Pre-parsing Efficiently - MA Thesis

Uriel Cohen Priva

September 12, 2006

Abstract

This work tries to model human parsing by assuming that some part of the parsing process is designed to complete its work in linear complexity. This part of the parser, the pre-parser, produces underspecified structure of which the correct parse tree can be re-built by subsequent modules. I demonstrate how such an assumption predicts a range of complex processing difficulties phenomena such as garden path and right association, while assuming little else about the parser.

Acknowledgments

I would like to thank my adviser, Fred Landman, whose guidance and wise criticism has led me through the mess of confused and confusing ideas that preceded this work. His precise observations, his devotion to the process, and most importantly his willingness to both accept and subdue my somewhat stubborn and alien approach to linguistics, have made this work possible.

I would also like to thank my thesis committee, Susan Rothstein and Alexander Grosu for their invaluable remarks, and on their vital role in making this thesis readable to fellow linguists. I am sorry I have made you read those preliminary versions, I really am.

I am grateful to Mira Ariel for believing in me during my years in the department, for always letting me think I can do better, and for explaining these views to others.

I am indebted to Nirit Kadmon for showing me I could enjoy being a teaching assistant. This realization had a major impact on my decision to proceed to study for a PhD.

To Tali Siloni and Outi Bat-El I owe my interest in theoretical linguistics. They have successfully nurtured both my excitement about the field and my criticism of it. I thank them for teaching me how theory should be built, what's important and what's argumentative.

Many thanks go to Tal Kedar, with whom I discussed this theory from its preliminary stages, all the way through its final outcome. His helpful remarks have helped me overcome many obstacles. For this and for his companionship, I am thankful.

This is the best place to express my gratitude to Roy Rosemarin, whose invaluable remarks and support have let me bring the thesis to its final state. I apologize for all those lost coffee meetings hours.

I am extremely grateful to my parents, Yael and Gabriel Cohen Priva, who have managed to accept I am doing what I desire, and for not tiring of asking me to explain what this thesis is all about.

Finally, to the 'A's of my life, for the good moments of these last few year, I give my love.

To My Parents

Contents

1 Introduction

1.1 The Human Parser

In the field of linguistics, it is rather surprising that the notion of the 'human parser' is not equivalent to that of 'language'. After all, every first year student of linguistics is told at the very first introductory course that the greatest change that modern linguistics represents is in trying to model language not as something that lies outside human cognition, but rather as a projection of a cognitive device, our language faculty. In other words, the study of human language should be equivalent to the study of the human parser: every claim we make about language is a claim about the way we utter and understand sentences. However, the modern study of language has also made a distinction between langue and parole, competence and performance, and the distinction between the two - the fact that our ability to judge sentence grammaticality is different from our ability to utter or construct sentences - has been used to differentiate between the two terms.

Kimball (1973) demonstrates that the difference between grammatical and 'acceptable' creates four sets of sentences: those that are both grammatical and acceptable, such as (1.a), those that are grammatical but unacceptable[1] such as (1.b), those that are ungrammatical but acceptable such as (1.c), and those which are ungrammatical and unacceptable, such as (1.d).

(1) (a) It is raining.

(b) Tom figured that that Susan wanted to take the cat out bothered Batsy out.

(c) They am running

(d) Tom and slept the dog.

Kimball does not distinguish here between sentence production and sentence understanding, as is clear from example (1.b). I would like to explain these distinctions by focusing more on sentence understanding, rather than production. Let us take two imaginary questionnaires. The first questionnaire requires the subject to say whether a sentence, some content and a context match: the subject has to decide whether the sentence can be used to express the content in the given context. The second questionnaire gives only the sentence, and requires the subject to say what content the sentence can be used to express in

[1] I find the equivalent examples in Hebrew ungrammatical

varying contexts. The first questionnaire relates directly to grammaticality tests, the ones we often use in syntax and semantics. The second relates to what we understand when we hear or read a sentence: speakers may fail to understand a grammatical sentence, or understand it only after a relatively long time; speakers may miss some of the meanings in an ambiguous sentence; finally, speakers may understand the content of ungrammatical or partial sentences. In cases where a speaker does not understand a grammatical sentence, or does not recognize a possible ambiguity, we witness a shortcoming of the human parser. A speaker's ability to understand a partial or ungrammatical sentence demonstrates the human parser's 'robustness' (a term used in computer science to describe a program's ability to handle faulty data). Both the shortcomings of the human parser and its error recovery ability demonstrate that the parser does have distinct properties.

This work focuses on some of the phenomena that demonstrate the human parser's existence: the various difficulties that we encounter when we process certain sentences. I will deal mainly with the most famous of these phenomena: garden path sentences, but I will also discuss some less glamorous phenomena that demonstrate processing difficulties, as well as other parser related peculiarities, such as missing ambiguities.

1.2 Mission and Disclaimer

Many works try to trace the workings of the human parser. The various approaches differ greatly by what they take for granted and what they try to achieve. Due to these differences, they differ also on what may be the legitimate means of achieving these objectives, and what phenomena should be account for the given explanation. This work tries to do the same, but begins by giving up as many assumptions as it can (or so I see it), and explain as much as it can, without focusing solely on one phenomenon.

To achieve this goal, I will not assume any specific syntactic framework my parser should model, but rather allow a modular analysis in which the parser can be followed by another syntactic module, a semantic module, or whatever one may wish to assume follows a parsing module. From this follows that I do assume a modular view of language. Another choice I make is to try and model the various limitations the human parser demonstrates by using computational complexity terms. Such a model may rely on a specific type of computational model, one that may be very different from the machinery our brain supplies. However, this should not bother the reader, as I do not try to describe the

human parser, but rather to model it. Just as syntax does not claim that, say, Merge has a direct physiological correspondent, my model of the human parser uses a particular computational model to express the way our brain parses sentences, without tracking down the exact mechanics the brain uses to realize this behaviour. My work is a formal one, and tries to model the behaviour of the human parser rather than mimic it. It makes no assumptions about the machinery provided by our brain and offers no insights about the actual psychological processes involved. I reduce the human parsing process to its outcome: meanings conveyed and difficulties encountered. My work tries to model just those aspects of human parsing. It *would* be interesting to try to see whether the products of this work are compatible with psycho-linguistic findings, but this is beyond the scope of this work.

1.3 Road map

The object of this work is to explain as much as it can using as few assumptions as possible. The object of research are the many features of the human parser, and the assumption I make is that the parser tries to correctly come up with structure using very tight complexity requirements.

In section 2, I describe the objectives and formal framework used to solve this problem. Section 2.1 draws the objectives of this work, describing the different parsing related phenomena I wish to explain - different kinds of processing difficulties and shortcomings - and the assumptions I make when approaching a solution: a modular algorithm that can resolve structure in linear complexity. In section 2.2, I describe in formal terms the differences between different kinds of parsers, and translate the notion of tree to that of dominance order relation, a notion that is used extensively in my work.

Section 3 tracks some of the current work on parsers and processing difficulties. A reader who is already familiar with similar work may wonder why important works of psycho-linguistic theory are missing from this section. The reasons are similar to those mentioned in my disclaimer: this is a formal work and I therefore compare it to other formal works. I rather focus on two formal approaches: parsing algorithms that rely on backtracking, and algorithms that manage to do without backtracking.

Section 4 is the main body of this work. In §4.1 I introduce the way I meet the objectives of this work, explain the way the complexity and modularity requirements can be met, and the formal representations of this solution. §4.2 provides the general outline of an implementation that relies on the way I meet those objectives. I then demonstrate in §4.3 how the algorithm manages to explain various processing difficulties. I also show what further adjustments have to be made so that other processing difficulties can also be explained, and introduce another possible complexity constraint. In §4.4 I talk about two goals that are not accomplished in the current implementations: moving syntactic features such as c-command into the parsing algorithms and explaining processing difficulties in head final languages.

2 Preliminaries

2.1 Objectives

2.1.1 Garden Paths

Garden path sentences are a class of sentences which are undeniably 'proper' syntactically, specify understandable content, but are not parsable - at least not as quickly as other sentences in the language. The term 'Garden Path' is not neutral and refers to the commonly accepted hypothesis that the reason those sentences are not easily parsable is as follows:

- The parser does not wait for the sentence to end in order to start building the sentence structure.
- At some point the parser may entertain two (or more) incompatible alternatives: that is, future data may prove one of those alternatives to be incorrect.
- The parser does not wait for disambiguating data, but rather chooses one of those alternatives.
- If future data does indeed prove the choice to be a wrong one, the parser is required to 'retrace' its steps, find the wrong decision, undo it, and try again.
- The retracing process takes effort and time, which causes the measurable cognitive effect of confusion or increase in response time.

For example, in (2) - perhaps the most famous garden path sentence - the parser cannot tell when it reaches the word 'raced' whether the verb is used in the matrix clause, or in a reduced relative clause.

(2) ¿[2] The horse raced past the barn fell

The parser seems to choose the former over the latter, and when proven wrong at the end of the sentence, as it encounters the second verb, 'fell', it has to go back and 'choose again', which requires the reassignment of the 'the horse' as the subject of 'fell', lowering 'raced . . . barn' to be a relative clause modifier of 'the horse'. This process is either not automatic or cognitively taxing, and we feel it takes us more time to figure out the correct underlying structure of the sentence. The term 'garden path' therefore correlates to descriptions in

[2] '¿' is a commonly accepted way to signal either 'garden path' or 'difficult to process'

which the parser might make a mistake, and when some error is encountered, an attempt to backtrack is made, which causes processing difficulties.

However, the description just provided is far from perfect, as there are cases in which we expect the parser to 'choose wrong', but no cognitive effort is encountered. In (3.a), 'Mary' should be interpreted as the direct object of the verb 'saw', and if the parser fails to do so, a correct interpretation of the sentence would be prohibited. However, that same sentence can be extended without effort to (3.b), in which 'Mary' does not serve as the object of 'saw', but rather as the subject of the embedded sentence. The difference between this example and the previous one is probably not due to the size of the reanalyzed part. In (4) [3], we supposedly choose wrong when we attach 'food' as the object of 'eats': just one word later we learn that we have made a mistake, and should re-attach 'food' as the subject of 'gets'. Supposedly, this is too late already.

(3) (a) John saw Mary.

(b) John saw Mary dance.

(4) ¿When Fred eats food gets thrown

From these cases two main explanatory paths diverge: either we can *sometimes* retrace our steps and backtrack, or we can assume that we made no wrong choice: we somehow did not choose wrong when we assigned 'Mary' the role of an object. The first sort of theories I would label *limited backtracking* theories, and the latter I would label (after Marcus (1978)) *deterministic* theories. Limited backtracking theories deal with defining the conditions in which backtracking does not cause conscious effort, while deterministic theories would have to create elaborate explanations to why reanalysis does not actually occur in these cases.

While all these theories describe the conditions under which garden path occurs, one of my objectives is to try and explain the data: to provide a mechanism through which garden path will follow from general principles. Such principles can only be general if they give rise to more than just the garden path phenomena. rather than merely describe it: such principles can only be valid if they would explain not only the garden path phenomena.

[3] From Ken Barker's list of garden path sentences, which can be found at http://www.site.uottawa.ca/~kbarker/garden-path.html

2.1.2 Processing Difficulties and Right Association

While garden path sentences are usually regarded as the most important phenomenon among sentences which are not easily processed - mainly since they provide evidence for the parser's need to choose one alternative over another - there are at least two other types of sentences which cause processing difficulties leading either to failure to understand a seemingly grammatical sentence, or to a significant delay in response times. Those are *center embedding* and *right association* sentences. These sentences are even more puzzling than garden path sentences, as they are usually unambiguous at every point, and thus do not require the parser to choose.

Center embedding sentences are interesting because we see no obvious reason that could explain the tremendous conscious effort required to understand them. Sentence (5.a) is an example of such a sentence. We would expect the sentence to be an alternative way of saying (5.b), built by combining (5.c) and (5.d).

(5) (a) ¿The boy the girl the dog bit saw shouted

(b) The boy that was seen by the girl that was bitten by the dog shouted.

(c) The boy the girl saw shouted.

(d) The girl the dog bit saw the boy.

However, for some reason embedding one sentence in this position is parsable, but embedding two sentences in the same position renders the sentence too difficult to parse. In fact, these sentences are not only difficult for the hearer to process but also for a speaker to produce, making it difficult to claim they are grammatical. Center embedded sentences are simply not used by speakers, which contrasts them with the other type I will discuss, right association sentences.

Kimball (1973) has described a range of phenomena he attributed to a basic parser's tendency to attach new nodes recently processed nodes, a tendency he labeled 'right association':

> Terminal symbols optimally associate to the lowest non terminal node

Kimball's description can be demonstrated by the difficulty encountered when reading sentences such as (6.a). This phenomenon is surprising, as similar sentences where the

object of the verb is not a clause, such as (6.b), sentences where the adverbial is a clause, such as (6.c) or even the same sentence, uttered with a pause preceding 'tomorrow' (6.d), are not problematic.

(6) (a) ¿John will tell the kids he has already bought them a dog tomorrow.

(b) John will tell the kids a story tomorrow.

(c) John will tell the kids he has already bought them a dog when he thinks it's appropriate.

(d) John will tell the kids he has already bought them a dog | tomorrow.[4]

Right association effect are more difficult to explain than garden path and center embedding sentences: they are used by speakers (unlike center embedding sentences), and do not originate in a mistake caused at an earlier stage: the parser has all the data it needs, but fails to use it. Kimball's descriptive rule of right association seems an arbitrary one.

I will not try to explain center embedding, although I believe the theory I will present does contain some leads to the proper explanation of the problem. I will only explain some right association effects in §4.3.4, but an ideal theory should manage to explain both problems using similar mechanics. I do consider using principles that cannot be applied to right association sentences a drawback of some garden path explaining theories.

2.1.3 'On-line' Parsing

Tomita (1987) demonstrates a relatively efficient algorithm for parsing a sentence of an arbitrary ambiguous[5] context free grammar. Such an algorithm can provide all the alternative interpretations of a sentence, and might represent an ideal parser that always manages to find the correct parse tree if one exists. The three types of processing difficulties mentioned above demonstrate that the human parser does not belong to this set of parsers.

By giving up on the attempt to achieve this goal, a parser can be more efficient in parsing most of the sentences, by exercising what is often called in computer science *greedy* behaviour. For algorithms, being greedy means trying to make the best choice given some subset of the information, at the expense of achieving the (best) solution every

[4]The pause mark (|) follows Kedar (2006) symbols

[5]That is, one which allows for some prefix of a grammatical sentence to support more than one valid interpretation.

single time. In parsing theories, acting 'on-line' means not delaying a decision for more information. Theories differ on what is considered 'delaying' a decision. A parser need not be assumed to make a decision for every single piece of input, but evidence shows that if the parser does indeed delays while waiting for more information it does not wait for too long: there are sentences (discussed below, in §2.1.4) which cause garden path effects in which the erroneous decision and the end of the sentence are not more than two or three words apart. It is not surprising that most theories that allow backtracking prohibit the use of any sort of delay or 'look-ahead'. In its most strict form, an 'on-line' parser is considered to make every decision right away.

Even if a theory does allow for some sort of delay to occur, the size of this delay window has to be considered and minimized. The smaller the window, the easier it would be to disprove the theory, thereby making it more elegant, and in my view, better. The theory I propose cannot be analyzed in terms of 'look-ahead' or 'on-line', and replaces those terms with the broader notion of complexity, discussed in the following section.

2.1.4 Complexity

Much of the research in the field of computer science has to do with the notion of complexity. This notion deals with evaluating different algorithms, all capable of performing some task. The evaluation is based on comparison of the processing time and storage space requirements of every algorithm in the comparison set, where processing time refers to the number of fixed time steps the algorithm takes to reach its final state, and storage space usually refers to the maximum number of fixed size items the algorithm keeps in memory until it reaches its final state. The comparison is based on the length of the input n. One of the most commonly used notations in this field, is the *Big Omicron* (or Big O) notation, as defined in Knuth (1976). The O notation defines a set of n based functions (denoted by '$g(n)$'), such that some function we choose (denoted by '$f(n)$') is bigger than or equal to these functions, for every input large enough (every input which has more than the first 'n_0' elements), disregarding constant multipliers (denoted by 'C').

> $O(f(n))$ denotes the set of all $g(n)$ such that there exist positive constants C and n_0 with $|g(n)| \leq C \cdot (n)$ for all $n \geq n_0$

This means that the complexity of an algorithm $f(n)$, $O(f(n))$, is the set of all algorithms $g(n)$ in a comparison set, that are at most in a constant way faster / take less space than

$f(n)$. For example, the set of functions define by $f(n) = n^2$ (which means we chose $O(n^2)$) would include:
$\{g(n) = an^2 + bn + c : a, b, c \in \mathbb{R}\}$
and other functions (for instance, $g(n) = \log n$). The fact that we can disregard constants helps us compare algorithms across different machines and programming languages. These constants may matter if the size of the input is guaranteed to always be small, or if we compare two algorithms with identical asymptotic complexity.

There are many uses for this notation. For example, when the number of items an algorithm stores is linearly dependent on the size of the input (that is, if the input doubles, the number of items doubles) we would say that the *space complexity* of the algorithm is $O(n)$ (a reminder: n is the length of the input). If the number of processing steps an algorithm needs to perform some task is independent of the input size, we would say the *time complexity* of the algorithm is $O(1)$. Real life examples for linear time or $O(n)$ missions include cleaning the floor: the time required to clean the floor would double if the floor space doubled. Here we can see how ignoring constants can be useful for us. Taking the mop out of the closet, for instance, is a constant in this example: the more we have to clean the less significant taking the mop out of the closer becomes. Real life example of an $O(1)$ task may be tying shoelaces: no matter how long the shoelaces may be, tying them takes a fixed amount of time (of course, if we defined the length of the input to be the *number* of shoelaces, this task would take linear time to complete).

