UNIVERSITY OF CALIFORNIA

Los Angeles

The S-Parameters: A Minimalist Approach to Syntax

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Linguistics

by

Andi Wu

1993

Contents

1 Introduction 1

2 The Spell-Out Parameters 5
- 2.1 The Notion of Spell-Out . . . 5
  - 2.1.1 The Minimalist Program . . . 5
  - 2.1.2 The Timing of Spell-Out . . . 8
  - 2.1.3 The S-Parameters . . . 10
- 2.2 The S(M)-Parameter and Word Order . . . 12
  - 2.2.1 An Alternative Approach to Word Order . . . 12
  - 2.2.2 The Invariant X-bar Structure Hypothesis (IXSH) . . . 14
  - 2.2.3 Modifying the IXSH . . . 19
- 2.3 Interaction of S(F)- and S(M)- Parameters . . . 24
- 2.4 Summary . . . 28

3 An Experimental Grammar 29
- 3.1 The Categorial and Feature Systems . . . 32
  - 3.1.1 Categories . . . 32
  - 3.1.2 Features . . . 33
  - 3.1.3 Features and Categories . . . 38
  - 3.1.4 The Spell-Out of Features . . . 39
- 3.2 The Computational System . . . 41
  - 3.2.1 Lexical Projection . . . 42
  - 3.2.2 Generalized Transformation . . . 49
  - 3.2.3 Move-α . . . 54
- 3.3 Summary . . . 66

4 The Parameter Space 67
- 4.1 The Parameter Space of S(M)-parameters . . . 69
  - 4.1.1 An Initial Typology . . . 70
  - 4.1.2 Further Differentiation . . . 73
  - 4.1.3 Some Set-Theoretic Observations . . . 76
- 4.2 Other Parameters . . . 78
  - 4.2.1 HD-Parameters . . . 78
  - 4.2.2 Value Combinations that Predict Auxiliaries . . . 81
  - 4.2.3 S(F)-parameters . . . 84
- 4.3 Case Studies . . . 86
  - 4.3.1 English: An SVO Language . . . 87
  - 4.3.2 Japanese: An SOV language . . . 90

4.3.3 Berber: A VSO Language . . . 93
4.3.4 German: A V2 Language . . . 95
4.3.5 Chinese: A Head-Final SVO Language . . . 98
4.3.6 French: A Language with Clitics . . . 100
4.4 Summary . . . 101

5 Setting the Parameters **102**
5.1 Basic Assumptions . . . 102
5.1.1 Assumptions about the Input . . . 102
5.1.2 Assumptions about the Learner . . . 104
5.2 Setting S(M)-Parameters . . . 105
5.2.1 The Ordering Algorithm . . . 105
5.2.2 The Learning Algorithm . . . 109
5.2.3 Properties of the Learning Algorithm . . . 110
5.2.4 Learning All Languages in the Parameter Space . . . 112
5.3 Setting Other Parameters . . . 114
5.3.1 Setting HD-Parameters . . . 114
5.3.2 Setting S(F)-Parameters . . . 117
5.4 Acquiring Little Languages . . . 120
5.4.1 Acquiring Little English . . . 121
5.4.2 Acquiring Little Japanese . . . 122
5.4.3 Acquiring Little Berber . . . 123
5.4.4 Acquiring Little Chinese . . . 125
5.4.5 Acquiring Little French . . . 126
5.4.6 Acquiring Little German . . . 128
5.5 Summary . . . 134

6 Parsing with S-Parameters **135**
6.1 Distinguishing Characteristics of the Parser . . . 135
6.2 A Prolog Implementation . . . 146
6.2.1 Tree-Building . . . 147
6.2.2 Feature-Checking . . . 148
6.2.3 Leaf-Attachment . . . 150
6.2.4 The Parser in Action . . . 151
6.2.5 Alternative Implementations . . . 160
6.2.6 Universal vs. Language-Particular Parsers . . . 161
6.3 Summary . . . 162

7 Final Discussion **163**
7.1 Possible Extensions . . . 163
7.2 Potential Problems . . . 167
7.3 Concluding Remarks . . . 172

A Prolog Programs **173**
A.1 . . . 173
A.2 . . . 176
A.3 . . . 179
A.4 . . . 180
A.5 . . . 183
A.6 . . . 184

A.7 190
A.8 201

B Parameter Spaces **205**
B.1 205
B.2 209
B.3 211
B.4 214
B.5 219
B.6 222
B.7 226

C Partial Ordering of Parameter Settings **229**

D Learning Sessions **238**
D.1 238
D.2 240
D.3 242
D.4 245
D.5 252
D.6 256

ABSTRACT OF THE DISSERTATION

The S-Parameters:

A Minimalist Approach to Syntax

by

Andi Wu

Doctor of Philosophy in Linguistics

University of California, Los Angeles, 1993

Professor Edward P. Stabler, Jr., Chair

This thesis explores a new parametric syntactic model which is developed from the notion of Spell-Out in the Minimalist framework (Chomsky 1992). The main hypothesis is that languages are identical up to the point of Spell-Out: the sets of movements and morphological features are universal but different languages can have different word orders and morphological paradigms depending on which movements or features are visible. We can thus account for a wide range of cross-linguistic variation by parameterizing the spell-out options.

The model proposed in this thesis has two sets of Spell-Out parameters. The values of S(M)-parameters determine which movements occur before Spell-Out in a given language. Different settings of these parameters may generate differernt word orders. The values of S(F)-parameters determine which features are morphologically realized. Different value combinations of these parameters may produce different morphological paradigms. The values of these two sets of parameters can also interact, resulting in such syntactic phenomena as auxiliaries, grammatical particles and expletives.

Computational experiments are conducted on a minimal version of this model in terms of language typology, language acquisition and language processing. It is found that the parameter space of this model can accommodate a wide variety of languages. In addition, all these languages are found to be learnable via a linguistically motivated parameter-setting algorithm. Finally, this new parametric system can also lead to the construction of a parser which is more universal. The experimental results, though preliminary in nature, indicate that the line of research suggested in this thesis is worth pursuing.

Chapter 1

Introduction

The goal of this thesis is to explore a new parametric system within the theoretical framework of *Principles and Parameters* (P&P theory hereafter) which is represented by Chomsky (1981, 1982, 1986, 1989, 1992) and many other works in generative syntax. The basic assumption of this theory is the following:

> Children are endowed at birth with a certain kind of grammatical knowledge called Universal Grammar (UG) which consists of a number of universal principles along with a number of parameters. Each of the parameters has a number of possible values and any possible natural language grammar results from a particular combination of those parameter values. In acquisition, a child's task is to figure out the parameter setting of a given language on the basis of the sentences he/she hears in this language.

At the present stage, the P&P theory is still more of a research paradigm than a fully-developed model. No final agreement has been reached as to what the principles are and how many parameters are available. In this thesis, I will propose a set of parameters and investigate the consequences of this new parametric system in terms of language typology, language acquisition and language processing.

The syntactic model to be explored in this thesis was inspired by some recent developments in syntactic theory exemplified by Chomsky (1992) and Kayne (1992, 1993). In his *A Minimalist Program for Linguistic Theory*, Chomsky introduced the notion of *Spell-Out* into our theory. Spell-Out is a syntactic operation which feeds a syntactic representation into the PF (phonetic form) component where a sentence is pronounced. The ideal assumption is that languages have identical underlying structures and all surface differences are due to Spell-Out. In my view, the notion of Spell-Out applies to both movements and features. When a movement is spelled out, we see overt movement. We see overt morphology when one or more features are spelled out. Different languages can have different word orders and different morphological paradigms if they can choose to spell out different subsets of movements or features. Since so many cross-linguistic variations have come to

be associated with Spell-Out, it is natural to assume that most parameterization is to be found this syntactic operation. We need a set of parameters which determine which movements are overt and which features are morphologically visible. The main objective of this thesis is to propose and test out such a set of parameters.

The new parameters to be proposed in this thesis are all related to Spell-Out. We will thus call those parameters *S-parameters*. There are two types of S-parameters: the S(M)-parameters which control the spell-out of movements and the S(F)-parameters which control the spell-out of features. We will see that the value combinations of these parameters can explain a wide range of cross-linguistic variation in word order and inflectional morphology. They also offer an interesting account for the distribution of certain functional elements in languages, such as auxiliaries, expletives and grammatical particles. In terms of language acquisition, this new parametric system also has some desirable learnability properties. As we will see, all the languages generated in this new parameter space are learnable. There exists a parameter setting algorithm whereby every language can have its correct parameter value combination identified. The new parameter system is interesting in terms of language processing as well. It will be demonstrated that the new model proposed here may enable us to construct a universal parser which can be used to parse any language in our parameter space by setting the parameters accordingly.

All the experiments on this parametric syntactic model were performed by a computer. The syntactic system and the learning algorithm are implemented in Prolog. The parameter space was searched exhaustively using a Prolog program which finds every language that can be generated by our grammar. The learning algorithm has also been tested against every possible language in our parameter space. It should be pointed out here that in this thesis the term *language* will often be used in a special sense to refer to an abstract set of strings. In order to concentrate on basic word order and basic inflectional morphology and study those properties in a wide variety of languages, I will represent the "sentences" of a language in an abstract way which shows the word order and overt morphology of a sentence but nothing else. An example of this is given in (1).

(1) `s-[c1] o-[c2] v-[tns,asp]`

The string in (1) represents a sentence in some SOV language. The lists attached to S, O and V represent features that are morphologically realized. The "words" we find in (1) are therefore

(a) a subject NP which is inflected for case;

(b) an object NP which is inflected for a different case; and

(c) a verb which is inflected for tense and aspect.

A language then consists of a set of such strings. Here is an example:

(2)
```
{ s-[c1] v-[tns,asp],
  s-[c1] o-[c2] v-[tns,asp],
  o-[c2] s-[c1] v-[tns,asp]
}
```

What (2) represents is a verb final language where the subject and object NPs can scramble. The NPs in this this language are overtly marked for case and the verb in this language is inflected for tense and aspect. Such abstract string representation makes it possible to let the computer read strings from any "language". In many situations we will be using the term "language" to refer to such a set of strings. To remind ourselves that this is not a real language, we will often put this term in quotes. The fact that we will be conducting computer experiments with these artificial languages does not mean that we will be detached from reality, however. Many real languages will also be discussed in connection with these simplified languages. Although the representation in (2) is fairly abstract, it is not hard to see what natural language it may represent. As a matter of fact, most languages generated in our parameter space can correspond to some natural languages. The results of our experiments are therefore empirically meaningful. In the course of our discussion, we will often furnish real language examples to illustrate those abstract languages.

The rest of this thesis is organized as follows.

Chapter 2 examines the notion of Spell-Out in detail and considers its implications for linguistic theory. After a brief description of the Minimalist model and a critique of some existing parameter systems, I propose an alternative syntactic model where cross-linguistic variations in word order and morphology are mainly determined by two sets of S-parameters: S(M)-parameters and S(F)-parameters. Arguments for this new model are given and the potential of the new system is illustrated with examples from natural languages.

Chapter 3 describes the syntactic model to be used in the experiments. In order to implement the new system in Prolog and use computer search to find out all the consequences of this system, our grammatical model must be specified in every detail. Since a full specification which defines every aspect of a syntactic model is impossible at the present stage, I will define a partial grammar which is sufficient for deriving the basic word order and basic morphological system of any language. The partial grammar includes a categorial system, a feature system, a parameter system, and a computational system whose basic operations are Lexical Projection, Generalized Transformation and Move-α. The specification follows standard assumptions in most cases, but some innovations are incorporated.

In Chapter 4, we consider the consequences of our experimental grammar in terms of the language typology it predicts. It is found that our new parameter space is capable of accommodating a wide range of linguistic phenomena. In terms of word order, we are able to derive all basic word orders

(SVO, SOV, VSO, VOS, OSV, OVS and V2) as well as many kinds of scrambling. In terms of inflectional morphology, we can get a variety of inflectional paradigms. We also find parameter settings which can account for the appearance of auxiliaries, grammatical particles and clitics. Many value combinations in the parameter space will be illustrated by examples from natural languages.

The topic of Chapter 5 is learnability. We consider the question of whether all the "languages" in our parameter space can be learned by setting parameters. As we will see, the languages can be identified through the learning paradigm of Gold's (1967) identification by enumeration if the hypothetical settings are enumerated in a certain order. It turns out that this ordering of hypotheses can be deduced from some general linguistic principle, namely the Principle of Procrastinate. Our experiments show that, with this linguistically motivated ordering algorithm, the learner can converge on any particular grammar in an incremental fashion without the need of negative evidence or input ordering.

In Chapter 6, we discuss the implications of our parametric syntactic model for language processing. We will see that this new model can result in a parser which is more universal in nature. The parser to be presented in this chapter is capable of processing every language in the parameter space without a single change in the parser itself. The only thing that changes from language to languages is the value combinations of parameters which the parser consults throughout the parsing process. The reason why we are able to do so is that the new model has made tree-building and chain-building almost invariable across languages. The only important language-particular decision the parser has to make is where to attach the leaves.

Chapter 7 concludes the thesis by considering possible extensions and potential problems of the present model. One extension to be discussed is how our approach can be applied to the word order variation within DP/NP and PP. It seems that we can account for the internal structures of these phrase using a similar approach. The main potential problem to be considered is the depedency of our model on certain systactic assumptions. We realize that the particular model we have implemented does rely on some theoretical assumptions which are yet to be proved. However, the general approach we are taking here can remain valid no matter how the specific assumptions change. Our model can be updated as the research in linguistic theory advances.

Chapter 2

The Spell-Out Parameters

In this chapter, we examine the notion of *Spell-Out* and consider its implications for cross-linguistic variations in word order and morphology. We will see that a considerable amount of word order variation can be explained in terms of the Spell-Out of movements, while the Spell-Out of features can account for much morphological variation. Two sets of Spell-Out parameters are proposed: the S(M)-parameters which determine the Spell-Out of movements and the S(F)-parameters which are responsible for the Spell-Out of features. We shall see that the parameter space created by these two sets of parameters can cover a wide range of linguistic phenomena.

This chapter will only present a very general picture of how things might work in this new model. The full account is given in Chapter 3 and Chapter 4. In the brief sketch that follows, we will start by looking at the notion of Spell-Out in Chomsky (1992). This notion will then be applied first to movement and then to infectional morphology. Finally, we will have a quick glance at how the Spell-Out of movements and the Spell-Out of features might interact.

2.1 The Notion of Spell-Out

2.1.1 The Minimalist Program

Spell-Out as a technical term is formally proposed in Chomsky's (1992) Minimalist Program for Linguistic Theory (MPLT hereafter), though the notion it denotes has been around for some time. The most salient feature of the Minimalist framework[1] is the elimination of D-structure and S-structure. Chomsky reduced the levels of representation to nothing but the two interfaces: *Phonetic Form* (PF), which interacts with the articulatory-perceptual system, and *Logical Form* (LF) which interacts with the conceptual-intentional system. Consequently, grammatical constraints have come to be associated with these two interface levels only. Most of the well-formedness conditions that

[1] Throughout this thesis I will try to make a distinction between MPLT and the Minimalist *framework*. The former refers to the specific model described in MPLT while the latter refers to the general approach to syntax initiated by MPLT.

used to apply at *D-structure* (DS) and *S-structure* (SS) have shifted their domain of application to either PF and LF. In this new model, *structural descriptions* (SDs) are generated from the lexicon and the SDs undergo syntactic derivation until they become legitimate objects at both PF and LF. Given a SD which consists of the pair (π, λ)[2], " ... a derivation D *converges* if it yields a legitimate SD; otherwise it *crashes*; D *converges at PF* if π is legitimate and *crashes at PF* if it is not; D *converges at LF* if λ is legitimate and *crashes at LF* if it is not" (MPLT p7). The legitimacy of PF and LF representations will be discussed later.

The derivation is carried out in the *computational system* which consists of three distinct operations: *lexical projection* (LP),[3] *generalized transformation* (GT), and *move-α*.

LP "selects an item X from the lexicon and projects it to an X-bar structure of one of the forms in (3), where $X = X^0 = [_x X]$." (MPLT, p30).

(3) (i) X

(ii) $[_{x'} X]$

(iii) $[_{x''}[_{x'} X]]$

The generation of a sentence typically involves the projection of a set of such elementary phrase-markers (P-markers) which serve as the input for GT.

GT reduces the set of phrase-markers generated by LP to a single P-marker. The operation proceeds in a binary fashion: it "takes a phrase-marker K^1 and inserts it in a designated empty position ø in a phrase-marker K, forming the new phrase-marker K^*, which satisfies X-bar theory" (MPLT, p30). In other words, GT takes two trees K and K^1, "targets" K by adding ø to K, and then substitutes K^1 for ø. The P-markers generated by LP are combined pair-wise in this fashion until no more reduction is possible.

Move-α is necessary for the satisfaction of LF constraints. Some constituents in the sentence have to be licensed or checked in more than one structural position and the only way to achieve this kind of multiple checking is through movement. Unlike GT which operates on pairs of trees, mapping (K, K^1) into K^*, move-α operates on a single tree, mapping K to K^*. It "targets K, adds ø, and substitutes α for ø, where α in this case is a phrase within the targeted phrase-marker K itself. We assume further that the operation leaves behind a trace t of α and forms the chain (α, t)." (MPLT, p31).

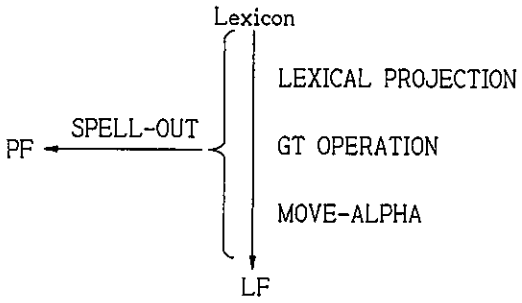
There is an additional operation called *Spell-Out* in the computational system. This operation feeds the SD being derived into the PF component. The derivation of a sentence can consist of a number of intermediate SDs but only one of them is actually pronounced or heard. The function of Spell-Out is to select such an SD. According to Chomsky, Spell-Out can occur at any point in the course of derivation. Given a sequence of SDs in the derivation, $< SD_1, SD_2, ..., SD_n >$, each SD_i

[2] π stands for the PF representation and λ for the LF representation.

[3] Chomsky did not use the term *lexical projection*, but the operation denoted by this term obviously exists.

representing a derivational step, the system can in principle choose to spell out any $SD_i, 1 \leq i \leq n$[4]. This notion of Spell-Out is illustrated in (4) where the curly bracket is meant to indicate that Spell-Out can occur anywhere along the line.

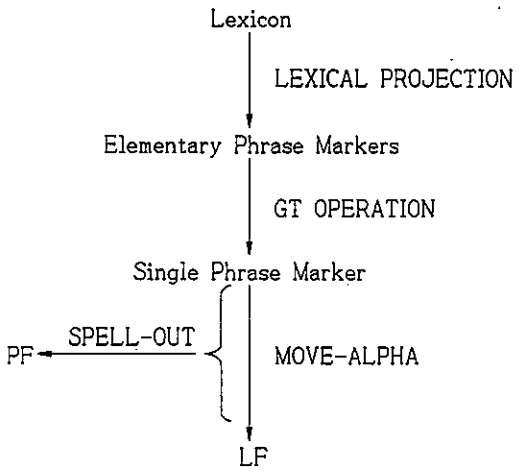
(4)



However, not every SD that we choose to spell out is acceptable to the PF component. Only those SDs which are legitimate objects at PF can be pronounced. In other words, the SD being fed into PF must at least satisfy the PF requirements. Once these requirements are met, an SD can be spelled out regardless of how many LF constraints have been satisfied.

One of the PF constraints that Chomsky has proposed requires that the input to PF be a single P-marker. If the representation being spelled out "is not a single phrase marker, the derivation crashes at PF, since PF rules cannot apply to a set of phrase markers and no legitimate PF representation π is generated." (MPLT, p30). In other words, a given P-marker cannot be spelled out until all its subtrees have been projected and reduced to a single tree. In normal cases, therefore, Spell-Out must occur after the completion of LP and GT.[5] This leads to the conclusion that Spell-Out can only apply in the process of move-α. So (5) is a more realistic picture of the derivational process.

(5)



[4] The derivation may proceed in more than one way. In that case, we can have different intermediate SDs depending on which particular derivational procedure is being used.

[5] We might get sentence fragments or a broken sentence if Spell-Out occurs before the completion of GT.

This diagram may seem to suggest a sequencing of the computational operations, with LP preceding GT which in turn precedes move-α. Such sequencing is possible but not always necessary. The picture is intended to be a logical description of linguistic theory rather than a flow chart for procedural computation. In actual language production and language comprehension, these computational operations can be co-routined. For instance, GT operations may be interleaved with movement operations. The real message the diagram is supposed to convey is the fact that only single trees can be accepted by PF and the logical consequence that Spell-Out can only occur after GT is complete.

2.1.2 The Timing of Spell-Out

Now let us take a closer look at Spell-Out which, as we have argued, normally occurs in the process of move-α where this operation is free to apply at any time. What is the consequence of this freedom? Before answering this question, we had better find out exactly what happens in move-α. In the pre-Minimalist P&P model, some movements are forced by S-structure requirements and some by LF requirements. The ones that are forced by SS constraints must take place in overt syntax. In our current terminology, we can say that these movements must occur before Spell-Out. The movements forced by LF requirements, however, can be either overt or covert. A typical example is wh-movement which is forced by the scope requirement on wh-phrases. It has been generally accepted since Huang (1982) that the scope requirement is satisfied at SS in languages like English and at LF in languages like Chinese. This is why the wh-phrase is sentence-initial in English but in situ in Chinese. Now that S-structure is gone, all movements are forced by LF requirements. Consequently, every movement has become an LF movement which, like wh-movement, can be either overt or covert. The Case Filter and the Stray Morpheme Filter (Lasnik 1981)[6] , for instance, have become LF *checking* requirements and the A-movement and head movement they motivate do not have to take place in overt syntax anymore.

It should be mentioned here that all LF requirements in the Minimalist framework are checking requirements. An SD is a legitimate object at LF only if all its features have been checked. In cases where the checking involves two different structure positions, movement is necessary. In fact, movement takes place for no other reason than feature-checking in this model. The visibility of a movement depends on the *timing* of feature-checking. It is visible if the relevant feature is checked before Spell-Out and invisible if it is checked after Spell-Out. Now the question is why some features are checked before Spell-Out. According to Chomsky's Principle of Procrastinate (MPLT, p43) which requires that overt movement be avoided as much as possible, the optimal situation should be the one where every movement is covert. There must be some other requirements that force a movement to occur before Spell-Out. In Chomsky's model, overt movement is forced by a PF constraint which

[6]This filter requires that morphemes designated as affixes be "supported" by lexical material at PF. It is the primary motivation for V-to-I raising or *do*-support.

requires that "strong" features be checked before Spell-Out. " ... 'strong' features are visible at PF and 'weak' features invisible at PF. These features (i.e. those features that are visible, AW[7]) are not legitimate objects at PF; they are not proper components of phonetic matrices. Therefore, if a strong feature remains after Spell-Out, the derivation crashes." (MPLT, p43) To prevent a strong feature from being visible at PF, the checking of this feature must be done before Spell-Out. Once a feature is checked, it disappears and no PF constraint will be violated. He cites French and English to illustrate this: "the V-features of AGR are strong in French, weak in English. ... In French, overt raising is a prerequisite for convergence; in English, it is not." (MPLT, p43) The combined effect of this "Strong Feature Filter" and the Principle of Procrastinate is a precise condition for overt movement: a movement occurs before Spell-Out if and only if the feature it checks is strong. This account is very attractive but it is far from perfect, as we will see later when we come to an alternative account in 2.1.3.

The timing of feature-checking and consequently the timing of movement are obviously relevant to word order. This is clearly illustrated by wh-movement which checks the scope feature. This movement is before Spell-Out in English and after Spell-Out in Chinese. As a result, the wh-phrases in these two languages have different surface distributions. Now that *every* movement has the option of being either overt or covert, the amount of word order variation that can be attributed to movement is much greater. As we will see in 2.2.2, given current syntactic assumptions which incorporate the VP-Internal Subject Hypothesis[8] (Koopman and Sportiche 1985, 1990, Kitagawa 1986, Kuroda 1986, Speas and Fukui 1986, Sportiche 1990, etc.) and the Split-Infl Hypothesis[9] (Pollock 1989, Belletti 1990, Chomsky 1991, etc.), it is possible to derive all the basic word orders (including SVO, SOV, VSO, V2, VOS, OSV and OVS) just from movement. This suggests that movement can have a much more important role to play in word order variation than we have previously thought. We may even begin to wonder whether *all* the variation in word order can be accounted for in terms of movement. If so, no variation in the X-bar component will be necessary. This idea has in fact been proposed in Kayne (1992, 1993) and implemented in a specific model by Wu (1992, 1993). We will come back to this in 2.2.2.

There is another assumption in the Minimalist theory which has made the prospect of deriving word order variations from movement a more realistic one. This is the assumption that all lexical items come from the lexicon fully inflected. In pre-Minimalist models, a lexical root and its inflectional morphology are generated separately in different positions. To pick up the morphology, the lexical root must move to the functional category where the inflections reside. For instance, a verb must move to Infl to get its tense morphology and a subject NP must move to the Spec of IP to be assigned its case morphology. Without movement, verbs and nouns will remain uninflected.

[7] Comment added by Andi Wu

[8] This hypothesis assumes that every argument of a VP (including the subject) is generated VP-internally.

[9] This hypothesis assumes a more articulated Infl structure where different functional elements such as Tense and Agreement count as different categories which can head their own projections.

This assumption that lexical roots depend on movement for their inflectional morphology runs into difficulty whenever we find a case where the verb or noun is inflected but no movement seems to have taken place. It has been generally accepted since Pollock (1989) that the English verb does not move to the position where agreement morphology is supposed to be located. To account for the fact that verbs are inflected for subject-verb agreement in English, we have to say that, instead of the verb moving up, the inflectional morphology is lowered onto the verb. In the Minimalist theory, however, lowering is prohibited. The requirement that each move-α operation must extend the target has the effect of restricting movement to raising only. At first sight, we seem to be in a dilemma: lowering is not permitted, but without lowering the inflectional morphology will be stranded in many cases. But this problem does not exist in the Minimalist model. In this model, words come from the lexicon fully inflected. Verbs and nouns "are drawn from the lexicon with all of their morphological features, including Case and ϕ-features" (MPLT, p41). They no longer have to move in order to pick up the inflections. Therefore, whether they carry certain overt morphological features has nothing to do with movement. Movement is still necessary, but the purpose of movement has changed from *feature-assignment* to *feature-checking*. The morphological features which come with nouns and verbs must be checked in the appropriate positions. For instance, a verb must move to T(ense) to have its tense morphology checked and a noun must move to the Spec of some agreement phrase to have its case and agreement morphology checked. These checking requirements are all LF requirements. Therefore, the movements involved in the checking can take place either before or after Spell-Out. The cases where lowering was required are exactly those where the checking takes place after Spell-Out. As far as the empirical coverage is concerned, this checking story is equivalent to the lowering story, but the former is conceptually more appealing since movement operations are now restricted to raising only.

2.1.3 The S-Parameters

We have seen that the timing of Spell-Out can vary and the variation can have consequences in word order. We have mentioned Chomsky's account of this variation: a movement occurs before Spell-Out just in case the feature it checks is "strong". Now, what is the distinction between strong and weak features? According to Chomsky, this distinction is morphologically based. He did not make this claim very explicit, but the idea he wants to suggest is clear: a feature is strong if it is realized in overt morphology and weak otherwise.[10] Let us assume that there is an underlying set of features which are found in every language. A given feature is realized in overt morphology when this feature is spelled out. Then the PF constraint in Chomsky's system simply says that a feature must be checked before it is spelled out. Given the Principle of Procrastinate, a movement will

[10]Chomsky did not use the term "overt morphology" and used "rich morphology" instead. The agreement morphology in French, for example, is supposed to be richer than that in English. In this way, French and English can be different from each other even though both have overt agreement morphology. Unfortunately, the concept of "richness" remains a fuzzy one. Chomsky did not tell us how the rich/poor differentiation is to be computed.

occur before Spell-Out just in case the morphological feature(s) to be checked by this movement is overt. This bijection between overt movement and overt morphology is conceptually very appealing. If it is true, syntactic acquisition will be easier, since overt morphology and overt movement will be mutually predictable in that case. The morphological knowledge children have acquired can help them acquire the syntax while their syntactic knowledge can also aid their acquisition of morphology. We will indeed have a much better theory if this relationship actually exists. Unfortunately, the bijection does not seem to hold in every language.[11] Counter-examples to this claim come in two varieties. On the one hand, there are languages like Chinese where we find overt movement but not overt morphology. As has been assumed in Cheng (1991) and Chiu (1992), the subject NP in Chinese moves out of the VP-shell to a higher position. But there is no morphological motivation for this movement, for this NP carries no inflectional morphology at all. On the other hand, there exist languages like English where we find overt morphology but not overt movement. In view of the fact that agreement features are spelled out in English, the verbs in English are expected to move as high as those in French. This is not the case, as is well known. If we insist on the "iff" relationship between overt movement and overt morphology, we will face two kinds of difficulties. In cases of overt movement without overt morphology, the Principle of Procrastinate is violated. We find movements that occur before Spell-Out for no reason. In cases of overt morphology without overt movement, the PF constraint will be violated which requires that overt features be checked before Spell-Out.[12]

There is an additional problem with this morphology-based explanation for overt/covert movement. Apparently, not all movements have a morphological motivation. V-movement to C and XP-movement to Spec of CP, for example, do not seem to be always morphologically related. They are certainly related to feature-checking, but these features are seldom morphologically realized.[13] Why such extremely "weak" features should force overt movement in many languages is a puzzle.

Since the correspondence between overt morphology and overt movement is not perfect, I will not rely on the strong/weak distinction for an explanation for the timing of feature-checking. Instead of regarding overt morphology and overt movement as two sides of the same coin, let us assume for the the time being that these two phenomena are independent of each other. In other words, whether a feature is spelled out and whether the feature-checking movement is overt will be treated as two separate issues. We will further assume that both the spell-out of features and the spell-out of the feature-checking movement can vary arbitrarily across languages. I therefore propose that two *Spell-Out Parameters (S-Parameters)* be hypothesized. The first S-parameter determines whether a given feature is spelled out. The second one determines whether a given feature-checking movement

[11] Chomsky can of course say that the strong/weak distinction is just a higher idealization. In this sense, many existing languages have deviated from the ideal grammar.

[12] All the movements in the Minimalist theory involves raising. Therefore, no checking can be performed through lowering.

[13] We do not want to exclude the possibility that these features can be realized in some languages or some special visible forms, such as intonation and stress.

occurs before Spell-Out. Let us call the first one the S(F)-parameter ("F" standing for "feature") and the second one the S(M)-parameter ("M" standing for "movement"). Both parameters are binary with two possible values: 1 and 0. When S(F) is set to 1, the feature it is associated with will be morphologically visible. It is invisible when S(F) is set to 0. The S(M)-parameter affects the visibility of movement. When it is set to 1, the relevant movement will be overt. The movement will be covert if S(M) is set to 0.

The S(F)-parameters determine the morphological paradigm of a language. A language has overt agreement just in case the S(F)-parameter for agreement features are set to 1, and it has an overt case system just in case the S(F)-parameter for the case features is set to 1. Given a sufficiently rich set of features, the value combinations of S(F)-parameters can result in any inflectional system we find in natural languages. All this is conceptually very simple and no further explanation is needed. The exact correspondences between S(F)-parameters and morphological paradigms will be discussed in Chapter 4 after we have defined the feature system in Chapter 3.

The S(M)-parameters, on the other hand, determine (at least partially) the word order of a language. How this works is not so obvious. So we will devote the next section (2.2) to the discussion of this question. In 2.3 we will consider the relationship between S(F)-parameters and S(M)-parameters.

2.2 The S(M)-Parameter and Word Order

In this section, we consider the question of how word order variation can be explained in terms of parameterization. We will first look at the traditional approach where word order is determined by the values of *head-direction parameters*[14] (hereafter HD-parameters for short) and then examine an alternative approach where S(M)-parameter values are the determinants of word order. The two approaches will be compared and a decision will be made as to what kind of parameterization we will adopt as a working hypothesis.

2.2.1 An Alternative Approach to Word Order

Traditionally, word order has been regarded mainly as a property of phrase structure. It is assumed that different languages can generate different word orders because their phrase structure rules can be different. In the Principles and Parameters theory, cross-linguistic variations in basic word order are often explained in X-bar-theoretic terms. The basic phrase structure rules of this theory are all of the following forms:

(6) $XP \Rightarrow \{ \bar{X}, (\textit{specifier}) \}$

$\bar{X} \Rightarrow \{ X, (\textit{complement}) \}$

[14] Various names have been given to this parameter in the literature. The one adopted here is from Atkinson (1992). Other names include *$\bar{X}$-parameters* (e.g. Gibson and Wexler (1993)) and *head parameters* (e.g. Ref??)

The use of curly brackets indicates that the the constituents on the right-hand side are unspecified for linear order. Which constituent precedes the other in a particular language depends on the values of HD-parameters. There are two types of HD-parameters: the specifier-head parameter which determines whether the specifier precedes or follows $\bar{X}$ and the complement-head parameter which determines whether the complement precedes or follows X. (*cf* Jackendoff (1977), Stowell (1981), Koopman (1983), Hoekstra (1984), Travis (1984), Chomsky (1986), Nyberg (1987), Gibson and Wexler (1993), etc.) The values of these parameters are language-particular and category-particular. When acquiring a language, a child's task is to set these parameters for each category.

It is true that the parameter space of HD-parameters can accommodate a fair range of word order variation. The parameterization can successfully explain the word order differences between English and Japanese, for instance. However, there are many word order facts which fall outside this parameter space. The most obvious example is the VSO order. If we assume that a direct object is the complement of a verb and therefore must be adjacent to the verb at D-structure, we will not be able to get this common word order no matter how the HD-parameters are set. The same is true of the OSV order. A more general problem is scrambling. It has long been recognized that this word order phenomenon cannot be accounted for in terms of HD-parameters alone (ref??). All this suggests that the HD-parameters are at least insufficient, if not incorrect, for the explanation of word order variation. To account for the complete range of word order phenomena, we need some additional or alternative parameters.

The observation that not all word order facts can be explained in terms of phrase structure is by no means a new discovery. Ever since Chomsky (1957), linguists have found it necessary to account for word order variation in terms of movement in addition to phrase structure. In fact, this is one of the main motivations that triggered the birth of transformational grammars. All the problems with HD-parameters mentioned above disappear once movement is accepted as an alternative explanation for word order issues. The VSO order can be derived, for example, if we assume that the verb and the object *are* adjacent at D-structure but the verb has moved to a higher position at S-structure. (Ref??) Scrambling can also receive an elegant account in terms of movement. As Mahajan (1990) has shown, many scrambled word orders (at least in Hindi) can be derived from A and $\bar{A}$-movements. As a matter of fact, most linguists agree that movement is at least partially responsible for cross-linguistic differences in word order. As has been mentioned in the previous section, some attempts (e.g. Huang 1982) have already been made to parameterize movement options. However, movement has seldom been considered a major source of cross-linguisitc word order variation. Until Kayne (1992, 1993), no one had proposed that word order be explained mainly in terms of movement, let alone a model where movement options are systematically parameterized. Why this should be the case is not hard to explain. In the standard P&P theory, very few movements have the option of being either overt or covert. The parameterization of movement, even if it were implemented, would not be able to account for a sufficiently wide range of word order phenomena.

Things are different in the minimalist framework, as we have seen in the previous section. In this model, every movement has the option of being either overt or covert. We have proposed that an S(M)-parameter be associated with each of the movements and let the value of this parameter determine whether the given movement is to occur before Spell-Out (overt) or after Spell-Out (covert). The word order of a particular language then depends at least partially on the values of S(M)-parameters.

Now that we can account for word order variation in terms of S(M)-parameters, we have to reconsider the status of HD-parameters. Since HD-parameters by themselves are insufficient for explaining all word order phenomena, we only need to consider two possibilities:

(7) (i) S(M)-parameters can account for all the word order facts that HD-parameters are able to explain. In this case, HD-parameters can be replaced by S(M)-parameters.

(ii) S(M)-parameters cannot account for all the word order facts that HD-parameters are able to explain. In this case we will need both types of parameters.

To choose between these two possibilities, we have to know *whether* all the word orders that are derivable from the values of HD-parameters can be derived from the values of S(M)-parameters as well. To find out the answer to this question, we must first of all get a better understanding of the parameter space created by S(M)-parameters. We will therefore devote the next section to the exploration of this new parameter space.

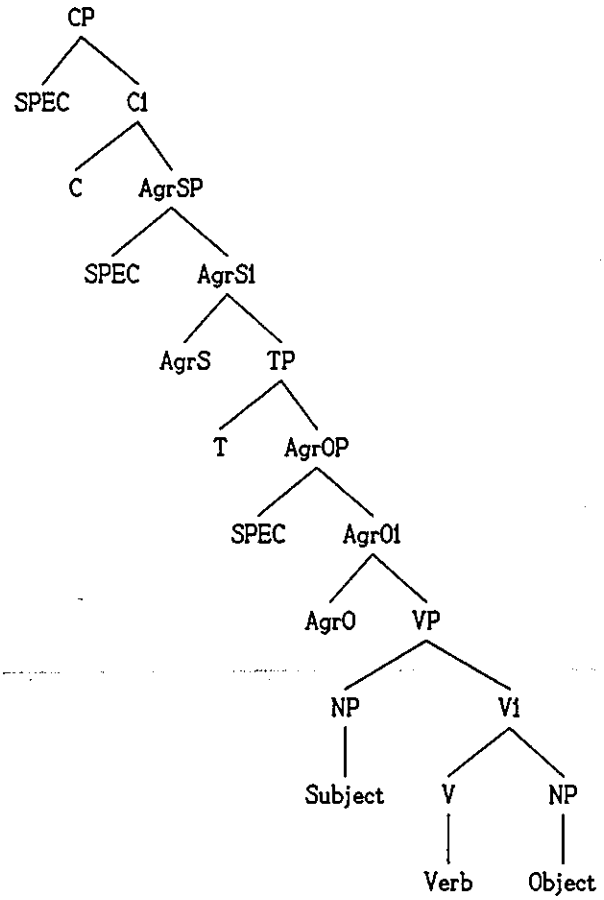
2.2.2 The Invariant X-bar Structure Hypothesis (IXSH)

If S(M)-parameters can replace HD-parameters to become the only source of word order differences, variations in phrase structure can be assumed to be non-existent. We will be able to envision a model where X-bar structures are invariant and all word order variations are derived from movement. Let us call this the *invariant X-bar structure hypothesis* (IXSH). This hypothesis has already been put forward by Kayne (1992, 1993). He argues that X-bar structures are asymmetrical. In the model he proposes, the specifier invariably precedes the head and the complement invariably follows the head. The structure is right-branching in every language and linear order always corresponds to C-command relations. According to this hypothesis, there is a single set of X-bar trees which are found in all languages. All variations in word order are results of movement. HD-parameters are thus non-existent.

Kayne's proposal has been explored in Wu (1992, 1993) where the new approach is tried out in the Minimalist framework. Wu (1993) experimented with the IXSH in a restricted model of syntax and showed that a surprising amount of variation in word order can be derived from a single X-bar tree. Since the results of this experiment will give us a more concrete idea as to what S(M)-parameters can do and cannot do, we will take a closer look at this model.

The invariant X-bar tree that the model assumes for a simple transitive sentence is given in (8).[15]

(8)



The set of LF requirements which force movements in this model are listed in (9).

(9) **(A)** The verb must move to AgrO-0 to have its ϕ-features checked for object-verb agreement.

(B) The verb must move to T0 to have its tense/aspect features checked.

(C) The verb must move to AgrS-0 to have its ϕ-features checked for subject-verb agreement.

(D) The verb must move to C0 to have its predication feature checked.

(E) The subject NP must move to Spec-of-AgrSP to have its case and ϕ features checked.

(F) The object NP must move to Spec-of-AgrOP to have its case and ϕ features checked.

(G) The XP which has scope over the whole sentence or serves as the topic/focus of the sentence must move to Spec-of-CP to have its operator feature checked.

[15] The tree for an intransitive sentence is identical to (8) except that the verb will not have an internal argument. It is assumed that AgrO exists even in an intransitive sentence, though it may not be active.

Each of the seven movements listed above, referred to as A, B, C, D, E, F and G, is supposed to be associated with an S-parameter whose value determines whether the movement under question is to be applied before or after Spell-Out. The seven S-parameters are referred to as $S(A), S(B), S(C), S(D), S(E), S(F)$, and $S(G)$. All the parameters are binary (1 or 0) except $S(G)$ which has three values: 1, 0 and 1/0. The last value is a variable which can be either 1 or 0. The overtness of movement will be optional if this value is chosen.

