A left-branching grammar design for incremental parsing

Petter Haugereid
Division of Linguistics
and Multilingal Studies
Nanyang Technological University
Singapore
Department of Computer Science
University of Haifa, Israel

Mathieu Morey
Division of Linguistics
and Multilingal Studies
Nanyang Technological University
Singapore
Laboratoire Parole et Langage
Aix-Marseille Université, France

Proceedings of the 19th International Conference on
Head-driven Phrase Structure Grammar

Department of Linguistics, Chungnam National University Daejeon, South Korea

Stefan Müller (Editor)

2012

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

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.

1 Introduction

Until recently natural language parsing was commonly conceived as a chart-based, head-driven process, in particular among the HPSG community (Ninomiya *et al.*, 2009; Ytrestøl, 2011). This conception has had a significant impact on the design of implemented HPSG grammars and even more so when parsing efficiency was desired. Psycholinguistic studies however suggest that human sentence processing is not head-driven nor chart-based but incremental and deterministic. Such findings are of wide relevance as they suggest different means of achieving efficient parsing, that, in turn, call for different grammar designs.

The notion of incremental parsing/processing is well established in the psycholinguistic literature, and refers to the notion of words being added to an overall syntactic structure one by one. This is evidenced by studies showing that sentences in head-final languages do not require a higher processing effort than a head-initial sentence, even though the head, which according to traditional constituent analysis is required to form a constituent, appears after several of its arguments. The example in (1) taken from Swets *et al.* (2008) shows how as many as 7 arguments and adjuncts may appear before the first verb in Japanese.

(1) John-ga denwa-de Mary-ni Tom-ga asa rokuji-ni inu-ni
John-NOM phone-by Mary-DAT Tom-NOM morning six-at dog-DAT
esa-o ageta ka kiita.
food-ACC gave if asked
John asked Mary by phone if Tom gave his dog food at six in the morning.

The notion of deterministic parsing refers to the aim of producing a unique analysis for a sentence, which, in an incremental setting, usually implies to make decisions at each step (Ytrestøl, 2011). This is suggested by evidence that humans parse structurally ambiguous sentences more efficiently than structurally unambiguous sentences. The examples in (2) are taken from Van Gompel *et al.* (2001).

[†]We would like to thank the audience at HPSG 2012, Daejeon, South Korea, and three anonymous reviewers for their useful comments and feedback. This research was supported in part by the Erasmus Mundus Action 2 program MULTI of the European Union, grant agreement number 2009-5259-5.

Experiments show that the ambiguous sentence in (2a) is processed faster than the unambiguous sentences in (2b) and (2c). This is contrary to what one would expect from a deep non-deterministic parser, which generally requires a higher processing effort to process ambiguous sentences than unambiguous sentences.

(2) a. The maid of the princess who scratched herself in public was terribly humiliated.

b. The son of the princess who scratched himself in public was terribly humiliated.

c. The son of the princess who scratched herself in public was terribly humiliated.

Much of the linguistic analysis in the psycholinguistic literature is conducted within the framework of GB/Minimalism. For example, 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 it is possible to achieve a similar analysis by means of an appropriately designed HPSG grammar that retains full compatibility with a standard bottom-up HPSG parser. This grammar design characteristically provides 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 fragment 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.[1]

[1]The grammar fragment presented is a modified version of a grammar for Norwegian, Norsyg, (`http://moin.delph-in.net/NorsygTop`) and is a part of the DELPH-IN effort (`http://www.delph-in.net/`)

2 A Left-branching grammar design

The grammar fragment presented in this paper has a left-branching grammar design, which allows for words to be incorporated into the overall structure one by one. The design can be compared to Left-Associative Grammar (LAG) (Hausser, 1989), which also combines words to the overall structure one by one, resulting in a binary left-branching tree. But where LAG does not construct anything corresponding to a conventional constituent tree, but rather makes the step directly from the binary left-branching syntactic tree to a semantic representation, our approach employs a feature STACK in order to represent the constituent structure. Similarly to LAG the semantic representation is constructed "on the fly," as the sentence is processed from left to right. In section 4 we will return to how the constituent structure is reflected by the STACK feature. In this section, however, we will give an introduction to a couple of the features involved in the left-branching grammar design.