Well, most theoretical linguists would ask, what has that got to do with *us*? I believe this is a very good question. Most linguistic theories are not defined in terms of processing time or storage requirements: they are abstract, independent of the underlying mechanism, and for a good reason: we do not know enough about the mechanism: our mind. But when we come to theoretical discussions about parsing, this view is no longer supportable. The human parser's "on-line" behaviour has been explained as a short term memory constraint. Kimball (1973) used the same terminology to account for right association. Moreover, the parser has been explicitly described, in (Pritchett, 1992; Weinberg, 1993, 1999) for instance, as trying to satisfy all the constraints for every input word as soon as it is encountered, and does not try to calculate the optimal solution for the entire input, what we have labeled above as a *greedy* behaviour: a complexity driven approach in computer science. Moving from vague terms of 'driven by restricted storage' to an actual analysis

of how is this storage limited is therefore necessary.

Surprisingly enough, the use of these terms for describing the motivations for the otherwise inexplainable behaviour of the human parser have remained outside the actual descriptions of such suggestions. Actual processing and storage requirements have not been discussed, and the different parsers have not been compared on this basis. Even when we analyze such solutions, it is usually very difficult to understand how complex the algorithm actually is. Pritchett's (1992) parser description, for instance, allows the parser to move from every structure to every structure (even in a non monotonic fashion), the only constraint being his On Line Locality Constraint (OLLC): "The target position (if any) assumed by a constituent must be *governed* or *dominated* by its source position (if any), otherwise attachment is impossible for the automatic Human Sentence Processor". Not a word about *how* these structures come to be considered. I find it therefore essential to try and delve into this prospect of parser evaluation, by trying to assess how difficult it is to identify the structure of sentences, starting with the lowest possible time and space complexity.

When discussing complexity, one needs to take into account some computational model framework. An algorithm that takes $O(n)$ in the commonly assumed model may take $O(n^2)$ in a Turing machine model. In a model in which an infinite number of instructions can be carried out 'at once', the above mentioned Tomita's (1987) model would take only linear time to compute all the alternatives. The model I will use to define complexity terms is the one most commonly used in computer science today: a Random Access Machine (RAM) model: a single instruction can be carried out at every given time, infinite storage is available for every instruction, and instructions include setting the value of every storage component to some value, or compare its value to some other storage component. This makes the RAM model very similar to today's computers, but perhaps very different from the brain.

Time complexity The time complexity of a parser cannot drop below $O(n)$, since by the time the parser finishes processing a sentence, it has at the very least read every word. Attaching every node where it should be attached may well take more than that, but a preliminary model should, I think, try to achieve an $O(n)$ time complexity.

Space complexity Bounding the space complexity is a much more intriguing question. Marcus's (1978) parser, one of the only parsers for which an algorithm has been provided, uses $O(1)$ storage for syntactic structures (that is, the storage size is finite and independent of the input length), but keeps the states of every node to which more elements can still be attached. The number of such states is the height of the syntactic tree, and since syntactic trees grow in an uneven manner (new words are usually attached to the last sentence or noun phrase as described in Kimball's (1973) right association guideline), we cannot guarantee the number of saved states to be less than $O(n)$, which brings Marcus's parser to a total of $O(n)$. This bound cannot be reduced for parsers which allow reanalysis, as a reanalysis process requires to "remember" words which have already been processed - possibly all the way up to the top node. Moreover, some element of syntax or semantics requires us to keep $O(n)$ previously processed elements: in sentences such as (7), the emphasized element should be attached to the VP of the matrix clause for the sentence to make sense, which means that verbs that have already been processed must still be available for further consideration, an arbitrary number of clauses away from the last clause ("Mary to ask" can be repeated as many times as we want).

(7) Dan told Mary to ask Jane to invite John to the party, *rather than do it himself.*

We cannot simply 'forget' what has already been processed. This means that if the parser I propose uses a linear amount of storage, it will not exceed the amount of storage already used. A model in which the space complexity is $O(n)$ is therefore a rather lean model, one which I find sufficient. However, a $O(1)$ model would be more compact and should be achieved if possible (see discussion in §4.3.2 and §4.3.4).

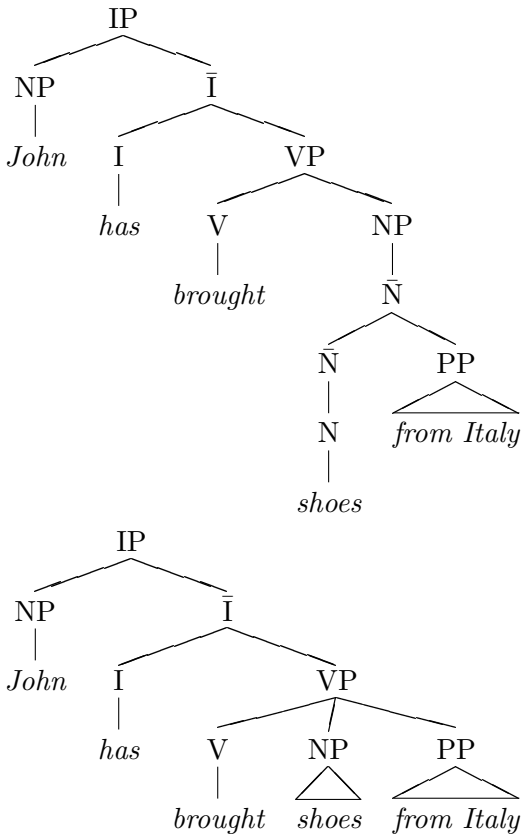
These bounds suggest that any syntactic process whose space and time complexity do not exceed $O(n)$ is at least as efficient as any parser we have encountered so far. My objective would therefore be not to exceed these bounds when trying to describe a parsing algorithm.

2.1.5 Modularity

It has generally been assumed, that syntactic processes cannot and should not handle complex semantic and pragmatic representations. Based on this assumption, it is clear that we should not expect syntax to resolve ambiguities such as in the following examples. In (8) '*from Italy*' can be attached to the verb as an argument, or to the noun as an adjunct

- the decision relies heavily on what we know already about John. In (9.1) we attach '*using Eudora*' to the lower verb because it makes more sense, while in (9.2), we attach '*on the phone*' to the higher verb for similar reasons. Under the modularity assumption, we would have to rule out any theory in which it is syntax that determines where to attach these elements.

(8) John has brought shoes *from Italy.*



(9) 1. Mary told Dan to mail a letter to Jane *using Eudora*[6].

2. Mary told Dan to mail a letter to Jane *on the phone.*

Since knowing where to attach these elements requires semantic and pragmatic information, we do not expect syntax to be able to decide where the attachment takes place, and we therefore assume that it does not. Any description of a parser should therefore include some explanation to what *really* happens in these circumstances. We have several options:

1. Syntax passes on every possible structure to the next module. That is, whenever we encounter an ambiguity, all the alternatives are taken, and each and every one

[6]Eudora is a popular e-mail program

survives to the 'surface': the interface with the next module.

2. The interface between syntax and the next module determines what is the best attachment site for the new element. Under this analysis, whenever a choice between two alternatives has to be taken, both options are 'offered' to the interface, the interface selects one, and processing resumes.

3. Syntax passes on an underspecified structure to the next module: a structure which the next module will extend correctly, using semantic and pragmatic data.

The first option is inconsistent with both our complexity requirement above, and the nature of the syntactic process in general: it requires syntax to maintain several possible structures at once. If this were possible, there would be no need for the parser to commit itself to certain syntactic structures too early - what we assume to cause the Garden Path phenomena - and the correct tree in right association sentences would be available. What's more, the number of possible solutions may grow exponentially: every PP attachment ambiguity may survive all the way up to the final interpretation.

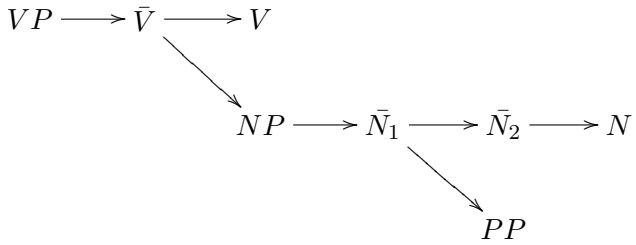
The second option is more consistent with the current analysis of syntax and with the garden path phenomena, but is inconsistent with the complexity constraint I chose: every possible attachment site has to be considered by the interface to be chosen or overruled. Therefore, the final time complexity will be greater than $O(n)$. This is not to say that this option is irrelevant or should not be pursued when dealing with the interface between the currently described algorithm and the next module, but one which I should at least try to do without.

The third option is appealing, as it seems to allow the creation of a syntactic process which adheres to both the modularity requirement and the complexity constraint. It should be noted though, that it makes the algorithm to be described at the very least 'less than a parser', as parsers are expected to deliver a structure or a finite number of complete structures, rather than an underspecified structure.

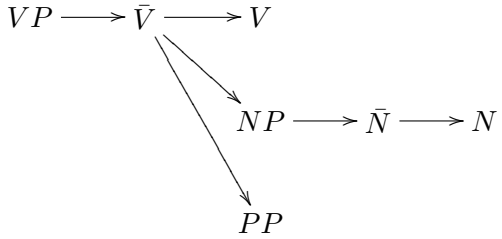
A question remains: how can some structure be underspecified? A possible idea can be borrowed from D-Theory (Marcus et al., 1983; Barton and Berwick, 1985; Weinberg, 1993)), although it is not used for this purpose in any of these articles. When we think

of a parser we imagine it building a tree. Trees, however, can be represented not only by the immediate dominance relation ('VP_1 immediately dominates $\bar{V}_1$'), but also by a dominance order relation (see also §2.2.1), in which we may say that some tree node A dominates some other node B if and only if A dominates the mother of B, or A is the mother of B. If the *immediate dominance* relations that represent the two alternatives for the VP in (8) above are:

$$\textit{Immediate Dominance}_1 = \left\{ \begin{array}{l} \langle VP, V\rangle, \langle VP, \bar{V}\rangle, \langle \bar{V}, NP\rangle, \langle \bar{V}, V\rangle, \\ \langle NP, \bar{N}_1\rangle, \langle \bar{N}_1, \bar{N}_2\rangle, \langle \bar{N}_2, N\rangle, \langle \bar{N}_1, PP\rangle \end{array} \right\}$$

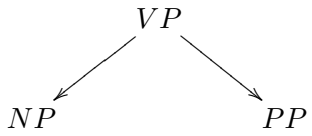


$$\textit{Immediate Dominance}_2 = \left\{ \begin{array}{l} \langle VP, V\rangle, \langle VP, \bar{V}\rangle, \langle \bar{V}, NP\rangle, \langle \bar{V}, V\rangle, \\ \langle \bar{V}, PP\rangle, \langle NP, \bar{N}\rangle, \langle \bar{N}, N\rangle \end{array} \right\}$$

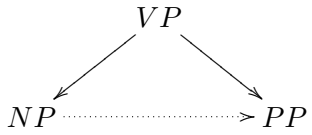


The following dominance relation can be extended to represent both options:

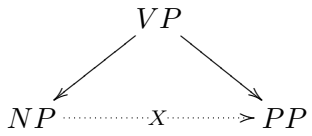
$$\textit{Dominance} = \{\langle VP, V\rangle, \langle VP, NP\rangle, \langle VP, PP\rangle\}$$



To rule out the second alternative, another pair would be added to the dominance relation: $\langle NP, PP\rangle$:



And to rule out the first alternative, it would be made explicit that $\langle NP, PP\rangle$ does not hold:



While this is not the underspecified structure I will use in my theory, it represents efficiently how underspecified structure can be used to solve the modality puzzle. Since I would like to solve this problem in my parser, I will formulate an alternative that builds on this idea.

2.1.6 Deterministic Parsing and Assertion Sets

We are already familiar with ambiguous sentences, sentences which may be represented by two distinct structures. Sentences can also be *locally* ambiguous. For instance, the grammar in (10) cannot give rise to more than one interpretation for any sentence, but sentence prefixes can be ambiguous. (11.a) can be the prefix of either (11.b) or (11.c). This does not allow parsers to parse the sentence incrementally one word at a time, and should either postpone some decisions or allow the parser to undo choices it already made.

(10) (a) $S \rightarrow A$

(b) $S \rightarrow CBB$

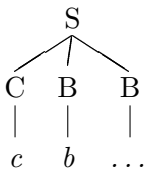
(c) $A \rightarrow CBA$

(d) $A \rightarrow a$

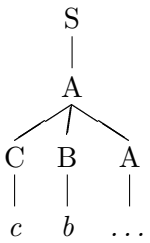
(e) $B \rightarrow b$

(f) $C \rightarrow c$

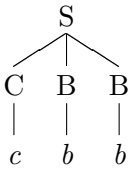
(11) (a) c b



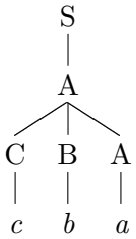
or



(b) c b b



(c) c b a



As I have mentioned before, a parser that deals with these situations must employ some strategy for such cases. Marcus (1978) presents the following strategies: explore all the alternatives using either *backtracking* or *pseudo-parallelism*, or postpone the decision making until enough information is gathered, using what some theories label as *look-ahead* in (11.a); this would mean waiting for the next input symbol), and commit itself to every decision made. Marcus uses the term 'deterministic parsing' for this strategy.

Parallelism in an ideal parser, processing a sentence one word at a time, would simply mean that whenever there is more than one option in a given situations, the parser 'splits' to two (or more) identical parsers, each following another option. Should any instance of such a parser reach the end of the sentence and not have a proper parse tree, that instance would terminate, and its incomplete parse purged. *Pseudo*parallelism in such an environment simply means that the ideal behaviour is simulated by copying the state and storage of the parser before the split occurs, and alternate between the various instances every step. This option is very wasteful in terms of processing 'time', and storage size (which grows exponentially), but some compact implementations of it manage to effectively reduce processing time (as in Tomita (1987). Backtracking, in essence not very different from parallelism, means that rather than follow each and every path, the parser picks some arbitrary option, and follows it until it fails, in which case it traces back its actions to the place where it last picked an option, and tries the alternatives one by one, until a valid parse for the entire sentence is encountered. Its advantage over any sort of parallelism is that it more efficient if the order in which each option is taken is calibrated to the most common cases. Even in the case of a statistically tuned parser, the result may not necessarily be the most "statistically plausible" parse, as it will reflect only the earliest valid choice. Both of these strategies do not correlate with the time complexity constraint I chose above, and do not match the behaviour of the human parser.

Even though the parallel and backtracking parsers are too strong to describe the human parser, Marcus's deterministic class of parsers are not the only alternative to those parsers. Limited backtracking parser are parsers in which some backtracking mechanism is implemented, but every decision does limit the set of alternatives. These two sets of parsers are obviously not disjoint: a parser with a finite look-ahead of three words is as powerful as a limited backtracking parser with ability to undo the last three decisions made. While every parser with finite lookahead can be described in terms of limited backtracking, limited

backtracking parsers can be allowed to manage operations more powerful than undoing a finite number of decisions.

In an attempt to widen Marcus's (1978) determinism following D-theoretic work (Marcus et al., 1983; Barton and Berwick, 1985; Weinberg, 1993), but refraining from the too powerful alternatives of backtracking, the very term 'determinism' has been formalized as follows (from Barton and Berwick (1985)):

> An assertion set parser develops its analysis *deterministically* if changes in its (global) assertion set are always *refinements* in the information-theoretic sense - that is if information is monotonically preserved.

This description of parsing is in terms of information growth: in the initial state we have asserted nothing: every possibility can be entertained. As information is encountered, assertions about the outcome are made and some options are made impossible, until at the final stage we remain with only the valid possibilities [7], but at no time can an assertion be withdrawn or modified. Determinism in this sense does not by itself guarantee online parsing: a choice can be put into the assertion set after an unbounded amount of information has been made available, though this would usually mean that we have been 'cheating' by evaluating alternatives outside the assertion set: in essence modifying a temporary assertion set. We can avoid this problem by making sure we do indeed have just one assertion set. This is the path I will follow in my analysis: a deterministic parser with a single assertion set.

[7] Note that this can be easily be used to explain ambiguities: at the end of the possibility elimination process, we have more than one valid possibility.

2.2 Formal Layout

2.2.1 Dominance Orders and Assertion Sets

Rooted trees, the basic notion used in linguistics to describe syntactic and semantic representations is a terms borrowed from graph theory, where a tree is a subtype of a graph. I will start by defining graphs, directed graphs, paths and cycles, notions needed for the computer science definition of rooted trees, and finish with the definition of tree.

Graphs are used to describe a set of basic elements and a set of pairs of these basic elements. We call the set of basic elements 'vertices', and the set of pairs 'edges'. The set of basic elements may be a set of people, and the set of edges may signify the 'know each other well' relation. In the basic definition, the pairwise relation is symmetric: we cannot use a basic graph to describe the set of people and the 'has heard of' relation.

Definition 12. *A* graph *is a pair* $G = \langle V^G, E^G \rangle$ *such that:*

1. V *is a set of vertices.*
2. $E \subseteq V \cdot V$ *is a set of edges:* $E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V\}$.
   We will use the shorthand $v_i v_j$ *rather than* $\{v_i, v_j\}$.