The application of those movements is subject to the two constraints in (10).

(10) **(i)** Head Movement Constraint.[16]

(ii) An NP must move to an Spec-of-Agr to have its case/agreement features checked before moving to Spec-of-C.[17]

It was shown that with all the assumptions given above, the parameter space consists of fifty possible settings.[18]. These settings and the corresponding word orders they account for are given in (11).

[16] This constraint requires that no intermediate head be skipped during head movement. For a verb to move from its VP-internal position all the way to C, for instance, it must land successively in AgrO, T and AgrS. This means that, if D occurs before Spell-Out, A, B and C will also occur before Spell-Out. Consequently, setting $S(D)$ to 1 requires that $S(A)$, $S(B)$ and $S(C)$ also be set to 1. So there is a transitive implicational relationship between the values of $S(A)$, $S(B)$, $S(C)$ and $S(D)$: given the order here, if one of them is set to 1, then the ones that precede it must also be set to 1.

[17] This means that $S(G)$ cannot be set to 1 unless $S(E)$ is set to 1.

[18] With 6 binary parameters and one triple-valued one, logically there should be 192 possible settings. But most of these settings are ruled out as syntactically impossible by the two constraints in (10)

#	*Values of S-parameters*							*Word Order*
	$S(A)$	$S(B)$	$S(C)$	$S(D)$	$S(E)$	$S(F)$	$S(G)$	
1	0	0	0	0	0	0	0	S V (O)
2	1	0	0	0	0	0	0	V S (O)
3	0	0	0	0	1	0	0	S V (O)
4	0	0	0	0	0	1	0	(O) S V
5	1	1	0	0	0	0	0	V S (O)
6	1	0	0	0	1	0	0	S V (O)
7	1	0	0	0	0	1	0	(O) V S
8	0	0	0	0	1	1	0	S (O) V
9	1	1	1	0	0	0	0	V S (O)
10	1	1	0	0	1	0	0	S V (O)
11	1	1	0	0	0	1	0	V (O) S
12	1	0	0	0	1	1	0	S (O) V
13	1	1	1	1	0	0	0	V S (O)
14	1	1	1	0	1	0	0	S V (O)
15	1	1	1	0	0	1	0	V (O) S
16	1	1	0	0	1	1	0	S V (O)
17	1	1	1	1	1	0	0	V S (O)
18	1	1	1	1	0	1	0	V (O) S
19	1	1	1	0	1	1	0	S V (O)
20	1	1	1	1	1	1	0	V S (O)
21	0	0	0	0	1	0	1	S V (O)
22	0	0	0	0	1	1	1	S (O) V, O S V
23	1	0	0	0	1	0	1	S V (O)
24	1	0	0	0	1	1	1	S (O) V, O S V
25	1	1	0	0	1	0	1	S V (O)
26	1	1	0	0	1	1	1	S V (O), O S V
27	1	1	1	0	1	0	1	S V (O)
28	1	1	1	0	1	1	1	S V (O), O S V
29	1	1	1	1	1	0	1	S V (O)
30	1	1	1	1	1	1	1	S V (O), O V S
31	0	0	0	0	0	0	1/0	S V (O)
32	1	0	0	0	0	0	1/0	V S (O)
33	0	0	0	0	1	0	1/0	S V (O)
34	0	0	0	0	0	1	1/0	(O) S V
35	1	1	0	0	0	0	1/0	V (O) S
36	1	0	0	0	1	0	1/0	S V (O)
37	1	0	0	0	0	1	1/0	(O) V S
38	0	0	0	0	1	1	1/0	S (O) V, O S V
39	1	1	1	0	0	0	1/0	V S (O)
40	1	1	0	0	1	0	1/0	S V (O)
41	1	1	0	0	0	1	1/0	V (O) S, O V S
42	1	0	0	0	1	1	1/0	S (O) V, O S V
43	1	1	1	0	1	0	1/0	S V (O)
44	1	1	1	0	0	1	1/0	V (O) S, O V S
45	1	1	0	0	1	1	1/0	S V (O), O S V
46	1	1	1	1	0	0	1/0	V S (O)
47	1	1	1	0	1	1	1/0	S V (O), O S V
48	1	1	1	1	1	0	1/0	V S (O), S V (O)
49	1	1	1	1	0	1	1/0	V (O) S, O V S
50	1	1	1	1	1	1	1/0	V S (O), S V (O), O V S

(11)

As we can see, the word orders accommodated in the parameter space include SVO, SOV, VSO, V2, VOS, OSV, and OVS. In other words, all the basic word orders seem to have been covered. The parameter space also permits a certain degree of scrambling.

In addition to this new word order typology, Wu (1993) also showed that the S-parameters assumed here can be successfully set. He proposed a parameter-setting algorithm which has the following properties.[19]

- Convergence is guaranteed without the need of negative evidence.
- The learning process is *incremental*: the resetting decision can be based on the current setting and the current input string only.
- The resetting procedure is *deterministic*: at any point of the learning process, there is a unique setting which will make the current string interpretable.
- Data presentation is order-independent. Convergence is achieved regardless of the order in which the input strings are presented, as long as all the *distinguishing* strings eventually appear.

It has been shown in Wu (1992) that the IXSH can also make the parser more universal. With the assumption that languages are invariant in terms of X-bar structure and LF movements, the parser will build the same trees and same chains no matter what language is being parsed. As a result, the number of possible trees is greatly reduced. The only language-particular decision the parser has to make is to determine, on the basis of the S-parameter values, where in the chain (head or

[19]The parameter-setting algorithm is based on the *Principle of Procrastinate* (MPLT) which basically says "avoid overt movement as much as possible" and *Principle of Greed* (MPLT) which says "do not move unless the movement can achieve a purpose". Following these principles, Wu assumes that all the S-parameters are set to 0 at the initial stage. The parameter-setting algorithm is basically the failure-driven one described in Gold (1967) (induction by enumeration). The learner always tries to parse the input sentences with the current setting of parameters. The setting remains the same if the parse is successful. If it fails, the learner will try parsing the input sentence using some different settings until he finds one which results in a successful parse. The interesting part is the algorithm that determines the *order* in which alternative settings are to be tried. Because of the existence of subset relations, the order must not violate the Subset principle, i.e. the order should ensure that, given two languages L1 and L2, L1 a proper subset of L2, the setting for L1 must be tried before the setting for L2. The ordering algorithm is as follows:

> Sort the possible settings into an ordered list where precedence is determined by the following sub-algorithms:
>
> (i) Given two settings $P1$ and $P2$, $P1 < P2$ if $S(G)$ is set to 0 in $P1$, but it is set to 1 or 1/0 in $P2$. (The setting where there is no overt movement to Spec-of-C is to be tried first.) Go to (ii) only if (i) fails to distinguish $P1$ and $P2$.
>
> (ii) Given two settings $P1$ and $P2$, $P1 < P2$ if $S(G)$ is set to 1 in $P1$, but it is set to 1/0 in $P2$. (If overt movement to Spec-of-C is required, the setting where the movement is obligatory is to be tried first.) Go to (iii) only if (ii) fails to distinguish $P1$ and $P2$.
>
> (iii) Given two settings $P1(i)$ and $P2(j)$ where i and j are the number of 1's in the respective settings, $P1 < P2$ if $i \leq j$. (The setting where there are fewer overt movements is to be tried first.) Go to (iv) only if (iii) fails to distinguish $P1$ and $P2$.
>
> (iv) Order the settings freely. (All settings that remain ordered correspond to languages which are disjoint or intersecting with one another. Their learnability is guaranteed.)

The resulting order is the one given in (11). What the learner does in cases of failure is try those settings one by one until he finds one that works. The learner is always going down the list and no previous setting will be tried again.

tail) a lexical constituent should appear. Technically, this means that the parser only has to decide for each terminal node whether it must contain spelled out lexical material or not. Consequently, different word orders can be parsed with a single parser which is universally given. Since all the parameterization is in the grammar, no learning will be needed in terms of parsing.

The fact that the S-parameter account of word order variation can make acquisition easier and the parser more universal is not a surprise. In the traditional model where the syntactic tree can vary in shape cross-linguistically, the parser and the learner may have to construct a new set of trees for every new language. This kind of tree construction can be a costly operation in both parsing and acquisition. X-bar theory has made tree-building easier by providing a universal set of "prefabricated" subtrees which are to be "assembled" by the parser or learner. Nonetheless, the ways of assembling those trees can vary so much that a different "assembly line" may have to be designed for each different language. In Kayne's or Wu's model, however, the assembly line is universal. The learner thus has much fewer decisions to make and the parser needs fewer adaptations.

2.2.3 Modifying the IXSH

So far the IXSH approach to word order has appeared to be very promising. We have an S-parameter space which can accommodate all the basic word orders and a parameter-setting algorithm which has some desirable properties. Many word order facts that used to be derived from the values of HD-parameters have proved to be derivable from the the values of S(M)-parameters as well. In addition, there are facts which are explained by the S(M)-parameters by not by HD-parameters. What we have seen has certainly convinced us that movement can have a much more important role to play in the derivation of word orders. However, we have not yet proved that HD-parameters can be eliminated altogether. In other words, we not yet sure whether S(M)-parameters can account for *everything* that HD-parameters are capable of accounting for.

Both Kayne (1992,1993) and Wu (1992, 1993) have assumed a base structure where the head invariably precedes its complement. This structure is strictly right-branching. One consequence of this is that linear precedence now corresponds to C-command relations in every case. Given two terminal nodes A and B where A asymmetrically C-commands B, A necessarily precedes B in the tree. In current theories, functional categories dominate all lexical categories in a single IP/CP, with all the functional heads asymmetrically C-commands the lexical heads. This means that all the functional heads precede the lexical heads in the base structure. This is apparent from the tree in (8). Let us assume that a functional head can be spelled out in two different ways: (i) as an affix on a lexical head or (ii) as an independent word such as an auxiliary, a grammatical particle, an expletive, etc. (This assumption will be discussed in detail in 2.3.) In Case (i), the lexical head must have moved to or through the functional head, resulting in the "amalgamation" the lexical head and the functional head. The prefix or suffix to the lexical head may look like a functional head preceding or following the lexical head, but the two cannot be separated by an intervening element. Case (ii) is

possible only if the lexical head has not moved to the functional head, for otherwise the functional head would have merged into the lexical head. Consequently, the lexical head such as a verb must follow the functional head such as an auxiliary in a surface string, since the former must be lower in the tree and asymmetrically C-commanded by the latter in this case. What all this amounts to is the prediction that we should never find a string where a functional element follows the verb but is not adjacent to it. In other words, the sequence in (12) is predicted to be impossible where F stands for any overtly realized functional head such as an auxiliary or a grammatical particle and X stands for any intervening material between the verb and the functional element(s).

(12) $[_{cp}$ ... $[_{ip}$ ... *Verb* X $F^{+}]]$

One may argue that this sequence *is* possible if *excorporation* (Koopman 1992) can occur. In excorporation, a verb moves to a functional head without getting amalgamated with it, and then moves further up. In that case, the verb will end up in a position which is higher than some functional heads. When these functional heads are spelled out as auxiliaries or particles, they will follow the verb. But this does not account for all cases of (12). Consider the Chinese sentence in (13)[20].

(13) *Ta* *kan-wan* *nei-bèn* *shu* *le* *ma*
you finish reading that book Asp Q/A
'Have you finished reading that book? / He has finished reading that book, as you know.'

This sentence fits the pattern in (12). The aspect marker *le*[21] and the question/affirmation particle *ma*[22] are not adjacent to the verb, being separated from it by a full NP. This order does not seem to be derivable from excorporation. The aspect particle *le* is presumably generated in AspP (aspect phrase) and the question/affirmation particle *ma* is generated in CP (Cheng 1991, Chiu 1992). In order for both the verb and the object NP to precede *le* or *ma*, the verb must move to a position higher than Asp^0 or C^0, and the object NP must also be in a position higher than Asp^0 or C^0. This is impossible given standard assumptions.[23] Therefore the sentence in (13), which is perfectly grammatical, is predicted to be impossible in Wu's model. This shows that there are linguistic facts which the Invariant X-bar Structure Hypothesis is unable to account for. By contrast, these facts can receive a very natural explanation if some HD-parameters are allowed for. We can assume that CP and IP (including AspP) are head-final in Chinese. The verb does not move overtly in Chinese,

[20] This sentence is ambiguous in its romanized form, having both an interrogative reading and a declarative reading. (They are not ambiguous when written in Chinese characters, as the two senses of *ma* are written in two different characters: " " and " ").

[21] There are two distinctive *les* in Chinese: an inchoative marker and a perfective marker (Teng 1973). These two aspect markers can co-occur and they occupy distinct positions in a sentence. The inchoative *le* is always sentence-final in a statement while the perfective *le* immediately follows the verb. The *le* in (13) is an instance of the inchoative marker. The other aspect marker in (13), *guo*, is called an experiential marker. It indicates that the event did happen in the past.

[22] Whether it is the interrogative *ma* or affirmative *ma* depends on the intonation.

[23] The fact that *ma* is sentence-final might be explainable if we accept Kayne's (1993) hypothesis that the whole IP can move to the Spec of CP. But it will not be justified for us to swallow this new hypothesis just to save this single construction. Moveover, the position of *le* would still be a mystery even if this hypothesis were taken.

so the heads of AspP and CP are spelled out as an aspect marker and a question/affirmation marker respectively. Since they are generated to the right of the whole VP, they must occur in sentence-final positions.

The assumption that CP and IP can be head-final is supported by facts in other languages. In Japanese, for example, we find the following sentences.

(14) *Yamada-sensei-wa* *kim-ashi-ta* *ka*
Yamada-teacher-Topic come-Hon-Past Q
'Did Professor Yamada come?'

(15) *Kesa* *hatizi* *kara* *benkyoosi-te* *i-ru*
this-morning 8 from study-Cont be-Nonpast
'I have been studying since eight this morning.'

In (14), we find the question particle *ka* following the verb. This particle is clearly a functional element.[24] As assumed in Bach (1970), Bresnan (1972) and recently Cheng (1992) and Fukuda (1993), question particles in Japanese are positioned in in C^0. In (15), the verb is followed by *i-ru* which is most likely located in T. In both cases, a functional head appears after the verb, which is to be expected if CP and TP (which is part of IP) are head-final in Japanese. The IXSH will have difficulty explaining this unless we assume that these particles and auxiliaries are in fact suffixes which come together with the verb from the lexicon. But there is strong evidence that *ka* and *i-ru* are not suffixes. We could also argue that the verb has managed to precede *ka* and *i-ru* by left-adjoining to T_0 and C_0 through head movement. But in that case the verb would be in C_0 and the word order would be VSO instead of SOV. The excorporation story is even less plausible, for the verb in (14) would have to move to a position higher than C in order to precede *ka*.

Arguments for the existence of HD-parameter in IP are also found in European languages. In German, an auxiliary can appear after the verb, as we see in (16).

(16) *dat* *Wim* *dat* *boek* *gekocht* *heeft*
that Wim that book bought has
'that Wim has bought that book.'

The clause-final *heeft* is located in the head of some functional projection. It is not a suffix which can be drawn from the lexicon together with the verb.[25] If we stick with the IXSH, the word order in (16) would not be possible. For the verb to get in front of *heeft*, it must move to a position at least as high as *heeft*. But the word order in that case would be SVO or VSO instead of SOV. However, (16) will not be a problem if we say that IP is head-final in German.

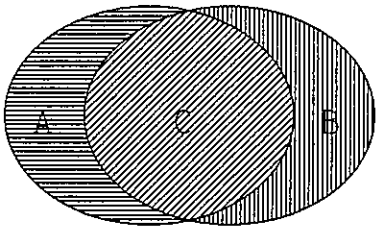
It has become evident now that a strong version of IXSH, where no HD-parameter exists, is very difficult, if not impossible, to maintain. We have seen that, although the S(M)-parameters can

[24] A number of other particles can appear in this position, such as *no*, *sa*, *yo*, *zo*, etc. There is no doubt that they are functional elements, though the exact functions they perform are controversial.

[25] The fact that *heeft* can appear in positions not adjacent to the verb is enough to show that it is not an affix.

explain things that HD-parameters fail to explain, there are also facts which are best explained by HD-parameters. In other words, we have come to the conclusion that the second possibility in (7) is more plausible. We have seen that that the word order facts covered by S(M)-parameters and HD-parameters intersect each other. While some facts can receive an explanation in terms of either S(M)-parameters or HD-parameters, there are word order phenomena which can be explained by S(M)-parameters only or HD-parameters only. The situation we have here is graphically illustrated in (17). Therefore we need both types of parameters.

(17)



A = facts covered by S(C)-parameters

B = facts covered by head-direction parameters

C = facts covered by both parameters

The next question is how to coordinate these two kinds of parameters in an account of word order variation. As (17) shows, the empirical grounds covered by these two types of parameters overlap a lot. If we keep both parameters in full, there can be too much redundancy. The number of parameters we get will be greater than necessary. Ideally the parameter spaces of these parameters should complement each other. There should be a new division of labor where the two kinds of parameters duplicate each other's work as little as possible. There are at least two ways in which this can be tried:

(i) Let word order be derived mainly through movement, with variations in phrase structures covering what is left out. In other words, the facts in the A and C areas of (17) are accounted for by S(M)-parameters and those in B by HD-parameters.

(ii) Let word order be mainly a property of phrase structure, with movement accounting for what is left out. In other words, the facts in the B and C areas of (17) are accounted for by HD-parameters and those in A by S(M)-parameters.

The choice between the two can be based on several different considerations. The model to be favored should be syntactically simpler and typologically more adequate. In addition, it should be a better model in terms of language acquisition and language processing. In order to have a good comparison and make the correct decision, we must have good knowledge of both approaches. The approach in (ii) has already been explored extensively. This phrase-structure-plus-some-movement

account has been the standard story for many years. The results are familiar to almost everyone. The approach in (i), however, has not received enough investigation yet. This is an area where more research is needed. For this reason, the remainder of this thesis will be devoted mainly to the first approach. We will examine this approach carefully in terms of syntactic theory, language typology, language acquisition, and language processing. It is hoped that such an investigation will put us in a better position to choose between different theoretical approaches.

Having decided on the main goal of the present research, we can now start looking into the specifics of a model which implements the idea in (i). One of the obvious questions that is encountered immediately is "how many S(M)-parameters are there and how many HD-parameters". This question will be addressed in Chapter 3 when we work on a full specification of the model. It will be proposed that the HD-parameters be restricted to functional categories only. In particular, there will be only two HD-parameters: a complement-head parameter for CP and a complement-head parameter for IP. The former determines whether CP is head-initial or head-final and the latter determines whether IP is. With the assumption that IP consists of a number of functional categories each heading its own projection, the number of heads in IP can be more than one. Instead of assuming that each of these functional projections has an independent parameter, I will assume that the complement-head parameter applies to the IP as a whole. This is to say that the value of this parameter will apply to every projection in IP. In other words, we will not consider the situation where, say, AgrP is head-initial while TP is head-final. This decision is based on the following considerations:

(i) All previous models incorporating HD-parameters have treated IP as a single unit.

(ii) There has been no strong evidence that IP can be "bidirectional" in the sense that some of its projections are left-headed and some right-headed.

(iii) The organization of IP-internal structure is still a controversial issue. There has not been enough consensus as to how many IP-internal categories there are and how they are hierarchically organized. It is risky, therefore, to attach a parameter to any specific "sub-IPs".

Similar arguments can be made for CP which is treated as a single unit in spite of the fact that some other CP-like categories, such as FP (Focus Phrase) (Brody 1990, Horvath 1992, etc.) and TopicP, have been proposed.

By reducing the number of HD-parameters to two, we have not destroyed the Invariant X-bar Structure Hypothesis completely. What are we are left with is a weaker version of the IXSH where the directions of specifiers are still fixed and the directions of complements are fixed except those in IP and CP. Compared with the standard model, the degree of invariability in X-bar structure is much higher. As a result, the number of possible tree structures has decreased considerably. The significance of this modified version of IXSH for language typology, language acquisition and language processing will be studied in Chapters 4, Chapter 5 and Chapter 6.

2.3 Interaction of S(F)- and S(M)- Parameters

In this section, we come back to the relationship between overt morphology and overt movement. We have assumed that these two phenomena are independent from each other. It has been proposed that each feature has two S-parameters associated to it: the S(F)-parameter that determines whether the feature is spelled out morphologically and the S(M)-parameter that determines whether the movement responsible for checking the feature occurs before Spell-Out. It is interesting to note that both the S(F)- and S(M)- parameters are feature-related. Given that both the S(F)-parameter and S(M)-parameter are binary (1 or 0)[26], there is a parameter space of 4 possible settings for each feature:

(18)

	S(F)	S(M)
(i)	1	1
(ii)	1	0
(iii)	0	1
(iv)	0	0

The question that follows is what linguistic facts these four settings are supposed to account for. Before answering this questions, let us have a closer look at how feature-checking works.

In the Minimalist framework, we can entertain the assumption that any feature that needs to be checked is generated in two different places, one in a functional category and one in a lexical category. Let us call these two instances of the same feature F-feature and L-feature respectively. For instance, the tense feature is found in both T and V. To make sure that the two tense features match, the verb (which carries the L-feature) must move to T (which carries the F-feature) so that the value can be checked. If the checking (which entails movement) occurs before Spell-Out, the F-feature and the L-feature will get unified and become indistinguishable from each other[27]. As a result, only a single instance of this feature will be available at the point of Spell-Out. If the checking occurs after Spell-Out, however, both the F-feature and the L-feature will be present at Spell-Out and either can be overtly realized. This assumption has important consequences. As will be discussed in detail in Chapter 4, this approach provides a way of reconciling the movement view and base-generation view of many syntactic issues. The checking mechanism we assume here requires both base-generation and movement for many syntactic phenomena. Many features are base-generated in two different places but related to each other through movement. I will not go into the details here. The implications of these assumptions will be fully explored in Chapter 4.

Now let us examine the values in (18) and see what can possibly happen in each case.

[26] We will see later on that the S(M)-parameter can have a third value: 1/0.

[27] This is similar to *amalgamation* where the functional element becomes part of the lexical element.

In (18(i)), both the S(F)-parameter and the S(M)-parameter are set to 1. This means both the feature itself and the movement that checks this feature will be overt. In this case, the F-feature and L-feature will become one and be spelled out on the lexical head in the form of, say, an affix. In addition, this lexical item will be in a position no lower than the one where the F-feature is located. If the features under question are agreement features, for example, we will see an an inflected verb in I/AgrS or a higher position, with the inflection carrying agreement information. This seems to be the "normal" case that occurs in many languages. One example is French where the verb does seem to appear in I/AgrS and it has overt morphology indicating subject-verb agreement (see (19)).[28]

(19) *Mes parents parlent souvent espagnol*
my parents speak-3P often Spanish
'My parents often speak Spanish.'

If the feature to be considered is the case feature of an NP, this NP will move to the Spec of IP/AgrP and be overtly marked for case. Japanese seems to exemplify this situation, as can be seen in (20).[29]

(20) *Taroo-ga Hanako-o yoku mi-ru*
Taroo-nom Hanako-acc often see-pres
'Taroo often sees Hanako'

If the feature to be checked is the scope feature of a wh-phrase, the wh-phrase will move to the Spec of CP, with the scope feature overtly realized in some way. English might serve as an example of this case, though it is not clear how the scope feature is overtly realized.

In (18(ii)), the S(F)-parameter is set to 1 but the S(M)-parameter is set to 0. This means that the feature must be overt but the movement that checks this feature must not. Since the checking movement takes place after Spell-Out, both the F-feature and the L-feature will be present at the point of Spell-Out and at least one of them must be overtly realized. There are three logical possibilities for spelling the feature out: (a) spell out the F-feature only, (b) spell out the L-feature only, or (c) spell out both. (a) and (b) seem to be exemplified by the agreement/tense features in English. The English verb does not seem to move to I before Spell-Out. Since the features appear both in I and on the verb, we can pronounce either the L-features or the F-features. When the L-features are pronounced, we see an inflected verb, as in (21). When the F-features are pronounced, we see an auxiliary, as in (22) where *does* can be viewed as the overt realization of the I-features. (In other words, *Do-Support* is a way of spelling out the head of IP.) The third possibility where the feature is spelled out in both places does not seem to be allowed in English, as (23) shows.

(21) *John loves Mary.*

(22) *John <u>does</u> love Mary.*

[28] The fact that the verb precedes the adverb *souvent* in this sentence tells us that the verb has moved to I/AgrS.

[29] We can assume that the subject NP in this sentence has moved to the Spec of AgrSP and the object has moved to Spec of AgrOP. This is supported by the fact that the adverb *yoku* is preceded by both NPs.

(23) * John does loves Mary.

In Swedish, however, "double" spell-out seems to be possible. When a whole VP is fronted to first position, which probably means that the verb did not have a chance to undergo head movement to T(ense), the tense feature is spelled out on both the verb and an auxiliary which seems to have moved from T to C. This is shown in (24).

(24) *Oeppnade doerren gjorde han*
open-Past door-the do-Past he
'He opened the door.'

The value in (18(ii)) can also be illustrated with respect to case/agreement features and the scope feature. It seems that in English the features in Spec of IP/AgrSP must be spelled out. When overt NP movement to this position takes place, the features appear on the subject NP. In cases where no NP movement takes place, however, Spec of AgrSP is occupied by an expletive (as shown in (25)) which can be viewed as the overt realization of the case/agreement features in Spec of AgrSP.

(25) *There came three men.*

The situation where the scope feature is overtly realized without overt wh-movement is found in German partial wh-movement. In German, wh-movement can be either complete or partial, as shown in (26), (27), (28) and (29).

(26) [*mit wem*]$_i$ *glaubst du* [$_{cp}$ t_i *dass Hans meint* [$_{cp}$ t_i *dass Jakob* t_i *gesprochen hat*]]
with whom believe you that Hans think that Jakob talked has

(27) *was*$_i$ *glaubst du* [$_{cp}$ [*mit wem*]$_i$ *Hans meint* [$_{cp}$ t_i *dass Jakob* t_i *gesprochen hat*]]
WHAT believe you with whom Hans think that Jakob talked has

(28) *was*$_i$ *glaubst du* [$_{cp}$ *was*$_i$ *Hans meint* [$_{cp}$ [*mit wem*]$_i$ *Jakob* t_i *gesprochen hat*]]
WHAT believe you WHAT Hans think with whom Jakob talked has

(29) **was*$_i$ *glaubst du* [$_{cp}$ *dass Hans meint* [$_{cp}$ [*mit wem*]$_i$ *Jakob* t_i *gesprochen hat*]]
WHAT believe you that Hans think with whom Jakob talked has
'With whom do you believe that Hans thinks that Jakob talked?'

These four sentences have the same meaning but different movement patterns. In (26) the wh-phrase (*mit wem*) moves all the way to the matrix CP. In (27) and (28), the wh-phrase also moves, but only to an intermediate CP. The Spec(s) of CP(s) which *mit wem* has not moved to are filled by *was* which is usually called a wh-scope marker. (29), which is ungrammatical, is identical to (28) except that the specifier of the immediately embedded CP does not contain *was*. It seems that the

into consideration, that some of these settings *can* generate non-empty languages. But we will first investigate a parameter space without S(F)-parameters.

The requirement that an NP must move to an Agrspec before moving to Cspec can also make certain value combinations produce an empty language. Consider the settings for S(M(spec1)), S(M(spec2)) and S(M(cspec)). If S(M(cspec)) is set to 1, Cspec must be filled at Spell-Out. To satisfy this requirement, some XP must move there. Suppose that the only XP that can move to Cspec in a particular sentence is the subject NP or object NP. When S(M(cspec)) is set to 1, at least one of them must move to Cspec. Since no NP can move there unless it has moved to an Agr1spec or Agr2spec, NP movement to these positions must not be blocked. But consider what will happen if we have the setting in (95).

(95) [. . . 0 0 1]

In (95), S(M(cspec)) is set to 1 while both S(M(spec1)) and S(M(spec2)) are set to 0. On the one hand, some NP is required to move overtly to Cspec; on the other hand, however, no NP is allowed to move Agr1spec or Agr2spec before Spell-Out. Since no NP can move to Cspec unless it has moved to Agr1spec or Agr2spec, the required movement is blocked. The derivation thus crashes and no string is generated.

It turns out that, with the two constraints on movement discussed above and our temporary restriction of Ā-movement to NPs only, the number of value combinations which can generate languages other than the empty one is 156 rather than 864. These settings and the corresponding sets of strings they generate are shown in Appendix B.1. In this list, each entry consists of a vector of S(M)-parameter values and the set of strings generated with this value combinations. We call each vector a *setting* and each set of strings a "language".

If we examine these setting-language pairs carefully, we will find that the correspondence between settings and "languages" is not one-to-one. For instance, both the setting in #1 ([0 0 0 0 0 0 0 0]) and the one in #3 ([0 0 0 0 0 1 0 0]) can generate the language [s v, s v o].[2] In fact, the correspondence is many-to-one in most cases. We see in Appendix B.1 that many settings generate identical languages. The language [v s, v s o], for instance, can be generated with 14 different settings including #4, #8, #12, #16, #20, etc.. There are 156 settings in the list, but the number of distinct languages that are generated is only 31. These 31 languages and their corresponding settings are listed in Appendix B.2. The significance of such many-to-one correspondences will be discussed in 4.1.2.

Looking at the languages listed in Appendix B.2, we find that our current parameter space is capable of accommodating all the basic word orders: SV(O) (#1), S(O)V (#4), VS(O) (#2), V(O)S (#8), (O)SV (#3), and (O)VS (#5). We also find a V2 language (#11). One of the settings

[2] It is important to note the distinction between strings and languages. A language comprises a set of strings, and two languages are identical only if they have exactly the same set of strings. For instance, [s v, s v o] and [s v, s o v] are not the same language in spite of the fact they both contain string "s v".

with which a V2 language can be generated is [1 1 1 1 1 1 1 1] where every movement is overt. According to this setting, the verb must move overtly all the way to C0, the NPs to their respective Agrspecs, and one of the NPs must move further on to Cspec. In an intransitive sentence, there is only one NP and this NP must move to Cspec. The resulting word order is SV. In a transitive sentence, either the subject NP or object NP can fill Cspec. We have SVO when the subject is in Cspec and OVS when the object is. This may well be what is happening in German root clauses. If so, the German sentences in (96(a)) and (97(a)) will have the structures in (96(b)) and (97(b)) respectively.

(96) a. *Der* *Mann* *sah* *den* *Hund*
the(nom) man saw the(acc) dog
'The man saw the dog.'

b. $[_{cp}$ **Der Mann**$_i$ $[_{c1}$ $[_{c}$ **sah**$_j$$]$ $[_{agr1p}$ e_i $[_{agr1-1}$ . . . $[_{agr2p}$ **den Hund**$_k$ $[_{agr2-1}$ $[_{agr2}$ $e_j]$ $[_{vp}$ e_i e_j $e_k]]]]]]]]

(97) a. *Den* *Hund* *sah* *der* *Mann*
the(acc) dog saw the(nom) man
'The dog, the man saw.'

b. $[_{cp}$ **Den Hund**$_k$ $[_{c1}$ $[_{c}$ **sah**$_j$$]$ $[_{agr1p}$ **der Mann**$_i$ $[_{agr1-1}$ . . . $[_{agr2p}$ e_k $[_{agr2-1}$ $[_{agr2}$ $e_j]$ $[_{vp}$ e_i e_j $e_k]]]]]]]]

In addition to basic word orders and V2, the current parameter space also allows for a considerable amount of scrambling. Scrambling is a general term referring to word order variation in a single language, usually the variation that results from clause-internal movements of maximal projections. In our present discussion, a "language" will be considered a scrambling one if it has more than one way of ordering S, V and O. For example, [o s v, s v, s v o] and [s o v, o s v, s v] are scrambling languages.

We can see that many of the "languages" in Appendix B.2 are scrambling languages. We do not know whether each of those languages is empirically attested, but at least some of them are. Let us take a look at the languages in #9, #16 and #30.

The "language" in #9, [s o v, o s v, s v], seems to be exemplified by Japanese and Korean. They are verb-final language where the subject and object can be put in any order as long as they precede the verb. The Japanese sentences in (98) and (99) illustrate this.

(98) *Taroo-ga* *Hanako-o* *nagut-ta*
Taroo-Nom Hanako-Acc hit-Past
'Taroo hit Hanako'

(99) *Hanako-o* *Taroo-ga* *nagut-ta*
Hanako-Acc Taroo-Nom hit-Past
'Taroo hit Hanako'

The "language" in #16 is [o s v, s o v, s v, s v o] where the subject must precede the verb but the object can appear anywhere in the sentence. Chinese seems to be an example of this, as we can see in (100), (101) and (102).

(100) *wo jian guo neige ren*
I see Perf that person
'I have seen that person before.'

(101) *wo neige ren jian guo*
I that person see Perf
'I have seen that person before.'

(102) *neige ren wo jian guo*
that person I see Perf
'That person, I have seen before.'

The "language" in #30 has very extensive scrambling, permitting all the following orders for a transitive sentence: [o v s, s v o, s o v, o s v, v s o]. All those orders are found in Hindi. For instance, the Hindi sentence in (103) has the alternative orders in (104)-(108).

(103) *raam-ne kelaa khaayaa* (SOV)
Ram-Erg banana ate
'Ram ate a banana.'

(104) *raam-ne khaayaa kelaa* (SVO)

(105) *kelaa raam-ne khaayaa* (OSV)

(106) *kelaa khaayaa raam-ne* (OVS)

(107) *khaayaa raam-ne kelaa* (VSO)

(108) *khaayaa kelaa raam-ne* (VOS)

The only order which is found in Hindi but not in #30 is VOS.

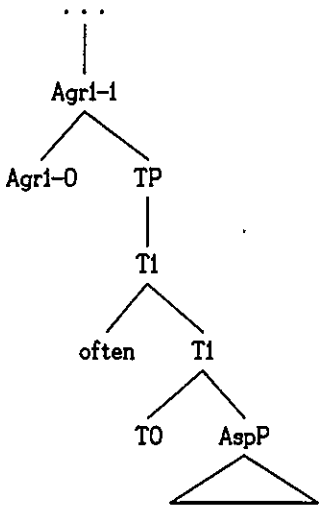
All the languages mentioned above – Japanese, Korean, Chinese and Hindi – have been described as "non-configurational" in the literature (??ref). They are supposed to be problematic for any strong version of X-bar theory where phrasal projections are assumed to be universal. This problem does not seem to exist in our present model. As we have seen, all the scrambled orders can be derived from a single configuration if some A and $\bar{A}$ movements are allowed to be optional at Spell-Out. This is in fact one of the current views of scrambling (e.g. Mahajan 1990).

4.1.2 Further Differentiation

One thing in Appendix B.2 that may cause concern is the fact that a single language can be derived with many different parameter settings. Those settings generate *weakly equivalent* languages[3] which cannot be distinguished from each other on the basis of surface strings. One reason why we cannot

[3]Two languages are weakly equivalent if they have identical surface strings but different grammars.

distinguish them is that many of the movements are string-vacuous. For example, in cases where the subject NP has moved to Agr1spec while the object NP remains in situ, there is no way of telling from an SVO string whether the verb is in Agr1-0, T0, Asp0, Agr2-0 or its VP-internal position. There is simply no reference point by which the movement can be detected. But this seems to be an accidental rather than an intrinsic property of the current model. The apparent indistinction between different settings is due at least in part to the artifact that only nouns and verbs have been taken into consideration in our grammar so far. But nouns and verbs are not the only entities in natural languages. There are other constituents which may serve as reference points for movement by virtue of the fact that some movements have to go "over" them. It has been widely assumed since Pollock (1989) that negative elements and certain adverbs are constituents of this kind. Once these reference points appear in the string, many movements will cease to be string-vacuous and many otherwise indistinguishable languages/settings will become distinct from each other. To illustrate this, I will introduce just one reference point into our grammar: an adverb of the "often" type. For easy reference I will simply call it *often*, meaning an adverb in any language which means "often". Furthermore I will assume that this temporal/aspectual adverb is a modifier of T and therefore left-adjoined to T1. The partial tree in (109) shows the part of the structure where *often* appears.



(109)

In terms of linear order, *often* appears between Agr1-0 and T0. This position can serve as a diagnostic for verb movement from T0 to Agr1-0 or NP movement to Agr1spec, both of which relates a position following *often* to a position preceding *often*. Any verb before *often* must have moved to Agr1-0 or higher and any NP before it must be in Agr1spec or a higher position. Given a string such as *O S often V*, for instance, we can tell that O is in Cspec and S in Agr1spec, since these are the only two NP positions before *often*. Moreover, we know that the verb cannot be in C0 or Agr1-0.

Appendix B.3 shows what the setting-language correspondences will be if the position of *often* is taken into consideration. The number of value combinations that produce non-empty languages did not change.[4] There are still 156 settings. However, the number of distinct languages that are generated with these settings has increased from 31 to 66. The addition of *often* into the strings has helped to differentiate many "languages" that are otherwise indistinguishable from each other. Take SVO "languages" as examples. In Appendix B.2, there is only a single SVO language which can be generated with 32 different settings. In Appendix B.3, however, four SVO languages are differentiated:

```
#1  [ (often) s v,  (often) s v o ] (2 settings)
#2  [ s (often) v,  s (often) v o ] (20 settings)
#12 [ s v (often),  s v (often) o ] (8 settings)
#20 [ s (often) v o,  s (often) v,  (often) s v,  (often) s v o\ ] (2 settings)
```

We are now able to distinguish between English and French which seem to correspond to #2 and #12 respectively. In English *often* precedes the verb while it follows the verb in French, as shown in (110) and (111).

(110) *John often visits China.*

(111) *Jean visite souvent la Chine*
John visits often China
'John often visits China.'

We can expect that, with more reference points (such as Neg and other adverbs) taken into account, even finer differentiation is possible. However, it is neither likely nor necessary that we should have enough reference points to differentiate every setting from every other setting. There seems no principled reason against having more than one way of generating the same language, as long as all the different *languages* are identifiable. The existence of weakly equivalent languages may just be a fact of life.

A natural question to ask at this moment is whether the reference points that have been assumed are always reliable. So far we have assumed that constituents such as Neg and adverbs do not move. However, this assumption does not seem well supported, although it has been around since Pollock (1989). The adverb *often*, for instance, does seem to move around in English. It can appear clause-initially as in (112), clause-medially as in (113), and clause-finally as in (114).

(112) *Often she plays the piano.*

(113) *She often plays the piano.*

[4] To have a fair comparison, we have kept the restriction that only NPs can move to Cspec. The number of settings will be greater if the AdvP *often* is allowed to move to Cspec as well. If Cspec can be filled by an AdvP, as it should be, then a non-empty language can be generated even if both S(M(spec1)) and S(M(spce2)) are set to 0.

(114) *She plays the piano very often.*

In (112), *often* seems to have moved to Cspec. This is possible if S(M(cspec)) is set to 1 or 1/0 in English. To account for (114), we have to assume that *often* may be adjoined to *either* T1 *or* V1 (i.e. it can be either a T modifier or V modifier). The order in (114) can be derived if *often* is attached to V1 while the verb has moved to T0 and the object NP to Agr2spec. This shows that the so called "reference points" may not always serve as a good indicator for the movement of other constituents. There might be a canonical position for each adverb and only this position serves as a reference point. But how a certain position can be identified as being canonical by the learner is a question that remains to be answered.

4.1.3 Some Set-Theoretic Observations

In this section, we consider the set-theoretic relations between the "languages" in our parameter space. We will also consider the parameter settings that are responsible for those relations. The languages to be examined will be those in Appendix B.2. There are only 31 distinct languages there and it does not take long to compute all the relations between those languages. (The Prolog program which does the computation is in Appendix A.2.) The number of languages we will have when all other parameters are taken into consideration is much greater than 31, but the properties we find in this small number of languages will hold when the parameter space is expanded with the addition of other parameters.

The set-theoretical relations can be considered in a pair-wise fashion. We take each of the 31 languages and check it against each of the other 30 languages.[5] The total number of pairs we get this way is 465 (30*31/2). There are three possible set-theoretic relations that can hold between the two languages in each pair: disjunction, intersection and proper inclusion.[6] Two languages are disjoint with each other if they have no string in common. An example of this is the relation between #1 [s v, s v o] and #2 [v s, v s o]. Two languages intersect each other if they are not identical but have at least one string in common. For instance, #1 [s v, s v o] and #4 [s o v, s v] intersect by virtue of the fact that (i) each of them has a string that the other does not have ([s v o] is in #1 only and [s o v] in #4 only.), and (ii) both of them contain the the string [s v]. A language L1 is properly included in another language L2 if the strings generated in L1 forms a proper subset of those generated in L2. This relationship is found, for example, between #1 [s v, s v o] and #10 [o s v, s v, s v o]. Each string in #1 is in #10, but not *vice versa*. Our computation shows that the subset relationship holds in 162 of the 465 pairs. The fact that subset relations do exist in a substantial number of cases will have important implications for the learning algorithm to be discussed in Chapter 5. We will not get into the learnability issues here. What we want to discover

[5] We are not interested in pairing each language with itself, since every language is identical to itself.