The tree in Figure 1 shows how a subordinate clause is analysed with the left-branching grammar design.[2] The head of the clause is the complementizer. The verbs and arguments attach to the complementizer projection from the right. Arguments are selected via the ARG(UMENT) feature with the valence rules. The ARG(UMENT) feature is a pivot for four different argument features, C-ARG1, C-ARG2, C-ARG3, and C-ARG4, corresponding to what in Government and Binding would refer to the 'external argument', 'direct object internal argument', 'indirect object internal argument', and 'goal/locative oblique', respectively. The grammar design has a mechanism that allows the grammar writer to constrain what combination(s) of arguments a verb can have. The rules that combine the arguments with the head projection (the valence rules) link the argument to the main predicate of the clause. Until the main verb is selected, the main predicate is left underspecified. This makes it possible to integrate the semantic linking of arguments before the main verb is encountered. The mechanism for constraining what combination(s) of arguments that can appear in a clause will not be a topic of this paper. (See Haugereid (2007, 2009, 2012) for detailed accounts of how arguments are linked, and how verbs are allowed to appear with different constellations of arguments.)

Verbs are selected via the VBL (VERBAL) feature with the verbal rule. As shown in Figure 1, a complementizer constrains the verb it selects to have the HEAD value *aux-verb*, which means that it is either an auxiliary or a main verb, and it also requires the TENSE value to be *finite*. When a verb is realized by the verbal phrase, the VBL value of the selected verb becomes the VBL value of the phrase. This allows a verb to constrain whether it will be followed by another verb and what kind of verb it is. The auxiliary in Figure 1 constrains the following verb to be a main verb past participle, while the main verb has the VBL constraint

[2]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. Also, the *force* rules that come on top of all parsed sentences in the implemented grammar have not been included.

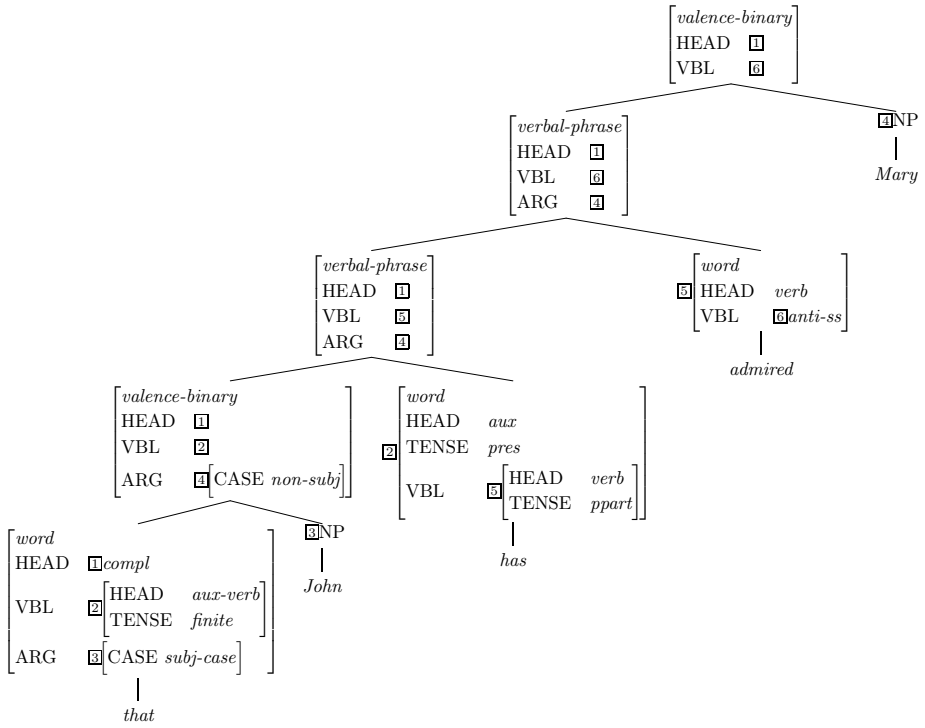

Figure 1: Selection of arguments and verbs in a subordinate clause

anti-synsem, which means that no more verbs can be selected.