Directed Graphs are used to describe just those cases in which the relation we wish to describe is not symmetric, such as 'has heard of'. In this case, the set of edges will contain ordered pairs, rather than simple pairs, so that we can correctly indicate that someone may have heard of someone else, but not the other way around. We will use a function called *indirect* to move us from a directed graph into its indirect counterpart. Naturally, this is not a *bijective* function, as different directed graphs may map into the same indirect graph. For instance: the graph made of the set of all people and the 'saw' relation, and the graph made of the same set of people and the 'was seen by' relation, map to the same indirect graph: the set of all people and the 'saw or was seen by' relation.

Definition 13. *A* directed graph *is a pair* $G = \langle V^G, E^G \rangle$ *such that:*

1. V *is a set of vertices.*
2. $E \subseteq V \times V$ *is a set of edges.*
3. *We define a surjective function from a directed graph to its undirected counterpart:* $\mathit{indirect}(G) = \langle V^G, \{\{v_i, v_j\} : \langle v_i, v_j \rangle \in E^G \vee \langle v_j, v_i \rangle \in E^G\} \rangle$

Paths are used to denote a recursive application of the relation denoted by the set of edges. We would say that there is a path between two vertices if they are connected to one another by a set of edges. This can describe how to get from some place to another using a simple map: if the vertices are the set of intersections, and the edges are the streets, you can get from one intersection to another by going through a street to get to some other intersection and so forth.

Definition 14. *A* path *between a and b is a non-empty graph $P = \langle V^P, E^P \rangle$ such that:*

1. $V^P = \{v_0, v_1, \ldots, v_k\}$ *such that* $\forall i \neq j \in \{1..k\} . v_i \neq v_j \wedge (v_0 = a) \wedge (v_k = b)$
2. $E^P = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\}$

If we care only for *whether* there is a path from one vertex to another, we can use a recursive notation, which I find more intuitive:
There is a path between vertex v and vertex u if and only if they share an edge, or there is some vertex w such that there is a path between v and w, and there is a path between w and u.

Directed Paths are used in directed graphs, where between two vertices v and u there may be and edge from v to u, but not the other way around. The obvious example is the one used above, except you use a car, and some streets are one-way.

Definition 15. *A* directed path *between a and b is a non-empty directed graph $P = \langle V^P, E^P \rangle$ such that:*

1. $V^P = \{v_0, v_1, \ldots, v_k\}$ *such that* $\forall i \neq j \in \{1..k\} . v_i \neq v_j \wedge (v_0 = a) \wedge (v_k = b)$
2. $E^P = \{\langle v_0, v_1 \rangle, \langle v_1, v_2 \rangle, \ldots, \langle v_{k-1}, v_k \rangle\}$
3. *We will use the notion $a \rightsquigarrow b$ to say 'there is a path from a to b.*

Here too, it is simpler to define a directed path if we care only for *whether* there is a directed path from one vertex to another. The recursive definition would be very similar: There is a directed path from vertex v to vertex u if and only if there is edge from v to u, or there is some vertex w such that there is a path from v to w, and there is a path from w to u.

Cycles are path that start and end at the same vertex.

Definition 16. *A* cycle *is a non-empty graph* $C = \langle V^C, E^C \rangle$ *such that:*

1. $V^C = \{v_0, v_1, \ldots, v_{k-1}\}$ *such that* $\forall i \neq j \in \{1 \ldots k-1\} . v_i \neq v_j$
2. $E^C = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_0\}$

Rooted Trees are what we have been trying to get to all along: these are directed graphs that have a root: some vertex from which there is a path to every other vertex, and contain no undirected cycles: in trees the path from the root to every vertex is unique. Rooted trees can be used to describe the trees we are familiar with from syntax: the vertices are the heads, intermediate and maximal projections, and the set of edges would be the 'immediately dominates' relation.

Definition 17. *A* rooted tree *is a directed graph* $T = \langle V^T, E^T \rangle$ *such that:*

1. *There is a directed path from some vertex in* V^T *to every other vertex in* V^T*:* $\exists v_0 \in V^T . \forall v \in V^T \setminus \{v_0\} . v_0 \rightsquigarrow v$
2. *There are no cycles in indirect* (T)*.*

While graph theory could be sufficient to describe linguistic structures, I find it easier to deal with them in some other form. I define tree structures as relations: *dominance relations*, which are maintained in structures that represent the assertion sets defined in Barton and Berwick (1985) (see §2.1.6). I will start by defining the most basic elements, dominance orders and the assertion sets that can contain them, and then prove that dominance orders can represent trees. I will use this machinery both to describe my own parser, but also to formalize some of the previous work, especially that of D-Theoretic attempts.

- **Elements**: I use the term 'elements' to be theory neutral, as what the basic elements actually are differs between theories. In generative syntax, the basic elements would include heads, intermediate and maximal projections. In my theory the basic elements would be either morphological particles, words or even complete phonological phrases. The difference lies in choosing what is allowed to dominate what: in generative syntax heads are not allowed to dominate other heads, and so there is a

need for basic elements which are not strictly part of the input. In my theory I do not assume projections of any kind and so the basic elements are usually chunks of input: the size of the chunk may differ between implementations and languages.

Definition 18. *M is a non empty finite set: the set of all elements or constituents.*

- **Dominance Orders**:
  - As I will prove below, *dominance orders* can be used to substitute the graph theoretic notion of trees. I find them easier to manipulate and maintain. We should think of dominance orders as the dominance relation in syntactic or semantic trees. Here too, the dominance relations differ between theories. In generative syntax an element would be dominated by another element if and only if it is the daughter of that element, or one of its daughters, recursively. In my theory, I only use semantic structure, and an element is dominated by another element if and only if it modifies it, or modifies one of its modifiers, recursively.

    Definition 19. *A* dominance order *D over a set M is a relation $D \subseteq M \times M$ such that:*
    - *D is a partial order (reflexive, antisymmetric and transitive)*
    - *D is non branching to the past:*
      $\forall \mu_1, \mu_2, \mu_3 \in C. [(\mu_1 D \mu_3 \wedge \mu_2 D \mu_3) \rightarrow (\mu_1 D \mu_2) \vee \mu_2 D \mu_1)]$
    - *D has a minimal element: there is a $\mu_0 \in T$ such that:*
      $\forall \mu \in C. [\mu_0 D \mu]$
  - While the general structures I use are dominance orders, *immediate dominance* can be defined as the intransitive counterpart of dominance orders: all we have to do is drop the recursive part of the definitions offered above. An element is immediately dominated by another element in generative syntax if and only if it is its daughter. Likewise, an element is immediately dominated by another element in my theory if it modifies it.

    Definition 20. *For every dominance order D, I^D is its* immediate *equivalent, an* immediate dominance order *such that:*
    $I^D = \left\{ \langle \mu_1, \mu_2 \rangle \in M^2 : (\mu_1 D \mu_2) \wedge \neg (\mu_2 D \mu_1) \wedge \neg \exists \mu \in M. [\mu_1 D \mu D \mu_2] \right\}$

– From these definitions we can demonstrate the relation between dominance orders and the rooted trees we are already familiar with.

Claim 21. *The pair* $\langle M, I^D \rangle$*, where* M *is a set of elements and* D *is a dominance relation, is a rooted tree.*

This follows from the definition of D: the root is the minimal element, and since D is non-branching to the past, I^D cannot have cycles in it.

- **Assertion Sets**:

– $\mathfrak{R}$ is the set of possible relations based on M, a union of all n place relations based on M. This definition is general enough to include more than 2-place relations. In practice, we can regard $\mathfrak{R}$ as $Pow(M \times M)$

Definition 22. $\mathfrak{R}$ *is the set of all relations based on* M*:*
$\mathfrak{R} = \lambda R.\exists n \in \mathbb{N}.R \subseteq M^n$

– A proposition is a property of relations: a set of relations. Using this terminology, being transitive is a proposition: the set of transitive relations.

Definition 23. *A* proposition p *is a subset of* $\mathfrak{R}$

– An assertion set is a mapping between labels and propositions. This would allow us to trace changes to multiple relations: applying the assertion set function to some label that represents the relation will provide us with the set of all the relations that relation can still be, based on the information we currently hold.

Definition 24. *An* assertion set *AS is a function from labels to propositions.*

– Refinements represent the way our knowledge grows: if one assertion set is a refinement of another assertion set, it means that we know *more* about all the relations contained in that assertion set: the set of possible relations (the proposition) related to every label has either narrowed or remained the same.

Definition 25. *An assertion set* AS' *is a* refinement *of an assertion set AS if and only if:*
$\forall l \in LABELS.\,[AS'(l) \subseteq AS(l)]$

This machinery can be used to demonstrate how information gained during a parsing process is narrowed down from a set of possible trees to the set of legitimate trees, and

if the sentence is unambiguous - hopefully just a single tree. The assertion sets discussed above can describe the information growth of any relations, not just dominance orders. We can therefore begin not with an empty assertion set, but rather one which already presupposes that some label is associated with a dominance order. This is the situation described below at the first stage of example (26). As we move from one stage to another information is added, the assertion set is refined, and the set of possible dominance orders is narrowed down until (in this case) we have a single possible dominance order.

(26) If T has three elements: a, b and c, and we begin knowing that there is some relation D that is the 'correct' dominance order of T, our assumption set would start off as something like this:

$$AS_1(l) = \left\{ \begin{array}{ll} \text{transitive} \cap \text{reflexive} \cap \text{antisymmetric} \cap \text{NBttP} \cap \text{ME} & \text{when } l = D \\ \mathfrak{R} & otherwise \end{array} \right\}$$

where:

$$\begin{aligned} \text{transitive} &= \lambda R.\forall x, y, z \in T.\, [(xRy \wedge yRz) \rightarrow (xRz)] \\ \text{reflexive} &= \lambda R.\forall t \in T.\, [tRt] \\ \text{antisymmetric} &= \lambda R.\forall x, y \in T.\, [(xRy \wedge yRx) \rightarrow (x = y)] \\ \text{NBttP (non-branching to the past)} &= \lambda R.\forall x, y, z \in T.\, [(xRy \wedge zRy) \rightarrow (xRz \vee zRx)] \\ \text{ME (minimal element)} &= \lambda R.\exists t_0 in T.\forall x \in T.\, [\mu_0 T x] \end{aligned}$$

At this stage, D can be any one of the possible dominance orders on three elements. If we now learn that aDb, D would be limited to the set of dominance orders in which aRb, and our assumption set would be refined to the following:

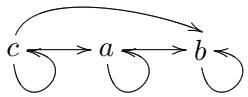
$$AS_2(l) = \left\{ \begin{array}{ll} AS_1(D) \cap (\lambda R.aRb) & \text{when } l = D \\ AS_1(l) \cap \mathfrak{R} & otherwise \end{array} \right\}$$

Which means that:

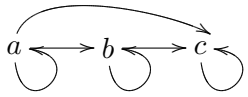
$$AS_2(D) = \left\{ \begin{array}{l} \{\langle a,a\rangle, \langle a,b\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle c,a\rangle, \langle c,b\rangle\}, \\ \{\langle a,a\rangle, \langle a,b\rangle, \langle a,c\rangle, \langle b,b\rangle, \langle b,c\rangle, \langle c,c\rangle\}, \\ \{\langle a,a\rangle, \langle a,b\rangle, \langle a,c\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle c,b\rangle\}, \\ \{\langle a,a\rangle, \langle a,b\rangle, \langle a,c\rangle, \langle b,b\rangle, \langle c,c\rangle\} \end{array} \right\}$$

And D would be either of the following:

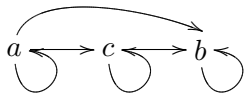
1.



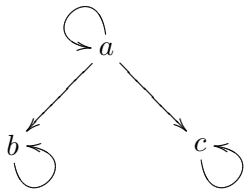
2.



3.



4.

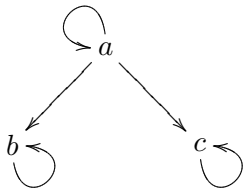


If further information reveals that $\neg bDc$ (which rules out 2) and $\neg cDb$ (which rules out both 1 and 3). Our assumption set is therefore refined to the following:

$$AS_3(l) = \left\{ \begin{array}{ll} AS_2(D) \cap (\lambda R.\neg bRc) \cap (\lambda R.\neg cRb) & \text{when } l = D \\ AS_2(l) \cap \mathfrak{R} & otherwise \end{array} \right\}$$

Note that now there is only one possible relation D, as $AS_3(D)$ contains only one element:

$$AS_3(D) = \{\{\langle a,a\rangle, \langle a,b\rangle, \langle a,c\rangle, \langle b,b\rangle, \langle c,c\rangle\}\}$$



2.2.2 Parsers

In this paragraph I will lay the basic machinery to describe all parsers I have mentioned so far, the ones I will describe in §3, and my parser. Since my parser is not strictly a parser as briefly mentioned in §2.1.5, the framework will allow me discuss 'less than parsers' as well as parsers using the same terms. The basic layout is that of not necessarily monotonous information growth, which builds relations representing the composition of the sentence.

Parse States are the basic building block of parsers: they represent a stage in the parse process of sentences. Different parsers will be formalized by the manner they move from one parse state to another. The definition of parse states consists of some subset of elements the parse state claims something about, an assertion set that represents the information the parser holds in the state, and a labeling partial function that I use to set apart actual input elements (to which the labeling function assigns a value) from non-input elements (which have no such value).

Definition 27. *A* Parse State *s is a triplet $\langle M^s, AS^s, l^s\rangle$ such that:*

- *M^s, the set of elements about which something is asserted is a subset of M (the set of all nodes).*
- *AS^s is a an assertion set such that: there is some α in the set of labels such that $AS(\alpha)$ is a set of dominance orders.*
- *l^s is a partial function from M the set of element to LEX the set of lexical items such that $\forall \mu \in M \setminus M^s . l^s(\mu) = \perp$*
  We will also define L_s, the set of elements to which l^s assigns a value and the value it assigns, such that:
  $L_s = \{\langle \mu, \alpha\rangle : (\langle \mu, \alpha\rangle \in l^s) \wedge (\alpha \neq \perp)\}$

We will use the symbol $\mathbb{S}$ for the set of all parse states.

The parse state s_1 of a parser that has just read the input elements 'John' and 'ate' might be:

- $M^{s_1} = \{a, b, IP, NP, VP, V\}$
- $l^{s_1}(\mu) = \left\{ \begin{array}{ll} \text{John} & \mu = a \\ \text{ate} & \mu = b \\ \perp & otherwise \end{array} \right\}$
- $L^{s_1} = \{\langle \text{John}, a\rangle, \langle \text{ate}, b\rangle\}$
- If D is the dominance order we build then:
  $AS^{s_1}(D) = \lambda R. \{\langle IP, NP\rangle, \langle IP, VP\rangle, \langle VP, V\rangle, \langle NP, a\rangle, \langle NP, b\rangle\} \subseteq R$

Parse Processes represent the possible ways for a parser to move from one parse state to another. In order to do that, I define parse processes as growth from some null parse state in which no input elements have been read (that is, the L^s of the initial state is empty) to some set of result states. The growth is represented by a strict partial order relation, so it would be possible to represent parsers in which different alternatives can be entertained. The set of parse states ordered by that relation is finite, as the input is finite and parsers are expected to stop at one stage or another.

Definition 28. *A* parse process P *is a triplet* $\langle S_P, s_0, \prec_P \rangle$ *such that:*

- $S_P \subsetneq \mathbb{S}$ *is a finite set of parse states*
- s_0 *is a special parse state such that* $s_0 \in S$ *and* $L_{s_0} = \emptyset$
- $\prec_P$ *is a strict partial order over* S *such that*
  - $\forall s \in S_P \setminus \{s_0\} \,.\, (s_0 \prec_P s)$
  - $\forall s_i, s_j \in S_P . (s_i \prec_P s_j) \rightarrow \left(L_{s_i} \subseteq L_{s_j}\right)$

Parsers are therefore simply set of parse processes. The only requirement is that for these different parse processes the parse states they contain will be consistent with regard to the label to which the final dominance order corresponds.

Definition 29. *A* parser $\mathbb{P}$ *is a set of parse processes such that there is some label* $\alpha^{\mathbb{P}}$ *such that in every parse state* s *in every parse process* P *in* $\mathbb{P}$, $AS^s\left(\alpha^{\mathbb{P}}\right)$ *is a set of dominance orders.*

On-line Parsers can now be formalize. I use the fuzzy term 'on-line parser' to formulate parsers that cannot maintain multiple information states at once: when information is encountered it is incorporated in some unspecified way. This rules out parallel parsers, but not parsers with unrestricted backtracking.

Definition 30. *An* on-line parser *is a parser* $\mathbb{P}$ *where in every state* s *of every parse process* P, $\prec_P$ *is a (strict) linear order.*

Strictly On-line Parsers are a closer swing at what is defined as 'on-line' in several theories (usually theories with backtracking, but excluding parsers with look-ahead). I define strictly on-line parsers as parsers that do not allow more than one possible tree at any given parse state.