[6] No languages in any pair can be identical to each other, since all 31 languages in Appendix B.2 are distinct.

here is the *source* of those subset relations. It will become clear that these relations are attributable to some specific parameter values.

For convenience we will refer to a language which properly includes some other language(s) a *superset language* and a language which is properly included in some other language(s) a *subset language*. These notions are of course relative in nature. We may have three languages L1, L2 and L3 where L1 properly includes L2 which in turn properly includes L3. In this case, L2 will be a subset language relative to L1 but a superset language relative to L3. Examining the languages in Appendix B.2, we find that all superset languages show scrambling.[7] Since only two types of sentences, (transitive and intransitive) are produced in our system, any language in Appendix B.2 that contains more than two distinct strings is a scrambling language. Now, what do those scrambling languages have in common with regard to their parameter values? An examination of the "languages" in Appendix B.2 reveals that the superset languages either have S(M(spec1)), S(M(spec2)) and S(M(cspec)) all set to 1 or have at least one of them set to 1/0. In other words, there is either overt $\bar{A}$-movement of both the subject and the object or there is at least one optional A or $\bar{A}$ movement.

When S(M(spec1)), S(M(spec2)) and S(M(cspec)) are all set to 1, either the subject NP or the object NP must move to Cspec. As a result, each transitive sentence will have two alternative orders, depending on which of the two NPs is in Cspec. Take the first setting in #9 as an example. In this setting, all the S(M)-parameters for head movement are set to 0. The verb therefore remains VP-internal at Spell-Out. The subject and object NP must move to Agr1spec and Agr2spec respectively, and one of them must continue to move to Cspec. When a sentence is transitive, the word order is SOV if the subject is in Cspec and OSV if the object is. Hence the scrambling phenomenon. Similar situations are found in the 2nd setting for #9, the 1st, 2nd and 3rd settings for #10, and the the 1st setting for #11.

We now consider the second source of scrambling: optional movement. That optional movement can create scrambling is fairly obvious. If the movement is not string-vacuous, we will have one word order if the movement does occur and another one if it does not occur. In terms of parameter settings, any setting that has 1/0 as the value for one of the parameters is equivalent to the merge of two different parameter settings, one with that parameter set to 1 and the other with that set to 0. For instance, the setting [0 0 0 0 0 1/0 1 0] is equal to [0 0 0 0 0 1 1 0] plus [0 0 0

[7] The only exceptions are the superset languages for #6 [o s v] and #7 [o v s]. For instance, #3 [o s v, s v] is a superset language for #6 [o s v] but there is no scrambling in #3. However, the languages in #6 and #7 are odd in the sense that they do not contain any string which consists of one NP only. In other words, no intransitive sentences can occur in these languages. Such patterns do not seem to occur in natural languages. Looking at the parameter settings for #6 and #7, we see that they share the property of having S(M(spec1)) set to 0, S(M(spec2)) set to 1 or 1/0, and S(M(cspec)) set to 1. In other words, they all require that Cspec be filled before Spell-Out but only the object can move there. There is no overt subject NP movement to Agr1spec and hence no movement of the subject to Cspec. This situation will not arise if our grammar has a constraint that says "no overt movement to Cspec is possible unless at least the subject NP can move to Agr1spec before Spell-Out." (i.e. S(M(cspec)) cannot be set to 1 unless S(M(spec1)) is .) In any case, the languages in #6 and #7 can probably be ignored. If so, all superset languages are scrambling languages.

0 0 0 1 0]. The language generated with this setting is the union of the language generated with [0 0 0 0 0 0 1 1 0] (i.e. #4 [s o v, s v]) and the one generated with [0 0 0 0 0 0 1 0] (i.e. #3 [o s v, s v]). This is why the language for [0 0 0 0 0 1/0 1 0] is #9 [s o v, o s v, s v]. This is also why #9 is a superset language for #3 and #4. In general, the parameter value 1/0 is a variable which can be instantiated to either 1 or 0 and a setting with n parameters set to 1/0 can be instantiated to 2^n settings. For instance, a setting with two parameters set to 1/0 – [... 1/0 1/0 ...] – can have the following four instantiations:

```
[ ... 1  1 ... ]        [ ... 1  0 ... ]
[ ... 0  1 ... ]        [ ... 0  0 ... ]
```

If none of the movements controlled by these parameters is string-vacuous, a language with n optional movements can then have 2^n subset languages. In Appendix B.2 there are many cases where the setting has one or more parameters set to 1/0, but the the language generated is not a scrambling one. These are all cases where the optional movement is string-vacuous. The subset languages are thus string-identical. Consider the setting [0 0 0 0 0 1/0 0 0] which is equivalent to [0 0 0 0 0 1 0 0] plus [0 0 0 0 0 0 0 0]. The movement which is optional here is subject NP movement to Agr1spec. Since both the verb and the object remain in situ (in the order of VO), we will have SVO whether the subject is in Vspec or in Agr1spec.

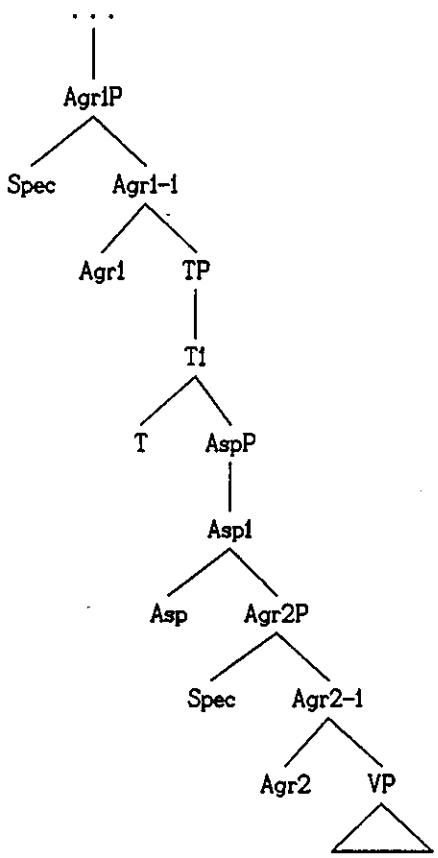
To sum up, there are two sources for scrambling or superset languages: overt Ā-movement and optional movement. This observation is important for learnability issues. The significance of this observation will show up in Chapter 5.

4.2 Other Parameters

In 4.1 we investigated the parameter space of S(M)-parameters. We kept the values of HD-parameters and S(F)-parameters constant in order to concentrate on the properties of S(M)-parameters. Now we will start taking those other parameters into consideration. In what follows, we will bring in the HD-parameters first, then some S(F)-parameters, and finally all the S(F)-parameters.

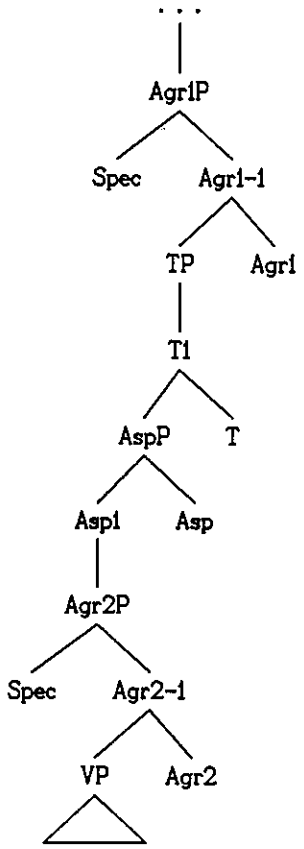
4.2.1 HD-Parameters

Only two HD-parameters are assumed in our system: one for CP and one for IP. There is no specifier-head parameter for any category and there is no complement-head parameter for lexical categories. The parameter for CP (HD1) determines whether CP is head-initial or head-final. The parameter for IP (HD2) determines for every segment of IP (i.e. Agr1P, TP, AspP and Agr2P) whether the head precedes or follows the complement. If the parameter is set to "I", all those segments will be head-initial and the whole IP will have the structure in (115(a)). The whole IP will be head-final if the parameter is set to "F", as shown in (115(b)).



(115)

(a)



(b)

It is evident that the values of HD-parameters can affect the linear order of a sentence. In our system, they work in conjunction with the values of S(M)-parameters in determining the word order of a language. Given a certain value combination of S(M)-parameters, the order may vary according to the values of HD-parameters. Our parameter vector now has 10 coordinates:

```
[ S(M(agr2))  S(M(asp))   S(M(tns))    S(M(agr1))  S(M(c))
  S(M(spec1)) S(M(spec2)) S(M(cspec)), HD1     HD2 ]
```

We can keep the values of S(M)-parameters constant and get different word orders by changing the values of HD1 and HD2. Consider the following two settings (a) [1 1 1 0 0 1 0 0 , I I] and (b) [1 1 1 0 0 1 0 0 , F F]. Both settings require that the verb move to T0, the subject NP move to Agr1spec, and the object NP stays VP-internal before Spell-Out, but T0 precedes the VP in (a) and follows the VP in (b). As a result, the language generated with (a) is SVO and the one with

(b) is SOV. It should be noted that the value of a HD-parameter has an effect on word order only if the relevant head is the final landing site of an overt head movement. In (a) and (b), overt head movement reaches T0 which is a head in IP but not C0 which is the head of CP. Consequently, the value of HD1 plays no role in determining the word order. The languages generated with [1 1 1 0 0 1 0 0 , I I] and [1 1 1 0 0 1 0 0 , I F] are identical. So are the languages generated with [1 1 1 0 0 1 0 0 , F F] and [1 1 1 0 0 1 0 0 , F I]. In a setting like [0 0 0 0 0 . . .] where there is no overt head movement, the language will be the same no matter what values HD1 and HD2 may receive.

We observed earlier that, modulo our current grammar, there are 156 settings of S(M)-parameters with which some non-empty language can be generated. Each of these settings can be combined with any of the four settings of HD-parameters: [I I], [F F], [I F], and [F I]. The number of settings is therefore 624. The languages generated with these 624 settings are given in Appendix B.4. As in Appendix B.2, I have grouped together all the settings that generate the same language. It is a surprise to see that, in spite of the four-fold increase in the number of value combinations, the number of distinct languages generated did not increase at all. We had 31 languages when both HD1 and HD2 are fixed to I. We expected to get more languages when the values of HD1 and HD2 are allowed to vary, but there are still exactly 31 languages. In other words, the languages generated when HD1 and HD2 are set to [I F], [F I] and [F F] respectively all form subsets of the languages generated with [I I]. As we can see in Appendix B.4, every language that can be generated by setting HD1 and HD2 to [I F], [F I] or [F F] can also be generated with the parameters set to [I I]. This might suggest that the parameterization of head directions is superfluous. What is the point of varying the directions if the variation accounts for no more word order facts? Why not eliminate the parameters and assume that all projections are universally head-initial? These will certainly be arguments in favor of the views in Kayne (1992, 1993) and Wu (1993).

However, the superfluity of HD-parameters is more apparent than real. The parameterization does not seem to increase the generative power of the grammar because the strings in Appendix B.2 and Appendix B.4 are too simple. They contain nothing more than S, O and V. As soon as other constituents are allowed to appear in the strings, the difference in generative power will show up. We can add *often* to the strings and see what happens.[8] When *often* is added while the HD-parameter values are fixed to [I I], 61 languages are distinguished. When the HD-parameter values are allowed to vary, we can distinguish 86 languages if *often* appears in the strings. (Due to space limitation, the list of settings and languages are not given in the Appendix.) The increase in number is not dramatic, but it does show that there are things that can fail to be generated if HD-parameters are eliminated altogether. One of the languages that cannot be generated if all projections are head-initial is [often s v, often s o v]. To get the SOV order while maintaining a strictly head-initial structure, the subject NP must move to Agr1spec and the object NP to Agr2spec. But the

[8] We assume as before that *often* is left-adjoined to T1.

movement to Agr1spec must go beyond *often*, putting the subject on its left side. Thus [s often o v] is possible while [often s o v] is impossible. However there is no problem in generating the latter if head direction is permitted to vary. We can get this order with the setting [1 1 1 0 0 0 0 0 , F F], for instance. This setting makes the verb move overtly to T0 while keeping everything else in situ. Since TP is head-final, the verb that lands in T0 will follow every other elements in the string. The string that results is [often s o v].

There are other languages which cannot be generated in the absence of HD-parameters. In Chapter 2, we mentioned languages with sentence-final auxiliaries and grammatical particles. In the next section, we will look at them again. We will find that the addition of the two HD-parameters can make a big difference in terms of generative power.

4.2.2 Value Combinations that Predict Auxiliaries

So far we have put all S(F)-parameters aside. This has several consequences. First of all, no "language" that has been generated exhibits morphology. Morphological variation has been kept out of the picture. Secondly, no functional heads are spelled out. The head of a functional category consists of a set of features but it has no lexical content. We assume that, unlike the head of a lexical category which is visible regardless of the value of S(F)-parameters, the head of a functional category is not visible unless its features are spelled out. When all S(F)-parameters are set to 0, no functional heads are visible and the only head that can undergo overt head movement is the verb. Once some functional heads are spelled out, however, the situation will be different. We have assumed earlier that the feature matrix of a functional head can be spelled out as auxiliaries, grammatical particles or expletives. Apparently, auxiliaries can also undergo head movement. This will have an important consequence on the number of non-empty languages that can be generated. We have noted in 4.1 that, due to the head movement constraint and our temporary assumption that the only kind of head movement is verb movement, many settings generate empty languages because some required movement is blocked. If functional heads can be spelled out and participate in head-movement, the picture will be different.

Let us consider the case where S(M(tns)) is set to 1-0. Recall that each S(F)-parameter can have two sub-parameters in the form F-L. The value of F tells us whether the F-feature (i.e. the feature residing in a functional category) is spelled out and L tells us whether the L-feature is spelled out. The value 1-0 for S(M(tns)) therefore means that the tense feature is spelled out in T0 but not on the verb (This is possible only if the verb does not move to T0 before Spell-Out. Otherwise, the F-feature and the L-feature will have merged into a single one which can appear on the verb only.) Let us assume that, when spelled out, T0 appears as an auxiliary. We will use "Aux" to represent such an auxiliary. Therefore the strings produced by our grammar may now contain Aux in addition to S, O and V. As a visible head, Aux can undergo head movement just as a verb does. This has the consequence of making it possible to generate non-empty languages with settings which

are previously capable of generating empty languages only. Look at [0 0 0 1 1 . . .] again. This setting requires that the verb remain in situ but something must move to Agr1-0 and then to C0. This is impossible when the only head that moves is the verb. Now that we have another head that moves, this setting is no longer impossible: the verb can stay in situ whereas the overt movement from T0 to Agr1-0 and C0 can be performed by Aux.

There are six value combinations of S(M(agr2)), S(M(asp)), S(M(tns)), S(M(agr1)) and S(M(c)) [9] which can start generating non-empty languages once T0 is spelled out as an auxiliary. They are [0 0 0 1 0 . . .], [0 0 0 1 1 . . .], [1 0 0 1 0 . . .], [1 0 0 1 1 . . .], [1 1 0 1 0 . . .], and [1 1 0 1 1 . . .]. The number of distinct languages that can be generated in this particular parameter space (with HD-parameter constantly set to I) is 83 instead of the original 31. Those 83 languages is listed in Appendix B.5. To save space, only one possible setting for each language is given, but this should be enough for the illustration of how each of the languages could be derived.

Now we try varying the values of HD-parameters. With S(F(tns)) set to 1-0 and all other S(F)) parameters to 0-0, the number of distinct "languages" that can be generated with all possible value combinations of HD-parameters and S(M)-parameters is 117. These languages are listed in Appendix B.6. As in Appendix B.5, only one setting is given for each language for the purpose of saving space. The result of this experiment shows again that there are languages that cannot be generated without HD-parameters. When we ignored the HD-parameter by allowing for head-initial constructions only, 83 languages were generated. Thirty-four additional languages are generated when the two HD-parameters are brought into play. There are languages which can be generated only if CP is head-initial and IP is head-final (e.g. #14). There are also languages which are possible only if CP is head-final and IP is head-initial (e.g. #18). So far there is no language which cannot be generated unless both CP and IP are head-final. But there will be such cases when more than one functional head is spelled out, as we will see.

The incorporation of auxiliaries into our system has given us a richer typology. In Appendix B.2 or Appendix B.4, where there is no auxiliary due to the fact that all S(F)-parameters are ignored, only one pure SVO language[10] is distinguished: [s v, s v o]. As a result of setting S(F(tns)) to 1-0 and thus allowing for the appearance of one auxiliary, we now have four different SVO languages: #1 [aux s v, aux s v o], #2 [s v aux, s v o aux], #4 [s aux v, s aux v o] and #20 [s v, s v o].

Certain predictions are made in this partial typology of SVO languages. Among other things it is predicted that no T0 auxiliary can appear between the verb and the object in an SVO language. The sequence [v aux o] is impossible because of the following contradiction. The fact that the T0 Aux precedes O shows that IP must be head-initial. In this case the verb must be higher than T0 in order to appear to the left of the T0 Aux. However, if the verb is higher than T0, it must have

[9] These are the 5 S(M)-parameters that are responsible for the spell-out of head movements.
[10] By "pure" I mean there is no scrambling.

moved through T0 and become unified with it. In this case, T0 will not be able to be spelled out by itself as an Aux. Now what if we do find the sequence [s v aux o] in natural languages? One potential example can be found in Chinese:

(116) *Ta mai le nei ben shu*
he buy Past that book
'He bought that book.'

The past tense marker *le*[11] looks like a T0 auxiliary that appears between V and O. However, this may not be a counter-example to the prediction under question. We may analyze *le* as a suffix of the verb, i.e. a tense feature which is spelled out on the verb. The sequence we are looking at here is therefore [s v-[tns] o] rather than [s v aux o]. The former can be generated if S(F(tns)) is set to 0-1 (spell out the L-feature only) instead of 1-0.

In some cases a tense marker can be analyzed either as an auxiliary or an affix. Take the Japanese sentence (117) as an example.

(117) *Taroo-ga Hanako-o mi-ta*
Taroo-nom Hanako-acc see-past
'Taroo saw Hanako'

The tense marker *ta* can be treated either as a suffix to the verb (the string being [s o v-[tns]]) or as a T0 auxiliary (the string being [s o v aux]). The former is possible in a setting, for example, where S(M(tns)) is set to 0-1 and the S(M) and HD-parameters are set to [0 0 0 0 0 1 1 0 , i i]. In this case, both CP and IP are head-initial. The subject and object move to Agr1spec and Agr2spec respectively and the verb remains in situ with its tense feature (the L-feature) spelled out. The latter sequence ([s o v aux]) is possible, for instance, when S(F(tns)) is set to 1-0 and the S(M) and HD-parameters set to [1 1 0 0 0 0 0 0 , f f]. In this case, both CP and IP are head-final. The subject and object remain VP-internal while the verb moves to Asp0. T0 has not merged with the verb and it is spelled out as an auxiliary. On the basis of (117), we cannot tell if Japanese is [s o v-[tns]] or [s o v aux]. The interesting observation is that a language like Japanese which has been regarded as a typical head-final language can be generated with either a head-initial or a head-final structure in our system.

We have so far only touched upon one type of auxiliary: an overtly realized T0. Other types of auxiliaries can be obtained by spelling out other functional heads such as C0 and Asp0. I will not explore these possibilities exhaustively as I did with the spell-out of T0. Some of them will be mentioned later on in this chapter when we consider the settings for some real languages, but it will basically be left for the reader to figure out what will happen when, say, S(F(pred)) or S(F(asp)) is set to 1-0.

[11] *Le* has traditionally be treated as an aspect marker. See Chiu (1992) for arguments for the treatment of *le* as a tense marker.

One desirable property of the present account of auxiliaries is that no extra machinery is needed to get a much richer typology. The S(M)-parameters and S(F)-parameters are not specifically designed to account for auxiliaries. We need them for independent reasons: S(M)-parameters for word order variation and S(F)-parameters for morphological variation. It just happens that certain value combinations of those two types of parameters predict the occurrence of auxiliaries. In other words, we are able to accommodate auxiliaries in our parameter space at no extra cost.

We conclude this section by pointing out that, no matter whether there are auxiliary movements or not, the verb always moves all the way up to C0 at LF. For reasons relating to the principle of Full Interpretation (Chomsky 1991, 1992), auxiliaries are assumed to be invisible at LF. Given a setting like [0 0 0 1 1 . . .] where there is no overt verb movement but T0 moves overtly to C0 as an auxiliary, the Aux along with the movement it has undergone will disappear at LF where the verb will move to Agr2-0, Asp0, T0, Agr1-0 and C0.

4.2.3 S(F)-parameters

Up till now we have examined the parameter space created by S(M)-parameters, HD-parameters and one S(F)-parameter (S(F(tns))). When we take the all other S(F)-parameter into consideration, combining their values with S(M)- and HD-parameters, the number of possible settings is huge and the number of languages that can be generated will be in the order of tens of thousands. It is impossible to list all those languages in the Appendix, not to mention the settings each of those languages can be generated with. The best we can do here is to look at a small subset of them and get some idea of what kinds of languages can be generated when all the S(F)-parameters enter the parameter space. One way to do it is to keep the values of S(M)-parameters relatively constant while varying the values of HD- and S(F)-parameters. In the following experiment, we will restrict the possible settings of S(M)-parameters to just two: [0 0 0 1 0 1 0 0 . . .] and [1 1 1 1 0 1 0 0 . . .]. The first setting represents the case where there is auxiliary movement but no overt verb movement; the second is a case where there is overt verb movement and no auxiliary shows up. The two HD-parameters will work as usual, with four possible settings. Of the six S(F)-parameters that have been assumed – S(F(agr)), S(F(case)), S(F(tns)), S(F(asp)), S(F(pred)) and S(F(op)) – two will be kept constant and the other four allowed to vary. The two S(F)-parameters whose values will be kept constant in the experiment will be S(F(pred)) and S(F(op)). They will always be set to 0-0. As a result, we will not see in this experiment any language where C0 or Cspec is spelled out. The other S(F)-parameters will have some of their values considered. S(F(agr)) will vary between three values 0-0 (no agreement features spelled out), 1-0 (agreement features spelled out on the auxiliary), and 0-1 (agreement features spelled out on the verb). S(F(case)) will vary between 0-0 (no case feature spelled out) and 0-1 (case feature spelled out on the noun). S(F(tns)) varies between 0-0 (no tense feature spelled out), 1-0 (tense feature spelled out on the auxiliary), and 0-1 (tense feature spelled out on the verb). The auxiliary which spells out T0 will continue to be called Aux. Finally,

S(F(asp)) varies between 0-0 (no aspect feature spelled out) and 0-1 (aspect feature spelled out on the verb).[12] Each parameter setting will now be a vector of 14 coordinates:

```
[S(M(agr2)) S(M(asp)) S(M(tns)) S(M(agr1))
 S(M(c)) S(M(spec1)) S(M(spec2)) , HD1 HD2 ,
 S(F(case)) S(F(agr)) S(F(tns)) S(F(asp)) ]
```

As we can see, even the S(F)-parameters which are active will not have its full range of value variation tried out in the experiment. Only a subset of their possible values is to be considered. All this is done for the purpose of illustrating the range of variation by looking at a very small sample of the "languages" that are generated. This small sample should be enough to give us some idea as to what languages can be accommodated in our parameter space when all parameters are fully active.

The value combinations and the languages that are generated in this very restricted parameter space is given in Appendix B.7 (only one of the possible settings is shown for each language). Forty-eight distinct languages are generated. These languages form a small subset of the SVO and SOV languages that can be generated in our system.

Looking at the strings in each language, we notice that every terminal symbol in these strings has a list attached to it. The list contains information about inflectional morphology. The appearance of a feature in the list indicates that this feature is morphologically visible (spelled out). Any symbol that has an empty list attached to it has no overt inflectional morphology. The list can contain more than one feature when the terminal symbol is inflected for more than one feature. The feature list is *unordered*, which means that the order in which the features are listed has no implication for the order of affixation or whatever other ordering. A symbol like v-[agr,tns] does not necessarily mean v-agr-tns where 'agr' and 'tns' are actual morphemes attached to the verb. It only indicates that the verb is inflected for those two features. How the inflection is morphologically represented is not our concern here.

The symbols that appear in Appendix B.7 and the syntactic entities they represent are displayed in (118).

(118) s-[]	a subject NP with no case-marking
s-[c(1)]	a subject NP overtly marked for Case 1
o-[]	a object NP with no case-marking
o-[c(2)]	a object NP overtly marked for Case 2
v-[]	a verb with no inflection
v-[agr]	a verb inflected for agreement
v-[tns]	a verb inflected for tense
v-[asp]	a verb inflected for aspect
v-[agr,tns]	a verb inflected for both agreement and tense
v-[agr,asp]	a verb inflected for both agreement and aspect

[12] The value 1-0 is impossible with the two settings of S(M)-parameters we are restricted to here.

v-[tns,asp]	a verb inflected for both tense and aspect
v-[agr,tns,asp]	a verb inflected for agreement, tense and aspect
aux-[tns]	the T0 auxiliary
aux-[tns,agr]	the T0 auxiliary inflected for agreement

From the sample in Appendix B.7, which mainly illustrates the range of morphological variation in our system, and Appendix B.4, which illustrates word order variation, we can tell how many typological distinctions can be made in the parameter space. With all the parameters working together, we can get languages with almost any basic word order and with many different types of inflectional morphology. We should list all the "languages" that can be generated in this parameter space and try to match each of them with a natural language. Given the huge number of languages in the parameter space, such listing is impossible in a thesis of the present size. However, to get a better understanding of the generative power of our present system, we will try to fit at least *some* real languages into the space. In what follows, therefore, we will choose some languages for case study. These case studies will put us in a better position to judge the potential and limitations of the present model.

4.3 Case Studies

In this section, we will look at a few natural languages and see to what extent they can be accommodated in the parameter space we have assumed. It is unrealistic to expect our parameter space to account for everything of any real language. There are many reasons why this should be so. The grammar we have been using is only a partial UG. There are other modules of UG which have not been taken into consideration so far. We are therefore bound to run into facts that cannot be explained until our model is interfaced with those other modules. The present module is only concerned with basic word order and basic inflectional morphology. Even in these domains we have further restricted ourselves to simple declarative sentences whose only components are S, V, O, Aux and possibly some adverb. Consequently, the "languages" generated in our parameters cannot be exact matches of natural languages. However, this does not prevent those "languages" from *resembling* certain natural languages or some subsets of natural languages. When we say that a certain language is accommodated in our parameter space, we mean that there is a parameter values combination that generates a "language" which is a rough approximation of this natural language. We have a long way to go before we can account for everything with our model, but there is no reason why we should not find out how much can be done in the the current partial model. In what follows, we will be considering some subsets of English, Japanese, Berber, German, Chinese and French. For convenience we will refer to these subsets as English, Jananese, etc., meaning some small subsets of those languages.

4.3.1 English: An SVO Language

The first question we have to deal with is how to represent English as a set of strings in the format we have been using here. In terms of word order, English is SVO. In additions, adverbs of the *often* type appears before the verb. The order OSV is found in topicalization. Morphologically, English pronouns are overtly marked for case. The verb in English shows overt tense and subject-verb agreement. We may therefore tentatively describe English as (119).

(119) s-[c(1)] (often) v-[agr,tns,asp]

s-[c(1)] (often) v-[agr,tns,asp] o-[c(2)]

o-[c(2)] s-[c(1)] (often) v-[agr,tns,asp]

The language in (119) can be generated with many different parameter settings. One of them is (120).

(120) [1 1 0 0 0 1 1 1/0 , i i , 0-1 0-1 0-1 0-1]

Other settings include:

(121) (a) [0 0 0 0 0 1 1 1/0 , i i , 0-1 0-1 0-1 0-1]
(b) [1 0 0 0 0 1 1 1/0 , i i , 0-1 0-1 0-1 0-1]
(c) [1 1 1 0 0 1 1 1/0 , i i , 0-1 0-1 0-1 0-1]
(d) [0 0 0 0 0 1 1/0 1/0 , i i , 0-1 0-1 0-1 0-1]
(e) [1 0 0 0 0 1 1/0 1/0 , i i , 0-1 0-1 0-1 0-1]
(f) [1 1 0 0 0 1 1/0 1/0 , i i , 0-1 0-1 0-1 0-1]
(g) [1 1 1 0 0 1 1/0 1/0 , i i , 0-1 0-1 0-1 0-1]
(h) [0 0 0 0 0 1 1 1, i i , 0-1 0-1 0-1 0-1]
(i) [1 0 0 0 0 1 1 1 , i i , 0-1 0-1 0-1 0-1]
(j) [1 1 0 0 0 1 1 1 , i i , 0-1 0-1 0-1 0-1]
(k) [1 1 1 0 0 1 1 1 , i i , 0-1 0-1 0-1 0-1]
(l) [0 0 0 0 0 1 1/0 1 , i i , 0-1 0-1 0-1 0-1]
(m) [1 0 0 0 0 1 1/0 1 , i i , 0-1 0-1 0-1 0-1]
(n) [1 1 0 0 0 1 1/0 1 , i i , 0-1 0-1 0-1 0-1]
(o) [1 1 1 0 0 1 1/0 1 , i i , 0-1 0-1 0-1 0-1]
etc.

According to the setting in (120), English is a strictly head-initial language. At Spell-Out, the verb moves to Asp0, the subject NP to Agr1spec and the object NP to Agr2spec. Furthermore, one of the XPs may optionally move to Cspec. We have the SVO order when Cspec is unfilled or filled by the subject. The OSV order occurs when the object moves to Cspec. If *often* appears in the

sentence, it may go to Cspec instead of the subject or object. We then have the strings in (122) in addition to the ones in (119).

(122) often s-[c(1)] O v-[agr(1),tns,asp]

often s-[c(1)] O v-[agr(1),tns,asp] o-[c(2)]

Morphologically this setting requires that the agreement features be spelled out on the verb, the case features spelled out on the noun, and the tense and aspect features spelled out on the verb.

Several questions arise immediately. First of all, the SVO and OSV orders are given equal status in (119). This seems undesirable for it fails to reflect the fact that the SVO order is more basic and occurs far more frequently than the OSV order. But this problem is more apparent than real. With our current setting, the OSV order occurs only if the object has undergone the optional $\bar{A}$-movement to Cspec. We know from the Principle of Procrastinate that, given the option of whether to move overtly or not, the default choice is always "do not move". Therefore the object will not move to Cspec unless this default decision is overridden by some other factor such as discourse context. As a result, we will find SVO most of the time and find OSV only in those situations where topicalization is required. Things would be very different if we have the setting in (123) or any of the settings in (121(h))-(121(o)).

(123) [1 1 0 0 0 1 1 1 , i i , 0-1 0-1 0-1 0-1 , 1]

This setting can also account for the strings in (119), but it requires that one of the XPs *must* move to Cspec. If this is the setting for English, we will have to find some other explanation for the peripheral nature of the OSV order. For this reason, the settings in (121(h))-(121(o)) are less likely to be the correct settings for English.

The second question concerns inflectional morphology. The values of S(F)-parameters are currently assumed to be absolute. Each parameter is set to a single value and no alternation between different values are permitted. This seems to create a number of problems:

(124) We have set S(F(case)) to 0-1 but not every NP in English is overtly marked for case. Only the pronouns are.

(125) S(F(asp)) is set to 0-1 but not every verb seems to inflect for aspect.

(126) S(F(agr)) and S(F(tns)) are set to 0-1, indicating that agreement and tense are to be spelled out on the verb only. This seems contrary to the fact that these features can also be spelled out in an auxiliary in English. This actually leads to the more general problem that the setting in (120) does not let auxiliaries occur in this language.

We will deal with these problems one by one.

The problem in (124) may be solved by refining our parameter system. So far we have not tried to differentiate various types of NPs. The S(F(case)) parameter, which is associated with the whole class of NP, is blind to the distinction, for example, between pronouns and other NPs. Since the value of this parameter does not seem to apply across the board to all types of NPs (at least in English), we may need to distinguish two S(F(case)) parameters, one for pronouns and one for other NPs. Once this distinction is made, the absolute nature of the S(F)-parameter values is no longer a problem. In fact, alternation of parameter values should not be permitted, for pronouns *must* be case-marked in English and other NPs *must not* be case-marked. The S(F(case)) parameter for pronouns is always set to 0-1 and that for other NPs always to 0-0. If the learner is able to distinguish between pronouns and other NPs, the parameters can be correctly set. How the learner becomes aware of this distinction is of course a different learning problem that needs to be addressed separately.

The problem in (125) is not a problem if we assume that a verb is inflected as long as it is morphologically different from other verbs. With regard to aspect marking, the progressive aspect is spelled out as *-ing* and the perfective aspect as *-ed*. When a verb has neither *-ing* nor *-ed* attached to it, we know that this verb has an aspect feature which is neither progressive nor perfective. In this sense, this verb has had its aspect features overtly marked through zero inflection.

Now we look at the problem in (126). The setting in (120) does not allow for the following set of strings which are actually found in English.

(127) s-[c(1)] aux-[agr,tns] (often) v-[asp]
s-[c(1)] aux-[agr,tns] (often) v-[asp] o-[c(2)]
o-[c(2)] s-[c(1)] aux-[agr,tns] (often) v-[asp]

Each of these strings contains an auxiliary where the agreement and tense features are spelled out. Verbs are inflected for aspect only. For this set of strings to be generated, S(F(agr)) and S(F(tns)) must be set to 1-0 rather than 0-1. In addition, S(M(agr1)) may have to be set to 1 so that the T0 auxiliary can move to Agr1 to pick up the agreement features. In other words, we need the following setting.

(128) [1 1 0 1 0 1 1 1/0 , i i , 1-0 0-1 1-0 0-1]

To generate the strings in both (119) and (127), we seem to need a setting which is a merge of (120) and (128), such as the following:

(129) [1 1 0 1/0 0 1 1 1/0 , i i , 1-0 1-0/0-1 1-0/0-1 0-1]

This setting raises several questions. First of all, the fact that S(M(agr)) is now set to 1/0 means that head movement can be optionally overt as well. This option is not available in our minimal model, but what we have seen here suggests that we may have to let S(M)-parameters for head

movement have the value 1/0 just like S(M(cspec)), S(M(spec1)) and S(M(spec2)). There is other evidence in English which shows that head movement can also be optional. So far we have limited our attention to declarative sentences only. As soon as we look at questions, we find that S(M(c)) must be set to 1/0 in English: head movement from Agr1-0 to C0 occurs overtly in questions but not in statements. If so, this movement will be covert unless the Principle of Procrastinate is overridden by some other requirement, such as the need of checking the predication feature (which is in C0) before Spell-Out when this feature has the value "+Q". In any case, it seems necessary that optional head movement should be incorporated into our parameter system.

Another question concerns the fact that S(F(agr)) and S(F(tns)) are set to 1-0/0-1. This setting is intended to represent the fact that (a) agreement and tense features must be spelled out in English, and (b) we can spell out either the F-feature or the L-feature, but not both. When the F-feature is spelled out, an auxiliary appears and this auxiliary may move to Agr1-0 or C0. When the L-feature is spelled out, the agreement and tense features appear on the verb and there is no auxiliary. The question is why we have to spell out the F-feature in some cases but the L-feature in some others. We do not find an answer to this question in our minimal model here. However, once this model is interfaced with other modules of the grammar, the choice may become explainable. It may turn out that negation requires the spell-out of F-features. This might explain why (130) and (131) are grammatical while (132) and (133)) are not.

(130) *John does not love Mary.*

(131) *John did not see Mary.*

(132) *John not loves Mary.*

(133) *John not saw Mary.*

It is also possible that the F-features of agreement and tense must be spelled out when the aspect feature is "strong" or "marked" in some sense. In English, this seems to happen when the aspect is progressive or perfective, as shown in (134) and (135).

(134) *John is writing a poem.*

(135) *John has written a poem.*

The auxiliaries *be* and *have* here are treated as some overtly realized functional heads and can be represented as "aux-[agr,tns]" in our system. Why "aux-[agr,tns]" is spelled out as *be* in some cases, *have* in some other cases, and *do* in most of the other cases has to be explained by theories which have not yet been incorporated into our system.

4.3.2 Japanese: An SOV language

Japanese is a verb-final language with a considerable amount of scrambling. The subject and the object must precede the verb but they can be ordered freely, resulting in SOV or OSV. Japanese NPs are always case-marked[13] and Japanese verbs come with tense markers.[14] There does not seem to be any overt agreement whose function is grammatical.[15]

In terms of surface strings, Japanese can be described as either (136) or (137) depending on whether we treat the tense marker as a suffix or grammatical particle. In (136) the tense marker *ta* is treated as a verbal suffix while in (137) it is treated as an auxiliary which spells out the tense feature in T0.

(136) s-[c(1)] v-[tns]
s-[c(1)] o-[c(2)] v-[tns]
o-[c(2)] [s-[c(1)] v-[tns]

(137) [s-[c(1)] v-[] aux
[s-[c(1)] o-[c(2)] v-[] aux
o-[c(2)] [s-[c(1)] v-[] aux

These patterns are illustrated by the Japanese sentences in (138), (139) and (140).

(138) *Taroo-ga* *ki-ta*
Taroo-Nom come-Past
'Taroo came.'

(139) *Taroo-ga* *tegami-o* *kai-ta*
Taroo-Nom letter-Acc write-Past
'Taroo wrote a letter.'

(140) *tegami-o* *Taroo-ga* *kai-ta*
letter-Acc Taroo-Nom write-Past
'Taroo wrote a letter.'

The languages in (136) and (137) can be generated with the settings in (141) and (142) respectively.

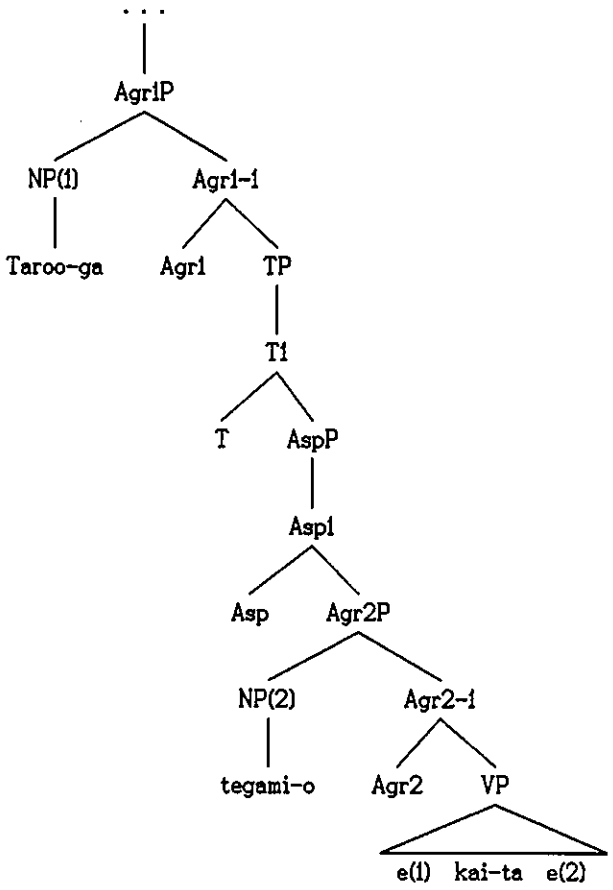
(141) [0 0 0 0 0 1 1 1/0 , i i , 0-0 0-1 0-1 0-0]

(142) [0 0 0 0 0 1 1 1/0 , f f , 0-0 0-1 1-0 0-0]

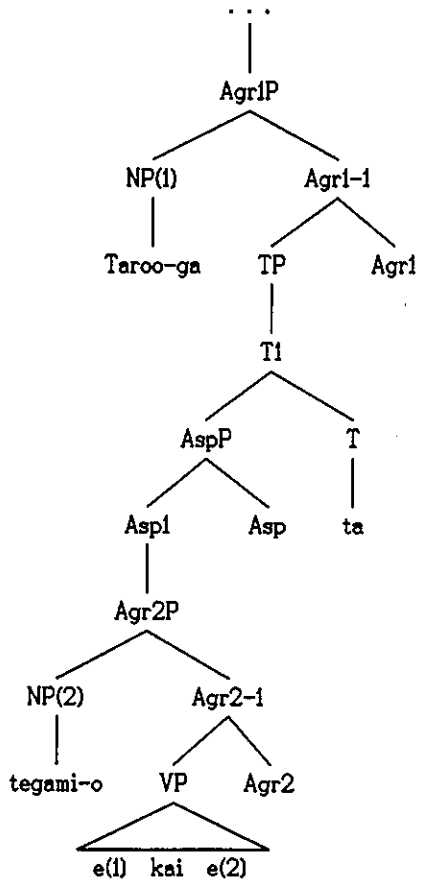
[13] Except in very casual speech.
[14] It is controversial whether there is aspect markers in Japanese. What we mean by tense marker here will include the aspect marker.
[15] There is, however, agreement with respect to levels of honorificness and politeness.