The fact that the main verb is selected by the overall structure, and that arguments can be linked before the main verb is encountered, is due to constructionalist design of the grammar. The underlying assumption is that the syntactic rules together with function words and inflections provide a skeleton that the open class words fit into. By splitting a construction up into *sub-constructions*, which are realized as single syntactic rules, function words, or inflections, the overall construction can be build incrementally, and the open class words are fitted into this construction as they appear.

3 Long Distance Dependencies

Contrary to the analysis of subordinate clauses just presented, the analysis of English main clauses assumed in the proposed grammar design implies the use of the HPSG SLASH feature.

The use of a slash to account for long-distance dependencies in a monostratal account was introduced by Gerald Gazdar (1981), where 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 trace-less account of long distance dependencies in Bouma *et al.* (2001) involves entering all arguments and modifiers of a verb onto a separate DEPS (DEPENDENTS) list and retrieving the slash from this list. The DEPS list is created by means of the Argument Structure Extension constraint shown in (3).

(3) **Argument Structure Extension:**

$$verb \Rightarrow \begin{bmatrix} \text{ARG-ST} & \boxed{1} \\ \text{DEPS} & \boxed{1} \oplus list\Big(adverbial\Big) \end{bmatrix}$$

Since there is no limit to the potential number of adjuncts added to the DEPS list by the Argument Structure Extension constraint, the number of possible lexical descriptions of a verb is infinite. This is problematic from a psycholinguistic perspective, since it means that the DEPS list cannot be fixed before the parsing of the sentence has reached a state where the number of adjuncts is determined (or possible to determine), and the SLASH mechanism ends up as a potential post-parsing process. This problem is acknowledged by the authors:

> The infinity which is a consequence of Argument Structure Extension is also similar to the infinity which arises as a consequence of recursive lexical rules (i.e. rules which may apply to their own output). For example, the Adjunct Lexical Rule allows a single lexical item to give rise to an infinite number of derived items. As argued in van Noord and Bouma (1994), the computational problem posed by this kind of recursion can be solved by reformulating lexical rules as recursive constraints on lexical entries, whose evaluation can be delayed to a point where only a finite number of solutions remain (typically, after some syntactic processing has taken place). (Bouma *et al.*, 2001, 15)

The account of long distance dependencies in this paper is similar to the Gazdar (1981) "trace" account, apart from the fact that the SLASH feature "percolates down" the tree, rather than "up". The tree in Figure 2 is an analysis of the Wh-question *Who does John admire?*[3]

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 (4). 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 (5).

[3]There has 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.