Definition 31. *A* strictly on-line parser *is an on-line parser* $\mathbb{P}$ *such that the set of dominance orders over* M^s*,* $AS^s\left(\alpha^{\mathbb{P}}\right) \cap Pow\left(M^s \times M^s\right)$ *contains a single relation*
or : $\forall P \in \mathbb{P}.\forall s \in S^P.\left|AS^s\left(\alpha^{\mathbb{P}}\right) \cap Pow\left(M^s \times M^s\right)\right| = 1$

Deterministic Parsers are parsers in which the information about the target state grows from one parse state to another. The definition is therefore rather simple (and follows Barton and Berwick (1985)):

Definition 32. *A* deterministic parser *is a an on-line parser* $\mathbb{P}$ *such that: If* $s_i \prec_P s_j$ *then* AS^{s_j} *is a refinement of* AS^{s_i}*.*
or: $\forall P \in \mathbb{P}.\forall s_i, s_j \in S^P.\left[\left(s_i \prec_P s_j\right) \rightarrow \left(\forall \alpha.\left[AS^{s_j}\left(\alpha\right) \subseteq AS^{s_i}\left(\alpha\right)\right]\right)\right]$

3 Previous Work

3.1 Limited Backtracking Parsers

3.1.1 Pritchett 1992

Pritchett's (1992) work is one of the most celebrated works in syntax-driven explanations to garden path phenomena. While these are not the mainstream approaches to handling garden path, they are grounded in theoretical work, unlike the main alternatives which focus on psycholinguistic reality. Pritchett's parser follows the following guidelines:

- The parser tries to build a syntactic tree[8].
- The parser does not start building structure until the first predicate specifying a theta grid (a verb) is encountered.
- Once a predicate is encountered, every new element is attached into the syntactic tree, assuming a maximal theta grid, and trying to satisfy every theta role. From this requirement it follows that if a verb is ambiguous between transitive and intransitive forms, the transitive form will be assumed.
- The tree need not grow in a monotonic way: reanalysis is possible. However, a reanalysis process can be *costly* or not. A costly reanalysis gives rise to garden path, while normal reanalysis is not noticed. Pritchett considers re-analysis to be a movement of constituents: if an NP was first attached as the object of some verb, and ended up as the subject of the following sentence, the tree position it first occupied is considered the *source* position, and the tree position it ended up attached to is considered the *target* position. We can now define a costly reanalysis in Pritchett: a reanalysis is costly if it violates the OLLC condition (mentioned in §2.1.4):

  > "The target position (if any) assumed by a constituent must be *governed* or *dominated* by its source position (if any), otherwise attachment is impossible for the automatic Human Sentence Processor"

  I will translate these terms to less theory specific terms:

  > The source position must dominate the target position, or be an argument of the first maximal projection (xP) dominating the target.

[8] Most parser try to build a syntactic tree. I note that because mine doesn't.

If we try to formalize Pritchett's theory, using the framework laid in §2.2.2, we can see that Pritchett's parser $\mathbb{P}_{Pritchett}$ is not a strictly on-line parser, as it waits for the first predicate before starting to build structure, but it can be assumed to be strictly on-line *after* the first predicate is encountered. In addition, we can assume there is some further restriction of parse processes that requires them not to violate the OLLC.

Pritchett uses these principles to explain a few kinds of garden path sentences (summed up in (Mulders, 2002)):

1. Main Clause - Relative NP Ambiguity

   (33) (a) The boat floated down the river.

   (b) ¿The boat floated down the river sank.

   This is the ambiguity that gives rise to the most famous garden path sentences. We can see how his principles make (33.b) a garden path sentence: when 'floated' is encountered, one of the two interpretations has to be taken, namely the one where it is the verb of the matrix clause, analyzing 'boat' as a part of the NP subject of 'floated'. When the second verb is reached, reanalysis places the target position more than two maximal projections below the source position. In (33.a), no second verb is reached, so reanalysis is not required, correctly predicting the sentence not to be a garden path sentence.

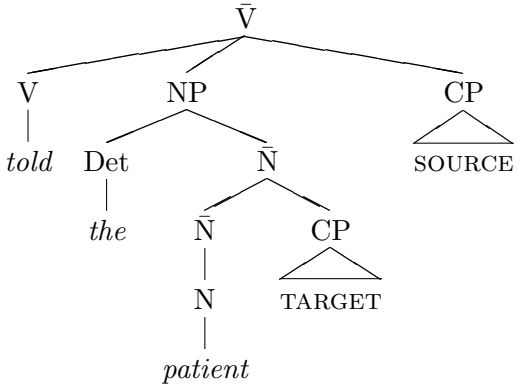
2. Complement Clause - Relative Clause Ambiguity

   (34) (a) The tourist persuaded the guide that they were having problems with their feet.

   (b) ¿The doctor told the patient that he was having trouble with to leave.

   In (34.b), when we get to 'that' we have a local ambiguity between a complement clause and a relative clause. Since we try to maximize theta attachment, we choose the complement clause option over the relative clause interpretation. When we reach 'to leave' we are forced to reanalyzed the sentence, pushing the sentential complement back to the second argument of the verb, a position separated by two maximal projections from the source position, making the reanalysis expensive, and hence causing a garden path. In (34.a) the assumption made in the complement clause / relative clause ambiguity is correct: this is indeed a complement clause, and

so no reanalysis is required, and the sentence is not a garden path.



3. Object-Subject Ambiguity

   (35) (a) John believed the ugly little man hated him.

   (b) ¿After Susan drank the water evaporated.

   In (35.b), 'the water' is first attached to 'drank' as the parser assumes a maximal theta grid. When 'evaporated' appears, 'the water' is reanalyzed as the subject of the matrix clause, and separated by a few maximal projections (PP, IP, VP) from its source position. In (35.a), a re-interpretation does occur: we first interpret 'the ugly little man' as the object of the verb, but a re-analysis is not costly, as it does not violate the OLLC.

4. Double Object Ambiguity

   (36) (a) Rex gave her presents to Ron.

   (b) ¿Todd gave the boy the dog bit a bandage.

   In (36.b) 'the dog' is pushed down from being the theme of 'gave' to the subject of a CP modifying the dative, separating the target position by more than two maximal projections (NP, CP, IP) from the source position

If we try to expand Pritchett's theory to other processing related phenomena, we see that it falls short of explaining them.

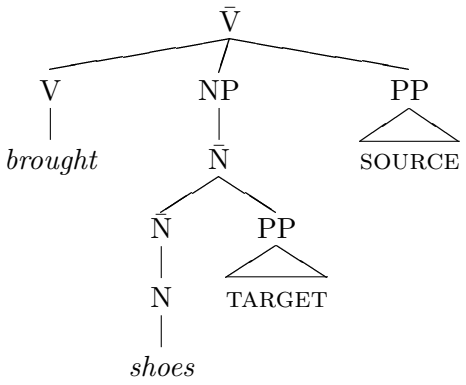
- **Right Association**: Pritchett's parser should not fail in analyzing sentences in which no reanalysis is required. Since processing difficulties in sentence such as (6.a), repeated here as (37) are not the result of such a process, Pritchett's explanation cannot deal with them.

(37) John will tell the kids he has already bought them a dog tomorrow.

- **Modularity**: Pritchett's parser contains no mechanisms that would allow it to provide more than one analysis for a given sentence, even if such different parse trees exist. Trying to check how the theory deals with (8), repeated here as (38.a), shows that Pritchett predicts an analysis in which 'from Italy' is attached as an argument of the verb rather than as a modifier of the noun, as the first option better satisfies the theta criterion. However, if the 'real' source role is revealed at the end later, as in (38.b), which is not (at least in Hebrew) difficult to process, Pritchett's theory is revealed to be too strong: 'from Italy' is 'pushed' under an intervening NP maximal projection and the reanalysis is wrongly predicted to yield a garden path.

(38) (a) John has brought shoes from Italy

(b) John has brought shoes from Italy from work.



- **Complexity**: Pritchett's parser cannot parse in linear time , not only because he does not want to mess with these notions, but because maximizing theta attachment requires the parser to check, for every new phrase, all the predicates to which a new argument can still be attached; since there can be $O(n)$ of these elements, and this procedure is carried out for every new element, and there are $O(n)$ such elements, the algorithm cannot drop below $O(n^2)$. It is also unclear how it can manage to evaluate what error led to the wrong structure, and how a new structure is to be built. An exhaustive search of wrong decisions can require undoing every decision and trying again until proved wrong. If every structure building takes $O(n^2)$, and a reanalysis of m elements takes $O(m^2)$, we may reach a worst case time complexity of $O(n^4)$. Even though on average the parser will probably not exceed $O(n^2)$, one may wonder whether assuming the parser to be able to evaluate such complex restructuring, does not assign it more power than is plausible.

- **Monotonic Growth**: Pritchett's parser allows reanalysis and therefore does not grown in a monotonic way.

3.1.2 Schneider 1999

Schneider (1999) uses underspecification to allow his parser more flexibility, but allows it to backtrack in certain cases. His basic framework is very similar to that of generative syntax, but extended with the possibility of not fully specifying features. Like Pritchett, he distinguishes between "cheap" and "costly" structure modifications. "costly" modifications result in a garden path, while "cheap" ones do not. Unlike Pritchett and following Frazier and Fodor (1978), he does not consider 'cheap' reanalysis to be free.

Schneider's (1999) parser is designed as a bottom-up parser. This means that the parser does not assert, for instance, that a sentence is being built, and therefore some VP node should be created, but rather builds the VP, as the evidence that calls for it (i.e. the verb) is encountered. Functional categories are created as a manifestation of a feature: a noun carrying dative case 'expects' a verb to assign it, a noun carrying nominative case needs an IP that would assign it. By disassembling linguistic properties into features, the parser is allowed to predict heads in head-final languages without being forced to change them later: dative case may be assigned by a post-position as well as a verb, so the parser would only predict a head that assigns the dative case, without actually specifying whether this head is a verb of a post-position. Thus, the parser manages to be a strictly online parser: every bit of information is integrated into the tree as soon as it is encountered.

The parser has the following preferences, modeling psycholinguistic data:

1. Attach as complement along the right edge of the tree (prefer recent attachments sites).

2. Attach as an adjunct along the right edge of the tree (prefer recent attachments sites).

3. Predict the head that would integrate the new element to the tree.

4. If these options fail, choose the first attachment site that can attach the new element (and / or its predicted head), detach its current attachment, incorporate it into the exiting material, and attach the combined element.

5. Fail the process (resort to other cognitive mechanism).

The first three preferences are kept in Pritchett's model as well, but the fourth one is designed to explain a phenomenon I do not explain in my parser (though it is not impossible to do so): Frazier and Fodor's (1978) RALR: reanalysis as a last resort. This means that in sentences where reanalysis is required in the local scope (this is also not quite a theory neutral term), and a solution exists in the global scope (for instance, via attachment to a higher verb), there would be a preference to resolve the problem in the global scope rather than perform a reanalysis.

From the first four attachment principles follows the fifth: garden paths occur if no attachment site can be found, not even by employing a 'legitimate' reanalysis. This means that the backtracking capabilities of the parser are limited to re-attaching elements which are *at the right edge* of the tree. Reanalyzing a sentence as a reduced relative for instance, would be an operation that does not involve the right edge only. Schneider's parser would be able to reanalyze (38.b), as re-attaching a PP to an NP which is on the right edge, once the PP is removed from its original attachment site, is possible.

If we review the other requirements we made for parser, we can see that Schneider's parser does not meet most of our desiderata:

- **Right Association**: Schneider's explanation for right association lies in the compatibility with psycho-linguistic findings. The parser scans for attachment sites bottom-up, and if it would give up on an attachment site, another might not present itself later. This, combined with a modular view of language, would allow the parser to attach an element to an improper attachment site, even if this means giving up the option to find the best attachment site: complements would always be attached to the last verb if possible, even if this leaves another verb missing an obligatory complement. The same goes for adverbials, which may be attached to semantically improper verbs.

- **Modularity**: As mentioned above, dealing with ambiguity would require the parser to preserve more than one 'state', as it builds a fully specified syntactic tree. The modular view that saves Schneider's parser from managing to interpret right association sentences properly, does not allow it to build more than one interpretation at

a time: only one structure is maintained, and underspecification is not used to hold ambiguities.

- **Complexity**: The parser reviews the entire tree depth for every operation that requires building a new head. Since there are $O(n)$ such operations when processing a sentence, the cumulative complexity is $O(n^2)$, provided that no reanalysis is performed. To keep the compatibility with psycho-linguistic evidence, which do not support non linear growth of processing time, Schneider would have to assume a parallel computational model; the model is apparently compatible with such enhancements.
- **Monotonic Growth**: Not kept. Reanalysis is possible.

3.1.3 Lewis 1998

Lewis' (1998) work tries to describe the human parser in terms of a computational framework: NL-Soar. This framework does not allow more than one computational state to be 'saved', and as such does not allow 'real' backtracking nor parallel processing, and actually requires the parser to be a strictly on-line parser. To solve erroneous decisions made, due to local ambiguities, Lewis introduces a structure destroying operation that he labels SNIP, and uses it, together with his structure building operation LINK, to describe how structure is built, and when it should be destroyed.

Lewis is not unaware of the appeals of deterministic growth of underspecified structure, but claims that such a procedure cannot be claimed to be more plausible than building and correcting a structure, since the actual structure implied by underspecification is a syntactic structure which has to be computed by the semantic module - which he finds odd[9] - and the immediate dominance relation grows in a non monotonic way[10]. This also means that the model he proposed cannot maintain ambiguities, as they would require a multiple final tree representations to be contained in the structure he builds.

Since structure rebuilding takes time, Lewis limits the kinds of structure SNIP can destroy to that of the maximal projection containing the inconsistency. This means that his SNIP operation should be able to correct structures which turn out to be garden paths,

[9] This is only odd if you assume the parser builds a syntactic representation.

[10] I show this is not alway true in §4.1.2.

such as (34b) repeated here as (39), since the inconsistency is contained within the VP to which the relative clause was attached as an argument to the verb.

(39) ¿The doctor told the patient that he was having trouble with to leave.

Lewis' model allows the structure that is built to be contained in the memory of the parser[11], and does not limit the possible applications of his LINK operations, thus allowing every attachment to take place, if there is a predicate that can take an element as its argument. This means that *deciding* which operation should take place takes well more than $O(n)$; however, as he is also aware that the entire parse process should take a finite number of steps, he only counts the actual number of applications of operations, i.e. how many times LINK and SNIP are used. While this is less efficient than what I desire to achieve, in other computational models (our mind may offer one) this may mean that $O(1)$ can still be achieved. Note that any model that manages to drop the $O(n)$ space requirement for an $O(1)$ storage requirement, such as the model I propose in §4.3.2, will be more efficient in any computational model.

Lewis' parser can be summarized as having the following features:

- **Right Association**: In Lewis' model, processing problems of the 'right association' type can only happen if LINK is constrained. Since Lewis' deals more with the constraints of SNIP, this is not achieved. However, we can come up with a Lewis-like parser which would limit both SNIP and LINK, possibly creating some basis for dealing with such sentences. Such an attempt would be rather ad-hoc, since Lewis does not propose that LINK operations grow more complex with more input, even though they hold more storage and more decisions can be made.

- **Modularity**: As mentioned above, dealing with ambiguity would require the parser to preserve more than one 'state', as it builds a fully specified syntactic tree. Lewis opposes that possibility for other reasons, which mean that the parser follows just one path. It is of course possible to stipulate that, given a choice of LINK decisions, semantics can play a role in deciding which one to choose; however, since those very LINK decisions can be undone by future SNIP operations, it is rather unclear when the semantic intervention should take place. A possible alternative to that may

[11] This makes his claim that this is derived by memory constraints somewhat implausible.

be found in Weinberg's (1999) parser, in which the parser only commits itself to some piece of structure when a merge operation fails: semantics may be allowed to re-interpret uncommitted structure only.

- **Complexity**: The complexity depends on the reader's view of Lewis' analysis. The parser operates in linear time, but apparently not by using the RAM computational model. Lewis views his LINK operations as something which can be carried out in a fixed time frame, but does not go into detail for how should the alternatives of LINK be considered.
- **Monotonic Growth**: Lewis claims this is unnecessary.

3.2 Deterministic Parsers

3.2.1 Marcus 1978

Marcus (1978) is not concerned with providing explanations for garden path related phenomena, but rather with providing a more efficient approach to parsing. Marcus compares his work to that of backtracking and parallel parsers, and demonstrates how well a rather simple machine working in $O(n)$ can parse (though he does not say that he aims to reach linear complexity). As I have mentioned above in my critique of Pritchett's work in §3.1.1, a backtracking parser might be very inefficient. Parallel parsers - another alternative prominent at the time of Marcus' work - is just as complex and memory consuming. This may have changed with time, but even Tomita's (1987) parallel parser is a great deal more complex that any linear parser such as Marcus'. Instead, Marcus suggests a parser which immediately incorporates data it encounters into the tree, and does not make any mistakes (and therefore requires no backtracking). Marcus is quite aware of the limitations imposed by what I have labeled in §2.1.5 above 'modularity': he does not expect his parser to decide in such cases, but rather to know how to interface with a semantic module to get this information. To avoid making mistakes, Marcus' parser has a three phrase buffer: other than the 'main' tree, the parser can keep up to three phrases (in a phrase only the top node is visible). These three phrases are supposed to give the parser all the information it needs to avoid wrong attachments in non problematic sentences such as (3.b), repeated here as (40):

(40) John saw Mary dance.

Marcus' parser works very much like a stack automaton. It contains three basic structures:

- A stack of *incomplete* nodes, that is nodes to which complete phrases can still be attached.
- An $O(1)$ buffer of complete phrases: this is how the parser implements its look-ahead. The size of this $O(1)$ buffer should be small, to reflect data constraints, and is set by Marcus to three phrases.
- A set of condition-sets. These reflect the state the parser is in, and the operations that are derived from its condition and the data.

The instructions in the condition set can test conditions that have to do with the $O(1)$ buffer storage, with the top two elements of the incomplete node stack (to deal with PP ambiguity: for cases in which we know that both the verb and the object can both attach a PP, and we have to choose what would have precedence). This means that the number of possible conditions is limited to the properties is $O(1)$ storage, and therefore evaluating a condition takes $O(1)$. Since the condition sets are part of the parser, and cannot be modified during the parser's execution, the number of possible conditions checked and executed cannot exceed $O(1)$, making the number of operations between each attachment or push into the incomplete node stack of $O(1)$ complexity. Over n elements, the total parsing time complexity is therefore $O(n)$, and the total space complexity, as mentioned briefly above in §2.1.4, of $O(n)$, as potentially every node can end up in the incomplete open node stack.