It is required in (141) as well as (142) that both the subject and object move to their case positions (Agr1spec and Agr2spec respectively) and the verb remains in situ. However, CP and IP are head-initial in (141) but head-final in (142). In addition, the value of S(F(tns)) is different in the two settings. In (141) it is set to 0-1 which requires the tense feature to be spelled out on the verb as an affix (spelling out the L-feature). In (142), on the other hand, this feature is to be spelled out in T0 by itself (spelling out the F-feature). The two settings produce very different tree structures, as shown in (143(a)) and (143(b)).

(143)



(a) Tree generated with (141)



(b) Tree generated with (142)

It is not possible to tell on the basis of (138), (139) and (140) which of the two structures is more likely to be the correct one for Japanese. If (143(a)) is the correct one, Japanese will not be a head-final language at all, contrary to common belief. What this shows is that a verb-final language is not necessarily a head-final one. When we look at more data from Japanese, however, we begin to see evidence that (143(b)) is probably the right choice. The following two sentences are examples in support of the setting in (142).

(144) *Taroo-wa* *ki-ta* *ka*
Taroo-Topic come-Past Q-marker
'Did Taroo come.'

(145) *Hanako-ga asoko de nai-te i-ru*
Hanako-Nom there at cry-Cont be-Nonpast
'Hanako is crying there.'

The question marker *ka* in (144) comes at the end of the sentence. The only way to account for it in (143(a)) is to treat *ka* as a verbal suffix attached to *ki*. In other words, two features are spelled on this verb, *ta* being the tense feature and *ka* a predication feature which will be checked in C0 at LF. However, this analysis does not seem to accord with the intuition of native Japanese speakers who usually regard *ka* as a separate word. If *ka* is indeed not part of the verb, we will have to adopt the analysis in (143(b)) where *ta* is an auxiliary or grammatical particle in T0 and *ka* in C0. Turning to (145), we again see the plausibility of (143(b)). To maintain (143(a)) we would have to say that *nai-te-i-ru* forms a single big verbal complex, which is again a bit far-fetched. In (143(b)), however, everything is comfortably accounted for: *nai-te* is in V0 and *i-ru* is in T0. It is also possible that *nai* is in V0, *te* in Asp0 and *i-ru* in T0.

4.3.3 Berber: A VSO Language

Berber is usually considered a VSO language, but other orders are also found. The most common alternative order is SVO which is usually used in topicalization.(Sadiqi 1986). Here are some examples:

(146) *i-ara hmad tabrat*
3ms-wrote Ahmed letter
'Ahmed wrote the letter.'

(147) *hmad i-ara tabrat*
Ahmed 3ms-wrote letter
'Ahmed wrote the letter.'

In terms of morphology, there is no overt case marking in Berber, but verbs are inflected for tense/aspect and agreement, as we can see in (146) and (147). This language thus have the following set of strings in our abstract representation:[16]

(148) v-[agr,tns,asp] s-[]

v-[agr,tns,asp] s-[] o-[]

s-[] v-[agr,tns,asp]

s-[] v-[agr,tns,asp] o-[]

This set of strings can be generated with the parameter setting in (149).

[16] The fact that the feature list is attached to the verb on the right in our representation does not have any implication as to whether the features are spelled out as prefixes or suffixes. It simply means those features are realized on the verb. They can appear as any kind of affix (prefix *or* suffix) or other forms of verbal conjugation.

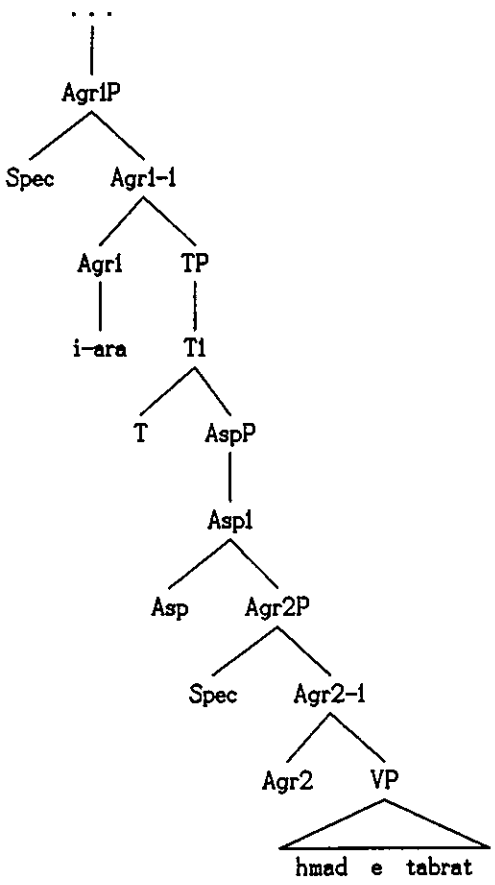
(149) [1 1 1 1 0 1/0 0 1/0 , i i , 0-1 0-0 0-1 0-1]

There are many alternative settings that are compatible with these strings. Here are some examples:

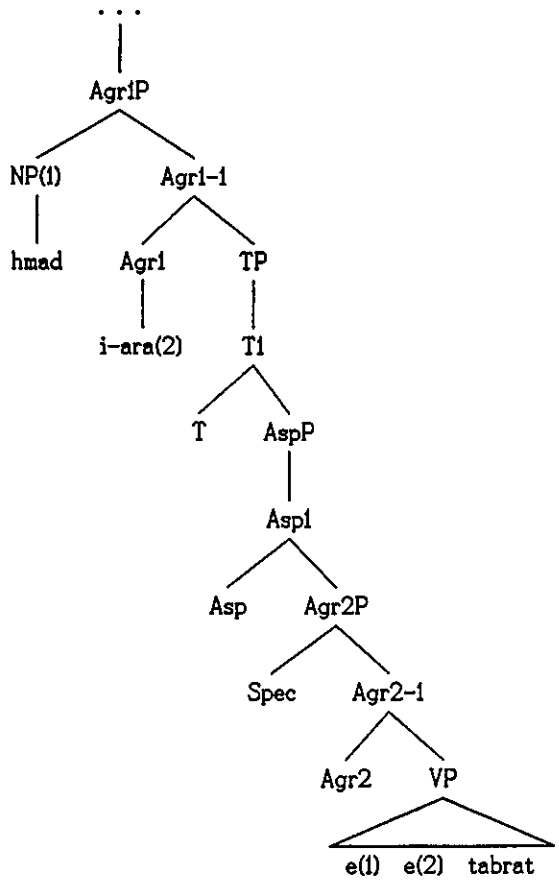
```
[ 1  0  0  0  0  1/0  0  1/0 ]        [ 1  1  0  0  0  1/0  0  1/0 ]
[ 1  1  1  0  0  1/0  0  1/0 ]        [ 1  1  1  1  1  1  0  1/0 ]
```

According to the setting in (149), the verb must move overtly to Agr1 and the object must stay in situ. The subject, however, can optionally move to Agr1spec and then to Cspec. If the principle of procrastinate is not overridden by other considerations, the subject will not move and the word order is VSO. When other factors call for overt movement, the subject can move to Agr1spec or Cspec. In either case the order is SVO. The tree structures for (146) and (147), according to (149), are (150(a)) and (150(b)) respectively.

(150)



(a)



(b)

One general question that can be raised at this point is whether the order in which the inflectional features appear in the list can imply anything about the actual sequence of affixes. We may be tempted, at least in Berber, to let our feature list have this extra ordering information. For instance, we may let v-[tense,agr] or [agr,tense]-v mean that the affix representing agreement appears outside the affix representing tense, as in the case of (146) and (147). Our string representation would certainly be more informative if the ordering is encoded there. If the Mirror Principle holds, this kind of encoding will not only be desirable but easy as well. Unfortunately, the Mirror Principle does not seem to hold everywhere, not in Berber at least. If we look at (146) and (147) only, we may conclude that agreement occurs outside tense. The verb is inflected for tense and the agreement affix is added to the tensed verb. However, we also find Berber sentences where the order is reversed. (151) is such an example.

(151) *ad-y-segh* *Mohand* *ijn* *teddart*
will(Tns)-3ms-buy Mohand one house
'Mohand will buy a house.'

In (151) tense clearly occurs outside agreement. To avoid such problems, we will insist that the feature list attached to each terminal symbol is *unordered.* They only tell us what features are spelled out in some form of verbal inflection. The order of affixation has to be handled separately. As a matter of fact, we cannot exclude the possibility that the ordering is arbitrary and the learner has to acquire it independently.

4.3.4 German: A V2 Language

German is a language where root clauses and subordinate clauses have different word orders. In root clauses, the verb must appear in second position, the first position being occupied by a subject NP, an object NP, an AdvP, or any other XP. This is illustrated in (152), (153) and (154).

(152) *Karl* *kaufte* *gestern* *das* *Buch*
Karl bought yesterday that book
'Karl bought that book yesterday.'

(153) *das* *Buch* *kaufte* *Karl* *gestern*
that book bought Karl yesterday
'That book Karl bought yesterday.'

(154) *gestern* *kaufte* *Karl* *das* *Buch*
yesterday bought Karl that book
'Yesterday Karl bought that book'

Assuming that German NPs are inflected for case[17] and German verbs are inflected for tense, aspect and agreement, we can abstractly represent German root clauses as the set of strings in (155) (where 'adv' stands for an AdvP like *yesterday* which is presumably left-adjoined to T1.)

[17] The case marking shows up on the determiner, though.

(155) s-[c(1)] v-[agr,tns,asp] (adv)

adv v-[agr,tns,asp] s-[c(1)]

s-[c(1)] v-[agr,tns,asp] (adv) o-[c(2)]

o-[c(2)] v-[agr,tns,asp] s-[c(1)] (adv)

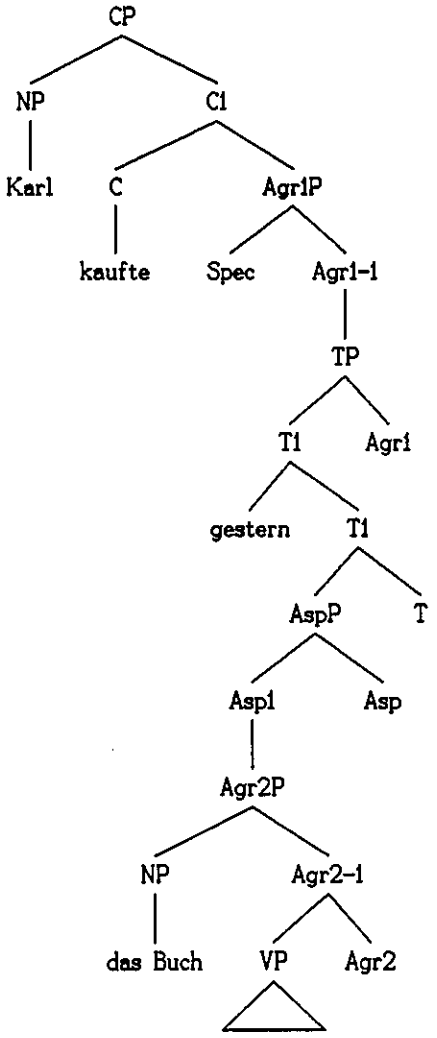
adv v-[agr,tns,asp] s-[c(1)] o-[c(2)]

This set of strings can be generated with the following parameter setting:

(156) [1 1 1 1 1 1 1 1 , i f , 0-1 0-1 0-1 0-1]

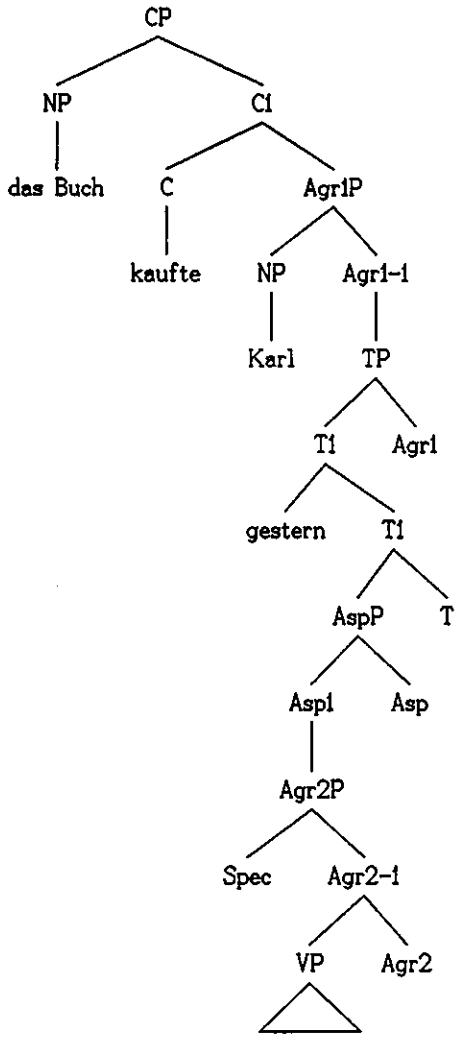
This setting requires that *every* movement be overt. By Spell-Out, the verb must move to C0, the NPs to Agrspecs, and one of the XPs to Cspec. We have (152) if the subject NP moves to Cspec, (153) if the object does, and (154) if the AdvP does. The setting also specifies that CP is head-initial and IP is head-final.

Incidentally, the structures predicted by this setting can also account for the fact that *gestern (yesterday)* can appear right after the verb in (152) but not in (153). We have assumed that a time adverb like *yesterday* can be left-adjoined to T1. In (152) the object has moved to Agr2spec but not to Cspec. This is why we can have the order SV(Adv)O. In (153) the object has moved to Cspec and the subject to Agr1spec. The resulting order can only be OVS(Adv), while OV(Adv)S is impossible. The tree structures for (152) and (153) are in (157(a)) and (157(b)).



(157)

(a)



(b)

German is similar to English in that the tense and agreement features are spelled out on the verb in some cases but in an auxiliary in others. When an auxiliary exists in a sentence, the auxiliary is in second position and the verb in final position. Here is an example:

(158) *Gestern hat Karl das Buch gekauft*
yesterday has Karl that book bought
'Karl bought that book yesterday.'

Obviously, the setting in (156) will fail to account for the word order found in this sentence. This problem may need to be handled in the same way as we handled the English case. We can assume

that tense and agreement features must be spelled out in German, either in an auxiliary (spelling out the F-feature) or on the verb (spelling out the L-feature), but not both. When the F-feature is spelled out, an auxiliary appears. This auxiliary moves to C0 and the verb moves to Asp0 only. When the L-feature is spelled out, there is no auxiliary and the verb will move to C0. Why we choose to spell out the F-features in some cases but the L-feature in some others is again an open question which cannot be answered until our model is interfaced with other components of the grammar.

We have so far only discussed the word order in German root clauses. The order in subordinate clauses is SOV rather than V2, as shown in (159) and (160).

(159) *dass Karl gestern dieses Buch kaufte*
that Karl yesterday this book bought
'that Karl bought this book yesterday.'

(160) *dass Karl gestern dieses Buch gekauft hat*
that Karl yesterday this book bought has
'that Karl bought this book yesterday.'

This fact again forces us to consider the possibility that some S(M)-parameters for head movement (in this case S(M(c))) must be allowed to have the value 1/0. If S(M(c)) is set to 1/0 in German, then the verb can move to Agr1-0, resulting in an SOV order, or move to C0 resulting in a V2 order. Apparently, the Principle of Procrastinate is overridden in the root clause. We may conjecture that the predication feature must be checked before Spell-Out in the root clause but not in the subordinate clause. This checking requirement overrides the Principle of Procrastinate and forces the verb to move to C0 overtly in the root clause.

4.3.5 Chinese: A Head-Final SVO Language

We have seen in (100), (101) and (102) that Chinese is a scrambling language, its possible word orders being SVO, SOV and OSV. All these orders can be generated with a parameter setting like (161) where both CP and IP are head-initial.

(161) [0 0 0 0 0 1 1/0 1 , i i , 0-0 0-0 0-1 0-1]

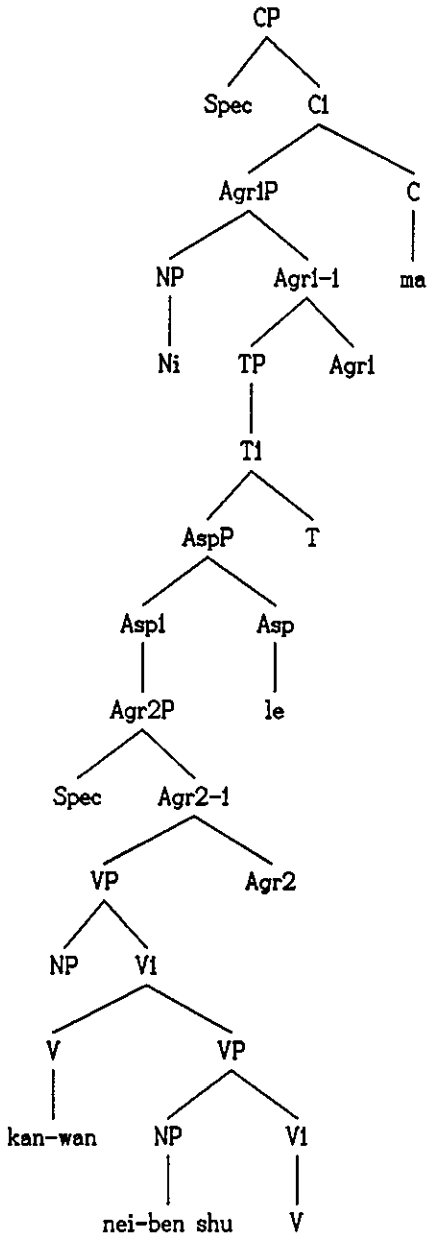
But this setting is not able to account for the following sentence where we find sentence-final particles which cannot possibly be spelled out on the verb.

(162) *Ni kan-wan nei-ben shu le ma*
you finish reading that book Asp Q/A
'Have you finished reading that book? / You have finished reading that book, as I know.'

In this sentence *le* and *ma* are not attached to the verb, since the object intervenes between the verb and those functional elements. A fair assumption is that *le* and *ma* are some overtly realized functional heads, the former being the head of AspP and the latter the head of CP. (This presupposes

that S(F(asp)) and S(F(pred)) are both set to 1-0.) These elements cannot appear in sentence-final positions unless both IP and CP are head-final. What this suggests is that Chinese is a head-final language (in terms of CP and IP).The structure for (162) should be (163) which illustrates how an SVO string can be generated in a head-final structure.

(163)



4.3.6 French: A Language with Clitics

In this case, we are not interested in the French language as a whole, but just its cliticization. Since we are only dealing with simple sentences with two arguments, only sentences like the one in (164) will be considered.

(164) *Je le-visitais*
I him-visited
'I visited him.'

There is a huge amount of literature on the analysis of clitics like *le* here, but we will not try to go through it in this short section. What I want to point out is that our current model may offer an alternative account of this well-known phenomenon. Recall that we observed in Chapter 3 that case and agreement features can be spelled out either on NPs or on the verb. (See Borer (1984) and Safir (1985) for similar ideas.) The parameter S(F(case)) has four values: 0-0 (no case feature spelled out), 0-1 (case features spelled out on the NP), 1-0 (case features spelled out on the verb)[18], and 1-1 (case feature spelled out on both the NP and the verb). We have further assumed that, when spelled out on the verb, the case-features together with the agreement features show up as clitics.

Now let us suppose that the S(F(case)) parameter has a value which is operative only when the object NP is a pronoun. Then the four values of this parameter will have the following effects. When it is set to 0-0, no case features are spelled out. Since a pronoun is nothing more than a set of case and agreement features, no pronoun will be visible in this case. We call this *pro-drop*. When S(F(case)) is set to 0-1, the features are spelled out as a pronoun. In cases where the value is 1-0, the features appear as a clitic instead of a pronoun. The features spelled out here are some F-features of Agr2. The verb acquired those features when it moved through Agr2-0 on its way to Agr1-0. In this sense, clitics are affixes of the verb which spells out some case/agreement features of the verb. This explains why clitics must be adjacent to the verb. Finally, we may have the value 1-1 which requires that the features be spelled out on both the verb and the NP. As a result, we may see the clitic as well as the pronoun, a case of clitic doubling. The value of S(F(case)) seems to be 1-0 in French.

In the GB literature there are basically two different accounts of cliticization. The base-generation account has the view that clitics are based generated on the verb. The movement account argues that clitics are generated in NP positions and later get attached to the verb through movement. Recently syntacticians have been trying to reconcile the two approaches and have proposed the view that cliticization involves both base generation and movement (e.g. Sportiche 1992). This is intuitively very similar to our present analysis. Clitics are base generated in the sense that the features are verbal features and they show up wherever the verb goes. They also involve movement because the verb has to move through Agr2-0 and the object NP has to move to Agr2spec. While

[18]The features to be spelled out in this case are the F-features which the verb can pick up and carry along when it moves through the functional categories.

the verb is in Agr2-0 and the NP in Agr2spec, the verb will get its case/agreement features checked against the object NP through spec-head agreement. It will take more work to see, however, whether the present account can cover all the empirical data that traditional approaches have succeeded in providing an explanation for.

The case studies above have given us some more concrete ideas as to what linguistic phenomena can be accommodated in our parameter space. The studies are incomplete, however, because the list of languages that can be studied this way is an open-ended one. We should have looked at many more languages but a complete survey is beyond the capacity of the present thesis.

4.4 Summary

In this chapter we have laid out the parameter space in our model. We have had a bird's-eye view at all the possible languages in this space as well as a worm's-eye view at some specific languages. We have seen the present parameter space is capable of accounting for a wide range of linguistic facts. In terms of word order and inflectional morphology, most natural languages can find a corresponding "language" in this parameter space. We have also discovered, however, that our present system has its limitations. In order to provide a more complete account of any natural language, the system must be enriched in the future.

Chapter 5

Setting the Parameters

This chapter will be devoted to the issue of learnability. We have defined an experimental grammar with a set of parameters. We have also seen that the parameter space thus created is capable of accounting for a wide range of cross-linguistic variation. The next question is how a learner can acquire different languages by setting those parameters. Is every language in our parameter space learnable? Is there a learning algorithm whereby the parameters can be set correctly? If so, what properties does this learning algorithm have? These are the questions that will be addressed in this chapter. We will see that there exists at least one algorithm which not only results in successful learning but has some desirable learnability properties as well.

5.1 Basic Assumptions

The study of language acquisition is an enormous project which involves many sub-areas of research. We are not trying to look at every aspect of language learning, however. The area we will focus on is a sub-part of syntactic acquisition. It is assumed that there are learning modules that are responsible for the acquisition of other linguistic knowledge such as phonology, morphology and semantics. The learning activities to be discussed here will thus take place in an idealized situation where other kinds of learning are supposed to be taken care of. We will take a number of things as given, and the the success or failure of the learning algorithm we will work out is to be viewed against the backgroud of those given assumptions. It is therefore important to state those assumptions explicitly at the beginning.

5.1.1 Assumptions about the Input

The input to the learning module we are concerned with comprises strings which are abstract representations of Degree-1 declarative sentences.[1] Each input string is assumed to be a CP (i.e. a

[1] A Degree-1 sentence is a sentence with no embedding. See Wexler and Culicover 1980 and Lightfoot 1991 for definitions of the degrees of input.

sentence) and every symbol in the string consists of a category label plus a feature list. Such input strings are usually referred to as *labeled strings* (ref??). It is assumed that some other learning mechanisms can enable the learner to segment a string correctly and figure out the grammatical category of each individual symbol. In addition, the learner is supposed to be able to identify the argument structure of each sentence. He/she can differentiate transitive verbs from intransitive ones and distinguish between subject and object NPs. How such "tagging" is achieved is not the concern of our present study. Finally, it is assumed that the learner can analyze the morphological structures of the target language. She can find out, for instance, that the word *does* in English is overtly marked for the tense and agreement features.

The input strings for our learner are more abstract than the prototypical labeled strings in that (i) no real words appear in the strings, and (ii) the inflections of the words are represented in a feature list. The category labels that can appear in the input strings include the following:

- s (subject NP)
- o (object NP)
- iv (intransitive verb)
- tv (transitive verb)
- aux (auxiliary or grammatical particle)
- often (adverb of the "often" type)

Each category label has a list of features attached to it. The features that appear in the list represent overt morphology, i.e. features that are spelled out. For instance, a string like [s-[c1] aux-[agr,tns] v-[asp] o-[c2]] represents a sentence where the subject and object are overtly marked for different cases, the auxiliary overtly inflects for agreement and tense, and the verb has overt inflection for aspect. The full array of possibilities has been illustrated in 4.2.3. It is taken for granted that the learner is able to identify the inflectional morpheme(s) in each word and the feature(s) encoded in each morpheme.

The language to be acquired by a learner is composed of a set of strings. These strings are to be presented in an *unordered* fashion. The learner can encounter any string at any time. It is assumed that every string in the set will eventually appear in the input and each string (which represents a certain *type* of sentences) can be encountered repeatedly.

All the input strings are supposed to be grammatical sentences in the target language. No sentence is marked "ungrammatical", telling the learner that this is not a sentence he/she should generate. In other words, no negative evidence is available (cf. Brown & Hanlon (1970), Wexler & Hamburger (1973), Baker (1979), Marcus (1993), etc.).

5.1.2 Assumptions about the Learner

The learner is equipped with Universal Grammar which has a set of parameters, each having two or more values. In our case, the UG is the experimental grammar defined in Chapter 3. Whenever an input sentence is encountered, the learner tries to parse it using the grammar and the current parameter setting. At the initial stage, the parameters can be either preset or unset. In the latter case, a setting has to be chosen before the parsing starts. As we will see, whether the parameters are preset or unset does not make a big difference in our model.

We adopt the hypothesis that the learner is failure-driven or error-driven (Wexler & Culicover (1980)). He/she will not change his/her current parameter setting unless he/she encounters a sentence which is not syntactically analyzable with the current setting. We also assume the Greediness Constraint (Gibson and Wexler 1993) according to which the learner will not adopt a new setting unless it can result a successful parse of the current input sentence. Finally, we share with most researchers in the field the assumption that the learner has no memory of either the previous parameter settings or the sentences previously encountered.

An ideal learning paradigm within which parameter setting can be experimented with the above assumptions is Gold's (1967) *identification by enumeration.* This is a failure-driven algorithm whereby the learner goes through a number of hypotheses until the correct one is found. In our case, the algorithm can be described as follows. Given a list of parameter settings, the learner attempts to parse an input sentence S with one of the settings in the list. If S can be successfully parsed, then the setting is retained. If S is unparsable, however, the current setting will be discarded and the next setting in the list will be tried. The settings are tried one by one until the learner reaches a value combination that results in a successful parse. A language is said to be learned or *identified in the limit* when the learner has come to a point where he/she can parse any S in the language without ever changing the parameter setting again.

A well-known property of this algorithm is that the enumeration of hypotheses (in our case the parameter settings) must follow the Subset Principle (Angluin 1978,1980, Williams 1981, Berwick 1985, Manzini and Wexler 1987, Wexler and Manzini 1987). Given two languages L_1 and L_2 and their respective parameter settings P_1 and P_2, P_1 must come before P_2 in the enumeration if L_1 constitutes a proper subset of L_2. We have seen in 4.1.4 that superset and subset languages do exist in our parameter space. This fact has significant implications for the enumerative process of our learning algorithm.

There are three types of parameters to be set in our model: S(M)-parameters, S(F)-parameters and HD-parameters. S(M) and HD parameters account for word order variation. They are reset if and only if the current setting fails to accept the word order of an input string. S(F)-parameters, on the other hand, are responsible for the morphological paradigm of a language. They are reset on the basis of visible morphology only. Thus the values of S(M)- and HD- parameters respond to word order only and the values of S(F)-parameters respond to morphology only. Since word

scope-feature in German has the value in (18(ii)) which requires that this feature be made overt. In (26) the wh-phrase has moved through all the Specs of CP, which means all the scope features have been checked before Spell-Out. So the F-features and L-feature have all gotten unified and we see the wh-phrase only. In (27) and (28), the checking movement is partial and all the unchecked scope features are spelled out as *was*. (29) is ungrammatical because one of the unchecked features is not spelled out, which contradicts the value of S(F) that requires that all scope features be spelled out.

The last example to illustrate the value in (18(ii)) is from French. Previous examples have shown that the specifiers of CP and AgrSP can be spelled out by themselves. The French example here is intended to show that the head of CP can also be spelled out in that way. Let us suppose that C^0 contains a certain predication feature which determines, for instance, whether the sentence is a statement or a question. Let us further suppose that this feature must be spelled out in a question in French. When I-to-C movement takes place, as in (30), this feature is not spelled out by itself. It is merged with the verb in C^0.

(30) *Apprenez vous le russe*
learn you Russian
'Do you learn Russian?'

In cases where no overt I-to-C movement takes place, however, the predication feature must be spelled out on its own. This is shown in (31) where *est-ce que* can be regarded as an overtly realized head of CP.

(31) *Est-ce que vous apprenez le russe*
you learn Russian
'Do you learn Russian?'

In (18(iii)), the S(F)-parameter is set 0 while the S(M)-parameter set to 1. This means the feature is checked through movement before Spell-Out but not overtly realized. We will see overt movement but not overt morphology. This seems to happen to the case/agreement features of the subject NP in Chinese, as (32) shows.

(32) *Zhangsan bu xihuan Lisi*
Zhangsan not like Lisi
'Zhangsan doesn't like Lisi.'

This NP does seem to move to the Spec of IP/AgrSP (Cheng 1991, Chiu 1992). One argument for this is the following: with the assumption that the subject is base-generated within VP and Neg is generated outside VP, the subject could not precede Neg had it not moved to Spec of IP/AgrSP. However, there is no case/agreement marking on this NP at all, which shows that the S(F)-parameter for case/agreement features is set to 0. More examples demonstrating the value in (18(iii)) can be found in Chapter 4.

In (18(iv)), both the S(F)-parameter and the S(M)-parameter are set to 0. In this case, there is neither overt morphology nor overt movement. The object-verb agreement features in many languages seem to exemplify this value: there is no overt agreement marker and the object does not seem to move. Examples of this will be given in Chapter 4.

The examples we have seen so far is sufficient to show that the interaction between S(F)-parameters and S(F)-parameters can give rise to a wide variety of syntactic phenomena. It has added a new dimension to our parameter space.

2.4 Summary

In this chapter, I have proposed a syntactic model which is built on some of the new assumptions in the Minimalist framework. By fully exploiting the notion of Spell-Out, we discovered a model where a wide range of linguistic facts can be accounted for in terms of two sets of S-parameters. The values of S(F)-parameters determine which features are morphologically realized; the values of S(M)-parameters determine which movements are overt. It is found that so much variation in word order can be derived from movement that the number of HD-parameters can be reduced. It is also found that the interaction between S(F)-parameters and S(M)-parameters can make interesting predictions for the distribution of functional elements, such as affixes, auxiliaries, expletives, particles and wh-scope markers. In short, we have found a parameter space which has rich typological implications. So far the model has been presented in a very sketchy way with many details left out. But the details are important. In the chapters that follow, we will make the model more specific and put it to the test of language typology, language acquisition and language processing.

Chapter 3

An Experimental Grammar

In the previous chapter I proposed a new approach to syntax where cross-linguistic variations in word order and inflectional morphology are attributed to three sets of parameters: the S(F)-parameters, the S(M)-parameters, and the HD-parameters. So far the discussion has been very general, with many details unattended to. To fully explore the consequences of this new hypothesis in language typology, language acquisition and language processing, we need to work with a more concrete model. We must have a grammar which is specific enough so that the consequences can be computed. The goal of this chapter is to specify such an experimental grammar.

Obviously, the presentation of a syntactic model which is complete in any sense is an unrealistic goal here. In the first place, the Minimalist theory has not been fully specified. Many issues that the standard P&P model has addressed have not been accommodated in the new program yet, not to mention those areas that even the standard model has left unexplored. In addition, I will not follow MPLT in every detail, though the theory I am proposing is in the Minimalist framework.[1] This leaves more issues open, for even those things that are supposed to have been discussed in MPLT may have to be reconsidered here. The best we can do at this moment is to come up with a partial model and use it as a test ground. The conclusions we draw from the test will not be definitive, but they can at least provide us with some way of evaluating the new theory. Such evaluation will give us some idea as to whether this line of research is worth pursuing at all. For this reason, the syntactic model to be presented in this chapter will be minimal. In particular, we will be concerned only with those parts of the grammar which are responsible for the *basic* word orders and morphological characteristics of languages. The discussion in later chapters may go beyond the grammar described here, but the extensions will be introduced as we go along.

We will start with a grammar which is restricted in the following ways.

[1] As has been stated in the last chapter, I will try to make a distinction between MPLT and the Minimalist *framework*. The former refers to the specific model described in MPLT while the latter refers to the general approach to syntax initiated by MPLT.

- Declarative sentences only. We will focus on cross-linguistic variations in statements first. In many languages, the word orders found in questions are different from those in statements. We do not want to get into this complication before we have a better understanding of how the parameters work in the "basic" type of sentences, i.e. declarative sentences. Therefore, I will put other sentence types aside in this chapter, though some of them will be picked up in Chapter 4.

- Matrix clauses only. We will start with simple sentences with no embedding. In other words, we will be concerned mainly with *Degree-0* sentences.[2] There are two reasons for this temporary exclusion of embedded clauses. First, main clauses and subordinate clauses have different word orders in some languages (e.g. German, Dutch, and many other V2 languages). Why there is this difference deserves some special discussion. We will come to that after matrix clauses have been analyzed. Second, the inclusion of embedded clauses will make it necessary to deal with "long-distance" movement whose application involves the notion of "barriers" or "minimality". How these notions are defined in the Minimalist framework is not clear yet. It is very likely that the standard definitions can be transplanted in the present model without too much tinkering. But I prefer to put these issues aside until we have worked on aspects of the grammar which are more directly related to basic word order and morphology.

- Two types of verbs only. Since I will be mainly concerned with the ordering of S(ubject), O(bject) and V(erb) in this experimental study, I will only look at two types of verbs: (a) intransitive verbs with a single NP argument (e.g. *swim*) and (b) transitive verbs with two NP arguments (e.g. *love*).

- IP/CP only. I will experiment with the ordering in IP/CP first and leave the internal structures of NP/DP aside for the moment. There have been many observations on the parallels between IP/CP and DP (Stowell 1981,1989, Abney 1986, Sabolsci 1989, Valois 1991, etc.). The new approach considered here can definitely apply to the word order phenomena within DP. It is very likely that the movement patterns in IP/CP and NP/DP are related (Koopman, 1993). However, I will single out IP/CP for analysis first, All NPs/DPs will be treated as unanalyzed wholes for the time being. Their internal structures and internal word orders will be given a very preliminary analysis in Chapter 7.

- No binding theory. Binding theory is one of those components of the grammar that needs major re-working in the Minimalist framework. In order not to get distracted from my main topic, I will not go into a Minimalist account of binding theory.

What is listed above does not exhaust the topics which are left out in this chapter. Other things

[2]Discussions on the "degrees" of sentences can be found in Wexler and Culicover 1980, Morgan 1986 and Lightfoot 1989, 1991.

will be noted in the course of presentation. We will see that, in spite of these simplifications and omissions, the model will be rich enough to spell out the basic properties of the present approach. The typology, the parser and the acquisition procedure based on this minimal model will not be complete, but they will be sufficient for the illustration of these properties.

It must be emphasized again that the model to be described below does not follow MPLT in every detail. The model is in the Minimalist *framework* in the sense that it keeps to the *spirit* of the Minimalist approach. The actual specifications are often different from what Chomsky has said. I will try to point out the differences as we go along. It should also be emphasized that the grammar to be described is not the only one where the new approach will work. I am simply trying out a particular instantiation of the theory to show that my proposal can be put to practice in *at least* one given version of the model. By the time when we have completed the experiment, we will realize that the main results of our experiment do not have to rely on this particular grammar. The approach should apply in general to many different specifications of the theory.

We now start on our particular model. To compute the relationships between the parameter values on the one hand and the variations in word order and morphology on the other, we must specify at least the following.

(i) The categorial system of the model. This provides the building blocks of linguistic structures.

(ii) The feature system of the model. Since both the S(F)- and S(M)- parameters are associated with features, we will not know how many S-parameters are needed unless we know what features can be spelled out and what features need checking.

(iii) The computational system of the model. This includes the following sub-systems:

- Lexical Projection (LP) which determines the phrasal projections of all categories.
- Generalized Transformation (GT) which determines how the phrasal projections are joined to form a single tree.
- Move-α which is responsible for feature-checking.

(iv) The PF constraints.

(v) The LF constraints.

The PF and LF constraints can be easily defined in this model. We will therefore specify (iv) and (v) first.

There is only one PF constraint in this model which requires that the input to PF be a single tree. Presumably, the violation of this constraint might result in "broken" sentences. This does not mean, however, that we are not allowed to produce sentence fragments. A single NP or PP can also constitute a single tree and thus be a legitimate object at PF. In MPLT there is another constraint

which rejects strong features that survive to PF, as I have mentioned in 2.1.2. Since we have chosen not to resort to the strong/weak distinction as a possible explanation for overt movement, this constraint does not exist in the present model.

The only LF requirement in this model is that all the features must be checked. Since checking requires movement, it actually requires a set of movements to take place in the derivation. The nature of these movements will become clear in 3.2. In what follows, we will look at these systems one by one.

3.1 The Categorial and Feature Systems

The categorial system and the feature system will be discussed in the same section, because they are closely related. Every feature is associated with one or more categories and every category is basically a bundle of features.

3.1.1 Categories

The categories to be used in this mini-grammar will be limited to the ones in (33).

(33) { C(omp), Agr(eement)1, Agr(eement)2, T(ense), A(spect), V(erb), N }

Agr1 and Agr2 is equivalent to what we usually call AgrS and AgrO. We prefer not to use AgrS and AgrO because AgrS is not always associated with the subject, nor is AgrO always associated with the object. The use of Agr1 and Agr2 will facilitate our discussion on ergative constructions, passive constructions, unaccusatives, etc.

We see that all categories except N are verbal in nature while some nominal categories like D(eterminer) are missing. This is because for the time being we will not look into the internal structures of NP or DP, all of which will be treated as a single unit. For instance, *John*, *the boy* and *the boy who smiled* will be treated identically as NPs. This is why the traditional term NP is used instead of the more up-to-date DP in referring to such phrases. Other common categories that are absent in the list include P(reposition), Adv(erb), Adj(ective) and Neg(ation). Some of them will be introduced into our system in succeeding chapters when they become relevant to our discussion.

We assume that the set of categories in (33) is universal. In other words, the categories are given as part of UG. The fact that some categories do not seem to show up in some languages can be explained in two different ways:

(i) Only a subset of those categories is used in each particular language. After the critical period of language acquisition, the categories that are not used are "trashed".

(ii) All the categories are present not only in UG but in every individual adult grammar as well. The fact that some categories are invisible simply means they are not spelled out.

The explanation to be accepted in our present model is the one in (ii). There are several arguments against (i). First of all, the assumption that some categories can be trashed after the critical period implies that different languages can have very different X-bar structures. If we assume (ii), however, the structures will be more uniform. Secondly, the "trashing" of categories is not a simple computational operation. It may mean a partial or total rehash of selectional relations among categories. Suppose that in UG C selects Agr as a complement, Agr selects T, and T selects Asp. If a language does not have overt tense and agreement, we have to remove all those selectional rules and replace them with a new rule which may let C select Asp as its complement. How this complicated operation can be triggered and accomplished is a question. Finally, even in languages where certain categories seem to be missing, the concepts represented by those categories appear to be present. Many people have analyzed Mandarin Chinese as a language where the category T is missing (e.g. Cheng 1991). This by no means indicates that speakers of this language are tense-insensitive. As a matter of fact, every Chinese sentence is interpreted in some tense frame. This is true even in cases where no time adverbial is present. The simplest explanation for this fact is that T is present though it is not overtly realized.