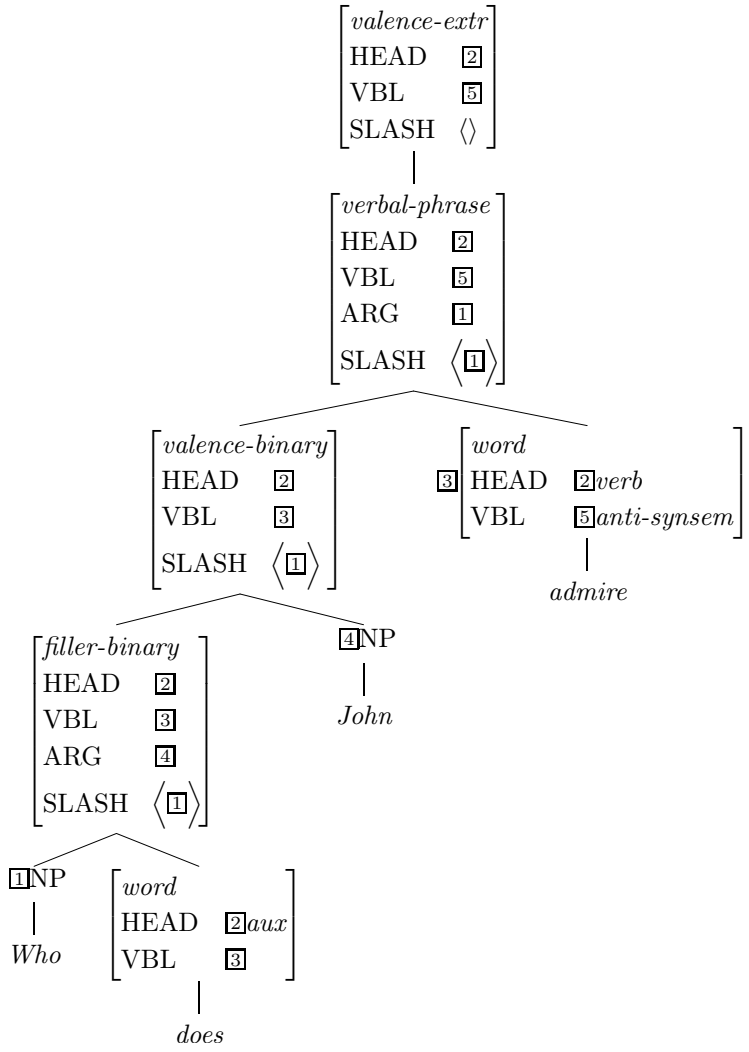

Figure 2: The SLASH feature: Fronted object.

(4)
$$\begin{bmatrix} \textit{filler-binary} \\ \text{SLASH} \quad \langle \boxed{1} \rangle \\ \text{ARGS} \quad \left\langle \boxed{1}, \begin{bmatrix} \text{SLASH} \quad \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

(5)
$$\begin{bmatrix} \textit{valence-extr} \\ \text{SLASH} \quad \langle\rangle \\ \text{ARGS} \quad \left\langle \begin{bmatrix} \text{ARG} \quad \boxed{2} \\ \text{SLASH} \quad \langle \boxed{2} \rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

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 3. 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 (6). The rule can be seen as the combination of the filled-in constituent and an empty auxiliary.

(6)
$$\begin{bmatrix} \textit{filler-unary} & \\ \text{HEAD} & \textit{aux} \\ \text{SLASH} & \langle \boxed{1} \rangle \\ \text{ARGS} & \langle \boxed{1} \rangle \end{bmatrix}$$

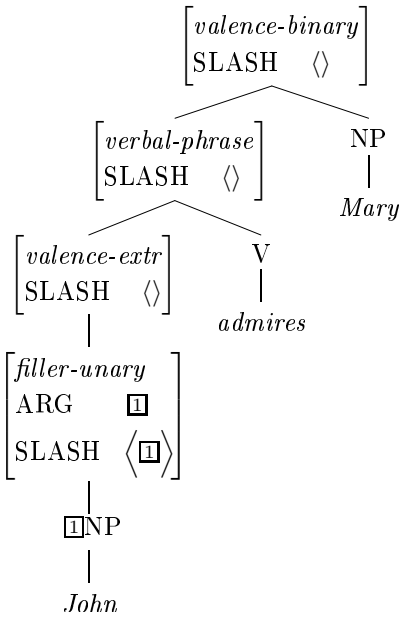

Figure 3: The SLASH feature: Fronted subject.

4 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.[4] 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.

[4] As mentioned, Hausser's Left-Associative Grammar is an exception.

4.1 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.[5] 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.[6]

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 (7) in Figure 4.[7]

(7) John says that he slept.

The use of the stack reflects the constituent structure of a parsed string. In (7), 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 5.

The ambiguous sentence in (8) has up to three levels of embedding (CP, PP, and DP). The two possible constituent structures of the sentence are given in Figure 6 and Figure 7. The different PP attachment is accounted for by letting the rule that pops the complementizer projection apply either after the PP embedding rule (low PP attachment) or before the PP embedding rule (high PP attachment).

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

[5] An exception to this principle is when the embedded constituent is an NP. (See discussion in Section 5.)

[6] 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), one of the motivations behind the no-trace account of long distance dependencies in Bouma *et al.* (2001).

[7] 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.