Marcus's framework deals nicely with the sort of sentences it was set to parse, parsing them a great deal more efficiently than parallel or backtracking parsers. However, when we choose constants for an $O(1)$ storage, we should be careful not to oversize them. Since Marcus' parser should choose correctly every single time, he has to allow it to hold three phrases at once before making a decision. While this does not mean that the parser is not an online parser, the constraint is not tight enough to fit the behaviour of the human parser. There are garden path sentences that have less than three phrases, and should therefore be analyzed correctly in Marcus' parser, but do cause a garden path condition.

(41) (a) ¿Without her contributions ceased.

(b) *xulca* *metayelet* *bavadi*
shirt/evacuated(PASSIVE) hikes/hiker in the creek
A hiking shirt in the creek (an improbable noun phrase, not a sentence).
¿A hiker was evacuated from the creek.

In (41.a), attaching 'her' to 'without' is not the attachment that causes the garden path. This means that Marcus' parser can perform this attachment and hold 'contributions' and 'ceased' in its buffer. In this case, the parser should have all the evidence it needs not to attach 'contributions' to the PP, but the sentence does cause a garden path. This might mean that 'contributions' is attached to 'her' *before* 'ceased' is encountered. The Hebrew example in (41.b), in which the garden path is caused by two lexical ambiguities, is even worse: the entire sentence is in the buffer when the wrong decisions are made. This shows

that even though the mechanisms are rather simple in terms of complexity, the parser's 'per-operation' constant is not small enough.

To sum up, Marcus' 1978 parser has the following properties:

- **Right Association**: Since Marcus's parser makes his choice using a finite number of storage elements, it is likely to make attachment errors even though the information needed to avoid them is still available. This makes it (at least in principle) susceptible to right association type errors.
- **Modularity**: Modularity is not implemented in the parser, it is one of the things the parser calls for: Marcus is quite aware he cannot decide between some competing options, and calls for intervention.
- **Complexity**: The parser uses linear time and storage to parse a sentence.
- **Monotonic Growth**: No assertion made by the parser can be retracted, and the assertion set does indeed grown deterministically.

3.2.2 Weinberg 1993

Weinberg's (1993) work is one of the prominent and complete attempts to use D-Theory in order to explain garden paths, using a deterministic parser that adds dominance statements rather than immediate dominance statements to its assertion set. The parser therefore can only 'correct' itself by pushing down an element by using intervening nodes, and cannot stipulate syntactic material without direct evidence for its existence, since it cannot retract what it has already stipulated. The parser employs the following principles:

Basic Parsing Algorithm:

(a) The parser uses an underspecified representation, written in terms of dominance and precedence predicates, to construct a local representation that is maximally licensed according to the submodules of the grammar. This creates an assertion set of dominance relations.

(b) The parser uses no lookahead. It scans tokens one word at a time and tries to shift a token onto the current phrase that it is building. If attachment to the current token is unlicensed, then the phrase currently being built

is reduced and the parser tries to shift the token onto a previously built phrase.

(c) The parser may add to its assertion set after seeing disambiguating material, either by adding a statement to the set or adding features to incompletely specified phrases using categorial notation as defined by the X-bar system. The parser may also establish and add to linear precedence relations.

Following the terms we have used in our objectives, the parser has the following properties:

- **Right Association**: Weinberg's parser may be able to handle some right association problems, as it tries to first allocate a token in the current phrase, and only then looks for attachment site further up the tree. If we strengthen this assumption to include some sort of modularity (for instance, if the parser is not even *aware* of higher nodes when it tries to perform the attachment) then we get the desired behaviour.

- **Modularity**: The very behaviour which lets the parser misinterpret right association sentences in the way the human parser does, is undesirable in attempting to explain ambiguity. For instance, if a PP is encountered after a noun, attaching it to the noun follows the parser's attempt to attach every token to the current phrase; this would mean that an D(N,PP) statement would be added to the assertion set, and the PP attachment ambiguity would be lost. The 'correct' behaviour would be to attach the PP to the verb, as this would leave the interpretation in which the NP dominates the PP still possible. This, however, would result in an ambiguous structure, and this is not what Weinberg (1993) describes.

- **Complexity**: Weinberg's (1993) parser does indeed have linear time $O(n)$ complexity if it follows the stated behaviour. However, allowing for the corrections that follow from its inability to trace ambiguities may well make it more complex. The space complexity of the parser is also linear.

- **Monotonic Growth**: Here we should note Lewis' (1998) criticism of Weinberg's approach: Weinberg (1993) tries to use the underspecified dominance statements as immediate dominance statements when there is a lack of counter evidence: The set of dominance relation assertions:

$\{D(V, NP), D(V, PP)\}$

is ambiguous between the two possibilities:

$\{I^D(V, NP), I^D(V, PP)\}$ and $\{I^D(V, NP), I^D(NP, PP)\}$,

but only the first option will be taken as true. This means that, indeed, the I^D relation does not grow in a monotonic way. This means that unless we maintain immediate dominance relations *outside* the assertion set, only the dominance relation would grow in a monotonic way.

4 My Parser

4.1 Implications on Implementation

4.1.1 Implementing Complexity

In §2.1.4 I have set the parser's complexity to be $O(n)$. For that to be achievable, the number of operations allowed for each element (word, syllable, phonological phrase etc.) has to be of $O(1)$: the number of operations has to be finite and independent of input length. This has some implications:

1. No large scale reanalysis is possible: the number of times an element can be 'reprocessed' should be independent of the size of the input. In other words, the algorithm may require one, two, or any number of elements to be reprocessed, as long as this is not an iterative process. Choosing too many elements here has other consequences which are independent of complexity issues. Example (41.a), repeated here as (42) demonstrated that for word elements in English, we cannot allow more than one word to be reprocessed: this is a garden path sentence, and it only requires the reanalysis of the last two words. If we could reprocess both 'ceased' and 'contributions', this sentence would not have been a garden path, but it is.

   (42) ¿Without her contributions ceased.

   This yields a deterministic (or monotonic) parser, as defined in Barton and Berwick (1985): every decision that is beyond the scope of legitimate (language dependent) reanalysis can be considered to be in the algorithm's assertion set. When reanalysis is not bound, no decision fulfills this requirement.

2. The number of elements involved in any decision making has to be finite and independent of the size of the input. The description of the algorithm should therefore have no more than a finite number of storage elements (or registers) available for any decision. For example, when we consider where to attach a PP, we may consider the immediately preceding NP, or the NP and the VP above it, but not any number of VPs above it. This automatically yields a 'Right Association' tendency as in (Kimball, 1973): if the parser does not hold the top nodes of the tree in its working storage, attaching to the lower nodes is the only available option. It should be noted though, that changing the representation used by the parser, as suggested

in §2.1.5 and below, in §4.1.2, might compromise that, and we would have to check that it is still achieved.

3. Ambiguities can only be resolved within the storage limited number of elements. In order to even *know* there are large scale ambiguities, the parser would have to 'climb' up the entire tree length for every element it encounters, so it may find alternative attachment points.

What these limitations mean, is that within the limits of linear processing time, it would be impossible to have a parser that can manage ambiguities, and would be severely limited in solving rather local ones. Thus, only allowing for a parser that would not be able to 'grasp' the full interpretation of a sentence, but rather to get its information incrementally, one bit of information at a time. This calls for a simpler and more limited representation of the parse tree, which I will demonstrate in the following section.

4.1.2 Implementing Modularity

Modularity, as one can conclude from the limitations imposed by linear processing time, is not only an aim to achieve, but a necessity: if ambiguity cannot be resolved by the parser, than it has to provide a structure that is modular, that is: a structure in which the ambiguities remain unresolved.

Our first shot at trying to achieve a modular structure would be using D-Theoretic representation. In D-Theory, as in Weinberg (1993), two orders are maintained: *precedence* and *dominance* (rather than immediate dominance). Precedence is evident from the input data and need not be maintained. Dominance, however, requires the parser to connect pieces which might be very far apart from each other, at least when certain ambiguities are resolved. Sentences such as (7), repeated here as (43):

(43) Dan told Mary to ask Jane to invite John to the party, rather than do it himself

require the parser to check every verb for possible attachment. The parser would have to check which of the following statements can be added to the assertion set:

1. $\lambda R.\, \langle \text{invite}, \text{rather than do it himself} \rangle \in R$
2. $\lambda R.\, \langle \text{ask}, \text{rather than do it himself} \rangle \in R$

3. $\lambda R.\, \langle \text{told}, \text{rather than do it himself} \rangle \in R$

This would require the parser to keep all the verbs in accessible storage, which we have already seen is impossible in linear complexity.

Parsers usually try to build syntactic structure. I find that in order to achieve the level of modularity I require I will have to do without syntactic structure. In the classical PP attachment ambiguity for instance, the differences between the two structures in syntactic terms are significant, while in semantic terms they are not: it's only a question of which head the PP modifies. Giving up syntactic structure for semantic structure gives me greater flexibility. I do not think this is too problematic, as the aim of a parser, after all, is to transform the linear phonological data into semantic representation: so having a semantic representation as the goal of the parser should be even more natural, in my view, than trying to achieve a syntactic representation. Note that this does not mean that there is no need for another syntactic phase or module: I do not try to claim that there is no need for syntax.

The alternative I offer is maintaining a relation that is not a dominance order, but rather determines a set of dominance orders: one or more of these dominance order should be the correct parse of the tree. There are several implications of this choice:

1. The set of dominance orders may contain dominance orders which are not the correct parse trees of the sentence. This can only be determined by subsequent modules, which can incorporate more data into the decision.

2. An ambiguous sentence should have all its valid readings in the dominance orders set: this predicts that, if for some reason, one of the expected readings is not in the dominance orders set, this reading would not be available, at least not to someone who hears or reads the sentence.

3. If the correct parse tree is not in the dominance orders set, the sentence would not be understood, or, in terms we have already used, the sentence would be difficult to process.

4. This should also go the other way around: if a sentence is difficult to process, we expect the correct parse tree not to be found in the dominance orders set.

The structure I provide is a linear order of input elements, which is unlike the original linear order provided by phonology, and adheres to the following two conditions:

- If an element modifies another element, it follows it. By *modifies* I refer to a semantic notion:

  Definition 44. *Element A modifies element B if and only if A is an argument of B or otherwise restricts B, or (recursively) A modifies some element C that modifies B.*

  For example, in the sentence 'John saw blue flowers', 'John' is a modifier of 'saw' because it is its argument, and 'blue' modifies 'saw' because it restricts 'flowers', which is a an argument of 'saw'. Since in the semantic tree of a sentence an element that modifies another element is dominated by the other element, I will often use the term 'dominates' rather than 'is modified by', and 'is dominated by' rather than 'modifies'.

- An element and all its modifiers are not intervened by another element. Combining this with the previous condition means that if A is modified by B, then every element between A and B modifies A as well.

This representation is very similar to a bracketed notation, in which the brackets are not specified (not specifying the brackets yields the underspecification). 'John saw flowers' might have the following representation:

- $saw \leq John \leq flowers$

Which should be interpreted as:

- $\left[saw \leq [John]_{John} \leq [flowers]_{flowers}\right]_{saw}$

While an ambiguous sentence 'John bought her jewels' might have the following representation:

- $bought \leq John \leq jewels \leq her$

This can be interpreted as either of the following, depending on context:

- $\left[bought \leq [John]_{John} \leq [jewels]_{jewels} \leq [her]_{her}\right]_{bought}$
- $\left[bought \leq [John]_{John} \leq [jewels \leq [her]_{her}]_{jewels}\right]_{bought}$

The subsequent module's main task would therefore be placing the correct 'brackets', and processing difficulties can be represented as supplying the wrong linear order. Subsequent modules would rely on the linear order, and if the correct interpretation is not the result of placing the correct brackets, we would experience a processing difficulty.

In formal terms, this means that I have to define both a linear order for the parser to build, and a dominance order, which will be what subsequent modules should reveal, based on context and other data.

Definition 45. *The structure built is defined as a pair:* $A = \langle M^A, \leq \rangle$ *such that:*

- M^A *is a finite set of elements.*
- $\leq$ *is a total linear order relation on* M^A

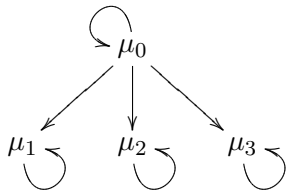
We already know, that, if we wish to keep linear complexity, it is impossible to handle anything but a set number of 'entrance' points to add elements in. This would naturally allow the parser to set an element *before* all the elements that are already set in the linear order or *after* all these elements. It would be possible to set a fixed number of other such elements: 'just after the first element', 'just before the last element', as long as the number of these special cases is not only of $O(1)$ space complexity, but also small. If this is kept, the linear order can be accessed as $O(1)$ storage, meeting the complexity requirements. A linear order, however, is just a single dominance order, and the modular underspecified structure requires for more than one such dominance order to be present. I therefore define the $\sqsubseteq$, a partial order, based on the guidelines I gave above.

Claim 46. *In every non-empty A there is at least one partial order relation* $\sqsubseteq$ *such that:*

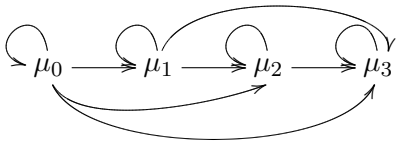
- $(\sqsubseteq) \subseteq (\leq)$
- *The pair* $\langle M^A, \sqsubseteq \rangle$ *is a semi-lattice.*
  That is, for every μ_1, μ_2 *in* M^A, *the least upper bound or* join *of* μ_1 *and* μ_2, $\mu_1 \sqcup \mu_2$ *is in* M^A.
- *For every two elements* μ_1, μ_2, *such that* $\mu_1 \sqsubseteq \mu_2$, *the pair:*
  $\langle \{\mu : \mu_1 \leq \mu \leq \mu_2\}, \sqsubseteq \rangle$ *is a semi-lattice.*

Samples of $\sqsubseteq$ may be:

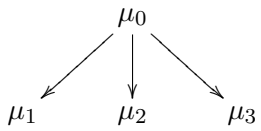
1. $\sqsubseteq = \left\{\langle \mu_0, \mu\rangle : \mu_0 \text{ is the minimal element in } M^A \text{ with regard to } \leq \text{ and } \mu \in M^A\right\}$



2. $\sqsubseteq = \leq$



The examples I gave above are dominance orders. Their immediate dominance order equivalent, $I^{\sqsubseteq}$ would be:

1. 

2. $\mu_0 \longrightarrow \mu_1 \longrightarrow \mu_2 \longrightarrow \mu_3$

What is obviously needed here, is to prove that $\sqsubseteq$ is a dominance order. This is rather easy to prove:

Claim 47. *$\sqsubseteq$ is a dominance order.*

Proof. This follows from the definition of dominance orders:

- $\sqsubseteq$ is a partial order by definition.
- $\sqsubseteq$ is non branching to the past. If there are two elements μ_1, μ_2 such that $\mu_1 \sqsubseteq \mu_3$ and $\mu_2 \sqsubseteq t_3$, then without loss of generality, $\mu_1 \leq \mu_2$, which means μ_2 is in the semi-lattice $\langle \{\mu : \mu_1 \leq \mu \leq \mu_3\}, \sqsubseteq\rangle$, and since for every μ in that semi-lattice $\mu_1 \sqsubseteq \mu$ as $\sqsubseteq \subseteq \leq$, it follows that $\mu_1 \sqsubseteq \mu_2$
- $\sqsubseteq$ has a minimal element as it is a semi-lattice defined over a finite set.

□

From this it should follow that what the next modules can come up with are trees, if we can produce a tree from the $\sqsubseteq$ relation.

Claim 48. $\langle I^{\sqsubseteq}, M \rangle$ *is a rooted tree. This follows directly from the claims 21 in §2.2.1 and 47 above.*

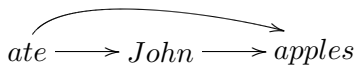
I will now re-define the two constraints I proposed above in a formal manner. This is what the definition of $\sqsubseteq$ boils down to. I will use these constraints in the following section to show how the requirements made by $\sqsubseteq$ are not met in sentences which are difficult to process.

- *Non-dominance*: If μ_1 and μ_2 are in M, and $\mu_1 \leq \mu_2$, then μ_2 does not dominate μ_1: μ_1 does not modify t_2 or any of its modifiers (recursively).
- *Bracketing*: An element (head) and all its arguments and modifiers are not intervened by a element which is not dominated by the head.

Here are a few demonstrations of what the $\leq$ and $\sqsubseteq$ relations can yield. In (49), the verb has two arguments.

(49) John ate apples.

In order to provide the correct parse tree, the verb must precede its argument. Therefore, the two valid possibilities for $\leq$ are:

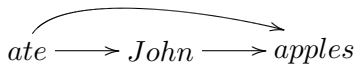
1. 

2. 

We can see straight away that if the parser yields the first option, $\sqsubseteq$ can be either of the following two options, of which only the first is the correct parse tree. This is also true, of course, if the parser provides the second option.