3.1.2 Features

The arguments we made above about the universal nature of categories can be applied to features in a similar way. The features to be assumed in this model are also supposed to be universal. They are given in UG and they remain in the grammar of every individual language. The fact that only a subset of those features is visible in a given language means that only this subset is spelled out. Therefore, we can assume the existence of a feature as long as this feature is visible in *some* languages. We have hypothesized that the visibility of a feature depends on the value of its S(F)-parameter rather than the availability of this feature itself. Some people may argue that the distinction we are making here has no empirical import. After all, what is the difference between invisible existence and non-existence? But there *is* a difference in terms of language acquisition and language processing. As we will see, both of them can be simplified with our assumptions.

Now let us specify the features of our model. It is commonly accepted that the basic features are of two types: the V-features and the NP-features. What these features should exactly be is an open question, but we can start with the tentative set in (34).

(34) V-features: θ-grid, case, tense, aspect, ϕ-features, predication features
NP-features: θ-role, case, ϕ-features, operator features

This set is by no means complete but it will be enough for experimental purposes. In what follows, I will give some justification for the inclusion of those features in our model, and specify for each feature whether there is a S(F)-parameter associated with it. The set of S(F)-parameters to be assumed will again be minimal in nature. We are not trying to attach an S(F)-parameter to every

feature whose spell-out can vary cross-linguistically. Instead, only a subset of such features will have their spell-out options parameterized.

The existence of θ-grids is relatively non-controversial. Whether this feature can be spelled out, however, is open to debate. On the one hand, we can say that it is always overtly realized in the argument structure of a sentence; on the other hand, there does not seem to be a language where the θ-grids are morphologically realized on the verb. But one thing is almost certain: the spell-out of θ-grids does not vary from language to language. We can think of this features as being either always spelled out (in the argument structure) or never spelled out (in verbal morphology). Therefore there is no reason to assume an S(F)-parameter for this feature.

It is also well accepted that NPs carry θ-roles, though their realization is mixed up with that of case features. There are suggestions (ref ??) that θ-roles and cases are two sides of the same coin, morphological case being the overt realization of θ-roles. If we accept this view, we could eliminate the case feature and regard case-markers as spelled out θ-roles. However, we will stay with the standard view that θ-roles and cases are two different animals. The arguments for this assumption are so familiar that I will not review them here (See ??). One implication of this assumption is that θ-roles are never morphologically realized. In other words, they are understood but never spelled out. Consequently, there is no reason to suppose that they have an S(F)-parameter associated with them.

The ϕ-feature is used as a cover term for all agreement features, such as Person, Number and Gender. It is both a V-feature and an NP-feature. As far as overt agreement is concerned, however, the Spell-Out of V-features is more important than that of NP-features. A language is considered to have overt agreement as long as the V-features are overt, regardless of the status of the NP-features[3]. In this experimental model, we will only be interested in those features which are involved in overt agreement. For this reason, the spell-out of NP ϕ-features will be ignored for the time being. When we say a ϕ-feature or an agreement feature is spelled out, we mean the V-feature is overt. Another fact we will temporarily ignore is that different ϕ-features can be spelled out independently. We could let each ϕ-feature be associated with an independent S(F)-parameter. This is justified because each feature can be overt or covert regardless of the status of other ϕ-features. For instance, we can find a language where person and number features are spelled out on the verb but the gender feature is not. However, such details do not affect the general approach we are taking. They can be easily added to the system after the big picture has been made clear. To simplify our parameter space so as to concentrate on the more interesting aspects of the theory, we will assume a single S(F)-parameter for the whole set of ϕ-features. Its value is 1 (spelled out) if any subset of the ϕ-features is overt.

The existence of the tense and aspect features is again non-controversial. They are overt in some languages and covert in others. We could put these two features in a single bundle and let

[3]Functionally speaking, case-marking and agreement perform the same role, i.e. identifying the grammatical functions of NPs. This function is realized on the NP when case is spelled out and realized on the verb when agreement is spelled out.

them be associated with a single S(F)-parameter, as we have done to the ϕ-features. However, the differentiation of these two features is more important than that of ϕ-features in the present model because we are focusing on the verbal system. How tense and aspect features can be realized on their own is of interest to us. We have decided in ?? that T(ense) and Asp(ect) constitute two different categories, each having its own features. Therefore we will let tense and aspect be associated two independent S(F)-parameters. This will enable us to have a more detailed analysis of the tense/aspect system.

The case feature is usually regarded as an NP-feature, for it is often overtly realized as case-markers on NPs. There is no doubt that there should be at least one S(F)-parameter for the case features. Potentially we can associate a parameter with each different case, but we will assume a simpler system where the spell-out of all case features is determined by a single S(F)-parameter. This parameter will have the value "1" in a language if there is any kind of morphological case.

It is not so obvious, however, whether case is also a V-feature. Most people would think, at least initially, that verbs do not have case features. But there is evidence that the verb does carry case features and these features are sometimes visible. One example is found in Tagalog (Schachter 1976). In this language, the verb can have a case marker and the case feature varies according to which NP in the sentence is being topicalized. Consider the sentences in (35), (36), (37) and (38). (The topic marker is *ang* while *ng* marks agent and patient, *sa* marks locative and *para sa* beneficiary.) We could treat those markers as spelled out theta-roles, but we will stick with our assumption that theta-roles are never spelled out. Whatever is spelled out is always the case feature.

(35) *Mag-salis ang babae ng bigas sa sako para sa bata*
A-will:take woman rice sack child
'The woman will take rice out of a/the sack for a/the child.'

(36) *Aalisin ng babae ang bigas sa sako para sa bata*
O-will:take woman rice sack child
'A/The woman will take the rice out of a/the sack for a/the child.'

(37) *Aalisan ng babae ng bigas ang sako para sa bata*
Loc-will:take woman rice sack child
'A/The woman will take some rice out of the sack for a/the child.'

(38) *Ipag-salis ng babae ng bigas sa sako ang bata*
A-will:take woman rice sack child
'A/The woman will take some rice out of a/the sack for the child.'

These examples suggest that the case feature exist in both nouns and verbs.

Another possible example of verbal case features is cliticization. We can think of cliticization as a process where case features are spelled out on the verb. This view has been expressed by Borer (1984) who calls this process *Clitic Spell-Out*. It is also reminiscent of the treatment of subject

clitics in Safir (1985). Clitics can be viewed as something between a pronoun and an affix. In fact, they are more like affixes than pronouns. If we are allowed to treat them as affixes, as in the lexical analyses of clitics, [4] they will start to look like case and agreement markers affixed to the verb. In the following French sentence, for example, *me* and *la* can be viewed as the overt realization of case and agreement features on the verb, the former being the feature matrix of [case:dat, person:1, number:s] and the latter [case:acc, person:3,number:s, gender:f].

(39) *Jean me-la-montre*
John me-it-shows
'John shows me it.'

Since languages can differ with respect to whether the verbal case feature is spelled out, we assume that this feature has an S(F)-parameter associated with it.

The feature "predication" is supposed to contain information about sentence type. It tells us, for instance, whether the verb (or the "predicate") is used in a statement or a question ([-Q] or [+Q]). Is this feature visible in some languages? The answer seems to be positive. One way in which this feature can be said to be realized is through intonation. It is very common for statements and questions to have different intonational contours. The fact the verb seems to be the main bearer of the clausal intonation suggests that there is a verbal feature which can be overtly realized. This feature also seems to show up morphologically sometimes. The English word *whether* can be regarded as a spell-out of [+Q] in an embedded CP. In Chinese, this feature can be morphologically realized as a question/affirmation particle, as shown in (13) repeated here as (40), or in the A-not-A construction, as in (41).

(40) *Ta kan-wan nei-ben shu le ma*
you finish reading that book Asp Q/A
'Have you finished reading that book? / He has finished reading that book, as you know.'

(41) *Ni he-bu-he pijiu*
you drink-not-drink beer
'Do you drink beer?

The verbal complex *he-bu-he* ('drink or not') in (41) can be viewed as an instance where the [+Q] feature is spelled out on the verb. It seems that this feature must be spelled out in Chinese either in a verbal form or as a grammatical particle.[5] We will assume that there is an S(F)-parameter associated with this predication feature, as languages can vary as to whether this feature is morphologically realized.

[4] Lexical analyses claim that a clitic is in effect a derivational affix modifying the lexical entry of a predicate. For instance, the alternation between *lire un livre* and *le lire* is taken to be an alternation between a transitive verb *lire* and an intransitive *le+lire*.

[5] The A-not-A construction never co-occurs with the question particle *ma*, which suggests that "double spell-out" is prohibited in Chinese.

The last feature we will discuss is "operator". This feature is used as a cover term for such features as "scope", "topic" and "focus". The status of these features is open to discussion. We will assume that quantifier-raising (QR), topicalization and focalization involve similar syntactic operations, i.e. putting a constituent in a prominent position. This is the view expressed by Chomsky: "The natural assumption is that C may have an operator feature (which we can take to be the Q or *wh-* feature standardly assumed in C in such cases), and that this feature is a morphological property of such operators as *wh-*. For appropriate C, the operators raise for feature checking to the checking domain of C: [SPEC,C], or adjunction to specifier (absorption), thereby satisfying their scopal properties. Topicalization and focus could be treated the same way." (MPLT p45) However, there seems to be evidence that these operations are syntactically distinct. In Hungarian, for instance, a quantified NP, a topic and a focus can apparently co-occur in a single sentence. Consider (42).

(42) *Mari* *mindenkinek* *Petit* *mutatta* *be*
Mary-Nom everyone-Dat Pete-Acc showed in
'Mary introduced Pete to everyone.'

In this sentence, *Mari* is the topic, *mindenkinek* the raised quantified NP, and *Petit* the focus. All three of them are raised to the beginning of the sentence and they must appear in the order of *Topic* < *QP* < *Focus*. To account for these facts, we will assume that there are distinct operator features but they are checked through the same syntactic operation, namely, by raising a constituent to the Spec of CP through A-bar movement. The fact that more than one constituent can be raised in this way simply means that there is more than one operator. The multiple A-bar movements that seem to be involved may be handled in a way analogous to the treatment of multiple wh-movement. We can have a layered CP where the specifier position of each layer contains one operator. The different layers might be named Topic-P, Focus-P, etc. We can also let the operators adjoin to CP one after another. We can even put an ordered list of operators in Spec of CP. For the purpose of our preliminary experiments, however, there is no need to commit ourselves to any of those options. Being minimal again, we will currently limit ourselves to those cases where only one operator feature is checked. This may be the scope feature, the topic feature, or the focus feature.

The next question is whether the operator feature is ever morphologically realized. The answer seems to be "yes". The wh-scope marker *was* in German, for instance, can be taken as an overt scope feature, while the topic marker *-wa* in Japanese can be considered an overt topic feature. A German example is given in (43) (same as (28)) and a Japanese example is given in (44).

(43) was_i *glaubst* *du* $[_{cp}$ was_i *Hans* *meint* $[_{cp}$ [*mit wem* $]_i$ *Jakob* t_i *gesprochen* *hat*$]]$
WHAT believe you WHAT Hans think with whom Jakob talked has

(44) *Taroo-wa* *sensei* *da*
Taroo-Topic teacher is
'Taroo is a teacher.'

We will associate an S(F)-parameter to the operator feature to account for the fact it is overt in some languages but not in others.

In sum, we will have six S(F)-parameters which are associated with the following features: case, agreement, tense, aspect, operator and predication. They will be called S(F(case)), S(F(agr)), S(F(tns)), S(F(asp)), S(F(op)) and S(F(pred)) respectively.

3.1.3 Features and Categories

So far we have been talking about V-features and NP-features as if only verbs and nouns had features. This is not true, of course. Every category, whether lexical or functional, has a set of features. Moreover, many features are found in more than one category. Typically, a feature appears in two different places, one in a lexical projection and one in a functional projection. The tense feature, for instance, exists in both T and V. Let us call features in functional projections *F-features* and those in lexical projections *L-features*. Whenever a feature resides in both a functional projection and a lexical projection, feature-checking is required. To make sure that the F-feature and the L-feature agree in their values, the lexical element bearing the L-feature must move to the position where the F-feature is located. The only features that seem to have L-features only are the θ-features. The θ-grid is found in the verb only and the θ-roles are found in NPs only. All other features are generated in more than one position. The following table lists the features and the categories that contain them.

(45)

	functional	lexical
θ-grid		V
θ-role		N
tense:	T	V
aspect:	A	V
ϕ(1):	Agr1	V, NP1
ϕ(2):	Agr2	V, NP2
case(1):	Agr1	V, NP1
case(2):	Agr2	V, NP2
predication:	C	V
operator:	C	XP

There are two sets of ϕ-features. ϕ(1) consists of the features involved in subject-verb agreement. ϕ(2) is related to object-verb agreement. NP1 and NP2 usually correspond to Subject and Object, but not always so. Technically, NP1 is just the higher NP and NP2 the lower one. Parallel to the ϕ-features, there are also two sets of case features. Case(1) is the case assigned/checked in Agr1, and case(2) the one in Agr2. The XP with which the operator feature is associated can be any full projection, such as an NP or AdvP.

Conceptually, we can think of the F-features as representing the information the speaker intends to convey and the L-features as the features to be physically realized. To ensure that "we say what we mean", so to speak, the lexical features must be checked against the functional features.[6] As we will see, the dual presence of F-features and L-features can account for many interesting linguistic phenomena.

Each feature has a set of values. For instance, the value of the tense feature can be instantiated to present, past, future, etc. However, we are not interested in those specific values in this abstract model. What we will be focusing on is the spell-out those features: whether they are morphologically realized in a given language, whatever their values may be.

3.1.4 The Spell-Out of Features

In 3.1.2, we have assumed six S(F)-parameters. Here is review of those parameters and the features they are associated with.

(46)

S(F)-Parameter	*Feature*
S(F(case))	case
S(F(agr))	agreement
S(F(tns))	tense
S(F(asp))	aspect
S(F(op))	operator
S(F(pred))	predication

When the S(F)-parameter of a given feature is set to 1, this feature will be spelled out in overt morphology. What is actually spelled out, however, can vary in different cases. First of all, there is a distinction between pre-Spell-Out checking and post-Spell-Out checking. When a feature is checked before Spell-Out, the lexical element carrying the feature will have moved to the functional position where this feature is checked. As a result, the L-feature and F-feature will get unified before Spell-Out and become indistinguishable from each other. We may say that the F-feature disappears after checking and what is available for spell-out is the L-feature only. Thus the feature is always spelled out on a lexical head which shows up inflected. By "inflected" we mean the lexical element has an affix, a special tone, or any other forms of conjugation. In cases where the feature is checked after Spell-Out, the L-feature and the F-feature will co-exist in the representation that is fed into PF. Consequently, both of them will be available for potential phonological realization. Logically speaking, then, there are four possibilities for the spell-out of features, as we have mentioned in the previous chapter:

[6] The fact that the θ-features need not be checked this way does not mean that they are not checked. They do get checked but the checking takes place in the process of lexical projection.

(47) **(i)** Only the F-feature is spelled out;

(ii) Only the L-feature is spelled out;

(iii) Both the F-feature and the L-feature are spelled out;

(iv) Neither the F-feature nor the L-feature is spelled out.

The possibilities in (i) and (iii) are available only if the feature is checed after Spell-Out.

All the four possibilities can be illustrated by examples from real languages. Let us take the tense feature as an example. In English, this feature must be spelled out and what is overtly realized can be either the F-feature or L-feature. In (48) the F-feature is spelled out. The verb does not move to TP and the head of TP is overtly realized by itself as *will.* In (49) the L-feature is spelled out as a suffix to the verb.

(48) *John will visit Paris.*

(49) *John visit-ed Paris.*

It seems that double spell-out, i.e. spelling out both the F-feature and the L-feature, is prohibited in English, for (50) is ungrammatical.

(50) **John did visit-ed Paris.*

In Swedish, however, both the F-feature and L-feature can be spelled out, as we have seen in (24) repeated here as (51).

(51) *Oeppnade doerren gjorde han*
open-Past door-the do-Past he
'He opened the door.'

In fact, the sentence will be unacceptable if the L-feature is not spelled out. In (52), the infinitive form of the verb *oeppna* is used instead of the past tense form *oeppnade*, and the sentence is out. [7]

(52) **Oeppna doerren gjorde han*
open (inf) door-the do-Past he
'He opened the door.'

On the other hand, there are also languages where neither the L-feature nor the F-feature are spelled out. Chinese offers examples of such null spell-out:

(53) *Ta meitian qu xuexiao*
he/she everyday go school
'He/she goes to school everyday.'

[7] This sentence and the judgment on it is from Platzack.

(54) *Ta* *mingtian* *qu* *xuexiao*
he/she tomorrow go school
'He/she will go to school tomorrow.'

(53) is in the present tense while (54) is in the future tense, but there is no overt tense marking at all.[8]

We have thus seen that all the four logical possibilities in (47) are empirically attested. This suggests that the S(F)-parameter can have two sub-parameters, one for the spell-out of L-features and one for the F-features. We will represent these two sub-parameters by splitting the value of each S(F)-parameter in the form X-Y. The value of X determines whether the F-feature is spelled out and Y the L-feature. Then the four possibilities in (47) will correspond to the following parameter values.

(57)	*Features Spelled Out*	*Values of S(F)-parameter*
(i)	F-feature only	1-0
(ii)	L-feature only	0-1
(iii)	both F and L feautres	1-1
(iv)	neither F nor L features	0-0

The values in (i) and (iii) are possible only if feature-checking takes place after Spell-Out because only in these cases will both the L-feature and F-feature be available at the time of Spell-Out. The significance of these sub-values will be further discussed in Chapter 4 where more examples will be given to illustrate those possibilities.

3.2 The Computational System

There three major operations in the computational system: lexical projection (LP), generalized transformation (GT) and move-α. We will specify them one by one.

[8]There are certain things in Chinese that are arguably tense markers. For example, *jiang* and *hui* can be treated as future tense markers, and the perfective marker *le* can be treated as a past tense marker (Chiu 1992), as we can see in (55) and (56).

(55) *Ta* *mingtian* *jiang* *qu* *xuexiao*
he/she tomorrow JIANG go school
'He/she will go to school tomorrow.'

(56) *Ta* *zuotian* *qu* *le* *xuexiao*
he/she tomorrow go LE school
'He/she went to school yesterday.'

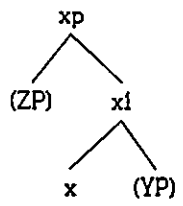
But even if these are tense markers, it still remains true that at least in some sentences the tense feature is not overt.

3.2.1 Lexical Projection

3.2.1.1. Basic Operation

In our system every category X projects the tree in (58).

(58)



This is different from the lexical projection described in MPLT. We assume that the projections are invariable, with every projection resulting in an XP (i.e. X2). In MPLT, however, the projection only goes "as far as it needs to" and what is projected can be an X0, an X1 or an XP, as has been illustrated in (3) in Chapter 2. In addition, the specifier and complement positions do not appear in the initial projection in MPLT. They are added later in the GT process. Consequently, the distinction between substitution and adjunction is in fact gone. In our system, however, this distinction is maintained. Generally speaking, any position which is *obligatory* is to be generated in the process of lexical projection. Attachment or movement to these positions is therefore substitution. All positions that are *optional* is not base-generated. They are to be added to the structure in the process of GT or move-α.

Now let us take a closer look at (58) which will be called an *elementary X-bar tree*. The fact that ZP and YP appear in upper case letters indicate that they are empty when the tree is projected. They contain a set of features but no lexical content. In the process of syntactic derivation, they will either be filled by a subtree or be licensed to remain empty. In the former case, they serve as *attachment points* for GT or move-α operations. The parentheses around ZP and YP indicate their optionality: not every projection contains them. The actual status of ZP and YP is determined by X. ZP can appear as a specifier of X if and only if X selects a ZP specifier. This selectional rule can be represented as (59).

(59) *specifier(X,Z)*

Similarly, YP can appear as a complement of X if and only if X selects YP as a complement. This can be represented as (60).

(60) *complement(X,Y)*

The specifier can be absent if the rule in (61) exists and the complement can be absent if (62) exists.

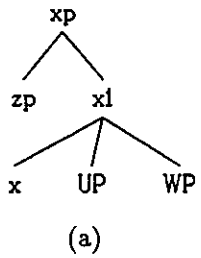
(61) *specifier(X,ϵ)*

(62) *complement(X,ϵ)*

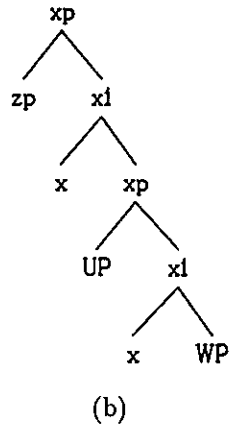
The rules in (59) and (60) do not tell us the directionality of the specifier or complement. In any actual implementation of these rules, however, the direction has to be specific. To generate the tree in (58), for example, we must make clear that the specifier is to occur to the left of the head and the complement to the right of the head. We have decided in 2.2.4 that we will adopt a weaker version of the Invariant X-bar Structure Hypothesis where there are only two HD-parameters: a head-complement parameter for CP and another one for IP. The specifiers always occur to the left of their heads. The complements always occur to the right of their heads except those in functional categories. The positions of heads in functional categories are determined by two HD-parameters: HD1 and HD2. The value of HD1 determines the position of head in CP and the value of HD2 determines the head-position in IP. HD1 and HD2 each have two values: I (head-initial) and F (head-final).

It is assumed here that each category can take at most one specifier and one complement. As a result, the tree to be generated will never be more than binary branching. The assumption that each category can take no more than one specifier (Spec) is well accepted. (Ref??) The single-complement assumption, however, may seem to be unsupported at first sight. There are obviously structures where two or more complements are found. The double-object construction is an example of this. However, the fact that some categories need more than one complement does not mean that any single elementary X-bar tree has to contain more than one complement position. Adopting the layered-VP hypothesis of Larson (1988), we can let a category project more than one elementary X-bar tree, with each tree taking only one complement. For instance, instead of (63(a)), we can have the tree in (63(b)) where both UP and WP are generated without sacrificing binary branching.

(63)



(a)



(b)

The structure in (63(b)) will be discussed further when we come to the projection of VP.

3.2.1.2. Selectional Rules (Simplified)

Now we define the selectional rules for each category. We will tentatively assume that UG contains the rules in (64).

(64)

specifier(c,x)	(i)
specifier(agr1,n)	(ii)
specifier(t,ϵ)	(iii)
specifier(asp,ϵ)	(iv)
specifier(agr2,n)	(v)
specifier(v,n)	(vi)
complement(c,agr1)	(vii)
complement(agr1,t)	(viii)
complement(t,asp)	(ix)
complement(asp,agr2)	(x)
complement(agr2,v)	(xi)
complement(v,v)	(xii)
complement(v,ϵ)	(xiii)

Some notes on these rules are in order here.

There are no HD-parameters associated with the specifier rules: the specifier selected by the head always occurs to the left of the head. The complement rules which are sensitive to the values of HD-parameters are (vii), (viii), (ix), (x) and (xi). The head in (vii) precedes the complement when HD1 is set to "I" and follows the head when HD1 is set to "F". The heads in (viii), (ix), (x) and (xi) precede their complements when HD2 is set to "I" and follow their heads when HD2 is set to "F". The fact that the values of HD2 applies in all the three rules reflects what we have assumed in 2.2.4: IP is treated as a whole regardless of how many independent projection it contains. In other words, Agr1-P, TP, AspP and Agr2-P are to be treated as segments of a single IP. Therefore they share the value of a single HD parameter. The application of (xii) is not subject to the value of any HD-parameter. A verb always precedes its complement.

The Spec of CP (Cspec hereafter) can be any maximal projection. We use "x" to represent this. The Specs of Agr1, Agr2, and V are all NPs in these rules. (From now on, we will refer to these positions as Agr1spec, Agr2spec, and Vspec respectively.) This is again a simplification. In a more complete system, CPs or IPs should also be able to appear in those Spec positions.

We notice that T and Asp do not have a Spec position. This does not mean that there is any principled reason against these categories having a specifier. There have been many syntactic arguments which rely crucially on the presense of this position. It is not present in this grammar simply because we are trying to keep the system as small as possible.

The rules in (64) are applicable to both transitive and intransitive sentences. In other words, both Agr1-P and Agr2-P will be projected no mather whether there is an object or not. When a sentence is intransitive, one of the agreement phrases may be inactive. I will basically follow Bobaljik (1992), Chomsky (1992), and Laka (1992) in assuming that there is an *Obligatory Case Parameter* whose value determines which case-assigner (Agr1 or Agr2) is active in an intransitive sentence. According to this assumption, we get a nominative/accusative construction if Agr1 is active and an ergative/absolutive construction if Agr2 is active. There is evidence that both Agr1 and Agr2 exist in a single intransitive sentence. In some languages case marking identifies the patient of a transitive verb with the intransitive subject, while agreement identifies the agent of a transitive verb with the intransitive subject. It seems that, in these situations, the case system is ergative while the agreement system is nominative. We have to conclude then that both Agr1 and Agr2 can be partially active in an intransitive sentence.

3.2.1.3. Selectional Rules (Featurized)

So far the selectional rules and the projection trees have been presented in a simplified form with a lot of information left out. The nodes in (58) contain nothing but the category label, and the selectional rules in (64) only tell us the categorial status of a specifier or complement. It is obvious that the nodes do not consist of category labels only. Each node is a set of features, the category label being just one of them. Take the verb *catches* as an example. It has at least the syntactic features in (65).

(65)

Category:	v
θ-Grid:	[agent, patient]
Tense:	present
ϕ-features:	[person:3,numer:s]

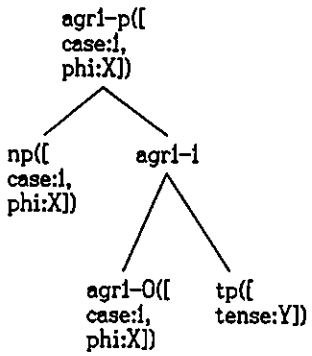
The specifier or complement selected by the category is also a bundle of features. For instance, Agr1spec may have the following features:

(66)

Category:	n
Case:	1
ϕ-features:	X

The value of the ϕ-features, represented here as a variable, is selected by the head of Agr1. This selectional relation can be seen in (67) which is the maximal projection of Agr1.

(67)



The variable "X" is found in both the specifier and the head. This indicates that the two nodes must have unifiable ϕ features.[9] This is the way Spec-head agreement is achieved. The actual value of this feature is not an intrinsic property of Agr1. They depend on (a) the verb that moves to Agr1-0 and (b) the NP that moves to Agr1spec. What Agr1 plays is a mediating role. It ensures that, whatever the value may be, it must be shared by the specifier and the head.

The structure in (67) also tells us that the NP specifier must have Case 1, i.e. the case assigned by Agr1. The NP to be attached or moved to this position must have the same case. In this way the case-marking of nouns will get checked. Notice that the case feature also appears in the Agr1-0 and the Agr1-P node. This means that case is an intrinsic feature of Agr. The specifier of Agr gets the value of its case feature via Spec-head agreement. The presence of the case feature in Agr also explains why case is a V-feature as well. A verb acquires the case feature or have a case feature checked when it moves to Agr-0 through head movement.

The complement in this projection tree is a TP whose tense feature has a variable "Y" as its value. The value will be instantiated when a TP is attached to this position.

To generate the tree in (67), we need two"featurized" selectional rules, such as the ones in (68).

(68) *specifier(agr1(case:1,phi:X), n(case:1,phi:X),l)*

complement(agr1,t,r)

Obviously, all the rules in (64) need to be featurized this way. But there is one more point to be elaborated on before we do this. I have mentioned earlier that, in order to preserve binary branching, we will adopt the "Layered-VP" hypothesis of Larson (1988). The VP structure assumed in this experimental grammar is a pseudo-Larsonian one which in a way carries Larson's idea to the extreme. In addition to the general layered-VP structure, we also assume the following:

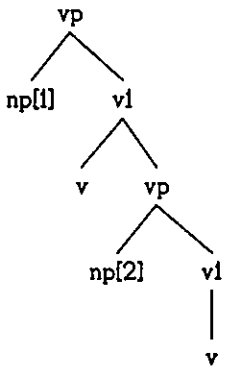
[9] "Unifiable" is used here in the standard sense of unification. (Ref??) Intuitively, two sets of features are unifiable if they do not have incompatible values. For instance, the feature matrices [person:X,number:p,gender:m] and [person:3,number:p, gender:Y] are unifiable (X and Y are variables meaning "any value"), while [person:2,number:p,gender:f] and [person:3,number:p,gender:f] are not unifiable. The values of person clashes with each other.

(i) The number of VP layers corresponds to the number of arguments a verb takes or the number of θ-roles it assigns. In other words, each layer of VP will contain exactly one argument.

(ii) The argument in a given VP layer appears uniformly in Vspec.

The assumption in (i) in fact follows from (ii). Each layer of VP can have only one specifier and therefore we need as many layers as the number of arguments. The VP tree for a transitive verb will look like the following:

(69)



These assumptions have the following consequences.

First, the distinction between internal and external arguments are eliminated. What remains is just a thematic hierarchy. An argument can just be relatively "higher" or "lower" than some others. What is traditionally called the "subject" is simply the highest argument in the VP-shell. The Extended Projection Principle is now translated into the requirement that every sentence must contain at least one argument. There is no longer the need to explain, for example, why an NP with a "Theme" role can be both an internal and external argument. The kind of argument promotion observed in (70a), (70b) and (70c) now receives a natural account.

(70) (a) The man opened the door with the key.

(b) The key opened the door.

(c) The door opened.

Passive and unaccusative constructions are also accounted for. The passive verb has had its first θ-role "absorbed". (Ref ??). Therefore the VP projected from a passive verb will not have the layer which is the top one in its active counterpart. The remaining arguments are thus promoted one layer up. The argument which is originally in the second layer is now in the first one and treated like a subject. The unaccusative verb has only one θ-role to assign, so only one layer of VP is projected. Since this single layer is also the top layer, the argument of an unaccusative verb can enjoy subjecthood.

Secondly, the VP structure assumed here lets θ-role assignment be performed uniformly in the Spec-Head configuration. This is conceptually appealing because θ-role assignment, case-checking and agreement-checking now involve the same type of operation, namely Spec-head agreement. We thus have a more general and more consistent notion of the Spec position being the checking domain of each category.

The structure in (69) consists of two elementary X-bar trees but they are in fact projected from a single head. The verb exists as a chain, each V0 being one of its links. The two links are identical except for the number of a θ-roles they contain. The higher one has two while the lower one has only one. This does not mean that two different verbs are involved here. The difference is used as a computational device which makes sure that the correct number of layers are projected. We have seen in (64) that there are two complement rules for V: it can take either a VP complement or no complement. The choice is determined by the θ-feature. If the θ-grid contains only one θ-role, this V will have no complement and the current VP will be the last layer. If the θ-grid contains $n+1$ (for any $n > 0$) θ-roles, this V will take a VP complement. The θ-grid of this VP complement will contain n θ roles, with the understanding that the other role has been assigned in the higher layer. In each layer, the θ-role being assigned is always the first one in the list. This role is removed from the list after the assignment so that the next role can be assigned in the next layer. The layer-by-layer stripping of θ-roles also ensures that eventually there will be only one role left so that the VP projection will terminate. In the case of (69), the verb has two θ-roles to assign. No more VP complement is permitted after the second layer because there is no more theta-role to assign.

Now we are ready for a "featurized" version of (64). The new lexical projection rules are given in (71). The structure "F:V" means that the feature F has V as its value.

(71)

specifier(c,n(op:+))	(i)
specifier(agr1(case:1,phi:X),n(case:1,phi:X))	(ii)
specifier(t,ϵ)	(iii)
specifier(a,ϵ)	(iv)
specifier(agr2(case:2,phi:X),n(case:2,phi:X))	(v)
specifier(v(θ-Grid:[Th,...],n(θ-role:Th)	(vi)
complement(c,agr1)	(vii)
complement(agr1,t)	(viii)
complement(t,asp)	(ix)
complement(asp,agr2)	(x)
complement(agr2,v)	(xi)
complement(v(θ-Grid:[Th1,Th2,...]),v(θ-Grid:[Th2,...])	(xii)
complement(v(θ-Grid:[Th]),ϵ)	(xiii)

The θ-grids in these rules contain variables like "Th1", "Th2", etc. instead of names like "agent" and "patient".[10] This is done for the sake of generality. The notation means that, given any two θ-roles, "Th1" is higher than "Th2" in the thematic hierarchy and is to be assigned in a higher layer of VP.

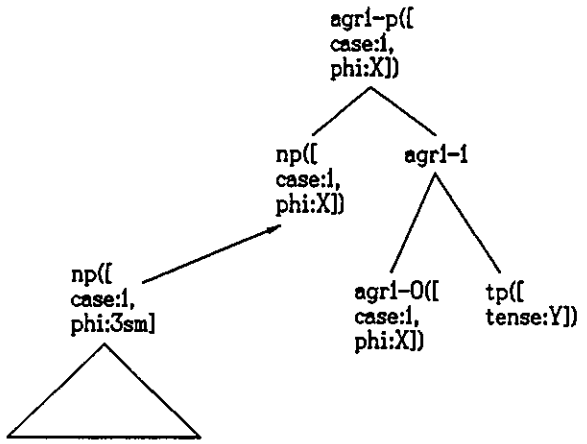
The op(erator) feature in Cspec has the value "+". This means that the NP or any XP to be substituted into this position is going to be the operator, i.e. it will receive the wide-scope, topic, or focus interpretation.

3.2.2 Generalized Transformation

The LP operation described in the previous section produces a set of elementary X-bar trees. The function of Generalized Transformation(GT) is to put those trees into a single tree. In MPLT, there is only one type of GT operation which subsumes both substitution and adjunction. In both cases, we add an empty position to the target tree (which looks like adjunction) and then substitute a subtree into this position. This will not be the version of GT to be assumed in the present model. As I have stated earlier, we will maintain the distinction between substitution and adjunction, the former associated with obligatory constituents and the latter with optional ones.

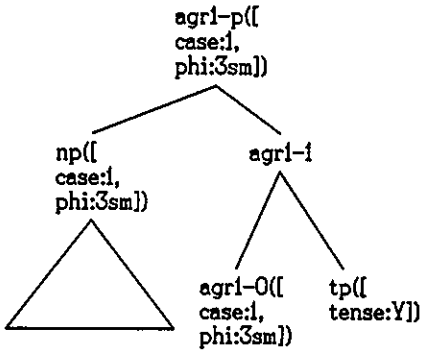
In GT substitution, a subtree K is substituted into an empty position ø in the target tree K', resulting in K*. The empty position ø in K' is either a specifier position or a complement position which has already been generated in the process of Lexical Projection. The position is empty in the sense that it has features but no lexical content. It is an attachment site into which another tree may be substituted. The substitution is possible only if the attachment site and the subtree to be attached have compatible (i.e. unifiable) features. For instance, the subtree to be attached to the Agr1spec in (67) must be an NP whose case feature has the value 1 and whose ϕ-feature has the value X. If X has already been instantiated to [person:3,numer:s] in Agr1, only a third person singular NP can be attached to this point. If the X in Agr1 is instantiated to [person:3,number:N] where N is a variable, however, either a singular or a plural third person NP can be substituted. (72) and (73) illustrate the two basic cases of GT substitution. In (72a), an NP is being substituted into Agr1spec. The tree that results is (72b). Notice the unification of feature values in the substitution process. ("3sm" is a short-hand form of [person:3,number:s,gener:m].) In (73a), a TP is being substituted into the complement position of Agr1-P. The result is (73b).

[10] The "..." in the θgrids represents the rest of the theta-roles which can be empty.

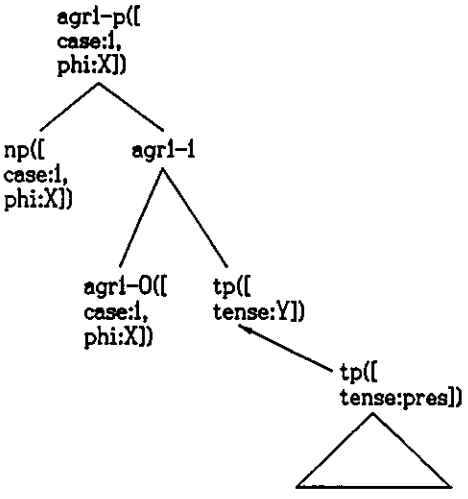


(72)

(a)

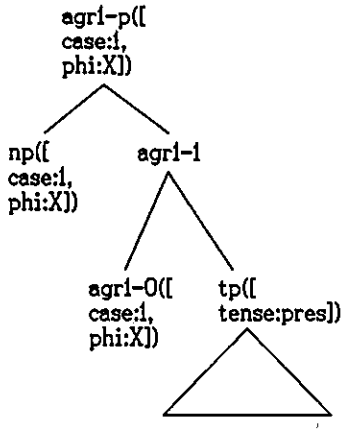


(b)



(73)

(a)

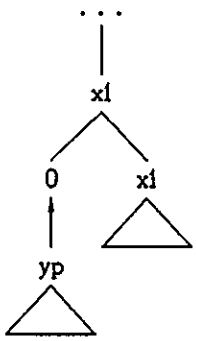


(b)

In GT adjunction, a subtree is added or inserted into a constituent. In this experimental model, we will only be concerned with adjunction to X1. In other words, we will only consider the adjunctions whose function is adding modifiers into the structure. The subtree to be adjoined to an X1 must be a maximal projection. In this GT process we create an additional segment of X1 which

contains an empty position ø and then attach a subtree to ø . In (74), an extra segment of X1 is created and AP is substituted into the empty position contained in this extra X1.