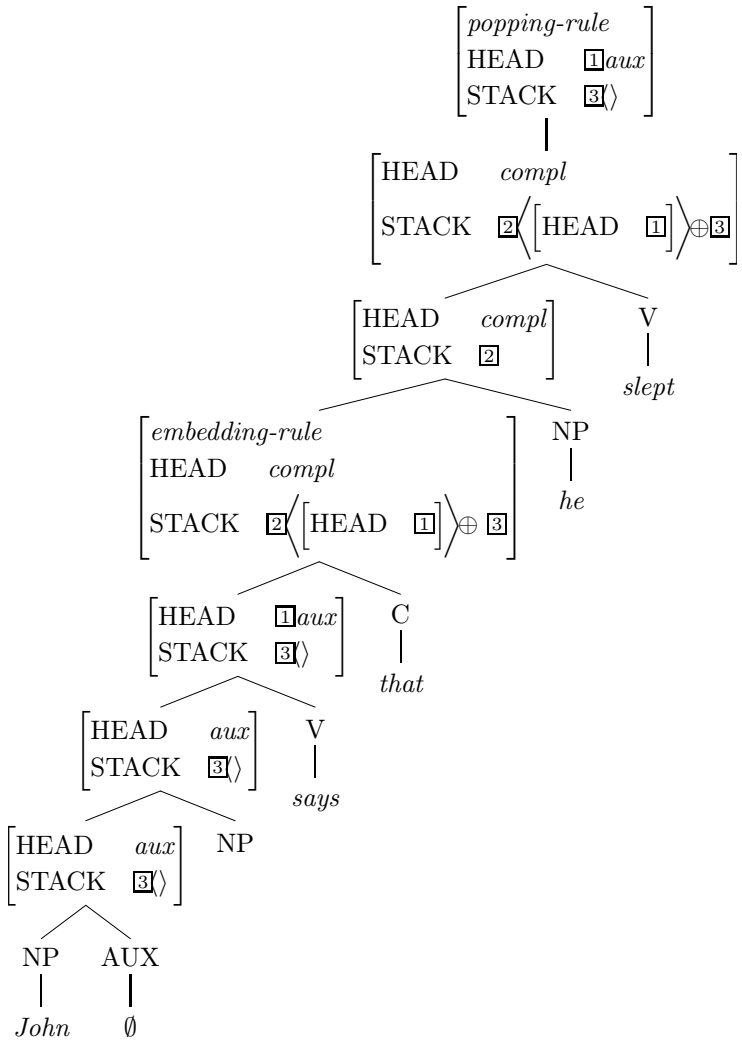

Figure 4: STACK mechanism in embedded clause

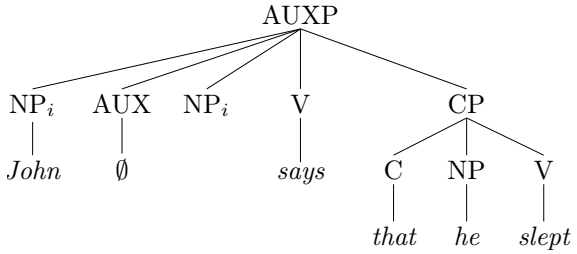

Figure 5: Constituent structure of sentence with subordinate clause

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

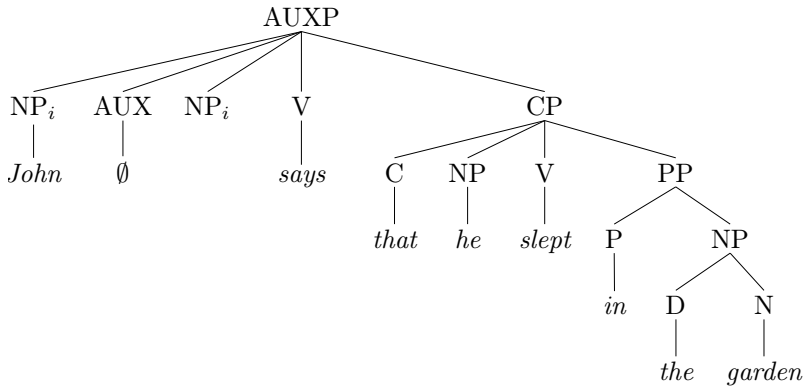

Figure 6: Constituent structure of sentence with subordinate clause. Low PP attachment.

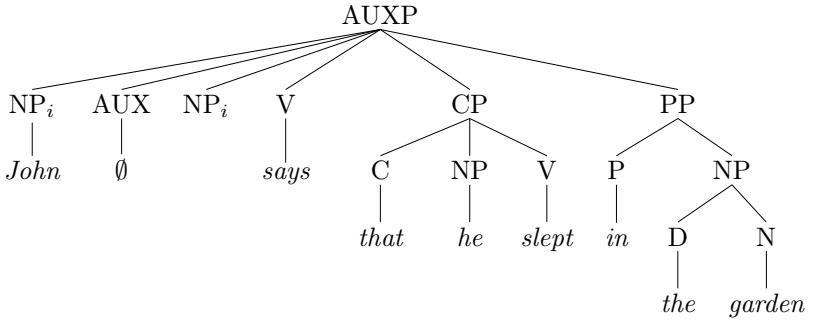

Figure 7: Constituent structure of sentence with subordinate clause. High PP attachment

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 is comparable 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.