1. $\sqsubseteq = \{\langle \mu_0, \mu \rangle : t_0$ is the minimal element in M^A with regard to $\leq$ and $\mu \in M^A\}$:

2. $\sqsubseteq = \leq$



The way of building the correct assertion set under such conditions, requires us to put the dependency between $\sqsubseteq$ and $\leq$ into the assertion set before any other information is provided, in much the same way as we have put the fact that the relation D is a dominance order in example 26 on page 24. The initial assertion set should contain at least the following statements (notice that since the information growth is deterministic, every relation set has to be a subset of the relation set assigned for that label in the previous assertion set):

$$AS_0(l) = \lambda l.\mathfrak{R}$$

$$AS_1(l) = \left\{ \begin{array}{ll} AS_{i-1}(l) \cap \text{linear order} & l = \leq \\ AS_{i-1}(l) \cap \lambda R.[\langle R, T\rangle \text{ is a semi-lattice}] \cap RL & l = \sqsubseteq \\ AS_{i-1}(l) & otherwise \end{array} \right\}$$

where:

$$RL = \lambda R.\left[\begin{array}{l} \exists S \in AS_i(\leq). \\ (R \subseteq S) \wedge \forall t_1, t_2 \in T.t_1 R t_2 \rightarrow (\langle R, \{t : t_1 S t S t_2\}\rangle \text{ is a semi-lattice}) \end{array} \right]$$

We can now begin parsing an ambiguous sentence, and see how both meanings are kept. I will use (8), repeated here (again) as (50), and treat 'from Italy' as a single element to make the example short.

(50) $t_1 = \text{John}, t_2 = \text{brought}, t_3 = \text{shoes}, t_4 = \text{from Italy}$

The following does not necessarily match what the parser I suggest does, but it does adhere to the definition of $\sqsubseteq$ used by my parser.

1. We decide that 'John' does not dominate 'brought', and we therefore put it after 'brought' in the linear order: $t_2 \leq t_1$:

   $$AS_2(l) = \left\{ \begin{array}{ll} AS_{i-1} \cap \lambda R.t_2 R t_1 & l = \leq \\ AS_{i-1} & otherwise \end{array} \right\}$$

   Notice that we still don't know how many elements there are, but they do not affect our interpretation. When we get to the end of the sentence we will only use the subset of the relations that deals with the elements we have said something about.

2. Now we decide that 'shoes' is also an argument of 'brought'. Since we would like not to keep the data about elements we have already used, we attach it to the end of the chain already maintained: $t_1 \leq t_3$.
$$AS_3(l) = \left\{ \begin{array}{ll} AS_{i-1} \cap \lambda R.t_1 R t_3 & l = \leq \\ AS_{i-1} & otherwise \end{array} \right\}$$
3. At last we decide to concatenate 'from Italy' to the end of the chain: $t_3 \leq t_4$.
$$AS_4(l) = \left\{ \begin{array}{ll} AS_{i-1} \cap \lambda R.t_3 R t_4 & l = \leq \\ AS_{i-1} & otherwise \end{array} \right\}$$

Notice now, that even though we have not modified $\sqsubseteq$ directly even once, it has changed, since its definition was derived from $AS_i\,(\leq)$, which was modified. Now we can check which are the valid $I^{\sqsubseteq}$ relations, and there are four of them.

(51) (a) $t_2 \longrightarrow t_1 \longrightarrow t_3 \longrightarrow t_4$

(b) $t_2 \longrightarrow t_1 \longrightarrow t_3$, $t_2 \longrightarrow t_4$

(c) $t_2 \longrightarrow t_1$, $t_1 \longrightarrow t_3 \longrightarrow t_4$

(d) $t_2 \longrightarrow t_1$, $t_2 \longrightarrow t_3$, $t_2 \longrightarrow t_4$

Of these, (51.a) and (51.b) are not valid parse trees of the sentence, but (51.c) and (51.d) are the two available ambiguous parses. (51.c) is the case where the PP is interpreted as an adjunct, and (51.d) is the case where the PP is interpreted as an argument to the verb. Notice also, that any subsequent module is very unlikely to accept the first two alternatives, as they require 'shoes' to modify 'John'. This hints that the number of interpretations does grow rapidly, but the task facing the subsequent module is not as complicated as it first seems.

4.2 Parser Overview

4.2.1 Data Structure

My analysis is built out of the following elements and operations:

- Units
  - The basic input element u (word, syllable) is a unit.
  - Every unit has two properties: its *category set* and its *label*. The category set is the set of possible categories for the unit. To refer to an element's category we will use the notation $cat(u)$. For simplicity, I will currently refer to the category set as just a set. In practice it is much more likely that each category is assigned some likelihood. The word 'man' for example, is more likely to be a noun than a verb. I will use a unit's category to store both its syntactic category and its required argument structure. The label of the category is the input it was based on: the word or the syllable. To refer to a unit's label we will use the notation $label(u)$.
    I use a rather loose notation for the category set of units, but the following guidelines do apply:
    - The first symbol is the unit's own category: if the category set of some unit is $\{V\}$, its category is V. If some unit's category set is $\{V, N\}$ it is ambiguous between categories V and N.
    - In subscript I write the list of arguments the unit can take. If some unit's category set is $\{V_N\}$, its category is V, and it can take an N category as an argument. If some unit's category set is $\{V_{N+V}\}$, its category is V, and it can take two arguments: N and V. Notice that if the unit's category set is $\{V_N, V_V\}$, its category is V and it can take *either* N or V.
    - In superscript I write the predicted role of the unit: if a unit's category set is $\{A^N\}$, its category is A, and it predicts a category N (that is the result of applying this unit to some future unit will result in an N). I use superscript for another similar use, to mark the thematic role of an argument: $\{N^{subj.}, N^{obj.}\}$: is the category set of a unit whose category is N, but is ambiguous between predicting the role of subject and object.
    - Superscript and subscript notations can be combined. Using both, we can

describe the category set of a V that takes an N subject: $\{V_{N^{subj.}}\}$.

- The concatenation of two units a and b, ab is a unit (recursively). The category set of the new unit is that of the left unit, or in this example, $cat(ab) := cat(a)$. The label of the new unit is the concatenation of the more basic labels: $label(ab) := label(a)label(b)$.

- Data Structures: there are three data structures for holding units. There are no other means of holding or accessing units.

  - The *open unit stack* is a stack (push down storage)[12] of units. The top n units of the unit stack are visible to comply with the complexity requirements, and $n = 1$ is the natural starting point (unless some language demonstrates that it is not the case for that language). This means that usually only the last unit to be inserted into the stack is visible. We will refer to the top unit in the stack as $open[1]$, the second visible element (if any) as $open[2]$ and so forth.

  - The *closed unit stack* is a stack of units of which the top n units are visible. Here again, $n = 1$ or just the last unit to be inserted into the stack is the natural starting point. We will refer to the top unit in the stack as $closed[1]$, the second visible element (if any) as $closed[2]$ and so forth. Note that it would be more efficient not to have a closed unit stack but rather an $O(1)$ storage to hold closed elements. This alternative is discussed in subsequent sections.

  - The *current unit(s)* is an $O(1)$ storage holding n: a finite and independent number of units, from the input. In the case of English, where units are words, $n = 1$, as $n = 2$ would have kept sentence (42), repeated here as (52), from being a garden path.

    (52) Without her contributions ceased

    We will refer to the first current unit as $current[1]$ and so forth. For convenience, we use the term *current* as equivalent to $current[1]$.

[12] A stack is a data structure that can contain countably many elements, but allows access to only a finite number of elements: usually just a single *top* element. Inserting a new element into a stack makes the new element the new top element, and removing it restores the previous top element. Therefore, inserting a new element into a stack is usually called *push* and removing the topmost element is usually called *pop*. In terms of complexity, both the *push* and *pop* operations can be achieved in $O(1)$ time.

For example, when we process a sentence such as "John saw a man", at the point where we first encounter the word 'man', the data structures are likely to contain the following data:

- Open unit stack:
  - $open[1] = \langle a, \{D\} \rangle \leftarrow$ Top of the stack
  - $open[2] = \left\langle saw, \left\{ V_{N^{subj.}+N^{obj.}}, V_{N^{subj.}+V} \right\} \right\rangle$
- Closed unit stack:
  - $closed[1] = \left\langle John, \left\{ N^{subj.}, N^{obj.} \right\} \right\rangle \leftarrow$ Top of the stack
- Current unit:
  - $current[1] = \left\langle man, \left\{ N^{subj.}, N^{obj.}, V, \ldots \right\} \right\rangle$

- Operations: the following operations are defined, each of which can be performed $O(1)$ times:
  - $u := NewUnit(label, category\ set)$: when a new unit is first encountered, it is assigned a finite set of possible categories.
  - $SetCategory(u, category\ set)$: remove or reduce categories in the unit category set. To reduce a category means to transform it to another category: the chain of possible transformations is bound and predefined. To remove a category means just that: remove it from the category set. Note that, since the number of categories is finite to begin with, this operation will not be repeated more than $O(1)$ times. A possible category set for 'man' may be: $\left\{ N^{obj.}, N^{subj.}, V_{N^{obj.}+N^{subj.}} \right\}$
  - $u := Attach(u_1, u_2)$: concatenate a unit to another unit: $u := \mu_1\mu_2$
  - $PushClosed(u)$: push a unit to the closed unit stack.
  - $PushOpen(u)$: push a unit to the open unit stack.
  - $u := PopClosed()$: pop a unit from the closed unit stack.
  - $u := PopOpen()$: pop a unit from the open unit stack.
  - $u := CreateUnit(category\ set)$: A unit may create a new functional[13] unit. The category set of the new unit is usually a singleton, that is comprised of a single element. Functional units can create new functional units themselves, as

[13] Functional means here not based on input

long as the number of successive functional units is independent of the input size and finite ($O(1)$).[14]

4.2.2 General Outline

The operation of the pre-parser is limited by the decisions it can make, due to the $O(n)$ complexity bound. Under these limits, the pre-parser should try to attach every new unit to the units which can still be extended, and not rely on the possible existence of better attachment sites which are not within view. The stored units are therefore split between two data structures: one which contains units to which units can still be attached (the open unit stack), and one which stores units to which other units cannot be attached, and which could not be attached to another unit while they have been processed (the closed unit stack).

The general pre-parser strategy in English should be something like this:

1. Check if the new unit ($current[1]$) can be attached to the unit at the top of the open stack ($open[1]$), based on the category sets of both elements. Every unit can be attached to the "top" of an empty open stack. If so, do the following:

   (a) Modify the possible category set of the current unit to reflect what the previous unit can have as its argument (a unit that can extend a verb that requires a direct object should not have the option of being a verb itself: if the top of the open stack contains 'drink' and the current unit is 'water', water will no longer have a verbal category in its category set)
   $SetCategory(current[1], \{\ldots\})$

   (b) Push the new unit to the top of the open stack
   $PushOpen(current[1])$

   (c) Read the next unit.
   $current[1] = NewUnit(\ldots, \{\ldots\})$

   For example, a parser in the following state, which complies with the condition for this rule:

[14] Creating functional units is not an essential part of the parser, but rather a trick, designed to store the parser's state in the data structures used to hold units, rather than in some other dedicated storage. This operation can be dropped in future implementations. Functional units are ignored in the output of the parser.

current: $water\left\{N^{obj.}, N^{subj.}, V_{N^{obj.}+N^{subj.}}\right\}$

open stack	**closed stack**
$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

will change by the rule to the following state:

current: *the next word*

open stack	**closed stack**
$water\left\{N^{obj.}\right\} \leftarrow$ Top of the stack $drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

2. If the new unit (*current*[1]) could not be attached, but the top of the open stack (*open*[1]) can be an argument of the new unit by some functional category (such as in the case of N^N N sequence where N^N can be a genitive pronoun)

   (a) Push the current top the open stack into the closed stack
   $PushClosed(PopOpen())$

   (b) Push an appropriate functional unit to the top of the open stack.
   $PushOpen(CreateUnit(\{...\})$

   For example, a parser in the following state, which complies with the condition for this rule:

   current: $coke\left\{N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$his\left\{N^{N^{obj.}}\right\} \leftarrow$ Top of the stack $drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

   will change by the rule to the following state, in which a functional unit is pushed into the stack. Do notice that the functional unit (marked here by $\emptyset$) take no part in the determination of $\leq$ or $\sqsubseteq$. In other words: functional elements are not really units but rather 'place holders', and we ignore them for any other purpose. The symbol I use to mark their category set, X^Y_Y, defines their proper use: their own category is X, which is disregarded, as they project some Y category, which also 'happens to be' the category they take as an argument. This is really an explicit way of telling the parser 'attach an element of category Y here'.

current: $coke\left\{N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$\emptyset\left\{X^{N^{obj}}_{N^{obj.}}\right\}$ ← Top of the stack	$his\left\{N^{N^{obj.}}\right\}$ ← Top of the stack
$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

In the next iteration rule 1 will apply again and *coke* will be pushed to the open stack with only $N^{obj.}$ in its category set.

current:

open stack	**closed stack**
$coke\left\{N^{obj.}\right\}$ ← Top of the stack	
$\emptyset\left\{X^{N^{obj}}_{N^{obj.}}\right\}$	$his\left\{N^{N^{obj.}}\right\}$ ← Top of the stack
$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

Note that I use the $\emptyset\left\{X^{\cdots}_{\cdots}\right\}$ notation here. The X category here has no effect, as a functional node will always have a projected value (after all they are inserted when their argument and result are already known). Moreover, they are marked with the $\emptyset$ symbol not only because they are not created by any actual input, but also because they do not affect the final linear order: we ignore them completely when we come to build both $\leq$ and $\sqsubseteq$.

3. If neither of the two conditions applied

   (a) While the top of the closed stack can be attached to the top of the open stack

      i. Attach the top of the closed stack to the open stack
      $PushOpen(Attach(PopOpen(), PopClosed())$

      ii. Modify the top of the open stack to indicate the attachment took place
      $SetCategory(open[1], ...)$

   For example, a parser in the following state, which complies with the condition for this rule:
   current:

open stack	**closed stack**
$coke\left\{N^{obj.}\right\}$ ← Top of the stack	
$\emptyset\left\{X^{N^{obj}}_{N^{obj.}}\right\}$	$his\left\{N^{N^{obj.}}\right\}$ ← Top of the stack
$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John\left\{N^{subj.}\right\}$

Will be transformed by this rule to:
current:

open stack	**closed stack**
$coke, his \left\{N^{obj}\right\} \leftarrow$ Top of the stack	
$\emptyset \left\{X^{N^{obj}}_{N^{obj.}}\right\}$	
$drank \left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John \left\{N^{subj.}\right\}$

(b) If the open stack has more than one unit in it:

i. Pop the top of the open stack and attach it to the new top of the open stack (the right argument of *Attach* is popped first).
$PushOpen(Attach(PopOpen(), PopOpen())$

ii. Modify the top of the open stack to indicate the attachment took place ($SetCategory(open[1], ...)$

This rule applies for the parser in the previous state, and it will be transformed by the current rule to:
current:

open stack	**closed stack**
$\emptyset, coke, his \left\{X^{N^{obj}}\right\}$	
$drank \left\{V_{N^{obj.}+N^{subj.}}\right\}$	$John \left\{N^{subj.}\right\}$

(c) If the open stack had no units other than the top unit, push the top of the open stack to the closed stack
$PushClosed(PopOpen())$
This means that if the only element in an open stack cannot attach the next unit, or be attached to it, it is pushed into the closed stack.

4. Repeat the process until there are no more units to read, and the closed stack contains exactly one element.

4.2.3 Parsing Samples

To demonstrate how the pre-parser works, I will start by using a few non problematic sentences. Each parse state will trace the three data structures from the previous section. Each data structure will be displayed as a list, with semicolons as separators. When a unit consists of more than one unit, a comma will separate the more basic elements.

(53) John saw Mary

(a) 'John' is encountered and is assigned a category.
current: $John\left\{N^{subj.}, N^{obj.}\right\}$

open stack	**closed stack**

(b) 'John' can be attached to the top of the open stack because it is empty. 'saw' is then read. This follows rule 1.
current: $saw\left\{V_{N^{obj.}+N^{subj.}}, V_{S_{obj.}+N^{subj.}}\right\}$

open stack	**closed stack**
$John\left\{N^{subj.}, N^{obj.}\right\}$	

(c) 'saw' cannot be attached to the top of the open stack, but the top of the open stack can be attached to 'saw' following rule 2.
current: $saw\left\{V_{N^{obj.}+N^{subj.}}, V_{N^{obj.}+N^{subj.}}\right\}$

open stack	**closed stack**
$\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(d) Rule 1 applies again, and 'Mary' is read. The new unit is assigned two possible categories, because 'saw' can attach both a sentence and a direct object.
current: $Mary\left\{N^{subj.}, N^{obj.}\right\}$

open stack	**closed stack**
$saw\left\{V_{N^{obj.}+N^{subj.}}, V_{N^{obj.}+N^{subj.}}\right\}$ $\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(e) 'Mary' can be attached to saw, and so rule 1 applies. No further input exists, and so the current unit is empty.
current:

open stack	**closed stack**
$Mary\left\{N^{subj.}, N^{obj.}\right\}$ $saw\left\{V_{N^{obj.}+N^{subj.}}, V_{N^{obj.}+N^{subj.}}\right\}$ $\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(f) Since no current unit is available, rule 3 applies. 'Mary' cannot attach 'John', and so rule 3a is skipped. Rule 3b now applies, 'Mary' is right-attached to

'saw', and 'saw' is modified to exclude the case it can attach a sentence, and reduced as not to allow it to attach another object.
current:

open stack	**closed stack**
$saw, Mary\{V_{N^{subj.}}\}$ $\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(g) Again, no current unit is available, and so we switch to rule 3. 'saw' can attach 'John' and so rule 3a applies. 'John' is attached to 'saw', and 'saw' is modified to indicate it does not need another subject.
current:

open stack	**closed stack**
$saw, Mary, John\{V\}$ $\emptyset\left\{X_V^V\right\}$	

(h) Rule 3b applies now to yield:
current:

open stack	**closed stack**
$\emptyset, saw, Mary, John\{V\}$	

(i) Rule 3c applies, and the only unit is moved to the closed stack. This is the final state, as it fits the stop conditions in rule 4.
current:

open stack	**closed stack**
	$\emptyset, saw, Mary, John\{V\}$

The expected result is a string of units in which the arguments of the verb are in non-dominance relation with the verb. This sentence can have the following correct $\sqsubseteq$ dominance order:

- $\left[saw \leq [Mary]_{Mary} \leq [John]_{John}\right]_{saw}$

The next example shows how the subject / object ambiguity is resolved.