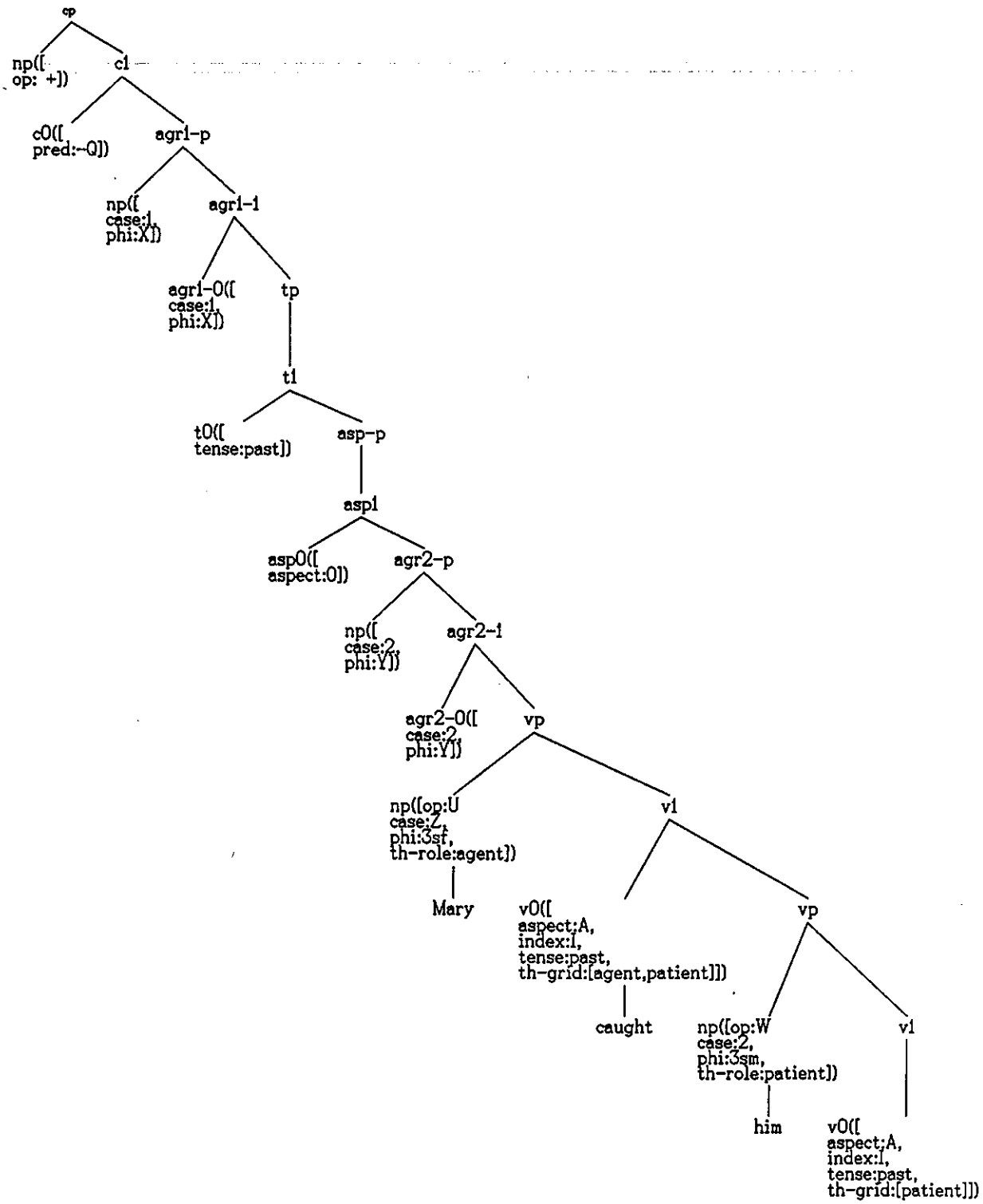
(74)



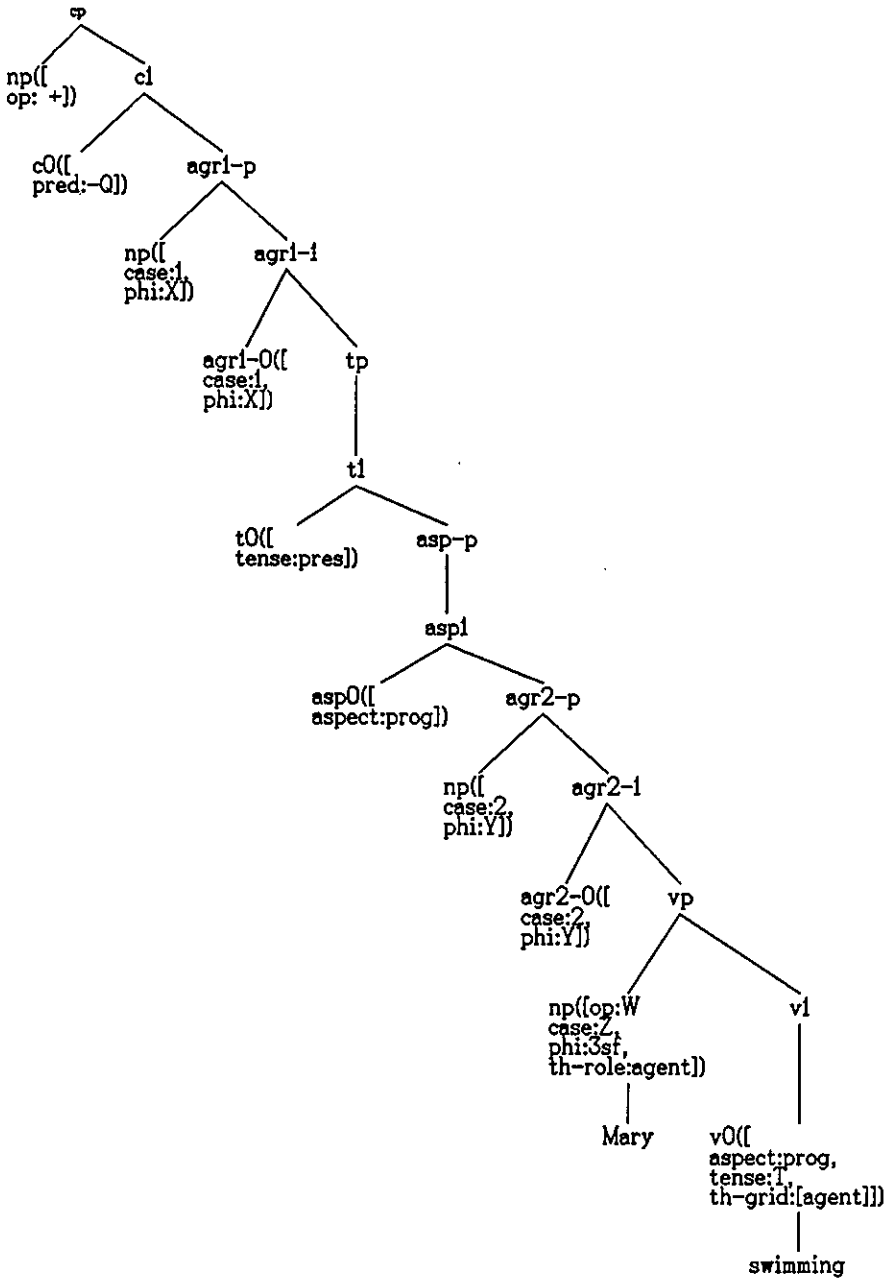
We assume that all adjunctions are left-adjunctions. We also assume that the attachment point created during the adjunction has certain selectional properties, so that each category will only accept a certain class of modifiers. For instance, the adjunction site in a V1 may require the adjunct to be an AdvP. Therefore we will not be able to adjoin an NP to a V1. If an adjunct is acceptable to two or more X1's, it can then choose to adjoin to any of them. I will not try to specify a full theory of modifier adjunction here. Some further discussion on this will be given when the need arises.

GT operations are applied recursively on pairs of trees until there is only a single tree left. If there are two or more subtrees left and no GT operation can apply to reduce them to a single tree, the derivation crashes.

At this point, we might be interested to see what structures are produced by LP and GT in our system. Given the rules in (71), we can get 4 different CP structures for each type of verb by varying the values of HD1 and HD2. For illustration, we will look at one of the 4 structures where both HD1 and HD2 are set to "I". We will demonstrate it with two types of verbs: a transitive verb that takes two NP arguments and an intransitive verb that takes one argument. The former will be illustrated by the English sentence *Mary caught him* and the latter by *Mary is swimming.* The structures generated by LP and GT for these two sentences are given in (75) and (76) respectively. (All the nodes in these trees should have features other than the category label, but to save space the other features are omitted in all but the terminal nodes.)


cp
np([op: +])
c1
c0([pred:-Q])
agr1-p
np([case:1, phi:X])
agr1-1
agr1-0([case:1, phi:X])
tp
t1
t0([tense:past])
asp-p
asp1
asp0([aspect:0])
agr2-p
np([case:2, phi:Y])
agr2-1
agr2-0([case:2, phi:Y])
vp
np([op:U case:Z, phi:3sf, th-role:agent])
v1
Mary
v0([aspect:A, index:I, tense:past, th-grid:[agent,patient]])
vp
caught
np([op:W case:2, phi:3sm, th-role:patient])
v1
him
v0([aspect:A, index:I, tense:past, th-grid:[patient]])

(75)



(76)

There are some specific points about these trees which are worth mentioning.

Firstly, We see that all the lexical items appear VP-internally. Each of the NPs is in a position to which a θ-role is assigned, but none of them , however, is in a position where its case can be checked. This is different from the traditional view that internal arguments are assigned cases VP-internally

under government. In our system, there is no internal arguments and government does not play a role in case-checking at all. Every NP is drawn from the lexicon together with its case-feature, but it cannot be checked VP-internally. To satisfy the checking requirement, it must move to the Spec position of one of the agreement phrases. This kind of movement will be discussed in 3.2.3.

Secondly, the copula *is* in *Mary is swimming* does not appear in the tree. This follows from our assumption that *is* is an expletive which is not base-generated. It is inserted in the Spell-Out process as a way of overtly representing the features in Agr1-0 and T0.

Finally, we find in those trees all the features we have assumed. The values of these features are constants in some cases and variables in others. (All the uppercase letters stand for variables.) The variables all represent unspecified values, but they can have different syntactic status depending on whether the feature is an F-feature (feature in a functional category) or an L-feature (feature in a lexical categories). The variables in functional projections are all used for agreement. Two nodes are supposed to have the same value for a certain feature if the same variable appears in both. For instance, the values of ϕ-features in both Agr1-P and Agr2-P are variables. The fact that ϕ has X or Y as its value in both the head and the Spec of AgrP ensures that the subject/object and the verb will agree in their ϕ-features. The values of these features will be instantiated when the VP-internal NPs move to the Specs of AgrPs and the verb moves to the heads of AgrPs.

The variables in the lexical projections indicate that the features in question are morphologically unspecified. In other words, there are no morphemes in the lexicon that represent the values of those features. In (75), for examples, the NP *Mary* is morphologically unspecified for the case feature and the verb *caught* is unspecified for the aspect feature. The features will get instantiated when movement takes place for feature-checking. In (76), *swimming* is specified for the aspect feature which is morphologically realized as the suffix *-ing*. But it is not morphologically specified for the tense feature. Hence the variable for the tense feature. The values of operator features in the two NPs (*Mary* and *him*) are also variables. When the sentence is used in a real situation, however, one of them can get the "+" value and only this NP can eventually move to Cspec.

3.2.3 Move-α

In our present system, movement takes place for no other reason than feature-checking. Following Chomsky's Principle of Greed (MPLT) which requires that no movement take place unless it is the only way to save a derivation from crashing, we will assume that a movement occurs if and only if there is an LF checking requirement whose satisfaction depends soly on this movement. We should be reminded at this point that the movements we are discussing here are LF movements which are universal. They take place in every language by LF, though only a subset of them may be visible in a particular language.

3.2.3.1. Movement as a Feature-Checking Mechanism

The necessity of movement in feature-checking can be viewed from two different perspectives. From the point of lexical items, we see that a given word may have two or more features, each of which must be checked in a different structural position. Take NP as an example. UG requires that it be assigned a θ-role and have its case checked. However, θ-roles are assigned in Vspecs only and cases are checked in Agrspecs only. To meet both requirements, an NP must move from one position to the other, which forms a chain linking the two positions. Once this occurs, the NP exists as a chain rather than a single node. It enters a structural relation whenever one of its links is in the required position for that relation. From the viewpoint of features, we see that most features are found in more than one node. In (75) and (76), for instance, the tense feature appears in both TP and VP. To make sure that a given feature has the same value throughout the whole structure, we have to form chains to link nodes which are related by feature-checking movements but are not in the same projection. All the chains in our system are formed in this way.

Since movement occurs for feature-checking only, we can find out all the movements by locating all the features whose checking requires movement. As we have seen, only those features which appear in more than one projection need to be checked through movement. Furthermore, in all the cases where a feature appears in two different projections, one of them is in a lexical projection and the other in a functional projection. This is clear in (45), (75) and (76). To see what movements are required, we only have to list all the features that are both L-features and F-features. According to (45), they include the following: tense, aspect, $\phi(1)$, $\phi(2)$, case(1), case(2), predication and operator.

The tense feature is found in both T and V. In order for the feature in V to be checked against the feature in T, the verb must move to T. The aspect feature is found in both Asp and V. Therefore, the verb must move to Asp for feature-checking. The predication feature is found in both C and V. Forced by the feature-checking requirement, the verb must move to C. The operator feature is found in both Cspec and NPs. For feature-checking, one of the NPs must move to Cspec. Since the value of the operator feature is always "+" in Cspec, only the NP which is the operator can move there. The case feature is found in both Agrspecs and NPs. Therefore, each NP must move to some Agrspec to have its case feature checked. NP1 must move to Agr1spec and NP2 to the Agr2spec. We assume that, when both NP1 and NP2 are present in the VP projection, NP1 cannot move to Agr2spec, nor can NP2 move to Agr1spec. There are various ways to account for this restriction. In MPLT, this restriction is supposed to be derived from the notion of *equidistance*. I will not go into the the mechanisms that implement this notion. At an intuitive level, we can view the restriction as a special way of observing the Principle of Economy which requires, among other things, that short moves be preferred over long moves. If we move NP1 to Agr1spec and NP2 to Agr2spec, both movements will be relatively short. If we move NP1 to Agr2spec and NP2 to Agr1spec, however, the NP1-to-Agr2spec movement will be very short but the NP2-to-Agr1spec movement will be longer than any of the two movements in the previous case. This economy-based argument is not well-

understood yet and it will not be incorporated into our model. We can get the same effect from some simpler principles. In our model we assume that the case hierarchy and the thematic hierarchy in a sentence must agree with each other. Given two NPs, NP_n and NP_{n+1}, and two θ-roles θ_n and θ_{n+1} with θ_n preceding θ_{n+1} in the θ-grid of the verb, NP_n must have its case checked in a higher case position if NP_n is assigned θ_n and NP_{n+1} assigned θ_{n+1}. (A case position (Agrspec) is higher than another one if the former asymmetrically C-command the latter.) Intuitively, this assumption simply means that the subject must be assigned the subject case and the object the object case. In passive constructions, the first θ-role in the θ-grid is suppressed and the one that follows it will become the the first. As a result, the NP assigned this promoted θ-role is free to move to the highest case position. In unaccusative constructions, the "subject" θ-role is missing from the θ-grid. Consequently, some other role will be the first in the grid and the NP assigned this role can go to the highest case position.

The ϕ-features are similar to the case features in that they are both NP-features and V-features. In terms of the NPs, the ϕ-features are found in both Agrspecs and the NPs. During feature-checking, NP1 must move to Agr1spec and NP2 must move to Agr2spec. The movement patterns are identical to those involved in case-checking. In terms of the verb, the ϕ-features are found in Agr1-0, Agr2-0 and V0. The verb therefore must move to Agr1-0 and Agr2-0 to have the features checked. During the movement, the verb will also pick up the case features in Agr1 and Agr2. This implies that case and agreement are two sides of the same coin. They have the common function of identifying grammatical relations.

To sum up, we list in (77) all the movements forced by feature-checking.

(77) **A.** The verb must move to Agr2-0 to have its ϕ & case features checked for object-verb agreement.

B. After moving to Agr2-0, the verb must move to Asp0 to have its aspect feature checked.

C. After moving to Asp0, the verb must move to T0 to have its tense feature checked.

D. After moving to T0, the verb must move to Agr1-0 to have its ϕ & case features checked for subject-verb agreement.

E. After moving to Agr1-0, the verb must move to C0 to have its predication features checked.

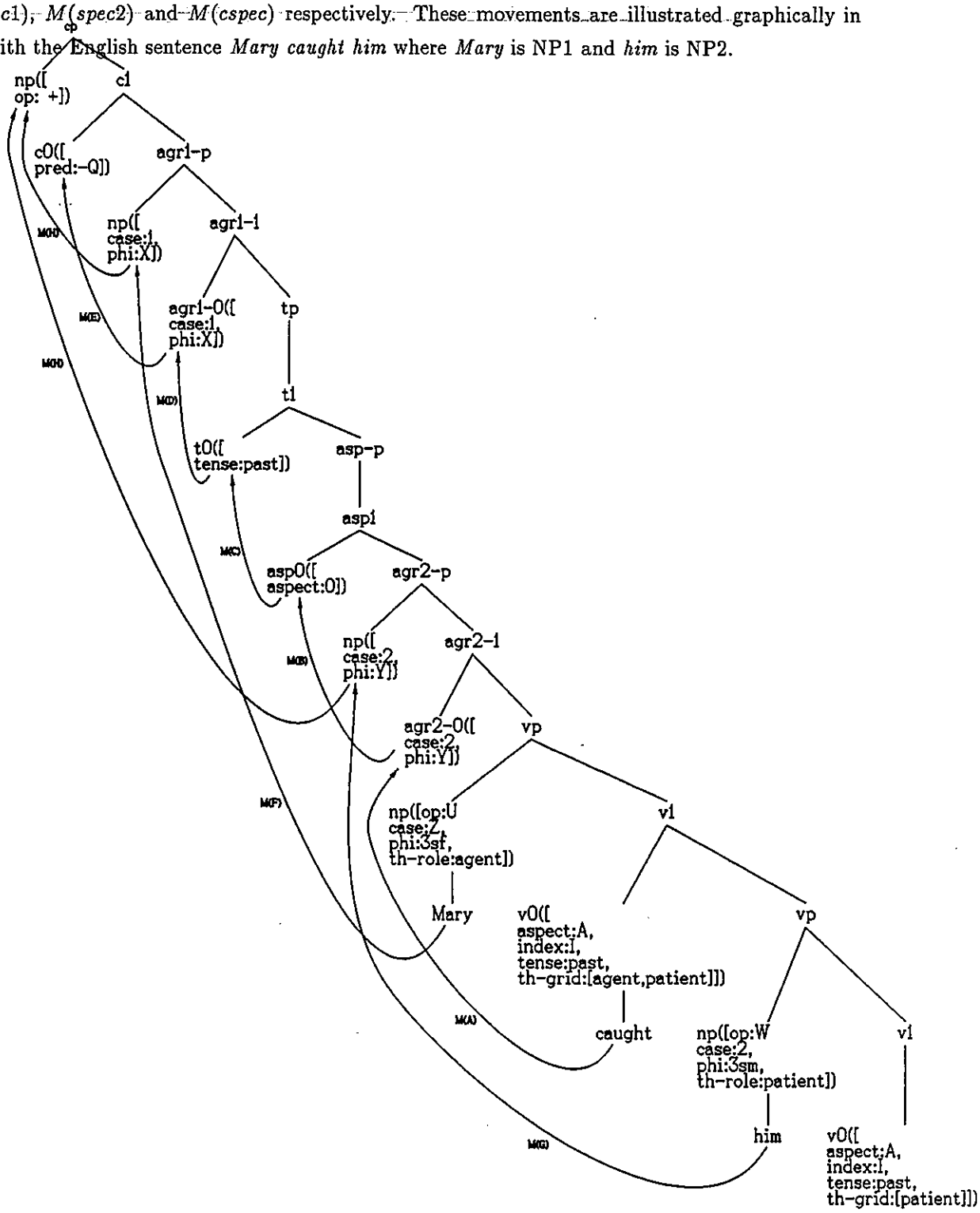
F. NP1 must move to Agr1spec to have its ϕ & case features checked.

G. NP2 must move to Agr2spec to have its ϕ & case features checked.

H. After moving to an Agrspec, one of the NPs must move to Cspec to have its operator features checked.[11]

[11] This implies that every sentence has an underlying topic or focus or an NP that receives a wide-scope interpretation.

From now on, we will refer to these movements as $M(agr2)$, $M(asp)$, $M(tns)$, $M(agr1)$, $M(c)$, $M(spec1)$, $M(spec2)$ and $M(cspec)$ respectively. These movements are illustrated graphically in (78) with the English sentence *Mary caught him* where *Mary* is NP1 and *him* is NP2.



(78)

We can see that $M(agr2)$, $M(asp)$, $M(tns)$, $M(agr1)$ and $M(c)$ are head movements, $M(spec1)$ and $M(spec2)$ are A-movements, and $M(cspec)$ is Ā-movement. There are two instances of $M(cspec)$ in the diagram. One involves NP1 moving to Cspec while the other involves NP2. In a particular sentence only one of the movements can occur. Which one occurs depends on which of the NPs is the topic or focus of the sentence.

3.2.3.2. Movement in Operation

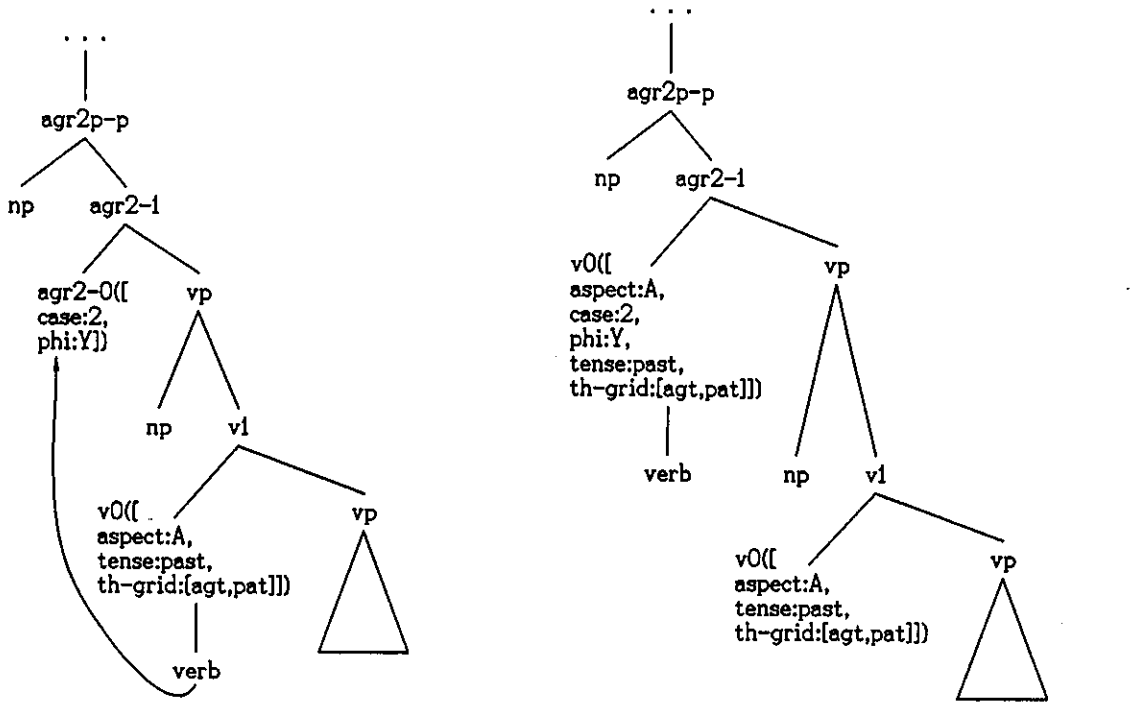
Having identified the set of movements involved in feature-checking, we will now take a closer look at the computational operation involved in these movements. It has been assumed that all the movements are raising movements in our system. Lowering is prohibited. Therefore, it is illegal to have any "yoyo" type of movement where a constituent moves up and then down or down and then up. Other operational constraints on movement are discussed below.

Movement as Substitution All the movements discussed here are substitution operations. The landing site of every movement is an existing attachment point which is an empty node created in lexical projection. The substitution is possible only if the moved element and the landing site have identical categories and compatible feature values. It is not possible, for example, to move an X0 to an XP or vice versa. Nor is possible to move an NP to a position where a different value is required for the case or ϕ feature. This guarantees that all the movements are *structure-preserving*.

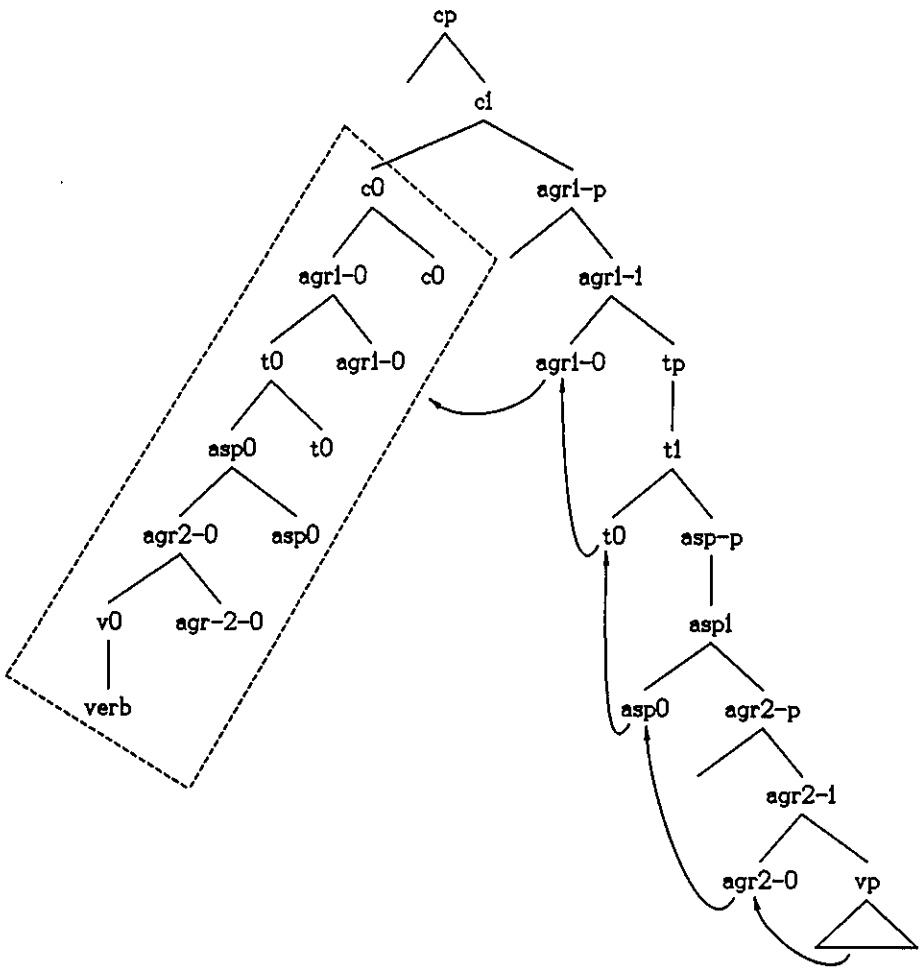
The substitution operation is obvious in the cases of A-movement and Ā-movement. The landing sites of these movements are all Spec positions projected in LP: Agr1spec in the case of $M(spec1)$, Agr2spec in the case of $M(spec2)$, and Cspec in the case of $M(cspec)$. In cases of head movement, however, this is less obvious. At first sight, substitution seems to be impossible. How can a V0 substitute for a T0 or C0, for instance? For a node to serve as the landing site of a movement, it must be (a) empty, and (b) have feature values which are unifiable with those of the moved element. The condition in (a) seems to hold. The landing sites of V-movement are all heads of functional categories which are feature matrices without lexical content. The condition in (b), however, looks a little problematic. For one thing, the landing site and the moved element do not seem to have the same categorial features. We seem to be substituting a V for a T, an Agr, a C, etc., which should be impossible. This is why head movements have been standardly treated as adjunction rather than substitution operations. But a second thought on the status of C, Agr, T and Asp suggests that the substitution story is plausible. These categories are after all extended V-projections. Since none of these functional categories has a lexical head, all of them can be said to have been projected ultimately from the verb. In other words, they are just some additional layers of the V-projection. Viewed in this way, C, Agr, T, Asp and V all belong to the same category and there should be no reason why substitution is impossible. We therefore assume in this experimental grammar that head movement involves substitution instead of adjunction. When a verb is substituted into the head of a functional category, the two heads will merge into one. We will call this new head "V", with the

understanding that all the features of the original functional head have been preserved. We choose to call it "V" rather than "T" or "Asp" because the features of this new head are spelled on the verb in the form of verbal inflection. The diagram in (79) illustrates the substitution involved in a head movement where a verb moves to the head of Agr2-0. (79a) shows the pre-movement structure and the movement which is taking place. (79b) shows the post-movement structure.

(79)



The kind of head movement assumed here fails to make some of the predictions that are made by the standard version of head movement. In head movement by adjunction, the moving head gets attached to the target head either from the left or from the right, so the head and the affix will appear in a certain linear order. In (80), for instance, the verb has moved to C0 through Agr2-0, Asp0, T0 and Agr1-0.



(80)

The successive adjunction results in a big verbal complex which is boxed in the diagram. Suppose that in this language agreement features and tense features are morphologically realized as suffixes. Then the structure of this verbal complex predicts that the inflected verb will be spelled out as V-T(ense)-Agr(eement) rather than V-Agr(eement)-T(ense). This prediction is based on the Mirror Principle (Baker 1985) which requires that morphological derivations reflect syntactic derivations (and vice versa). In the substitution story of head movement, this prediction is gone. The movement just results in a complex feature structure where no linear order is implied. This result can be good or bad depending on whether the Mirror Principle is really valid. If it is, our version of head movement will be less desirable because it has missed an important generalization. However, counter-examples to the Mirror Principle do exist. In terms of the T-suffix and the Agr-suffix, both orders seem to be possible. In Italian (81) and Chichewa (82), for example, we find T inside Agr while in Berber (83)

and Arabic (84), we find Agr inside T.

(81) *legge-va-no*
read-imp(Asp/Tns)-3ps(Agr)
'They read'

(82) *Mtsuko u-na-gw-a*
waterpot SP(Agr)-past(Tns)-fall-Asp
'The waterpot fell'

(83) *ad-y-segh Moha ijn teddart*
fut(Tns)-3ms(Agr)-buy Moha one house
'Moha will buy a house.'

(84) *sa-ya-shtarii Zayd-un dar-an*
fut(Tns)-3ms(Agr)-buy Zayd-Nom house-Acc
'Zayd will buy a house.'

In order to preserve the Mirror Principle, some people (e.g. Ouhala 1991) have proposed that the hierarchical order of AgrP and TP be parameterized, i.e. in some language AgrP dominates TP while in other languages TP dominates AgrP. But the price we pay here to save the Mirror Principle seems too expensive. In our system, such reshuffling in the tree structure is not necessary. What syntax provides for each node is a feature matrix. The linear order in which the features are spelled out can be treated as an independent matter which probably falls in the domain of morphological theory. Different languages may simply choose to spell out the features in different orders. In acquisition the linear order can be learned in the same way that other ordering rules in morphology are learned.

Some Barriers to Movement I have mentioned earlier in this chapter that the bounding theory may need some revision in the Minimalist framework. In the standard model, there is the distinction between SS movement and LF movement. It is assumed that some of the barriers which constrain SS movements do not apply to LF movement. Now that all movements are LF movements, it is no longer clear what the barriers are. Fortunately, this status of affair does not seem to affect our experimental model very much, since we are currently only concerned with simple sentences. Some barriers do exist within a single clause, but we can for the time being describe them in a case-by-case manner without attempting a general account. In what follows, we will look at head movement, A-movement and $\bar{A}$-movement one by one and discuss the constraints on each of them.

For head movement, we will assume the Head Movement Constraint (HMC) which requires that no intermediate head be skipped during the movement. Given three heads, H_1, H_2 and H_3, where H_1 asymmetrically C-commands H_2 and H_2 asymmetrically C-commands H_3, no X0 can move from H_3 to H_1 without moving to H_2 first. For a verb to move from its VP-internal position to C0, for example, it must move successively to Agr2-0, Asp0, T0 and Agr1-0 first.

For A-movement, there will be no clause-internal barriers. We usually assume that A-movement has to be local. According to Sportiche (1990), for instance, A-movement has to go in a Spec-to-Spec fashion. A movement is blocked whenever it has to go through a Spec position which is already filled by some other XP or one of the links of an XP chain. Obviously there would be problems if the locality constraint were imposed on the A-movements in our present system. For NP1 to move to Agr1spec, it would have to go through Agr2spec, but this is impossible.[12] A similar problem exists for NP2 which would have to go through NP1 to reach Agr2spec. To account for the fact that $M(spec1)$ and $M(spec2)$ are possible, we will assume that the domain of XP movement can be extended by head movement. As a result, all the projections that a single head has moved through will be transparent to each other. In our system, the verb moves all the way to C through Agr2, Asp, T and Agr1. So the whole CP tree is transparent for XP movement. Within this single CP tree an NP can move to any Spec position without crossing any barriers.

This extended domain for XP-movement also applies to Ā-movement. As a result, any NP within a single CP can move to Cspec without crossing any barriers. But there is an independent constraint which prevents an NP from moving from its VP-internal position directly to Cspec. It is required in our grammar that every NP move to a Agrspec to have its case & agreement features checked. If an NP moves directly to Cspec, skipping all Specs of AgrPs, the the case & agreement features will fail to be checked. Once in Cspec, an NP will not be able to move to an Agrspec any more, since lowering is prohibited. Consequently, an LF constraint is violated and the derivation will crash. To avoid the crash, an NP must move to a position to have its case & agreement features checked before moving to Cspec. In other words, NP1 must move to Agr1spec first and NP2 to Agr2spec first.

If we go back to (78) now, we will realize that all the constraints discussed above are observed there. In fact, the movements illustrated there represent not only all the possible movements in our system but also all the possible *paths* for these movements. In particular, each movement has a unique path and results in a unique chain.

Before we close this section, I will mention an apparent problem related to head movement. We have assumed that the verb always moves all the way up to C0. Superficially, however, there seem to be many cases where the verb only moves half-way up and what moves to Agr1 or C is an auxiliary. It looks as if the checking movement were broken up into two parts, one performed by verb movement and one by auxiliary movement, resulting in two separate chains. I will argue that, even in these cases, what moves to C0 at LF is still the verb and there is only a single chain. After Spell-Out the verb will move further up to the positions which the auxiliaries seem to have moved through. The movement is not blocked because auxiliaries are invisible at LF and their features are incorporated into the verb. Why the movement seems to be split at Spell-Out will be explained in Chapter 4. We will see that there are particular settings of S(M)-parameters which are responsible

[12] This is impossible because (i) the NP moving to Agr1spec must have Case 1 and will not be able to unify with Agr2spec which has Case 2, and (ii) Agr2spec belongs to the chain headed by NP2 and therefore it is already filled and should serve as a barrier for the movement of NP1.

for this superficial phenomenon.

3.2.3.3. The S(M)-Parameters

In 3.2.3.1, we identified a set of 8 movements: $M(agr2)$, $M(asp)$, $M(tns)$, $M(agr1)$, $M(c)$, $M(spec1)$, $M(spec2)$ and $M(cspec)$. We assume that each of these movements can occur either before or after Spell-Out. In other words, each of them has an S(M)-parameter associated with it. We will call those parameters $S(M(agr2))$, $S(M(asp))$, $S(M(tns))$, $S(M(agr1))$, $S(M(c))$, $S(M(spec1))$, $S(M(spec2))$ and $S(M(cspec))$ respectively. When $S(M(X))$ is set to 1, $M(X)$ will be overt. It is covert when $S(M(X))$ is set to 0.

Now the question is whether an S(M)-parameters can have a third value, namely 1/0, which is a variable. What this says is that the relevant movement can be either overt or covert, hence the optionality of the movement. Our immediate reaction to this idea might be negative. According to the Principle of Economy in general and the Principle of Procrastinate in particular, no movement should be optional. If a movement can be either overt or covert, it should always be covert. In addition, there are both acquisitional and processing arguments against optional movement. Optional rules are more difficult to acquire. They also make the parsing process less deterministic. However, there is empirical evidence which shows that the "no optionality" assumption is too strong. It runs into difficulty whenever a language has alternative word orders. If we insist on the binarity of S(M)-parameters values, any given movement will be either always overt or always covert. As a result, only a single word order will be permitted in any language. The fact most languages do have alternative word orders shows that the binarity is too restrictive. We can of course say that any given language has a canonical word order. This order is determined by the obligatory movements and all the optional movements are "stylistic" or "discourse-driven". But this leads to the assumption that there are two independent sets of movements: one syntactic and one stylistic. This assumption is not totally implausible, yet the necessity of identifying a different set of movements in addition to the checking movements we now have makes the theory more complicated. We will have a simpler theory if we assume that there is only a single set of movements and the "syntactic" and "stylistic" movements are overt manifestations of the same set of movements. In this way, we will not need to define a separate set of movements in addition to the movements we have defined here. All the "stylistic" movements correspond to movements whose S(M)-parameters are set to 1/0. This value is a variable which can be instantiated to either 1 or 0. As far as the S(M)-parameter values are concerned, therefore, both overt and covert movements are allowed. In stylistically neutral or unmarked cases, the Principle of Economy will dictate that the variable be instantiated to 0. As a result, the movements are invisible and the "canonical" order surfaces. In contexts where other factors call for overt movement, the Principle of Economy may be overridden. Consequently, the variable will be instantiated to 1 and the movement is visible. In short, when an S(M)-parameter is set to 1/0, the movement with which the parameter is associated will be covert unless there are some

stylistic or discourse factors calling for overt movement. So the movement is not really optional. Once we have a stylistic or discourse theory which defines precisely when overt movement is needed, the choice will be clear. In any given context, the variable can only be instantiated to a single value. However, the model we are describing here is a purely syntactic one which does not include a stylistic or discourse module. This other module is absolutely necessary, but it falls outside the scope of the present study. The issues involved there need to be addressed in a separate project. What we can do in syntax is providing all the options. The choice will be made when the syntactic module is interfaced with other modules. For this reason, we will allow some movements to be optionally overt in our grammar. In particular, we will let the three S(M)-parameters associated with XP/NP movement – S(M(spec1)), S(M(spec2)) and S(M(cspec)) – have three values: 1, 0 and 1/0. This by no means implies that head movement cannot be optional. We have simply chosen to experiment with optional movement on A-movement and Ā-movement first. There are two motivations for this choice. First, we want to try out *some* optional movements and find out their basic properties before generalizing optionality to all movements. Second, the main purpose of permitting optional movement in our grammar is to account for those scrambling facts which involve A-movement or Ā-movement. Optional head movement will be discussed briefly in this chapter but will be be put aside in the full discussion of parameter space.

To give the above argument more substance, we will look at two specific cases where the S(M)-parameters seem to be set to 1/0, one involving optional A-movement and one Ā-movement.

For optional A-movement we can find an example in English. In (85) and (86) (same as (25)), we see an alternation between overt and covert NP movement.

(85) *Three men came.*

(86) *There came three men.*

In (85), $M(spec1)$ (NP movement to Agr1spec) is overt. It is covert in (86). We thus conclude that $S(M(spec1))$ is set to 1/0 in English. This explains why both orders are possible. However, in a particular context only one of them will be appropriate. (86) seems to be the unmarked case where there is no reason for overt movement. In (85), however, the Principle of Economy has apparently been overridden by some discourse considerations.

An example of optional A-bar movement can also be found in English where topicalization produces a word order other than SVO.

(87) *John likes apples.*

(88) *Apples, John likes.*

In our system we assume that topicalization involves XP-movement to Cspec. Then it seems that *apples* has moved to Cspec in (88) but not in (87). We can conclude then that $M(cspec)$ is optional

in English and $S(M(cspec))$ is set to 1/0. In unmarked cases the movement does not occur overtly due to the Principle of Economy. When a constituent needs to be overtly topicalized, however, the Economy principle is overridden and the movement becomes visible.

Although we will put optional verb movement aside for the time being, we will assume that it is possible in principle. An example of this kind of optionality can be found in French. There we find the word order alternation between statements and questions, as shown in (89) and (90).

(89) *Nous allon à la bibliothèque*
we go to the library
'We are going to the library.'

(90) *Allez vous à la bibliothèque*
go you to the library
'Are you going to the library?'

(89) is a statement where the verb is presumably in Agr1-0 while (90) is a question where the verb is supposed to have moved to C0. It seems that the verb movement from Agr1-0 to C0 is optional in French, since both orders are possible. We can therefore assume that $S(M(c))$, the S(M)-parameter for verb movement to C0, is set to 1/0. This is why the verb can either precede or follow the subject. However, the movement is non-optional in any particular case. Let us suppose that the declarative sentence constitutes the unmarked case where there is no special motivation for Agr1-to-C movement. Thus the Principle of Economy will apply and the sentence will be ungrammatical if the movement is overt. In the case of interrogative sentences, there seems to be a special need for overt movement. We will not discuss what the need is here, but apparently it can override the Principle of Economy and require that the movement be overt. The Principle of Economy thus looks like a default principle. It applies only if no other principle is being applied.

In terms of the values of $S(M(c))$, French can be contrasted with V2 languages on the one hand and Chinese and Japanese on the other. In V2 languages, the Agr1-to-C movement seems to occur overtly regardless of whether the sentence is a statement or question. This shows that $S(M(c))$ is set to 1 rather than 1/0 in these languages. This is why the movement is always obligatory. In Chinese and Japanese, on the other hand, the Agr1-to-C movement is never visible. This suggests that $S(M(c))$ is set to 0 in these languages. In this case, the verb does not have the option to move to C even if this movement is motivated in some way.

We will see in Chapter 4 that the value 1/0 for S(M(spec1)), S(M(spec2)) and S(M(cspec)) can account for many interesting facts which would otherwise be left unexplained. The addition of this value will of course make the task of acquisition and parsing more challenging, but the challenge will give us a better understanding of the acquisitional and parsing processes.

3.3 Summary

In this chapter we have defined an experimental grammar upon which our study in syntactic typology, syntactic acquisition and syntactic processing in later chapters will be based. We defined a categorial system, a feature system and a computational system. The feature system includes a set of features and a set of S(F)-parameters which determine the morphological visibility of those features. The computational system is composed of three sub-components: lexical projection (LP), generalized transformation (GT), and move-α. For LP we defined a set of selectional rules which determine the specifier and complement each category takes and two HD-parameters which determine the position of heads in functional projections. No parameterization exists in GT which is performed in a universal way. For move-α we defined a set of feature-checking movements, each of which has a S(M)-parameter that determines the visibility of the movement. In the next chapter we will put this grammar to work. We will examine the parameter space created by the parameters and the language variations accommodated in the parameter space.

Chapter 4

The Parameter Space

In this chapter we consider the consequences of our experimental grammar in terms of the language typology it predicts. The parameters we have defined in the previous chapter can have many value combinations, each of which making the grammar generate a particular language.[1] Those different value combinations form our *parameter space* and the languages that are generated in this parameter space form a particular language typology. We will explore the parameter space and try to find out its main properties and the languages it accommodates.