4.2 Efficient processing of ambiguous structures

Even though the left-branching grammar design is incremental, it expresses the same ambiguities as other constraint-based grammars. In traditional chart-based parsing, ambiguities always add complexity, thus 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 that of corresponding unambiguous structures.

The left-branching grammar design however naturally lends itself to incremental processing and is thus inherently compatible with deterministic parsing strate-

gies. Instead of conducting a full analysis of all possible readings of an ambiguous utterance and performing a parse ranking after all the analyses are finished, an alternative strategy consists in having the parser make local decisions after each word is processed given the information available at that stage, that is, the structure that has been built so far and the word that is added to the structure. Assuming that at each step, a default analysis is available, parsing an ambiguous structure can in fact turn out to be more efficient on average than an unambiguous structure.

Unambiguous sentences can lead a parser using a deterministic incremental strategy into garden paths where it has to backtrack and do parts of the analysis over. Incremental deterministic parsers have been proposed for HPSG by Ninomiya *et al.* (2009) and Ytrestøl (2011), in the form of shift-reduce parsers with backtracking mechanisms.

5 Discussion

In the presentation of long distance dependencies in Section 3, 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, NP constituents are also 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 (5) and (6) 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.[8] 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).

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

6 Conclusion

The grammar design that has been presented is radically different from standard HPSG. The most striking difference is probably 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 deterministic incremental processing in line with psycholinguistic findings.

References

Bouma, G., Malouf, R., and Sag, I. A. (2001). Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory*, **1**(19), 1–65.

Carpenter, B. (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.

Chung, S. (1998). *The Design of Agreement: Evidence from Chamorro*. Folktales of the World Series. University of Chicago Press.

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. CSLI publications.

Culicover, P. W. (1997). *Principles and Parameters. An Introduction to Syntactic Theory*. Oxford University Press.

Gazdar, G. (1981). Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, **12**, 155–184.

Haugereid, P. (2007). Decomposed phrasal constructions. In S. Müller, editor, *Proceedings of the 14th International Conference on Head-Driven Phrase Structure Grammar*, Stanford, CA. CSLI Publications.

Haugereid, P. (2009). *A constructionalist grammar design, exemplified with Norwegian and English*. Ph.D. thesis, NTNU, Norwegian University of Science and Technology.

Haugereid, P. (2012). A grammar design accommodating packed argument frame information on verbs. *International Journal of Asian Language Processing*. To appear.

Hausser, R. (1989). *Computation of language: An essay on syntax, semantics, and pragmatics in natural man-machine communication*. Springer-Verlag.

Larson, R. K. (1988). On the double object construction. *Linguistic Inquiry*, **19**, 335–391.

McCloskey, J. (1979). *Transformational Syntax and Model-Theoretic Semantics*. Dordrecht: Reidel.

Ninomiya, T., Matsuzaki, T., Shimizu, N., and Nakagawa, H. (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, Athens, Greece. Association for Computational Linguistics.

Phillips, C. (2003). Linear order and constituency. *Linguistic Inquiry*, **34**, 37–90.

Ross, J. R. (1967). *Constraints on variables in syntax*. Ph.D. thesis, MIT.

Swets, B., Desmet, T., Clifton Jr., C., and Ferreira, F. (2008). Underspecification of syntactic ambiguities: Evidence from self-paced reading. *Memory & Cognition*, **36**(1), 201–216.

Van Gompel, R., Pickering, M., and Traxler, M. (2001). Reanalysis in sentence processing: Evidence against current constraint-based and two-stage models. *Journal of Memory and Language*, **45**(2), 225–258.

van Noord, G. and Bouma, G. (1994). Adjuncts and the processing of lexical rules. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 250–256. Association for Computational Linguistics.

Ytrestøl, G. (2011). Optimistic backtracking - a backtracking overlay for deterministic incremental parsing. In *Proceedings of the ACL 2011 Student Session*, pages 58–63, Portland, OR, USA. Association for Computational Linguistics.