(54) John saw Mary dance

Processing starts as it did in the previous case, right up the step (d). It then proceeds in the following manner:

(e) 'Mary' can be still be attached to saw, and so rule 1 applies. 'dance' is read and is assigned categories. Since it is not clear whether 'dance' is a noun or a verb, it is assigned more than one possible category.
current: $dance\left\{V_{N^{subj.}}, N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$Mary\left\{N^{subj.}, N^{obj.}\right\}$	
$saw\left\{V_{N^{obj.}+N^{subj.}}, V_{S_{obj.}+N^{subj.}}\right\}$	
$\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(f) 'dance' cannot be attached to 'Mary', but 'Mary' can be attached to 'dance' if we add a functional category, and so rule 2 applies: we move the top of the open stack to the top of the closed stack, and add a sentence functional unit to the top of the open stack.
current: $dance\left\{V_{N^{subj.}}, N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$\emptyset\left\{X_V^V\right\}$	
$saw\left\{V_{N^{obj.}+N^{subj.}}, V_{S_{obj.}+N^{subj.}}\right\}$	$Mary\left\{N^{subj.}, N^{obj.}\right\}$
$\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(g) Now rule 1 can apply, since the current unit can be attached to the top of the open stack. 'dance' is therefore pushed to the top of the stack, and its category set is modified to reflect its future role. No more input is available.
current:

open stack	**closed stack**
$dance\left\{V_{N^{subj.}}\right\}$	
$\emptyset\left\{X_V^V\right\}$	
$saw\left\{V_{N^{obj.}+N^{subj.}}, V_{S_{obj.}+N^{subj.}}\right\}$	$Mary\left\{N^{subj.}, N^{obj.}\right\}$
$\emptyset\left\{X_V^V\right\}$	$John\left\{N^{subj.}, N^{obj.}\right\}$

(h) Rule 3 is now relevant. Rule 3a is checked, and indeed 'dance' can attach an argument from the closed stack. 'Mary' is popped and attached. Now 'dance' cannot attach any more elements.

current:

open stack	**closed stack**
$dance, Mary\,\{V\}$ $\emptyset\,\{X_V^V\}$ $saw\,\left\{V_{N^{obj.}+N^{subj.}}, V_{V_{obj.}+N^{subj.}}\right\}$ $\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}, N^{obj.}\}$

(i) 'dance' is now attached to the second open stack element, following rule 3b.

The result is:
current:

open stack	**closed stack**
$\emptyset\,\{X_V^V\}$ $saw\,\left\{V_{N^{obj.}+N^{subj.}}, V_{V_{obj.}+N^{subj.}}\right\}$ $\emptyset, dance, Mary\,\{\}$	$John\,\{N^{subj.}, N^{obj.}\}$

(j) The process is repeated for 'saw' and $\emptyset$.

current:

open stack	**closed stack**
$saw, \emptyset, dance, Mary\,\{V_{N^{subj.}}\}$ $\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}, N^{obj.}\}$

(k) Rule 3a is repeated:

current:

open stack	**closed stack**
$saw, \emptyset, dance, Mary, John\,\{V\}$ $\emptyset\,\{X_V^V\}$	

(l) Rule 3b applies again:

current:

open stack	**closed stack**
$\emptyset, saw, \emptyset, dance, Mary, John\,\{V\}$	

(m) And finally rule 3c is applied to yield:

current:

open stack	**closed stack**
	$\emptyset, saw, \emptyset, dance, Mary, John\,\{V\}$

The final assertion set is the one we expected to have, and its growth has been deterministic: there was no need to withdraw any decision, simply because the role of object / subject does not become critical until the attachment occurs. A valid $\sqsubseteq$ dominance order can therefore be found:

- $\Big[saw \leq \Big[dance \leq [Mary]_{Mary}\Big]_{dance} \leq [John]_{John}\Big]_{saw}$

The case of the next example, follows the same path, except the functional category is a D rather than an S.

(55) John gave her books.

current:

open stack	**closed stack**
	$\emptyset, gave, \emptyset, books, her, John\,\{V\}$

Note that the final outcome of this sentence allows two readings. 'her' does not dominate books, but the reading in which it is not dominated by books is also kept, and the ambiguity is decided by the next module:

- Dative 'her': $[gave \leq [books]_{books} \leq [her]_{her} \leq [John]_{John}]_{gave}$
- Genitive 'her': $\big[gave \leq [books \leq [her]_{her}]_{books} \leq [John]_{John}\big]_{gave}$

The same goes for the example I gave against Pritchett's analysis of PP attachment, sentence (38.b) on page 32, repeated here as (56):

(56) John brought shoes from Italy from work.

current:

open stack	**closed stack**
	$\emptyset, brought, shoes, from, Italy, from, work, John\,\{V\}$

The sentence does not require re-interpretation, as both PPs can be dominated by the verb, and other semantic and syntactic factors determine that it is unlikely for the verb to be modified by a source argument twice [15].

[15] In some contexts it would be fine, of course: 'John brought me shoes from the top shelf from behind the pink shoes', which is fine in Hebrew.

4.3 Data Overview

4.3.1 Explaining Garden Path Sentences

What we know about garden path sentences, is that the parser cannot correct a mistake it has made before. For deterministic parsers (and my pre-parser), this means it cannot make a mistake, as mistakes cannot be corrected. I would now like to follow the parse process of some garden path sentences. I will not follow every parse state, but move to the point where the mistake is made.

(57) Without her contributions ceased.

(a) At some stage the parser reads the word 'contributions'.
current: $contributions\left\{N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$her\left\{N, N^{N^{obj.}}\right\}$ $without\left\{P_{N^{obj.}}\right\}$	

(b) Since 'contributions' can be extended by 'her' via a functional category, rule 2 is used, 'her' is pushed into the closed stack, and a functional category is pushed into the open stack:
current: $contributions\left\{N^{obj.}, N^{subj.}\right\}$

open stack	**closed stack**
$\emptyset\left\{N^{N^{obj.}}_{N^{obj.}}\right\}$ $without\left\{P_{N^{obj.}}\right\}$	$her\left\{N^{N^{obj.}}\right\}$

(c) Rule 1 is used, followed by 3a, to yield:
current:

open stack	**closed stack**
$contributions, her\left\{N^{obj.}\right\}$ $\emptyset\left\{N^{N^{obj.}}_{N^{obj.}}\right\}$ $without\left\{P_{N^{obj.}}\right\}$	

(d) The next rules will bring us to the following parse state, in which the bracketing condition is not obeyed.

current: $ceased\,\{V_{N^{subj.}}\}$

open stack	**closed stack**
$without, \emptyset, contributions, her\,\{P\}$	

The order of the units in the open stack cannot be changed now: even if we had operations that could put new units in between the existing units, the following would still hold:

$without \leq contributions \leq her$

Since 'her' modifies 'without', every unit that intervenes between them should also modify 'without'. This is not the case, as 'contributions' intervenes between the two, no $\sqsubseteq$ can be the correct tree, and we get a processing difficulty. The cognitive confusion we get is a feeling that 'ceased' is left without a subject. This is predicted by the violation of the bracketing condition: 'conditions' is between 'without' and its modifier 'her' and therefore must modify 'without'. As a modifier of 'without', it cannot serve as a subject.

The same problem occurs when processing:

(58) After Susan drank the water evaporated

(a) At some point we reach the following state:
current: $evaporated\left\{V_{N^{subj.}}, V_{N^{subj.}+N^{obj.}}\right\}$

open stack	**closed stack**
$water\left\{N^{obj.}\right\}$	
$the\left\{D^{N^{obj.}}\right\}$	
$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$	
$\emptyset\left\{X_V^V\right\}$	
$After\,\{P_V\}$	$Susan\left\{N^{subj.}, N^{obj.}\right\}$

(b) Since 'evaporated' cannot be attached to 'water' and 'water' cannot be a subject of a sentence (since 'drank' cannot take a sentential argument), rule 3 applies. 'Susan' cannot be attached to 'water' and so rule 3b is used:

$evaporated\left\{V_{N^{subj.}}, V_{N^{subj.}+N^{obj.}}\right\}$ $\quad the, water\left\{N^{obj.}\right\}$

$drank\left\{V_{N^{obj.}+N^{subj.}}\right\}$

$\emptyset\left\{X_V^V\right\}$

$After\,\{P_V\}$ $\quad Susan\left\{N^{subj.}, N^{obj.}\right\}$

(c) Since 'evaporated' cannot be attached to 'the,water' either (note that since the category D projects a noun, the remaining category set has a noun and not a determiner), and 'the,water' cannot be the subject of a sentence, for the same reason as above, rule 3 applies again. 'Susan' cannot be attached to 'the,water' either, and so rule 3b is used again:
current: $evaporated\left\{V_{N^{subj.}}, V_{N^{subj.}+N^{obj.}}\right\}$

open stack	**closed stack**
$drank, the, water\left\{V_{N^{subj.}}\right\}$ $\emptyset\left\{X_V^V\right\}$ $After\left\{P_V\right\}$	 $Susan\left\{N^{subj.}, N^{obj.}\right\}$

(d) The next stage attaches 'Susan' to 'drank', yielding the following state, in which the bracketing condition is violated by the unit 'water':
current: $evaporated\left\{V_{N^{subj.}}, V_{N^{subj.}+N^{obj.}}\right\}$

open stack	**closed stack**
$drank, the, water, Susan\left\{V\right\}$ $\emptyset\left\{X_V^V\right\}$ $After\left\{P_V\right\}$	

In much the same fashion it can be shown that another sort of garden path type characterized by Pritchett can be explained using the same mechanisms. The famous examples of main clause / relative NP ambiguity, repeated here as (59), are parsed until the verb is reached. This yields the following two near final states:

(59) (a) The boat floated down the river.
current:

open stack	**closed stack**
$floated, down, the, river\left\{N^{subj.}\right\}$ $\emptyset\left\{X_V^V\right\}$	 $the, boat\left\{N^{subj.}, N^{obj.}\right\}$

(b) ¿The boat floated down the river sank.
current: $sank\left\{V_{N^{subj.}}\right\}$

open stack	**closed stack**
$floated, down, the, river\left\{N^{subj.}\right\}$ $\emptyset\left\{X_V^V\right\}$	 $the, boat\left\{N^{subj.}, N^{obj.}\right\}$

In both cases 'the boat' will now be attached to the top element of the open stack, yielding a successful order in the first case, and violating the non dominance constraint in the latter, as 'boat' is modified by 'floated' and therefore dominates it, and should therefore not be allowed to follow it in $\leq$. Had 'floated' not been ambiguous between active and passive form, 'boat' could be prevented from acting as its argument (for instance, by pushing on top of it a functional barrier at the top of the closed stack, or by preventing a verb to take its argument from another phase, a strategy that can be supported by other arguments as well), which would save the normal reduced relative from becoming a garden path as well.

The current mechanism is not sufficient to differentiate between sentences characterized by Pritchett as Complement Clause / Relative Clause Ambiguity. Using the current algorithm description, both (60) sentences will end up having a correct dominance order limited by $\leq$.

(60) (a) The man told the child that he was entertaining some people
$told \leq the \leq child \leq that \leq was \leq entertaining \leq some \leq people \leq he \leq the \leq man$

(b) ¿The man told the child that he was entertaining to smile
$told \leq the \leq child \leq that \leq was \leq entertaining \leq he \leq to \leq smile \leq the \leq man$

The correct complement order has been achieved: every argument follows its predicate, and the bracketing condition will not be breached, which is obviously undesirable. There are two ways for dealing with this. One follows inserting units that lack phonological input (such as traces, PROs etc.). The other has complexity motivated reasons, and would be dealt with in the next section.

4.3.2 The Desired Order of Complements

The current parser cannot deal with the difference between (60.a) and (60.b), repeated here as (61.a) and (61.b).

(61) (a) The man told the child that he was entertaining some people.

(b) ¿The man told the child that he was entertaining to smile.

The main reason is this: the parser builds its arguments in an order which is not directed by the data, nor by any of the objectives I have set above. The order of the arguments is determined arbitrarily by the initial description that I have provided for the algorithm. The parser, as it has been described so far, attaches arguments that follow the verb in the order they appear, and attaches the subject of the verb when all the other arguments have been exhausted. While this mimics the classical description of first building the VP and then the various functional heads, it causes this failure, as well as other problems I discuss below.

First and foremost, this causes a problem with the storage constraint. It is rather evident, that the main reason that the closed stack constraint is $O(n)$ rather than $O(1)$, has to do with the fact that we may accumulate subjects of embedded sentences. A sentence such as (62) can have an infinite number of subjects, as we can repeat 'that John told Mary' as many times as we wish.

(62) Dan told Jane *that John told Mary* that he is tired.

There is a possible solution for this problem, if we required the subject of the sentence to be attached to the verb before a new sentence begins. In this case we would keep only the subject of the last clause until it is attached, and the closed 'stack' would only require a finite amount of storage, or $O(1)$ storage. The open stack will still grow linearly, but perhaps this can also be dealt with.

Evidence in favour of this assumption can be drawn from sentences in Hebrew, which are not interpreted as ambiguous, although syntactically, they are. Consider the following data in Hebrew. The phenomena of 'right association', already familiar in other contexts, such as example (63), may explain why (64.a) is not perceived as an ambiguous sentence (even though syntactically it is). Once we associate the dative complement with 'said', we are reluctant to associate it again with another verb, especially since the other verb has only a rather weak need for a dative complement. However, this is not the case when we move the dative complement into a topic position, as in (64.b) and (64.c). The topicalized element remains available for interpretation as the argument of every verb that needs it, and the actual interpretation is determined by many factors, such as 'how much' each verb would 'miss' a dative argument, emphasis on the verb (the emphasized verb would be interpreted as missing the argument) and context.

(63) *dani* *yesaper* *layeladim* *ʃe-hu* *kvar* *kana* *lahem* *kelev*
Dani will-tell DAT-the-kids COMP-he already bought DAT-them dog
maxar
tomorrow
¿'Dani will tell the kids that he has already bought them a dog tomorrow'

(64) (a) *dani* *siper* *ʃe-hu* *natan* *sefer* *le-yosi*
Dani said COMP-he gave book DAT-Yosi
'Dani told (someone) he gave the book to Yosi'
Inaccessible: 'Dani told Yosi he gave the book (to someone)'

(b) *le-yosi* *dani* *siper* *ʃe-hu* *natan* *sefer*
DAT-Yosi Dani told COMP-he gave book
preferred: 'Dani told Yosi he gave the book (to someone)'
dis-preferred: 'It is Yosi, that Dani said he gave the book to'

(c) *le-yosi* *dani* *hevtiax* *ʃe-hu* *yaazor*
DAT-Yosi Dani promised COMP-he would-help
preferred: 'Dani promised (someone) that he would help Yosi'
dis-preferred: 'Dani promised Yosi that he would help (someone / Yosi)'

There are just two accounts that would allow a topicalized element to be interpreted as attached to the lower verb in some cases, and to the main verb in other cases. One is that topicalized elements are not placed by the parser in any position, and are interpreted at a later stage by some other module. A more economic account would find a place in the final unit chain that would allow the dative complement to be interpreted either as a complement of the main verb or as a complement of the embedded verbs. This position is apparent when we add yet another verb that can take a dative argument, such as in (65), which allows for all three interpretations. Since we have three bracketed sentences here, the only position that would allow this ambiguity is at the very end of the sentence, but only if the subject of each verb is attached before the clausal complement.

(65) *le-yosi* *dani* *siper* *ʃe-dana* *hevtixa* *ʃe-amos* *azar*
DAT-Yosi Dani told COMP-Dana promised COMP-Amos helped
'Dani told (someone) that Dana promised (someone) that Amos helped Yosi'
this is the preferred reading. Two other readings are also possible:
'Dani told Yosi that Dana promised (someone) that Amos helped (someone)'
'Dani told (someone) that Dana promised Yosi that Amos helped (someone)'

Using brackets, it is easy to see the difference between the two possibilities. In the original order of complements, the linear order created for this sentence would be (I add the

bracketing condition's brackets for clarity):

$$\Big[told \leq \Big[promised \leq [helped \leq Amos]_{helped} \leq Dana\Big]_{promised} \leq Dani\Big]_{told}$$

There is no position we can push 'to Yosi' that would allow it to be interpreted as an argument of both 'told' and 'helped'. However, if we attached the subject *before* the embedded clause, we would get:

$$\Big[told \leq Dani \leq \Big[promised \leq Dana \leq [helped \leq Amos]_{helped}\Big]_{promised}\Big]_{told}$$

All the closing brackets are aligned, which allows us to place the topicalized elements right after them, in order to get one of the three possible interpretations:

Dani told Yosi that Dana promised (someone) that Amos would help (someone)

$$\Big[told \leq Dani \leq \Big[promised \leq Dana \leq [helped \leq Amos]_{helped}\Big]_{promised} \leq Yosi\Big]_{told}$$

Dani told (someone) that Dana promised Yosi that Amos would help (someone)

$$\Big[told \leq Dani \leq \Big[promised \leq Dana \leq [helped \leq Amos]_{helped} \leq Yosi\Big]_{promised}\Big]_{told}$$

Dani told (someone) that Dana promised (someone) that Amos would help Yosi

$$\Big[told \leq Dani \leq \Big[promised \leq Dana \leq [helped \leq Amos \leq Yosi]_{helped}\Big]_{promised}\Big]_{told}$$

The question remains: why would topicalization re-create a missing ambiguity, why was the ambiguity missing in the first place. For these I refer the reader to §4.3.4, as this is, indeed, a right-association kind of problem.