We should be reminded here that the term "language" is used in a special sense here. In most cases we will be using the quoted form of this term to mean a set of strings which are composed of abstract symbols like S(ubject), O(bject) and V(erb). A string such as *S V O* represents a sentence where the subject precedes the verb and the object follows the verb. In addition, each symbol can carry a list of features. The features in this list represent overt morphology, i.e. the features that are spelled out. For instance, V-[agr,tns] represents a verb which is inflected for agreement and tense. A typical "language" in our system looks like (91) which tells us the following facts: (a) this "language" has an SOV word order; (b) the NPs in this "language" carry case markers; (c) the verbs in this "language" are inflected for agreement and tense; (d) this "language" has both transitive and intransitive sentences; and (e) this is a nominative-accusative language where the subject in an intransitive sentence has the same case-marking as the subject in a transitive sentence.

(91)
```
{   s-[c1]  v-[agr,tns],
    s-[c1]  v-[agr,tns]  o-[c2]
}
```

This set of strings may resemble some subset of a real language, but it is far from a perfect representation of any natural language. It is only an abstract representations of certain properties of a human language. The properties we are interested in are word order and inflectional morphology. When we say a set of strings corresponds to an existing language, we mean that it reflects the word

[1] The language generated can be empty, i.e. it contains no string.

order and morphology in this language. All the languages that are generated in our systems are such abstract languages. In spite of their abstractness, however, it will not be hard to see what languages they may represent. We will see in this chapter that many "languages" accommodated in our parameter space have real language counterparts and most real languages can find an abstract representation in our parameter space.

Let us start the exploration by reviewing the parameters we have assumed.

(i) S(M)-parameters. These parameters determine what movements are overt in a given language. There are eight S(M)-parameters corresponding to the eight movements assumed in our theory:

S(M(agr2)) [V-to-Agr2]	S(M(c)) [Agr1-to-C]
S(M(asp)) [Agr2-to-Asp]	S(M(spec1)) [NP1-to-Agr1spec]
S(M(tns)) [Asp-to-T]	S(M(spec2)) [NP2-to-Agr2spec]
S(M(agr1)) [T-to-Agr1]	S(M(cspec)) [XP-to-Cspec]

The movement in brackets is overt (before Spell-Out) if the corresponding S(M)-parameter is set to 1 and covert (after Spell-Out) if the parameter is set to 0. We have assumed in Chapter 3 that A and $\bar{A}$ movements (M(spec1), M(spec2) and M(cspec)) can be optional before Spell-Out. Therefore the value of S(M(spec1)), S(M(spec2)) or S(M(cspec)) can be a variable – 1/0 – which indicates that the associated movement can be either overt or covert.

(ii) S(F)-parameters. These parameters determine what morphological features are overt in a language. Six of them are assumed: S(F(agr)), S(F(case)), S(F(tns)), S(F(asp)), S(F(pred)) and S(F(op)). Each of these parameters can have four values: 0-1 (spell out the L-feature only), 1-0 (spell out the F-feature only), 1-1 (spell out both the L-feature and the F-feature), and 0-0 (spell out neither the L-feature nor the F-feature). Recall that most features in our system are base-generated in two positions, one in a lexical category (the L-feature) and one in a functional category (the F-feature). The two features are checked against each other via movement.

(iii) Two HD-parameters: HD1 which determines whether the head of CP precedes or follows its complement, and HD2 which determines whether the heads in IP precede or follow their complements. These two parameters can be set to either I (head-initial) or F (head-final). The value of HD2 applies to every segment of IP: Agr1P, TP, AspP and Agr2P.

Putting these parameters together, we have 8 binary-valued parameters, 3 triple-valued ones, and 6 quadruple-valued ones. They make up a parameter space where there are 14,155,776 (i.e. $2^7 \times 3^3 \times 4^6$) value combinations. Two questions arise immediately:

(92) Does every existing human language has a corresponding "language" in our parameter space?

(93) Does every "language" in our parameter corresponds to some natural language?

If the answers to these questions are all "yes", our system will be more than just plausible. Any ideal parameter space should those properties. As we will see in this chapter, the answers are basically positive.

In order to get the answers to these questions, we must first of all get all the value combinations, try to generate some language with each setting, and collect the languages that are generated together with their corresponding settings. This is a straight-forward computational task and it can be accomplished using the Prolog program in Appendix A.1. There is obviously an expository problem as to how the results of this experiment can be presented and analyzed, since simply listing all the settings and the languages they generate may take a million pages. In order to describe the whole parameter space in a single chapter, I will break down the parameter space into natural sub-spaces and look at them one at a time. This can be done by varying the values of certain parameters while keeping the others constant. Some properties of the parameter space are local in the sense that they are properties of a particular sub-space or a particular type of parameters. We can get a very clear idea about those properties by examining the relevant sub-spaces. In areas where different types of parameters interact, we will concentrate on some representative cases instead of exhaustively listing all the possibilities. Such sampling will hopefully enable us to envision the potential of the entire parameter space.

In what follows, we will look at the space of S(M)-parameters first and then expand it to include the HD-parameters. After that we will bring some S(F)-parameters into the picture and consider their interaction with S(M)-parameters. As we will see, the interaction provides us with an interesting account of auxiliaries. Other S(M)-parameters will eventually enter the scene. Many natural language examples will be cited in the course of discussion to illustrate the relevance of our experimental results to empirical linguistic data.

4.1 The Parameter Space of S(M)-parameters

In this section, we will single out the S(M)-parameters and explore the range of language variation they can account for. To do this we need to look at all the value combinations of S(M)-parameters while keeping the values of all other parameters constant. In the following experiment, the HD-parameters are always set to "I" (head-initial). In other words, we will be restricted to the tree structures in (76) and (75) (Chapter 3) where every head precedes its complement. We will not be concerned with morphology at this moment. The settings of S(F)-parameters will be temporarily ignored. The "words" that appear in strings will therefore be simplified as *s* (subject) *o* (object) and *v* (verb) which are to be interpreted as NPs and verbs with any inflectional morphology.

4.1.1 An Initial Typology

We have assumed eight S(M)-parameters and we will represent their values in a vector of eight coordinates:

[S(M(agr2)) S(M(asp)) S(M(tns)) S(M(agr1)) S(M(c)) S(M(spec1)) S(M(spec2)) S(M(cspec))]

A setting like [1 0 0 0 0 1 0 0] means that S(M(agr2)) is set to 1, S(M(asp)) set to 0, S(M(tns)) set to 0, and so on.

With 5 binary-valued parameters (S(M(agr2)), S(M(asp)), S(M(tns)),S(M(agr1)), S(M(c))) and 3 triple-valued ones (S(M(spec1)), S(M(spec2)), S(M(cspec))), we have a parameter space of 864 settings. However, not all those settings can result in a non-empty language. Some of the value combinations may make the grammar generate no strings. Exactly what settings produce empty languages depends on our syntactic assumptions. In our current sub-space where only the S(M)-parameters are active, a setting may generate an empty language because of the two syntactic constraints discussed in 3.2.3: the Head Movement Constraint (HMC) and the constraint that an NP must move to an Agrspec before moving on to Cspec.

The HMC requires that no intermediate head be skipped during head movement. For a verb to move from its VP-internal position to C0, for example, it must move successively to Agr2-0, Asp0, T0 and Agr1-0 first. In other words, verb-movement to C must consist of 5 short movements: M(agr2) (V-to-Agr2), M(asp) (Agr2-to-Asp), M(tns) (Asp-to-T), M(agr1) (T-to-Agr1) and M(c) (Agr1-to-C). The verb cannot be in C0 if any of those successive movements fails to occur. If the only X0 in a grammar that can undergo head movement is the verb, there will be a transitive implicational hierarchy in the form of M(agr2) < M(asp) < M(tns) < M(agr1) < M(c). No movement on the right-hand side of an "<" can occur without the one(s) on the left-hand side occurring at the same time. If any of the intermediate movements are blocked, the movement as a whole will be blocked. However, there do exist settings where some of the intermediate settings seem to be blocked. Consider the setting in (94).

(94) [0 0 0 0 1 . . .]

This setting requires that the verb move from Agr1-0 to C0 before Spell-Out. But the verb cannot be in Agr1-0 unless it has moved through Agr2-0, Asp0 and T0. Since S(M(agr2)), S(M(asp)), S(M(tns)) and S(M(agr1)) are all set to 0, the verb is not allowed to move to Agr1-0 before Spell-Out. Consequently, no verb can move from Agr1-0 to C0 and the required movement will fail to occur overtly. The fact that some movement is required to be overt but cannot occur overtly makes the derivation crash. Thus the language generated is empty. Other settings which can result in empty languages due the HMC include [0 0 0 1 0 . . .], [0 0 1 0 0 . . .], [0 1 0 0 0 . . .], [1 0 1 0 1 . . .], etc. We will see later on, when the values of S(F)-parameters are taken

order and overt morphology are assumed to be independent of each other in our model, there is no dependency between the values of S(M)/HD parameters and S(F)-parameters. In other words, the former and the latter can be set independently. We can therefore consider them separately. In what follows, we will look at the word order parameters first. We will start with S(M)-parameters, adding HD-parameters to the parameter space later on, and finally get to the setting of S(F)-parameters.

5.2 Setting S(M)-Parameters

In this section we consider the setting of S(M)-parameters. The values of other parameters will be held constant for the moment, with all HD-parameters set to "i" and S(F)-parameters set to 0-0. Since no feature is spelled out when all S(F)-parameters are ser to 0-0, the feature list will be temporarily omitted in the presentation of strings. The string [s v o], for example, is understood to be abbreviated form of [s-[] v-[] o-[]]. In addition, the symbol "v" will often be used to stand for both "iv" and "tv".

5.2.1 The Ordering Algorithm

As we have seen in 4.1.4, some languages in the parameter space of S(M)-parameters are properly included in some other languages. This implies that the learning algorithm we have assumed can fail to result in convergence for some languages if the enumeration of parameter settings is random. In order for every language in the parameter space to be learnable, the hypothetical settings must be enumerated in a certain order. In particular, the settings of subset languages must be tried *before* the settings of their respective superset languages. Let us call the parameter setting for a subset language a *subset setting* and the one for a superset language a *superset setting*. A superset setting must then be ordered *after* all its subset settings. To ensure learnability for every language, we can simply calculate all subset relations in the parameter space, find every superset setting and its subset settings, and enumerate the settings in such a way that all subset settings comes before their relative superset settings. Such an ordering is not hard to obtain. In fact, the enumeration can be made to satisfy this ordering condition in more than one way. However, arbitrary ordering of this kind is not linguistically interesting. It can certainly make our learning algorithm work, but we cannot expect a child to know the ordering unless it is built in as part of UG. We are thus in a dilemma: the learning may not succeed if there is no ordering of parameter values, but the assumption that the ordering is directly encoded in UG seems extravagant.

However, there is a way to get out of this dilemma. The child can be expected to know the ordering without it being directly encoded in UG if the following is true: the ordering can be *deduced* from some linguistic principle in UG. Such a principle does seem to exist in our current linguistic theory. One possible candidate is the Principle of Procrastinate (Chomsky 1992) which requires that movement in overt syntax be avoided as much as possible. This principle has the

following implications for the parameter setting problem considered in our model.

(165) All S(M)-parameters should be set to 0 at the initial stage. Let us suppose that the Principle of Procrastinate is operative in children's grammar from the very beginning. According to this principle, an "ideal" grammar should have no overt movement. Therefore, children will initially hypothesize that no movement takes place before Spell-Out in their language. They will consider overt movement (i.e. setting some S(M)-parameters to 1) only if they have encountered sentences which are not syntactically analyzable with the current setting.

(166) In cases where children are forced to change their hypothesis by allowing some movement(s) to occur before Spell-Out, they will try to move as little as possible. They will not hypothesize more overt movement(s) than is absolutely necessary for the successful parsing of the current input sentence. As a result, given two settings, both of which can make the current input parsable, the setting with fewer S(M)-parameters set to 1 should be preferred and adopted as the new hypothesis.

(167) If the Principle of Procrastinate is adhered to rigorously, there should not be any optional overt movement. Given the option of moving either before or after Spell-Out, the principle will always dictate that the movement occur after Spell-Out. Setting an S(M)-parameter to 1/0 is therefore no different from setting it to 0. So why should the value 1/0 be considered in the first place? If a movement *has to* occur before Spell-Out, then its S(M)-parameter must be set to 1 rather than 1/0. Consequently, the value 1/0 should not be tried unless it is the only value which can make all the strings in a given language parsable.

(168) In cases where overt movement is absolutely necessary, the Principle of Procrastinate will require that the movements which are more "essential" be considered first. Now which movements are more essential? According to Chomsky, the Principle of Procrastinate can be overridden to let a movement occur before Spell-Out only if the feature to be checked by this movement is "strong" i.e. realized in overt morphology. In view of the fact that A-movement and head-movement often occur for morphological reasons while $\bar{A}$-movements do not, the former are more essential than the latter. In our model, overt movement is independent of overt morphology, so the morphological explanation may not be available. But there is a common assumption that A-movement and head-movement are more closely related to the *basic* word order of a language than $\bar{A}$-movements which are more likely to be associated with interrogation, focusing and topicalization. In this sense, A-movements and head movements are more essential than A-bar movements. If overt movement is to be considered at all, priority should be given to the former rather than the latter.

To sum up, the Principle of Procrastinate provides certain constraints on or preferences for the choice of the next parameter setting to be tried in the learning process. In particular, the following

ordering rules seem to be deducible from this general principle:

(169) **(i)** Given two parameter settings P_1 and P_2, with N_1 and N_2 $(0 \leq N_1, 0 \leq N_2)$ being the respective numbers of S(M)-parameters set to 1/0 in P_1 and P_2 , $P_1 \prec P_2$ if $N_1 < N_2$. In other words, the setting which allows for fewer optional overt movements is to be considered first.

(ii) Given two parameter settings P_1 and P_2, $P_1 \prec P_2$ if S(M(cspec)) is set to 0 in P_1 and 1 in P_2. In other words, the setting which does not require overt $\bar{\text{A}}$-movement is to be considered first.

(iii) Given two parameter settings P_1 and P_2, with N_1 and N_2 $(0 \leq N_1, 0 \leq N_2)$ being the total numbers of S(M)-parameters set to 1 in P_1 and P_2 , $P_1 \prec P_2$ if $N_1 < N_2$.

These ordering rules are to be applied in the sequence given above. The second rule is applied only if the first one fails to decide on the precedence, and the third applied only if the second fails to do so. This order of rule application is not directly deducible from the Principle of Procrastinate, but it is not totally stipulative, either. Comparing optional overt movement and overt $\bar{\text{A}}$-movement, we find the latter "less evil" than the former which, according to the principle, should not exist at all. In our particular parameter space, optional movements *always* result in subset relations while overt $\bar{\text{A}}$-movements do so only in *some* contexts. This also suggests that optional movement should be the last choice. Here is a situation where linguistic and computational considerations seem to agree with each other. The ordering of (ii) and (iii) is less justified by the Principle of Procrastinate, though. We assume here that a setting *without* overt $\bar{\text{A}}$-movement is to be preferred over a setting *with* it even if the total number of overt movements in the former is greater than that in the latter. The decision here is made on qualitative rather than quantitative grounds. Overt non-$\bar{\text{A}}$-movements are assumed to be "less evil" than overt $\bar{\text{A}}$-movements. Therefore the latter should be avoided even at the cost of having more other movements. So far this choice has been motivated by computational considerations more than linguistic arguments. Subset relations may arise from settings with overt $\bar{\text{A}}$-movement while they never arise from settings without it. By putting off overt $\bar{\text{A}}$-movements as much as possible, learnability can be guaranteed. The linguistic intuition in support of our preference here is that $\bar{\text{A}}$-movements seem to be more "peripheral" than A-movements and head movements on the whole. Whether this intuition is correct or empirically justifiable is an open question. In any event, we will suppose for the time being that there are qualitative differences between different movements. We assume that quantitative arguments apply only in cases where qualitative considerations yield no result. In this sense, (iii) acts as a default rule which applies only if nothing else works. It should be pointed out that there are many settings which will remain unordered to each other after all the precedence rules have been applied. Fortunately, these settings are never in subset relations and they can be enumerated in any order without affecting learnability.

Sorting all the settings in our parameter space, using the precedence rules above, we get an

ordered list of S(M)-parameter settings. This list is given in Appendix C. The Prolog program which implements the ordering rules as well as the sorting function is given in Appendix A.3. The settings in Appendix C are listed in 50 groups and numbered in the order in which they are to be tried in the parameter setting process. We notice that the first setting in the list is [0, 0, 0, 0, 0, 0, 0, 0] which requires no overt movement, and the last setting is [1, 1, 1, 1, 1, 1/0, 1/0, 1/0] which allows for the maximal number of optional movements in addition to requiring every other movement to be overt. Each group number is accompanied by three digits. The first shows the number of parameters set to 1/0, the second indicates whether S(M(cspec)) is set to 1, and the last is the total number of parameters set to 1 or 1/0 in a setting. The settings which appear in the same group have no precedence determined among themselves. They are therefore unordered with respect to each other within that group.

It turns out that the Subset Principle can be observed if our enumerative learner goes through the hypothetical settings in the order given in Appendix C. This is not a surprise. We noted in 4.1.4 that subset relations arise from two types of settings. The first type consists of settings where one or more S(M)-parameters are set to 1/0. Given two settings P_1 and P_2 which are identical except that some parameter(s) are set to 1/0 in P_2 but not in P_1, P_1 is a subset setting of P_2. In order not to violate the Subset Principle, P_1 must be enumerated before P_2. This condition is obviously met in the ordering we have obtained. We can see in Appendix C that all the settings with the value 1/0 come after the settings without this value. In addition, for parameter settings where the value 1/0 does exist, a setting with n parameters set to 1/0 always comes before a setting where $n+1$ parameters are set to 1/0. We can thus rest assured that no subset relations arising from optional movement will cause any problem for the learner.

The second type of superset settings share the characteristic that they have S(M(spec1)), S(M(spec2)) and S(M(cspec)) all set to 1. Given two settings P_1 and P_2 which are identical except that these three parameters are all set to 1 in P_2 but not in P_1, P_1 can be a subset setting of P_2.[2] According to the Subset Principle, P_1 must occur before P_2 in the enumeration. This condition is again met in our ordering. For settings which have identical values for S(M(agr1)), S(M(asp)), S(M(tns)), S(M(agr2)) and S(M(c)), the setting [. . . 1, 1, 1] is always encountered before [. . . 1, 1, 0], [. . . 1, 0, 1], [. . . 0, 1, 1], [. . . 1, 0, 0], [. . . 0, 1, 0], [. . . 0, 0, 1] or [. . . 0, 0, 0]. Take the settings where the first 5 parameters are all set to 0 as an example. The setting [0, 0, 0, 0, 0, 1, 1, 1] is found in Group 11 while the other settings are found in Group 1, Group 2, Group 9 and Group 10. This ordering is achieved by the joint effect of (ii) and (iii) in (169).

We may wonder what happens to the settings within a single group where there is no ordering. Are there subset relations in any group? The answer is negative. The languages generated with settings in the same group are either disjoint or intersecting with each other. To illustrate this, let us look at Group 5. There are 35 settings in this group. When the all HD-parameters are set to

[2] The use of "can" indicates that this is a necessary but not sufficient condition for the subset relation.

"i" and all S(F)-parameters set to 0-0, three distinct non-empty languages can be generated with the settings in this group. These languages are (a) [s (often) v (o)], (b) [(often) v (o) s] and [v (often) s (o)]. As we can see, none of these languages is properly included in any other. Therefore it does not matter how the settings of those languages are ordered relative to each other. Interested readers may check the other groups to see that this holds in every group. It may happen that two settings in a single group generate languages that are identical to each other. This is no problem because the language can be acquired no matter which setting is selected.

5.2.2 The Learning Algorithm

We can now describe our learning algorithm as follows.

(170) **(I)** Get all value combinations in a given parameter space and place them in a list P.

(II) Sort P according to the precedence rules in (169) and get the ordered list P_{ord} as output.

(III) Start learning language L.

(i) Select any string S from L and try to parse S.

(ii) If S is successfully parsed, go back to (i).

Otherwise, reset the parameters to the first value combination P_1 in P_{ord} and remove P_1 from P_{ord}. Go to (i).

If L is in the given parameter space, the learning process will eventually stay in (i) and never leave it. At this point, we can generate $L\prime$ which is the set of strings that can be successfully parsed with the current parameter setting. If $L\prime = L$, then L is learnable. We have converged on the correct value combination. If $L \subset L\prime$, then L is not learnable. We have converged on a superset setting for the grammar of L. If L is not in the given parameter space at all, P_{ord} will eventually becomes empty and the learning process will get stuck in (ii).

The Prolog program that implements the algorithm in (170) is given in Appendix A.4. The ordered list of parameter settings is computed off-line using the `get_settings/0` predicate.[3] (The next setting to be tried at each point can also be computed on-line, and the result will be the same. The off-line computation just makes the calculation simpler and the execution of the learning procedure more efficient.) The learning session is initiated by calling `sp/0` which keeps putting out the "`Next?`" prompt at which we can type in

a. a string from a language for the learner to process;

b. "`current_setting`" to have the current setting displayed;

c. "`generate`" to get the complete set of strings that can be generated with the current setting;

[3]The predicates in Prolog are referred to by the form X/Y where X is the predicate name and Y is the number of arguments in the predicate.

d. "`initialize`" to put the learner back to the initial stage; or

e. "`bye`" to terminate the session.

Appendix D contains a number of Prolog sessions. D.1 and D.2 illustrate the process in which the language [s (often) iv, s (often) tv o, s (often) o tv, o s (often) tv] (which can be Chinese) is acquired with the program in Appendix A.4. The input strings are numbered "%1", "%2", ... in the two sessions. The successive settings are numbered "%a", "tt %b", etc. In D.1, the strings %1 and %2 can be parsed with the initial setting, so the parameter values remained unchanged. Each of the strings in %3–%18 triggered a resetting of the parameters. After each resetting, the "`generate`" command is given to have all the strings accepted by the current setting generated, so that we can see the language that the learner "speaks" at that particular point. The learner converged on the correct setting at %18, after which all of the possible strings in the language (%19, %20, %21, %22, %23, %24 %25 and %26) became analyzable and no further resetting is triggered. As the output of "`generate`" shows, the current language is exactly the language we have tried to acquire, there being neither overgeneration nor undergeneration. In the D.2 session, the input strings were presented in a different order, but the final result is the same.

5.2.3 Properties of the Learning Algorithm

Several comments can be made on the learning sessions described above.

(171) (i) The learner converged on the correct setting on the basis of positive evidence only. Every input string presented to the learner is a grammatical sentence in the language.

(ii) The learning procedure is incremental. All resetting decisions are based on the current setting and the current input string only. The learner does not have to remember any of the previous strings, nor does she have to memorize the previous settings.[4]

(iii) The convergence does not depend upon any particular ordering of the input strings. The string to be presented at the next prompt can be selected randomly. The sessions in D.1 and D.2 differ in terms of the order in which the input strings are presented, but their final outcome is the same. The learner does require, however, that (a) all the possible (types of) strings in the language be presented in a finite amount of time, and

[4] One may argue that the learner has to memorize the previous settings in the sense that no setting that has been tried in the past can be tried again. But in most cases the learner can tell from the current setting which settings have been previously tried. As the settings being tested have progressively more overt movements, all previous settings should have fewer S(M)-parameters set to 1. The only previously-tested settings she may have to remember are those in the same group. Those settings have the same number of parameters set to 1, so she cannot tell from the current setting which of the other settings in that group have been tried before. But even there memorization may not be necessary. As the settings in the same group generate disjoint or intersecting languages, the settings can be selected randomly. If one of the settings results in a successful parse, she can keep that setting and forget about all the other settings in that group. If none of the settings in that group succeeds, however, she may grope in the dark for a while, but eventually she will realize that no setting in that group can be the target one and decide to consider the next group of settings which requires more overt movement.

(b) the presentation of some (types of) strings be repeated. In D.1 the string [s tv o] was presented ten times and eight of them triggered resetting. These requirements are empirically plausible. Children do get exposed to all sentence types in a finite amount of time and common sentence patterns do get repeated over and over again in the input.

(iv) The learner has to go through a number of intermediate grammars before she arrives at the correct setting. The intermediate settings that are traversed in the learning process can vary according to the way input strings are presented. At a given point in the learning process, different input strings can cause the parameters to be reset to different values. A comparison of the sessions in D.1 and D.2 shows this. For example, after arriving at the setting [0, 0, 0, 0, 0, 1, 1, 0] (%b in both sessions), the learner was presented different strings in the two sessions. In D.1, she was given [s tv o] which triggered the setting [1, 0, 0, 0, 0, 1, 0, 0]; in D.2, she was given [o s tv] which triggered a different setting: [0, 0, 0, 0, 0, 1, 1, 1]. Due to the fact that the presentation of input strings is different in the two sessions, the intermediate settings are different. While most settings appeared in both sessions, some settings were traversed in one session only. We notice that there are fewer intermediate settings in D.2 and the learner converged on the same correct setting with fewer input strings. The general picture seems to be the following: given a set of input strings, there is a definite set of intermediate settings that *can* be traversed. In a particular learning situation, however, only a subset of the settings will be reached and the members in this set can vary according how the input strings are presented sequentially.

(v) No Single Value Constraint (Clark 1988, 1990, Gibson and Wexler 1993) is imposed on the parameter setting process. As we can see in the sessions, two successive settings can differ by more than one parameter values. In D.1, for instance, the settings in %a and %b differ by two values while %e and %f differ by 5 values. It is not the case, however, that the learner can arbitrarily choose the next setting. Nor is the learner non-conservative. In fact, the learner always tries to make the current input string analyzable by making the smallest change in the parameter values. Bigger changes are attempted only after the smaller changes have failed to result in a successful analysis. This is clear from the ordered list in Appendix C. In this list, all adjacent settings differ minimally. While resetting the parameters, the learner tries the settings one by one and she will adopt a more drastic change only if the less drastic ones have failed to produce any result.

(vi) No Pendulum Problem (Randall ??) exists for the current learning algorithm. Why this is so is obvious. The learner proceeds through the list of hypothetical settings in a unidirectional way. Once a setting is considered incorrect, it will not be considered again. Such determinism is made possible by the fact that the learner is conservative and would never entertain settings with more overt movements if settings with fewer overt

movements had not been considered yet. Therefore, once a setting is reached, there is no need to consider settings that require fewer overt movements.

5.2.4 Learning All Languages in the Parameter Space

The sessions in D.1 an D.2 have only shown that at least some language is learnable in our present system. To prove that every language in our parameter space is learnable, we need to run a session for each of the languages. This exhaustive testing can be performed by calling `learn_all_langs/0` which is defined in the Prolog program in Appendix A.4. We first use `get_pspace/0` to get (a) the ordered list of parameter settings (done by calling `get_settings/0`) and (b) all the possible languages in the parameter space (done by calling `get_languages/0`). The `learn_all/1` predicate then feeds the languages one by one into `learn1/1` which conducts a learning session for each particular language. The real work is done by `learn/1` which keeps resetting the parameters until all the strings in the language become parsable. At this point, `generate/1` is called to get the complete set of strings generated with the current setting. A language is learnable if the set of strings generated is exactly the target language presented to the learner and not learnable if it is a superset of the target language[5]. Note that this Prolog program and all the other programs mentioned so far presupposes the existence of a parser which implements our experimental syntactic model. This parser will be discussed in Chapter 6.

In Appendix D.3 and D.4, we find two Prolog sessions run with `learn_all_langs/0`. The two sessions differ in that in D.3 *often* does not appear in the input strings while it does appear in D.4. In D.3, all languages are learnable except two: [o tv s] and [o s tv]. We have observed in Chapter 4 that these are the two languages whose superset languages has neither optional movement nor overt Ā-movement. We have decided that these two languages, which are odd in that they do not allow for intransitive sentences, can probably be ignored. Our syntactic model can be made more restrictive so that such languages are not generated at all. In D.4, however, all the languages, including these two odd languages, are proven to be learnable. The appearance of *often* has made some otherwise indistinguishable languages distinct from each other. One result of this is that the two odd languages are found to be not properly included in any other language.

One thing we notice in these sessions is that, while all the languages in the parameter space are learnable, not every setting in the parameter space can become a final setting for some language. This is not a surprise, though. We have seen in Chapter 4 that the relationship between settings and languages can be many-to-one. A single language can be generated with more than one setting. In the learning process, however, only one of the possible setting will be converged on as the final setting for a language. As we can see in Appendix B.2, the language [s iv, s tv o] can be generated with 32 different settings. The setting which the learner has converged on in the D.1 session is the

[5] A language is also not learnable if no setting in the parameter space can make every string in this language analyzable. But this will not happen here because such a language would fall outside the given parameter space thus not considered a possible language to be acquired in the first place.

initial setting. Once this setting is in place, all the strings in the language will be parsed successfully and no resetting will be considered any more. In D.4, the appearance of *often* made four pure SVO languages distinct from each other: [(often) s v (o)], [s (often) v (o)], [s v (often) (o)], and [s (often) v (o), (often) s v (o)]. The final settings reached for these languages by the learners are respectively [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0], [1, 1, 1, 1, 0, 1, 0, 0] and [0, 0, 0, 0, 0, 1/0, 0, 0]. Looking at all the possible settings for these languages (shown in Appendix B.3), we see that the final setting reached for each language is always the setting which is to be ordered first among alternative settings by the precedence rules in (169). In general, the learner always tries to select the setting with fewest overt movements possible to fit the current data. Further movements are considered only if there is evidence that the current setting is inadequate. Given the string [s iv] or [s tv o], the learner will stay with the initial setting, but a new string like [s tv often o] will make her reset the parameters to [0, 0, 0, 0, 0, 1, 0, 0].

Similar tests to those in D.3 and D.4 have also been run on parameter spaces where some S(F)-parameters are set to 1-0 so that auxiliaries appear. Due to space limitation, these sessions are not given in the appendices. Those tests show that our current learning algorithm works just as well in acquiring languages with auxiliaries and grammatical particles.[6]

We will conclude this section by observing a special property that the current learning algorithm has with regard to noisy input. As we have seen, the learner in our model takes every piece of input data seriously and treats it as a grammatical sentence in the language. This in principle should cause problems when the input is degenerated. However, those problems are sometimes accidentally remedied in our current system. Ungrammatical input may trigger wrong settings, but as long as those settings are ordered before one of the possible target settings,[7] the learner still has a chance to recover from the error. This is illustrated in D.5. The target language to be acquired in this session is the same as the one in D.1 and D.2, i.e. [s (often) iv, s (often) tv o, s (often) o tv, o s (often) tv]. The learner was presented those strings plus a number of strings which are not in the language (%2, %4, %6, %13, %18, %21, %28, %33, %37, %42, %46 and %47). The first 8 deviant strings triggered some wrong settings, but the learner still managed to converge on one of the correct settings (the setting at %z). This is however the last target setting the learner can ever reach. If she for any reason leaves this state and tries some settings further down the list, there will be no more chance of convergence. In the D.5 session, the learner was presented more deviant strings after %z was reached. These strings made the learner adopt the settings %b1 and %c1 which generate superset languages of the target languages. The deviant strings that appear after %c1 made things even worse. The learner eventually ran out of further hypotheses and the learning ended in failure.

The implications of this peculiar property of the learning algorithm are not clear. It may mean that the current system can provide a mechanism for language change. When the input data is

[6] The test sessions are available upon request.

[7] Recall that there can be more than one setting which is compatible with a given language.

perfect, only one of the possible settings for a language will be reachable. When the input contains deviant strings, however, alternative settings will be considered, as we have seen in D.5 where four of the possible settings (%l, %p, %w and %a1) are reached at some point of the learning process. The languages generated with these settings are just weakly equivalent to the target language. Underlyingly, each of those settings can potentially generate a different language. At this stage, this account for language change is purely speculative. Much more careful work has to be done before it can be taken seriously. What is certain, however, is that the learning algorithm as it stands now is not robust enough. Further research has to be done to make it more empirically plausible.

5.3 Setting Other Parameters

In this section we discuss how the other two types of parameters – HD-parameters and S(F)-parameters – can be set together with S(M)-parameters. HD-parameters interact with S(M)-parameters in determining the word order of a language. We want to know whether the learning algorithm presented in the previous section can be modified to set HD-parameters as well as S(M)-parameters without losing its basic properties. The S(F)-parameters can be set independently using a separate algorithm, though their values can interact with the values of S(M)- and HD-parameters, resulting in such linguistic phenomena as auxiliaries, grammatical particles and expletives.

5.3.1 Setting HD-Parameters

When HD-parameters are kept out of the picture, there is only one kind of parameters to reset when an input sentence is found to be syntactically unanalyzable in terms of word order. Now that both S(M)- and HD-parameters are available, we are given a choice. Upon failing to parse an input string, we have to decide which type of parameters to reset. This may seem to be a problem. In Gibson and Wexler (1993) (G&W hereafter) which addresses a similar problem, some target settings, called *local maxima*, are found to be unlearnable. The main contributor to this problem is the fact that there are two types of word order parameters that can be set when a parsing failure occurs. The parameters in their parameter space are $\bar{X}$-parameters, which are called HD-parameters in our model, and the V2-parameter which is similar to our S(M)-parameter in that it also determines whether a certain movement is overt. When an input sentence fails to be analyzed by the current grammar, the learner can reset either the V2-parameter or one of the $\bar{X}$-parameters, but not both. It is discovered in G&W that, with the Single Value Constraint, the Greediness Constraint, but no parameter ordering in any sense, the learner can get into an incorrect grammar from which she is never able to escape. We may wonder if the same problem will occur in our system, since we also have to set two types of parameters either of which may be responsible for the word order of a language.

Upon closer inspection, however, we find that the situation here is very different from the one

in G&W where local maxima occur. First of all, we only have one kind of $\bar{X}$-parameter – the complement-head parameter – while both the specifier-head and complement-head parameters are present in the model G&W assumes. They discovered that local maxima can be avoided if the value of the specifier-head parameter can be fixed before the V2-parameter and the complement-head parameter are set. But this condition is satisfied by default in our system, since there is no specifier-head parameter in our model at all. It is as if the specifier-head parameter were set before the other parameters are considered. According to G&W, this should be sufficient to prevent local maxima. Secondly, the Single Value Constraint is not assumed in our system. G&W has shown that local maxima can also be avoided by removing the Single Value Constraint. This is another reason why local maxima should not occur in our model. We do assume the Greediness Constraint. G&W show that the removal of this constraint can help avoid local maxima as well, but will leave the learning algorithm so unconstrained that the correct grammar can only be found by chance.

The solution G&W finds most plausible for the prevention of local maxima is parameter-ordering. There are several ways of ordering the parameters and they favor the one where $\bar{X}$-parameters are set before the V2-parameter. We agree with them on that movement operations are costly and should not be considered unless a simple flip of the $\bar{X}$-parameter fails to solve the problem. We will therefore basically adopt this ordering hypothesis. When a change of parameter values is required, the learner is to try HD-parameters first. Resetting of S(M)-parameters is attempted only if the resetting of HD-parameters has failed to make the input string syntactically analyzable. However, the actual implementation of the ordering has to be more sophisticated than the one suggested in G&W. While there is only one "movement parameter" – the V2-parameter – with two possible values in G&W's model, we have eight movement parameters with 864 possible value combinations. The values of the two HD-parameters in our system can interact with any of the 864 value combinations, producing a total of 3456 possible settings. In particular, we must allow the four possible value combinations of HD-parameters, [i, i], [i, f], [f, i] and [f, f], to interact with each value combination of S(M)-parameters. To do this we need the following algorithm.

(172) Given: an ordered list of S(M)-parameter settings P_{ord}.

(i) Initially set all S(M)-parameters to 0 and HD-parameters to any values.

(ii) Select any string S from the target language L and try to parse S.

(iii) If S is successfully parsed, go to (ii);
Otherwise, go to (iv).

(iv) Reset HD-parameters and try to parse S with the new setting.

If S is successfully parsed, retain the new HD-parameter setting and go back to (ii);

If none of the settings of HD-parameters results in a successful parse of S, reset S(M)-parameters to the first value combination P_1 in P_{ord}, and remove P_1 from P_{ord}. Go back to (ii).

Intuitively, what the above algorithm does is the following. For each of the S(M)-parameter settings, try combining it with any of the four HD-parameter settings. If none of the four makes the input string analyzable, then pick the next S(M)-parameter setting from the ordered list and try the combinations again. In other words, each setting in the ordered list is expanded into four different settings. The setting [1, 1, 0, 0, 0, 1, 1, 0], for example, will become the following four settings:

(173) [1, 1, 0, 0, 0, 1, 1, 0, i, i]
[1, 1, 0, 0, 0, 1, 1, 0, i, f]
[1, 1, 0, 0, 0, 1, 1, 0, f, i]
[1, 1, 0, 0, 0, 1, 1, 0, f, f]

This being the case, we can have an alternative algorithm which is equivalent to (172) in consequence but computationally simpler. We can expand the ordered list of S(M)-parameters in Appendix C by turning each setting in the list into a group of four settings, each having a different value combination of HD-parameters. The ordering of settings in the original list now becomes the ordering of groups of settings, with the original order maintained. Within each group, the four settings can be ordered in any way. For the moment, we will order them arbitrarily in the order shown in (173). This ordering has the implication that the default setting for HD-parameters is head-initial. This does not have to be correct, however, and the success of our learning algorithm does not depend on this arbitrary choice. The learning algorithm will work the same way no matter how these settings are ordered within each group, because the four languages generated with the four settings of a single group never properly include each other.

The outcome of the above expansion is obvious. Due to the length of this expanded list, we will put it in the appendices. But the beginning of the list is given below to give the reader some concrete idea of what the new list looks like.

(174) [0,0,0,0,0,0,0,0,i,i]
[0,0,0,0,0,0,0,0,i,f]
[0,0,0,0,0,0,0,0,f,i]
[0,0,0,0,0,0,0,0,f,f]
[1,0,0,0,0,0,0,0,i,i]
[1,0,0,0,0,0,0,0,i,f]
[1,0,0,0,0,0,0,0,f,i]
[1,0,0,0,0,0,0,0,f,f]
[0,1,0,0,0,0,0,0,i,i]
[0,1,0,0,0,0,0,0,i,f]
[0,1,0,0,0,0,0,0,f,i]
[0,1,0,0,0,0,0,0,f,f]

```
[0,0,1,0,0,0,0,0,i,i]
[0,0,1,0,0,0,0,0,i,f]
[0,0,1,0,0,0,0,0,f,i]
[0,0,1,0,0,0,0,0,f,f]
...
```

With this list in place, we can use the simple algorithm in (170) to set both the S(M)-parameters and HD-parameters. As a result, the Prolog program in Appendix A.4 can be used, with very little modification, to set both types of parameters.

Testing sessions have been run with this new list of parameter settings. The session in D.6 illustrates how an individual language can be acquired using the `sp/0` predicate. The target language in this case is [s iv aux, s o tv aux, o s tv aux]. The parameters being set here are the eight S(M)-parameters and two HD-parameters. All S(F)-parameters are constantly set to 0-0 except S(F(tns)) which is constantly set to 1-0 to allow the occurrence of some auxiliaries. We can find in this session all the properties listed in (171). The only thing new is the setting of HD-parameters.

The learnability of all the languages in this expanded parameter space has been exhaustively tested using the `learn_all_langs/0` predicate. Some sessions are run with all S(F)-parameters constantly set to 0-0 and some with one of the S(F)-parameters (S(F(tns)) set to 1-0. Auxiliaries appear when S(F(tns)) is set to 1-0. The results are again very similar to those obtained when HD-parameters have fixed values. There are a few languages which are not learnable. These languages are again [o tv s] and [o s tv] which are odd in not having intransitive sentences. When S(F(tns)) is set to 1-0, the unlearnable languages are [o s tv], [o s tv aux] and [o tv s aux]. No language is unlearnable, however, when the position of *often* is taken into consideration. Apparently, the appearance of *often* can make these odd languages so distinct that they are not properly included in any other language. The log files of those sessions are not included in the appendices for reasons of space, but they are available to anyone who wants to see the actual process.

In sum, the fact that there are two kinds of word order parameters in our system does not seem to create any problem for learnability. The success of parameter setting may be attributable to parameter ordering, the absence of the Single Value Constraint, or the non-existence of specifier-head parameters. It is important to note that the parameter ordering here is not artificially or arbitrarily imposed on the learning system. It is deducible from some general linguistic principle, namely the Principle of Procrastinate. This principle not only tells us how the S(M)-parameters should be ordered but also explains why we should try resetting HD-parameters before resetting S(M)-parameters. Thus the ordering is linguistically motivated.

