrestøl. 2011. Optimistic backtracking - a backtracking overlay for deterministic incremental parsing. In *Proceedings of the ACL 2011 Student Session*, pages 58–63. Association for Computational Linguistics, Portland, OR, USA. URL http://www.aclweb.org/anthology/P11-3011.