If we accept this evidence, apparently needed to explain the lack of ambiguity in Hebrew sentences, and called for by our attempt to reduce processing complexity, we can use this as a solution to the sentence the original version of the parser failed to explain (60.b). If the algorithm places the subject of the main verb of (60), repeated here as (66), before it began the sentential complement, it has to choose between two options that are not compatible with each other: either place the verb *before* the complement clause, or *after* the relative clause. It chooses the complement interpretation (both are available since the noun 'knows' it can be a subject, which means the verb can take a complement clause), and this leads to the following final states:

(66) (a) The man told the child that he was entertaining some people
told ≤ *the* ≤ *child* ≤ *the* ≤ *man* ≤ *that* ≤ *was* ≤ *entertaining* ≤ *some* ≤ *people* ≤ *he*

(b) ¿The man told the child that he was entertaining to smile
told ≤ *the* ≤ *child* ≤ *the* ≤ *man* ≤ *that* ≤ *was* ≤ *entertaining* ≤ *he* ≤ *to* ≤

smile

While the first sentence still has an appropriate $\leq$ relation, the garden path sentence does not, as 'the man' intervenes between 'the child' and its sentential modifier (the relative clause), violating the bracketing condition, and correctly predicting a garden path.

Following the change suggested above, we get a uniform solution to both garden path sentence and the lack of expected ambiguity in Hebrew sentences (I think these examples can be translated to English as well). This is certainly beyond the strength of most deterministic and limited backtracking parsers.

4.3.3 A Swing at a Tighter Parser

As I have already suggested in the previous section, it may be possible to reduce the storage complexity of the parser to $O(1)$. In the various examples I have used above to track the working of the parser, two general tendencies could be seen: the reason for limiting the number of units the closed stack contains to $O(n)$ has only to do with the availability of embedding a sentence within another sentence, and this need would be eliminated by the modification suggested in the previous section.

This modification can draw support from Marathi, where sentential complements usually follow the verb, while noun complements usually preceded it, as in (67), given to me by Tejaswini Deoskar.

(67) *N1-ni* *N2-la* *sangitle* *ki* *N3-ni* *N5-la* *N4* *dile*
N1-erg N2-dat told COMP N3-erg N5-dat N4 gave
'N1 told N2 that N3 gave N4 to N5'

Marathi is generally considered to be a head-final language, so we would expect the complement clause to follow the verb, which is not the case. Marathi's configuration allows it to know all it has to know about the verb before commencing the interpretation of another verb with another subject.

The closed stack is not the only $O(n)$ storage in the parser's original description. The open stack grows just as quickly, as it gathers the verbs of every sentence and keeps them until the parse process finishes. To solve this, we should note that units pushed into the open stack usually get attached to the previous top, with a few notable exceptions such

as modifiers that precede their argument. This means that there might not be a need to maintain an $O(n)$ storage in the open stack either, and the algorithm could do with just $O(1)$ storage. This can allow us to restate our complexity goal from the objectives section: parsing in $O(1)$ storage complexity constraint would be better than parsing with $O(n)$ storage constraint. Such a constraint would make the linear time complexity requirement redundant, as linear time complexity follows from finite storage complexity.

4.3.4 Explaining Right Association

The effort to explain garden path as a reanalysis problem, for whatever reasons and descriptions used, cannot account for sentences that are difficult to process due to what I have labeled in §2.1.2 right association sentences. The parsing guidelines I offered above can manage to explain sentences such as (6.a), repeated here as (68.

(68) John will tell the kids he has already bought them a dog tomorrow

Consider the following parser state reached in the described algorithm when 'tomorrow' is the next word (I present this in terms of the original description, prior to the changes that should follow §4.3.2): they would not change essential bits of the data):
current: $tomorrow\,\{Adv.\}$

open stack	**closed stack**
$dog\,\{N^{obj.}\}$	
$a\,\left\{D^{N^{obj.}}\right\}$	
$bought, them\,\{V_{N^{obj.}+N^{subj.}}\}$	
$has\,\{V_V^V\}$	
$\emptyset\,\{X_V^V\}$	$already\,\{Adv.\}$
$tell, the, kids\,\{V_{V+N^subj.}\}$	$he\,\{N^{subj.}\}$
$\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}\}$

'tomorrow' can be attached neither to 'dog', nor to 'a', which will soon enough bring us to the following state:
current: $tomorrow\,\{Adv.\}$

open stack	**closed stack**
$bought, them, a, dog\,\{V_{N^{subj.}}\}$	
$has\,\{V_V^V\}$	
$\emptyset\,\{X_V^V\}$	$already\,\{Adv.\}$
$tell, the, kids\,\{V_{V+N^subj.}\}$	$he\,\{N^{subj.}\}$
$\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}\}$

'tomorrow' cannot modify 'bought' at the semantic level, but modularity requires that the syntactic module be unaware of word content. As an adverbial it can modify a verb and 'bought' is a verb. It is therefore attached, bringing us to the next state, in which the sentence ends:

current:

open stack	**closed stack**
$bought, them, a, dog, tomorrow\,\{V_{N^{subj.}}\}$	
$has\,\{V_V^V\}$	
$\emptyset\,\{X_V^V\}$	$already\,\{Adv.\}$
$tell, the, kids\,\{V_{V+N^subj.}\}$	$he\,\{N^{subj.}\}$
$\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}\}$

Everything is now popped out of the stack, as no further attachment can be made:

current:

open stack	**closed stack**
$bought, already, them, a, dog, tomorrow, already, he\,\{V\}$	
$has\,\{V_V^V\}$	
$\emptyset\,\{X_V^V\}$	
$tell, the, kids\,\{V_{V+N^subj.}\}$	
$\emptyset\,\{X_V^V\}$	$John\,\{N^{subj.}\}$

This immediately causes a violation of the bracketing condition: 'tomorrow' is not dominated by 'bought', and 'he' is. Notice that this analysis requires the parser to disregard that 'tomorrow' is incompatible with the (possibly) past marked verb that takes it as a modifier. Right association can also be demonstrated by sentences that do not require such parser 'blindness'[16].

That explanation holds, of course, for the Hebrew sentences that are not interpreted as ambiguous, although their syntactic structure is. Sentence (64.a), repeated here as (70), is considered unambiguous by readers, although it has two valid parse trees: one in which the PP is an argument of 'told' and the other in which the PP is an argument of 'gave'.

[16] Though the parser can be demonstrated to be that blind. I will use again Ken Barker's useful list of garden path sentences from http://www.site.uottawa.ca/~kbarker/garden-path.html:

(69) Every woman that admires a man that paints likes Monet

In reading this many speaker read 'like' rather than 'likes'. This may mean the parser does not use agreement features to assist it in the parsing process.

(70) *dani siper ʃe-hu natan sefer le-yosi*
Dani said COMP-he gave book DAT-Yosi
'Dani told (someone) he gave the book to Yosi'
Inaccessible: 'Dani told Yosi he gave the book (to someone)'

If we trace the parse process of the sentences, we would arrive to a result state in which 'he' follows 'to Yosi', and the unavailable interpretation would violate the bracketing condition.

I find it appealing, that in this model garden path and other sentences which are difficult to process arrive at the same condition when processed by the algorithm. Even better, the same explanation applies to sentences that do not demonstrate any processing difficulty, but rather a processing quirkiness: the unavailability of an ambiguous reading. It is appealing to explain these three phenomena by a single mechanism. Note that this mechanism by itself is derived primarily for independent reasons: the conditions have not been set to explain the data, but rather derive from general principles: complexity constraints.

4.3.5 On-line fixing

One of the interesting things about some garden path and right association sentences, is that *hearing* them does not confer the sense of confusion and cognitive effort that characterizes the reading of the very same sentences. Following Kedar (2006), it is easy to demonstrate that with a few already familiar examples. Sentence (35.b), repeated here as (71.a), does not cause conscious effort when we pause after the word 'drank' (I use the symbol | following Kedar (2006) to indicate a pause here, as in (71.b).

(71) (a) ¿After Susan drank the water evaporated.

(b) After Susan drank | the water evaporated.

Not all theories can deal with that, as it is not clear that we should now wait for another predicate and not attach 'water' to 'drank' right after the pause, which would cause a garden path in both Pritchett (1992) and Lewis (1998). Some theories do incorporate possible solutions, such as Kedar (2006) whose parser does not commit itself to structure until the end of the phonological phrase and Weinberg (1999) whose parser does not commit itself when a merge operation fails. My parser, both before and after the changes suggested in §4.3.2 can deal with this problem easily, as a pause may simply trigger an 'expect no further input' behaviour. When 'water' appears, no new elements can be

attached to 'drank'. This is not a plausible behaviour for a parser in which there is no notion of different states for phrase, parsers that treat the entire structure as available for processing.

More intriguing examples from Kedar (2006) are the classical main clause / relative NP local ambiguity sentences such as (33.b) repeated here as (72.a) which require not only a pause, but also a high tone to avoid making the wrong decision, as in (72.b)[17].

(72) (a) ¿The boat floated down the river sank.

(b) The boat floated down the river^{H} | sank.

(c) ¿The boat | floated down the river sank.

(d) ¿The boat^{H} | floated down the river sank.

(e) The boat | floated down the river^{H} | sank.

Here too, there is little hope for theories which to do not distinguish between phrases to which more elements can still be added, and phrases to which no more elements can be added, but the fact that a pause alone does not suffice, correlates well with my parser(s). Simply assuming 'no more input' will not do, as the parser will simply attach 'the boat' as the subject of the verb (since this is its default behaviour when there is no more input). The high tone here may signal 'perform unorthodox attachment' (attach the verb phrase to the noun rather than the other way around). Any alternative in which the erroneous attachment has already taken place at this stage does not allow the sentence to be saved by a high tone that appears afterward. Such theories would expect some phonetic variation at the site of the wrong attachment, such as the speculative (72.c) or (72.d), but this is not the case. However, combining the two pauses and the high tone as in (72.e), does provide the best cue to avoid the processing difficulty: apparently a better cue than the one provided exclusively by a high tone and a pause before the second verb.

[17] Here too, I follow Kedar's (2006) high tone symbol, x^{H}

4.4 Desiderata

4.4.1 Preserving Syntactic Notions

Every new mechanism for structure building requires preservation of notions which lie at the heart of previous theories. An example is Epstein's (1999) attempt to preserve generative syntactic terms such as c-command in Minimalism. I have already claimed that the (pre-)parsers I present are compatible with a variety of syntactic theories. However, if these parsers can be claimed to have some insight into the actual workings of the human parser, it may be desirable to use the trace of the algorithm to re-phrase some of these notions, thus ridding the rest of syntax of the need to provide an explanation for them.

4.4.2 Dealing with SOV languages

There is no commonly accepted explanation for the existence of the few garden path patterns that exist in head-final languages such as Japanese. Some even doubt whether the known examples really are garden paths, or simply an accumulation of different processing difficulties. Pritchett (1992), for example, correctly predicts (73) to be a garden path, as 'Frank-ni', first analyzed as the dative complement of 'syookai', does not dominate or govern the dative complement position of 'iwaseta' where it should end up, thus violating the OLLC.

(73) Pritchett (1992) example (362)

Frank-ni	*Tom-ga*	*Guy-o*	*syookai*	*suru*	*to*	*John-wa*	*iwaseta*
Frank-DAT	Tom-NOM	Guy-ACC	introduce	COMP	John	top	said-CAUSE

¿'John made Frank say Tom introduced Guy'

However, Prithcett incorrectly predicts sentences like (74) to cause garden path as well, as 'Yumiko-o' has to be re-analyzed as the object of the verb of the matrix clause though it is first analyzed as the object of the verb of the embedded clause.

(74) Mulders (2005) example (17)

∅	*Yumiko-o*	*yobidasita*	*kissaten-ni*	*nagai*	*koto*	*mata-seta*
∅	Yumiko-ACC	summoned	tea	room-LOC	long	time made wait

'(someone) made Yumiko wait for a long time in the tea room to which he summoned her'

Here too, the source position does not dominate or govern the target position. Pritchett's constraint, the OLLC, is violated, but no processing difficulty is encountered. There is quite a bit of discussion about the actual causes of this inconsistency. Mazuka and Itoh (1995) use it to claim garden paths in Japanese are caused not by reanalysis failure but rather by cumulative processing difficulties that lead to conscious effort, while Mulders (2005) opposes that view and manages to explain these counter-examples to Pritchett (1992) using a revised reanalysis constraint (but one which incorrectly predicts (73) to be incorrect as well).

My basic algorithm does not deal with garden paths in Japanese, simply because the basic Japanese structure already has an inverse $\leq$ order, for which there is a $\sqsubseteq$ order that is the correct parse tree of the sentence. Even when argument order is scrambled by some operation such as topicalization, a correct $\leq$ is kept. This is obviously a problem, and I think it can be traced back to the problems raised above in sections 4.3.2 and 4.3.3.

The real problem with analyzing Japanese as having an inverse $\leq$, is that the complexity constraints laid in the preliminaries are strained to the limit when Japanese sentences are parsed. While during the parsing of sentences in head-initial language the closed stack grows only by the subjects of the embedded sentences, the closed stack in head final languages contains each and every argument until the final verb is encountered, forcing the stack to contain the entire sentence in storage. Such an approach does not allow the next module to get partial information: the next module has to wait until the sentence is over. I speculate, but I have yet to prove, that the sort of parser suggested in §4.3.3 might be able to do the trick, and future work will have to look into that. Such work might benefit from the inclusion of other processing difficulties that I think have the same causes as garden paths: perhaps Japanese lacks 'proper' garden path sentences, but is abundant with 'unambiguous ambiguities', for instance.

5 Summary

In this work I tried to demonstrate how many attributes of the human parser can be explained by a model that requires the parser to complete the parse process in linear time, and under a (small and) finite storage constraint. The outcome of this work is not a parser per se, but rather a pre-parser, as a parse tree is not achieved. Surprisingly enough, though, many attributes of the human parser are indeed explained by this assumption: garden paths, right association, unambiguous ambiguities, and even the modularity assumption are all derived by this seemingly trivial assumption.

This work is yet to be re-implemented in a way that would be able to explain some more phenomena of processing, namely center embedding, and processing difficulties in head final language. Future work should address this issue, most probably by further constraining the complexity assumptions made in this work.

References

Barton, G., Berwick, R., 1985. Parsing with assertion sets and information monotonicity. In: Proceedings of IJCAI-85. IJCAI, Inc., Los Angeles, CA.

Epstein, S. D., 1999. Un-principled syntax: The derivation of syntactic relations. In: Epstein, S. D., Hornstein, N. (Eds.), Working Minimalism. MIT Press, pp. 317–345.

Frazier, L., Fodor, J. D., 1978. The Sausage Machine: a new two-stage parsing model. Cognition 6, 291–325.

Kedar, T., 2006. Music to my parser. Given as a talk at the Tel Aviv University Linguistics Department Colloquium. Ask the author for a copy: kedartal*at*gmail.com.

Kimball, J., 1973. Seven principles of surface structure parsing in natural language. Cognition 2, 15–47.

Knuth, D. E., 1976. Big omicron and big omega and big theta. SIGACT News 8, 18–24. URL `http://doi.acm.org/10.1145/1008328.1008329`.

Lewis, R. L., 1998. Reanalysis and limited repair parsing: Leaping off the garden path. In: Fodor, J. D., Ferreira, F. (Eds.), Reanalysis in Sentence Processing. Kluwer Academic, pp. 247–284.

Marcus, M. P., 1978. Theory of syntactic recognition for natural languages. Ph.D. thesis, MIT. URL `http://hdl.handle.net/1721.1/16176`.

Marcus, M. P., Hindle, D., Fleck, M. M., 1983. D-theory: Talking about talking about trees. In: Proceedings of the 21th ACL. Cambridge, MA.

Mazuka, R., Itoh, K., 1995. Can japanese speakers be led down the garden path? In: Mazuka, R., N., N. (Eds.), Japanese Sentence Processing. Lawrence Erlbaum Associates, pp. 295–329.

Mulders, I., 2002. Transparant parsing. Head-driven processing of verb-final structures. Ph.D. thesis, Utrecht University.

Mulders, I., 2005. Transparent parsing: phases in sentenece processing. In: McGinnis, M., Richards, N. (Eds.), MITWPL 49. MIT Working Papers in Linguistics, pp. 237–264.

Pritchett, B. L., 1992. Grammatical Competence and Parsing Performance. Chicago: University of Chicago Press.

Schneider, D. A., 1999. Parsing And Incrementality. Ph.D. thesis, University of Delaware. URL `http://continuity.ist.psu.edu/559093.html`.

Tomita, M., 1987. An efficient augmented-context-free parsing algorithm. Computational Linguistics 13, 31–46. URL `http://doi.acm.org/10.1145/30000.26390`.

Weinberg, A., 1993. Parameters in the theory of sentence processing: Minimal commitment theory goes east. Journal of Psycholinguistic Research 22, 339–364.

Weinberg, A., 1999. A minimalist theory of human sentence processing. In: Epstein, S. D., Hornstein, N. (Eds.), Working Minimalism. MIT Press, pp. 283–316. URL `http://www.umiacs.umd.edu/users/weinberg/lamp-024.html`.