5.3.2 Setting S(F)-Parameters

As mentioned above, the S(F)-parameters can be set independently on the basis of morphological evidence only. We have assumed that each S(F)-parameter can have two sub-parameters, represented

as F-L, with the value of F (1 or 0) determining whether the F-feature is spelled out and the value of L (1 or 0) determining whether the L-feature is spelled out. There are therefore four value combinations for each S(F)-parameter: 0-0, 0-1, 1-0 and 1-1. The two sub-parameters in each parameter can again be set independently. The value of F is based on the morphological properties of function words (such as auxiliaries) only and the value of L on the morphology of content words (such as nouns and verbs) only. What we have to do in parameter setting is the following. We have to look at the function words, if there are any, and examine their morphological make-up. Since no auxiliary can appear unless F is set to 1 in at least one S(F)-parameter, the appearance of an auxiliary in the input string tells us that at least one S(F)-parameter is set to 1-X. ("X" is a variable indicating that L can have any possible value.) Exactly how many S(F)-parameters should be set to 1-X depends on how much information the auxiliary carries. If it is inflected for tense and agreement, for instance, then both S(F(tns)) and S(F(agr)) should be set to 1-X. We also have to look at the content words and see what features are morphologically represented. For example, a noun inflected for case will tell us that S(F(case)) is set to X-1.

It has been assumed that there is a separate learning module which is responsible for finding out whether a word is morphologically inflected and what features are represented by the inflection. The learner in our system takes the output of this module as its input. The input strings in our case consist of symbols which are of the form C-F where C is a category label such as s, o, iv, tv and aux, and F is a list which contains zero or more features.[8] A feature appears in the list only if it is morphologically realized in the language under question. Words that carry no inflectional morphology will have an empty list attached to it. Here is a sample string.

(175) [s-[] aux-[agr,tns] iv-[asp]]

The string in (175) represents a sentence which consists of the following "words" from left to right: a subject NP with no inflection, an auxiliary inflected for agreement and tense, and an intransitive verb inflected for aspect. The features that are in the feature list of "aux" are overt F-features and the one in the feature list of "iv" is an overt L-features. We assume that the learner is able to get the representation in (175) using some independent learning strategies. For instance, she is supposed to be able to conclude from words like *do*, *does*, *did* and *have*, *has*, *had* that English auxiliaries are inflected for agreement and tense. Thus the main auxiliary in English should be represented as aux-[agr,tns]. Once representations like the one in (175) are available, the setting of S(F)-parameters is straight-forward. For instance, the string in (175) can tell us that, in this language, S(F(agr)) and S(F(tns)) are set to 1-X while S(F(asp)) is set to X-1.

At first sight, we might think that there exist relations of proper inclusion in the morphological patterns presented here. For instance, "aux-[tns]" may seem to be properly included in "aux-[agr,tns]". A second thought tells us that this is not true. A feature appears in the list *if and only*

[8] The only exception is *often* which will appear by itself without a feature list attached to it.

if this feature is morphologically visible. Therefore, "aux-[tns]" is acceptable to the parser only if S(F(agr)) is set to 0-X, and "aux-[agr,tns]" is acceptable only if S(F(agr)) is set to 1-X. There is no setting with which both "aux-[tns]" and "aux-[agr,tns]" can be grammatically analyzed. If S(F(agr)) is originally set to 1-X, the analysis of "aux-[tns]" will end in failure which will trigger the resetting of this parameter to 0-X; Likewise, "aux-[agr,tns]" cannot be analyzed with S(F(agr)) set to 0-X, which will cause it to be reset to 1-X. No setting in the parameter space of S(F)-parameters is a proper subset of another. Because of this, the failure-driven learning algorithm we have assumed will always succeed in setting the S(F)-parameters on the basis of positive evidence only. No parameter-ordering or default setting is necessary. However, in view of the fact that inflectional morphology is generally absent in children's speech at the initial stages of language acquisition, we will assume that all S(F)-parameters are initially set to 0-0. They will retain this value if no overt inflectional morphology is found in the target language. If the target language does have overt morphology, they will be reset to 0-1, 1-0 or 1-1 when the inflections are detected and analyzed by children in the acquisition process. Notice that this "initial-0" hypothesis is empirically-based rather than computationally-based. We assume this in order to make our learning algorithm more natural, but the success of our algorithm does not depend on this hypothesis.

Now that we have both the morphological parameters (i.e. S(F)-parameters) and word order parameters (i.e. S(M)- and HD- parameters), we have to decide which parameters to reset first in the event of a parsing failure. A parsing failure may occur because (a) some S(F)-parameter has the wrong value value, (b) some S(M)- or HD- parameter has the wrong value, or both (a) and (b). In order to make the input sentence interpretable, we may have to (a) reset S(F)-parameter(s) only, (b) reset S(M)/HD parameters only, or (c) reset both types of parameters. Since S(F)-parameters can always be set to the right values in one step once the morphologically pattern is correctly identified, their values should be checked first. If no S(F)-parameter is found to have the wrong value, then some S(M)/HD parameters must be reset. If some S(F)-parameter does have the wrong value, we will first reset this parameter and see if the resetting can result in a successful parse. If so, no S(M)/HD parameters need to be reset. Otherwise, we will have to reset some S(M)/HD parameter(s) in addition to the S(F)-parameter(s).

The algorithm that checks S(F)-parameter values can be described as follows.

(176) Go through the feature list of every word in the input sentence and do the following whenever a feature is encountered. For an F-feature (which is found in an auxiliary in our system), leave the S(F)-parameter value of this feature untouched if it is already set to 1-X; otherwise reset it to 1-X. For an L-feature (which is found on nouns and verbs), leave the S(F)-parameter value of this feature untouched if it is already set to X-1; otherwise reset it to X-1.

Embedding this sub-algorithm in our general acquisition procedure, we now have the following learning algorithm:

(177) Given: an ordered list of value combinations of S(M)- and HD- parameters P_{ord}.

(i) Initially set all S(M)-parameters to 0, all HD-parameters to i, and all S(F)-parameters to 0-0.

(ii) Select any string S from the target language L and try to parse S with the current setting.

(iii) If S is successfully parsed, go to (ii);
Otherwise, go to (iv).

(iv) Check the values of S(F)-parameters using the sub-algorithm in (176) and parse S again.
If S is successfully parsed, go to (ii);
Otherwise, go to (v).

(v) Reset S(M)-/HD- parameters to the first value combination P_1 in P_{ord} and remove P_1 from P_{ord}. Go back to (ii).

The Prolog program that implements this new algorithm can be found in Appendix A.6. We can again use `sp/0` to set the parameters on-line for a given language, `learn1/1` to find out if a given language is learnable, and `learn_all_langs/0` to check out if all the languages in the parameter space are learnable. The search space in this case is huge, comprising tens of thousands of possible languages. Exhaustive testing by computer has shown that all the languages in this big parameter space are learnable,[9] but we do not have to depend on the computer search in order to know that it is true. We know that the S(M)- and HD- parameters can set correctly by themselves for every language in the parameter space. We also know that the S(F)-parameters can be set correctly on its own as well. Since there is no value dependency between S(F)-parameters and S(M)-/HD-parameters, the combination of those parameters does not result in any additional complexity. We can thus conclude that learnability is guaranteed for all the languages that can be generated in our experimental model.

5.4 Acquiring Little Languages

In this section we see how the complete set of parameters is set in some little languages. Due to the huge number of languages in the parameter space, it is not possible to put the log file of running `learn_all_langs/0` in the appendices. To see the behavior of the learner in this bigger parameter space, we will look at some Prolog sessions where the learnability of several individual languages is tested. The languages being tested are Little English, Little Japanese, Little Berber, Little Chinese, Little German and Little French, which are small subsets of the corresponding real languages. We will run `learn1/1` on Little English, Little Japanese, Little Berber and Little Chinese to test their learnability and then run `sp/0` on Little French and Little German to see the on-line processes whereby these two languages are acquired.

[9] Again there are those odd languages which are learnable only if *often* appears in the input strings.

5.4.1 Acquiring Little English

The portion of English to be acquired is (178).

(178) Little English:

```
{ s-[c1]  aux-[agr,tns]  (often) iv-[asp],
  s-[c1]  aux-[agr,tns]  (often) tv-[asp]  o-[c2]
}
```

These strings are instantiated by the English sentences *He was reading* and *He was chasing her.* The copula BE is treated as an auxiliary carrying agreement and tense information. It is a visible T0 which has moved to Agr1-0 before Spell-Out. "Aux-[agr,tns]" can be spelled out as *HAVE*, *BE* or *DO* in English. The choice of auxiliary in a particular sentence seems to be associated with the aspect feature of the sentence. It is spelled out as *BE* with progressive aspect, *HAVE* with perfective aspect, and *DO* with zero aspect.

Here is the Prolog session where Little English is acquired.

```
| ?- see(english),read(L),seen,learn1(L).
Trying to learn [[s-[c1],aux-[agr,tns],iv-[asp]],[s-[c1],aux-[agr,tns],tv-[asp]
,o-[c2]],[s-[c1],aux-[agr,tns],often,iv-[asp]],[s-[c1],aux-[agr,tns],often,tv-[
asp],o-[c2]]] ...
s(f(case)) is reset to 0-1
s(f(agr)) is reset to 1-0
s(f(tns)) is reset to 1-0
s(f(asp)) is reset to 0-1
Final setting: [0  0  0  1  0  1  0  0  i  i
 agr(1-0)  asp(0-1)  case(0-1)  pred(0-0)  tns(1-0)  ]
Language generated:
[s-[c1],aux-[agr,tns],iv-[asp]]
[s-[c1],aux-[agr,tns],tv-[asp],o-[c2]]
[s-[c1],aux-[agr,tns],often,iv-[asp]]
[s-[c1],aux-[agr,tns],often,tv-[asp],o-[c2]]
The language [[s-[c1],aux-[agr,tns],iv-[asp]],[s-[c1],aux-[agr,tns],tv-[asp],o-
[c2]],[s-[c1],aux-[agr,tns],often,iv-[asp]],[s-[c1],aux-[agr,tns],often,tv-[asp
],o-[c2]]] is learnable.

yes
| ?-
```

According to the final setting reached by the learner, Little English is a head-initial language where the subject NP moves to Agr1spec and T0 moves to Agr1-0. Agreement and tense features are spelled out on the auxiliary, aspect feature on the verb and case features on NPs.

5.4.2 Acquiring Little Japanese

The subset of Japanese to be acquired is (179).

(179) Little Japanese:

```
{ s-[c1]  (often) iv-[tns],
  s-[c1]  (often)  o-[c2]  tv-[tns],
  o-[c2]  s-[c1]  (often)  tv-[tns]
}
```

Instances of these strings can be found in (138), (139) and (140), repeated here as (180), (181) and (182).

(180) *Taroo-ga* *ki-ta*
Taroo-Nom come-Past
'Taroo came.'

(181) *Taroo-ga* *tegami-o* *kai-ta*
Taroo-Nom letter-Acc write-Past
'Taroo wrote a letter.'

(182) *tegami-o* *Taroo-ga* *kai-ta*
letter-Acc Taroo-Nom write-Past
'Taroo wrote a letter.'

Let us test its learnability by running `learn1/1`:

```
| ?- see(japanese),read(L),seen,learn1(L).
Trying to learn [[s-[c1],iv-[tns]],[s-[c1],o-[c2],tv-[tns]],[o-[c2],s-[c1],tv-[
tns]],[s-[c1],often,iv-[tns]],[s-[c1],often,o-[c2],tv-[tns]],[o-[c2],s-[c1],oft
en,tv-[tns]]] ...
s(f(case)) is reset to 0-1
s(f(tns)) is reset to 0-1
Final setting: [0  0  0  0  0  1  1  1  i  i
 agr(0-0)  asp(0-0)  case(0-1)  pred(0-0)  tns(0-1)  ]
Language generated:
[[s-[c1],iv-[tns]]
[s-[c1],o-[c2],tv-[tns]]
```

```
[o-[c2],s-[c1],tv-[tns]]
[s-[c1],often,iv-[tns]]
[s-[c1],often,o-[c2],tv-[tns]]
[o-[c2],s-[c1],often,tv-[tns]]
The language [[s-[c1],iv-[tns]],[s-[c1],o-[c2],tv-[tns]],[o-[c2],s-[c1],tv-[tns
]],[s-[c1],often,iv-[tns]],[s-[c1],often,o-[c2],tv-[tns]],[o-[c2],s-[c1],often,
tv-[tns]]] is learnable.

yes
| ?-
```

According to the final setting, Little Japanese is a head-initial language where the subject NP and object NP move to Agr1spec and Agr2spec respectively. In addition, one of them must move further up to Cspec. The fact that Little Japanese is identified as a head-initial rather than a head-final one shows that a simple SOV string is not sufficient for the identification of a head-final language. There are alternative settings for Little Japanese among which are (183) and (184) where IP is head-final. But (183) is not chosen because it requires more overt movements, and (184) is not chosen because it requires optional movement.

```
(183) [ 1  0  0  0  0  1  1  1  i  f
        agr(0-0)  asp(0-0)  case(0-1)  pred(0-0)  tns(0-1)
      ]

(184) [ 0  0  0  0  0  1  1  1/0  i  f
        agr(0-0)  asp(0-0)  case(0-1)  pred(0-0)  tns(0-1)
      ]
```

5.4.3 Acquiring Little Berber

Little Berber consists of the strings in (185).

(185) Little Berber:

```
{ iv-[agr,tns]  (often)  s-[],
  s-[]  iv-[agr,tns]  (often)
  tv-[agr,tns]  (often)  s-[]  o-[],
  s-[]  tv-[agr,tns]  (often)  o-[]
}
```

The 3rd and 4th strings in this set are exemplified by (146) and (147), repeated here as (186) and (187):

(186) *i-ara* *hmad* *tabrat*
3ms-wrote Ahmed letter
'Ahmed wrote the letter.'

(187) *hmad* *i-ara* *tabrat*
Ahmed 3ms-wrote letter
'Ahmed wrote the letter.'

I have no data on the position of *often* in Berber. Its position in Little Berber is purely hypothetical.[10] Let us see how the parameters are set for this language:

```
| ?- see(berber),read(L),seen,learn1(L).
Trying to learn [[iv-[agr,tns],s-[]],[s-[],iv-[agr,tns]],[tv-[agr,tns],s-[],o-[
]],[s-[],tv-[agr,tns],o-[]],[iv-[agr,tns],often,s-[]],[s-[],iv-[agr,tns],often]
,[tv-[agr,tns],often,s-[],o-[]],[s-[],tv-[agr,tns],often,o-[]]] ...
s(f(agr)) is reset to 0-1
s(f(tns)) is reset to 0-1
Final setting: [1  1  1  1  0  1/0  0  0  i  i
 agr(0-1)  asp(0-0)  case(0-0)  pred(0-0)  tns(0-1)  ]
Language generated:
[iv-[agr,tns],s-[]]
[s-[],iv-[agr,tns]]
[tv-[agr,tns],s-[],o-[]]
[s-[],tv-[agr,tns],o-[]]
[iv-[agr,tns],often,s-[]]
[s-[],iv-[agr,tns],often]
[tv-[agr,tns],often,s-[],o-[]]
[s-[],tv-[agr,tns],often,o-[]]
The language [[iv-[agr,tns],s-[]],[s-[],iv-[agr,tns]],[tv-[agr,tns],s-[],o-[]],
[s-[],tv-[agr,tns],o-[]],[iv-[agr,tns],often,s-[]],[s-[],iv-[agr,tns],often],[t
v-[agr,tns],often,s-[],o-[]],[s-[],tv-[agr,tns],often,o-[]]] is learnable.

yes
| ?-
```

In this session, Little Berber is identified as a language where the verb must move to Agr1-0 and the subject NP can optionally move to Agr1spec. The verb is required to have its agreement and tense features spelled out.

[10]The grammar employed in these learning sessions assumes that *often* can optionally appear in every language. Consequently any language without *often* will be found to be unlearnable. This is why I have to let *often* appear in Little Berber.

5.4.4 Acquiring Little Chinese

Little Chinese consists of the following strings:

(188) Little Chinese:

```
{ s-[]  (often)  iv-[]  aux-[asp]  aux-[pred],
  s-[]  (often)  tv-[]  o-[]  aux-[asp]  aux-[pred],
  s-[]  (often)  o-[]  tv-[]  aux-[asp]  aux-[pred],
  o-[]  s-[]  (often)  tv-[]  aux-[asp]  aux-[pred]
}
```

Here are some actual instances of those strings. The grammatical particle *ma* in the following sentences can be either a question marker or an affirmative marker depending upon the intonation. So they are all ambiguous in their romanized forms, having both an interrogative reading and a declarative reading. (They are not ambiguous when written in Chinese characters, as the two senses of *ma* are written in two different characters: “ ” and “ ”).

(189) *Ta lai le ma*
he come Asp Q/A
'Has he come? / He has come, as you know.'

(190) *Ta kan-wan nei-ben shu le ma*
you finish reading that book Asp Q/A
'Have you finished reading that book? / He has finished reading that book, as you know.'

(191) *Ta nei-ben shu kan-wan le ma*
you that book finish reading Asp Q/A
'Have you finished reading that book? / He has finished reading that book, as you know.'

(192) *Nei-ben shu ta kan-wan le ma*
that book you finish reading Asp Q/A
'Have you finished reading that book? / He has finished reading that book, as you know.'

Here is the session where Little Chinese is acquired.

```
| ?- see(chinese),read(L),seen,learn1(L).
Trying to learn [[s-[],iv-[],aux-[asp],aux-[pred]],[s-[],tv-[],o-[],aux-[asp],a
ux-[pred]],[s-[],o-[],tv-[],aux-[asp],aux-[pred]],[o-[],s-[],tv-[],aux-[asp],au
x-[pred]],[[s-[],often,iv-[],aux-[asp],aux-[pred]],[s-[],often,tv-[],o-[],aux-[
asp],aux-[pred]],[s-[],often,o-[],tv-[],aux-[asp],aux-[pred]],[o-[],s-[],often,
tv-[],aux-[asp],aux-[pred]]] ...
s(f(asp)) is reset to 1-0
```

```
s(f(pred)) is reset to 1-0
Final setting: [0  0  0  0  0  1  1/0  1  f  f
 agr(0-0)  asp(1-0)  case(0-0)  pred(1-0)  tns(0-0)  ]
Language generated:
[s-[],iv-[],aux-[asp],aux-[pred]]
[s-[],tv-[],o-[],aux-[asp],aux-[pred]]
[s-[],o-[],tv-[],aux-[asp],aux-[pred]]
[o-[],s-[],tv-[],aux-[asp],aux-[pred]]
[[s-[],often,iv-[],aux-[asp],aux-[pred]]
[s-[],often,tv-[],o-[],aux-[asp],aux-[pred]]
[s-[],often,o-[],tv-[],aux-[asp],aux-[pred]]
[o-[],s-[],often,tv-[],aux-[asp],aux-[pred]]
The language [[s-[],iv-[],aux-[asp],aux-[pred]],[s-[],tv-[],o-[],aux-[asp],aux-
[pred]],[s-[],o-[],tv-[],aux-[asp],aux-[pred]],[o-[],s-[],tv-[],aux-[asp],aux-[
pred]],[[s-[],often,iv-[],aux-[asp],aux-[pred]],[s-[],often,tv-[],o-[],aux-[asp
],aux-[pred]],[s-[],often,o-[],tv-[],aux-[asp],aux-[pred]],[o-[],s-[],often,tv-
[],aux-[asp],aux-[pred]]] is learnable.

yes
| ?-
```

The learner identified Little Chinese as a head-final language where overt movement to Agrspec is obligatory for the subject NP and optional for the object. Cspec must be filled at Spell-Out by the subject or by the object if it has moved to Agr2spec. The verb remains VP-internal while Asp0 and C0 are spelled out in situ.

5.4.5 Acquiring Little French

Here is the subset of French to be acquired.

(193) { s-[c1] iv-[tns,asp] (often)
  s-[c1] tv-[tns,asp] (often) o-[c2]
}

It differs from Little English in that (i) there is no auxiliary, and (ii) *often* appears post-verbally. The strings in (193) are exemplified by (194) and (195).

(194) *Il* *nage* *souvent*
he swims often
'He often swims.'

(195) *Il visite souvent Paris*
he visit often Paris
'He often visits Paris.'

The following session shows how Little French is acquired on-line.

```
| ?- sp.
The initial setting is
[0  0  0  0  0  0  0  0  i  i
 agr(0-0)  asp(0-0)  case(0-0)  pred(0-0)  tns(0-0)  ]
Next? [s-[],iv-[]].
Current setting remains unchanged.
Next? [s-[c1],iv-[]].                                    %1
Unable to parse [s-[c1],iv-[]]
Resetting the parameters ...

s(f(case)) is reset to 0-1
Successful parse.
Next? [s-[c1],iv-[tns]].                                 %2
Unable to parse [s-[c1],iv-[tns]]
Resetting the parameters ...

s(f(tns)) is reset to 0-1
Successful parse.
Next? [s-[c1],iv-[tns,asp]].                             %3
Unable to parse [s-[c1],iv-[tns,asp]]
Resetting the parameters ...

s(f(asp)) is reset to 0-1
Successful parse.
Next? [s-[c1],tv-[tns,asp],o-[c2]].                      %4
Current setting remains unchanged.
Next? [s-[c1],iv-[tns,asp],often].                       %5
Unable to parse [s-[c1],iv-[tns,asp],often]
Resetting the parameters ...

Word order parameters reset to: [1  1  1  1  0  1  0  0  i  i  ]
Successful parse.
Next? [s-[c1],tv-[tns,asp],often,o-[c2]].                %6
```

```
Current setting remains unchanged.
Next? current_setting.
[1  1  1  1  0  1  0  0  i  i
 agr(0-0)  asp(0-1)  case(0-1)  pred(0-0)  tns(0-1)  ]
Next? generate.
Language generated with current setting:
[s-[c1],iv-[tns,asp]]
[s-[c1],iv-[tns,asp],often]
[s-[c1],tv-[tns,asp],often,o-[c2]]
[s-[c1],tv-[tns,asp],o-[c2]]

Next? bye.

yes
| ?-
```

The presentation of input strings here is inconsistent with regard to overt morphology: %1 has no overt morphology whatsoever; %2 has overt case only; %3 has overt tense in addition to overt case; and %4 has all the overt morphology in Little French. This is done intentionally in order to mimic children's gradual acquisition of morphological knowledge. It is a common observation that children usually fail to notice or ignore inflectional morphology at the early stages of acquisition. When this happens, the sentence which is actually being analyzed by children are strings where part or all of the overt morphology is missing. The string `[ s-[] iv-[] ]` thus represents a piece of input data whose inflectional markings are not noticed by children. As acquisition progresses, the morphology is gradually worked out and becomes available for syntactic analysis. In the above session, case morphology becomes visible at %2 where S(F(case)) is reset to 0-1. Tense and aspect morphology appears in the input (or rather intake) at %3 and %4 where S(F(tns)) and S(F(asp)) are successively set to 0-1. Up till %5, however, Little French has been treated as a language which has no overt movement. The SVO order found so far is compatible with the structure where every lexical item is in its VP-internal position. The trigger for a different setting came at %6 where *often* appeared. The new string cannot be parsed unless the subject moves to Agr1spec and the verb moves to Agr1-0. Hence the new setting. After that, every string in (193) is acceptable. The language generated by the current setting is exactly Little French. We thus conclude that this language has been correctly identified in the limit.

5.4.6 Acquiring Little German

Little German is a V2 language which contains the following strings:

```
(196) { s-[c1]  aux-[agr,tns]  (often)  iv-[asp],
       s-[c1]  aux-[agr,tns]  (often)  o-[c2]  tv-[asp],
       o-[c2]  aux-[agr,tns]  s-[c1]  (often)  tv-[asp],
       often  aux-[agr,tns]  s-[c1]  o-[c2]  tv-[asp]
     }
```

This is the small subset of German main clauses where the auxiliary is in second position and the verb is clause-final. The following shows how Little German is acquired on-line.

```
| ?- sp.
The initial setting is
[0  0  0  0  0  0  0  0  i  i
 agr(0-0)  asp(0-0)  case(0-0)  pred(0-0)  tns(0-0)  ]
Next? [s-[],aux-[tns],iv-[]].                                    %1
Unable to parse [s-[],aux-[tns],iv-[]]
Resetting the parameters ...

s(f(tns)) is reset to i-0
Word order parameters reset to: [0  0  0  0  0  1  0  0  i  i  ]
Successful parse.
Next? [s-[],aux-[tns],o-[],tv-[]].                               %2
Unable to parse [s-[],aux-[tns],o-[],tv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  0  0  0  0  1  1  0  i  i  ]
Successful parse.
Next? generate.
Language generated with current setting:
[s-[],often,aux-[tns],iv-[]]
[s-[],often,aux-[tns],o-[],tv-[]]
[s-[],aux-[tns],iv-[]]
[s-[],aux-[tns],o-[],tv-[]]

Next? [o-[],aux-[tns],s-[],tv-[]].                               %3
Unable to parse [o-[],aux-[tns],s-[],tv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  0  0  0  0  0  1  1  i  i  ]
```

```
Successful parse.
Next? generate.
Language generated with current setting:
[often,aux-[tns],o-[],s-[],tv-[]]
[often,aux-[tns],s-[],iv-[]]
[o-[],often,aux-[tns],s-[],tv-[]]
[o-[],aux-[tns],s-[],tv-[]]

Next? [s-[c1],aux-[tns],iv-[]].                                    %4
Unable to parse [s-[c1],aux-[tns],iv-[]]
Resetting the parameters ...

s(f(case)) is reset to 0-1
Word order parameters reset to: [1  0  0  0  0  1  0  1  i  i  ]
Successful parse.
Next? generate.
Language generated with current setting:
[often,s-[c1],aux-[tns],iv-[]]
[often,s-[c1],aux-[tns],tv-[],o-[c2]]
[s-[c1],often,aux-[tns],iv-[]]
[s-[c1],often,aux-[tns],tv-[],o-[c2]]
[s-[c1],aux-[tns],iv-[]]
[s-[c1],aux-[tns],tv-[],o-[c2]]

Next? [often,aux-[tns],s-[c1],iv-[]].                              %5
Unable to parse [often,aux-[tns],s-[c1],iv-[]]
Resetting the parameters ...

Word order parameters reset to: [1  0  0  0  0  0  1  1  i  f  ]
Successful parse.
Next? generate.
Language generated with current setting:
[often,aux-[tns],o-[c2],s-[c1],tv-[]]
[often,aux-[tns],s-[c1],iv-[]]
[o-[c2],often,aux-[tns],s-[c1],tv-[]]
[o-[c2],s-[c1],tv-[],aux-[tns]]
```

```
Next? [s-[c1],aux-[tns],iv-[]].                                    %6
Unable to parse [s-[c1],aux-[tns],iv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  1  0  0  0  1  0  1  i  i  ]
Successful parse.
Next? generate.
Language generated with current setting:
[often,s-[c1],aux-[tns],iv-[]]
[often,s-[c1],aux-[tns],tv-[],o-[c2]]
[s-[c1],often,aux-[tns],iv-[]]
[s-[c1],often,aux-[tns],tv-[],o-[c2]]
[s-[c1],aux-[tns],iv-[]]
[s-[c1],aux-[tns],tv-[],o-[c2]]

Next? [often,aux-[tns],s-[c1],iv-[]].                              %7
Unable to parse [often,aux-[tns],s-[c1],iv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  1  0  0  0  0  1  1  i  i  ]
Successful parse.
Next? [o-[c2],aux-[tns],s-[c1],tv-[]].                             %8
Current setting remains unchanged.
Next? [s-[c1],aux-[tns],iv-[]].
Unable to parse [s-[c1],aux-[tns],iv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  0  1  0  0  1  0  1  i  i  ]
Successful parse.
Next? [s-[c1],aux-[agr,tns],iv-[]].                                %9
Unable to parse [s-[c1],aux-[agr,tns],iv-[]]
Resetting the parameters ...

s(f(agr)) is reset to 1-0
Word order parameters reset to: [0  0  0  1  0  1  0  1  i  i  ]
Successful parse.
Next? [often,aux-[agr,tns],s-[c1],iv-[]].                          %10
```

```
Unable to parse [often,aux-[agr,tns],s-[c1],iv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  0  0  1  0  0  1  1  i  i  ]
Successful parse.
Next? [o-[c2],aux-[agr,tns],s-[c1],tv-[]].                    %11
Current setting remains unchanged.
Next? [s-[c1],aux-[agr,tns],o-[c2],tv-[]].
Unable to parse [s-[c1],aux-[agr,tns],o-[c2],tv-[]]
Resetting the parameters ...

Word order parameters reset to: [0  0  0  1  0  1  1  1  i  i  ]
Successful parse.
Next? [o-[c2],aux-[agr,tns],s-[c1],tv-[]].                    %12
Unable to parse [o-[c2],aux-[agr,tns],s-[c1],tv-[]]
Resetting the parameters ...

Word order parameters reset to: [1  0  0  1  1  0  1  1  i  f  ]
Successful parse.
Next? [s-[c1],aux-[agr,tns],o-[c2],tv-[]].                    %13
Unable to parse [s-[c1],aux-[agr,tns],o-[c2],tv-[]]
Resetting the parameters ...

Word order parameters reset to: [1  0  0  1  0  1  1  1  i  i  ]
Successful parse.
Next? generate.
Language generated with current setting:
[often,s-[c1],aux-[agr,tns],iv-[]]
[often,s-[c1],aux-[agr,tns],o-[c2],tv-[]]
[o-[c2],s-[c1],aux-[agr,tns],often,tv-[]]
[o-[c2],s-[c1],aux-[agr,tns],tv-[]]
[s-[c1],aux-[agr,tns],often,iv-[]]
[s-[c1],aux-[agr,tns],often,o-[c2],tv-[]]
[s-[c1],aux-[agr,tns],iv-[]]
[s-[c1],aux-[agr,tns],o-[c2],tv-[]]

Next? [s-[c1],aux-[pred,agr,tns],o-[c2],tv-[]].               %14
```

```
Unable to parse [s-[c1],aux-[pred,agr,tns],o-[c2],tv-[]]
Resetting the parameters ...

s(f(pred)) is reset to 1-0
Word order parameters reset to: [0  0  0  1  1  1  1  1  i  i  ]
Successful parse.
Next? [s-[c1],aux-[pred,agr,tns],o-[c2],tv-[asp]].          %15
Unable to parse [s-[c1],aux-[pred,agr,tns],o-[c2],tv-[asp]]
Resetting the parameters ...

s(f(asp)) is reset to 0-1
Successful parse.
Next? [o-[c2],aux-[pred,agr,tns],s-[c1],tv-[asp]].          %16
Current setting remains unchanged.
Next? [often,aux-[pred,agr,tns],s-[c1],o-[c2],tv-[asp]].%17
Current setting remains unchanged.
Next? current_setting.
[0  0  0  1  1  1  1  1  i  i
 agr(1-0)  asp(0-1)  case(0-1)  pred(1-0)  tns(1-0)  ]
Next? generate.
Language generated with current setting:
[often,aux-[pred,agr,tns],s-[c1],iv-[asp]]
[often,aux-[pred,agr,tns],s-[c1],o-[c2],tv-[asp]]
[o-[c2],aux-[pred,agr,tns],s-[c1],often,tv-[asp]]
[o-[c2],aux-[pred,agr,tns],s-[c1],tv-[asp]]
[s-[c1],aux-[pred,agr,tns],often,iv-[asp]]
[s-[c1],aux-[pred,agr,tns],often,o-[c2],tv-[asp]]
[s-[c1],aux-[pred,agr,tns],iv-[asp]]
[s-[c1],aux-[pred,agr,tns],o-[c2],tv-[asp]]

Next? bye.

yes
| ?-
```

The first input string is not compatible with the initial setting. It became analyzable when S(M(spec1)) is reset to 1 and S(F(tns)) is set to 1-0. The next string (%2) triggered the resetting of S(M(spec2)) to 1. The language generated at this point is an SOV language instead of a V2 language. The next

input string (%3) informed the learner of the error and resetting occurred. The learner is trying to identify the input language as a non-scrambling one at first, but she did not succeed. Resetting continued through %4, %5, %6, and %7. The setting at this point was able to accept a language where the Aux is in second position while the first position can be occupied by an Adverb or an Object. But no subject can appear clause-initially, which shows the setting was still incorrect. Therefore more resetting occurred until after the string at %15 was presented. Meanwhile, the recognition of morphological inflections caused S(F(case)) and S(F(asp)) to be reset to 0-1 whereas S(F(agr)) and S(F(pred)) were reset to 1-0. From this point on, every string in Little German were acceptable. In addition, the language generated with the current setting is not a superset of the target language. The learner has thus converged on the correct grammar for Little German.

5.5 Summary

We have seen in this chapter that the parameter space associated with our experimental grammar has certain desirable learnability properties. The languages generated in this parameter space are not only typologically interesting but learnable as well. There is at least one algorithm whereby the parameters can be correctly set for each language in the parameter space. The precedence rules employed in the learning algorithm, which are crucial for the success of the learnability, are deducible from a general linguistic principle, the Principle of Procrastinate. This makes our learning algorithm linguistically plausible as well as computationally feasible. There are many issues to which our acquisition model is potentially relevant, but they have not been addressed so far. We may want to examine the empirical implications of this model for language development. One of the questions we can ask is how the intermediate settings the learner goes through in our model relate to the developmental stages that children undergo. Our model may also be theoretically interesting to the study of language change. The existence of weakly equivalent languages in our parameter space may provide an explanation for the kind of reanalysis phenomena where the grammar changes without immediately affecting the language generated. Another issue we may want to pursue is how plausible our learning algorithm will be when the input is noisy and what modifications are needed to make the learning procedure more robust. All these issues are yet to be explored and a serious discussion on them requires much more work than has been done in this thesis.

Chapter 6

Parsing with S-Parameters

When discussing the parameter space in Chapter 4 and the parameter setting algorithms in Chapter 5, we have assumed that there is a parser which implements the experimental grammar defined in Chapter 3. In Chapter 4, the parser was used to generate all possible strings of all possible languages in our parameter space. In Chapter 5, it was used by the learner to process incoming sentences or generate all the strings in her language.[1] So far the parser itself has not received any discussion. It is the goal of this chapter to describe this parser.

The parser to be discussed is supposed to be an implementation of our version of UG, namely, the experimental grammar we have assumed. It is not strictly speaking an axiomatic representation of the grammar, but it is equivalent to the grammar in that it accepts or rejects exactly the same set of sentences/languages as the grammar does. The parser is *universal* in the sense that it is capable of processing any language in the parameter space. In other words, the parsing procedures can be applied in a uniform fashion no matter what the grammar of the individual language is. The only thing that can change from language to language is the parameter setting. This should be no surprise because the parser is in fact the embodiment of UG which does not vary except for the parameter values.

6.1 Distinguishing Characteristics of the Parser

Our discussion of the parser will not be concerned with general issues of sentence processing which everyone who builds a parser has to address. For instance, everyone who builds a top-down parser has to deal with the problem of left-recursion and everyone who builds a bottom-up parser has to face the problem of empty categories. These issues have been discussed extensively in the literature and whatever applies to other parsers applies to our parser as well. What we are interested in are those properties which are absent in other parsers.

[1] The Prolog programs `pspace.pl` in Appendix A.1, `sp.pl` in Appendix A.4 and `sp1.pl` in Appendix A.6 cannot run without this parser.

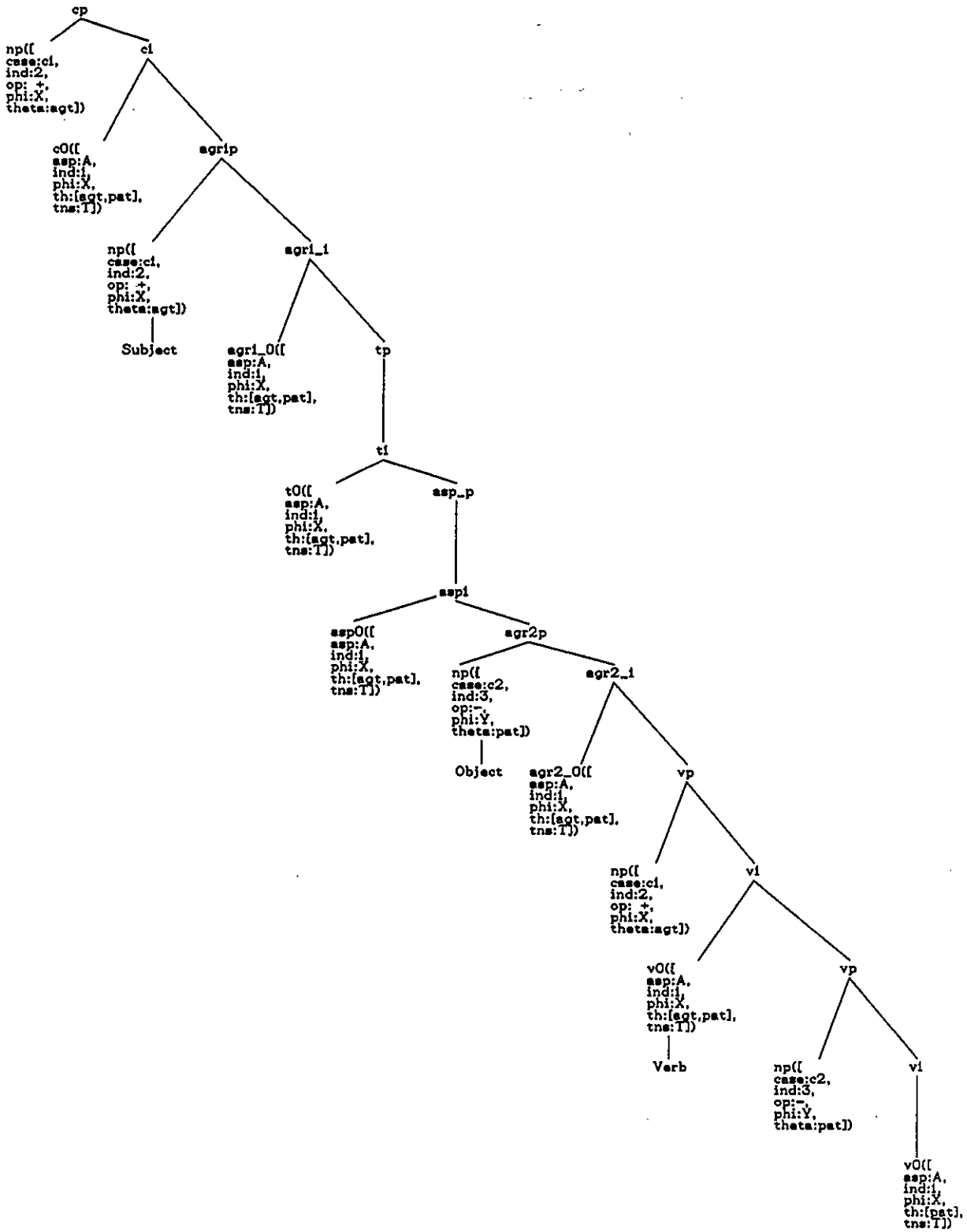
The reason why the parser we build here can be different from all other parsers is that the grammar to be implemented is different. The most salient feature which distinguishes our grammar from all other versions of UG is the uniform treatment of movement. In traditional grammars, most movements are S-structure movements. The movements that a parser deals with are only those movements which are overt. The parse trees built by the parser are either S-structure representations or combinations of S-structure and D-structure. In the latter case, certain nodes in the parse tree form *chains*. The constituent that moves is found at the *head* of the chain and the *tail* or *foot* of the chain consists of a trace. A chain is therefore a simultaneous representation of both the S-structure and D-structure positions of a constituent. Since S-structure movements can vary from language to language, the chains to be built by the parser can vary according to which language is being parsed. Consequently, the parse trees built for different languages can vary in shape depending on what S-structure movements occur in those languages. This makes it necessary to "tune" the parser for different languages. Suppose a wh-question is being parsed and we have come to the point where the Cspec node is constructed. If the language being parsed is English, then the parser must find a lexical item in this position (namely a wh-word) and start building a chain from this point. If the language being parsed is Chinese, however, the parser must *not* find a wh-word in this position and no chain will be built. These specific, language-particular instructions must be built into the parsing procedure. As a result, we may need an English parser to parse English and a Chinese parser to parse Chinese. These two parsers may be very similar on the whole, but the Chinese parser cannot be used to parse English without some procedural fine tuning.

Things are different in our present model. We have assumed that there is an underlying set of movements which occur in every language. All those movements are LF movements in the sense that they are all necessary for the successful derivation of the LF representation. The derivation will crash if any of those movements fails to occur. This implies that the parser in our model must build an LF representation in order to find out if a sentence is grammatical. Looking at LF, we see that all languages are identical at this level of representation in that the movement chains found there are the same cross-linguistically. If a sentence is grammatical at all, then its LF representation must have those chains, no matter what language this sentence is from. This means that the LF representation can be built in a universal way as far as chain formation is concerned. The parser can go ahead and construct an invariant set of chains regardless of what language is being parsed. If the only representation we need were LF, then we could have a truly universal parser which needs no fine tuning when used to process different languages.

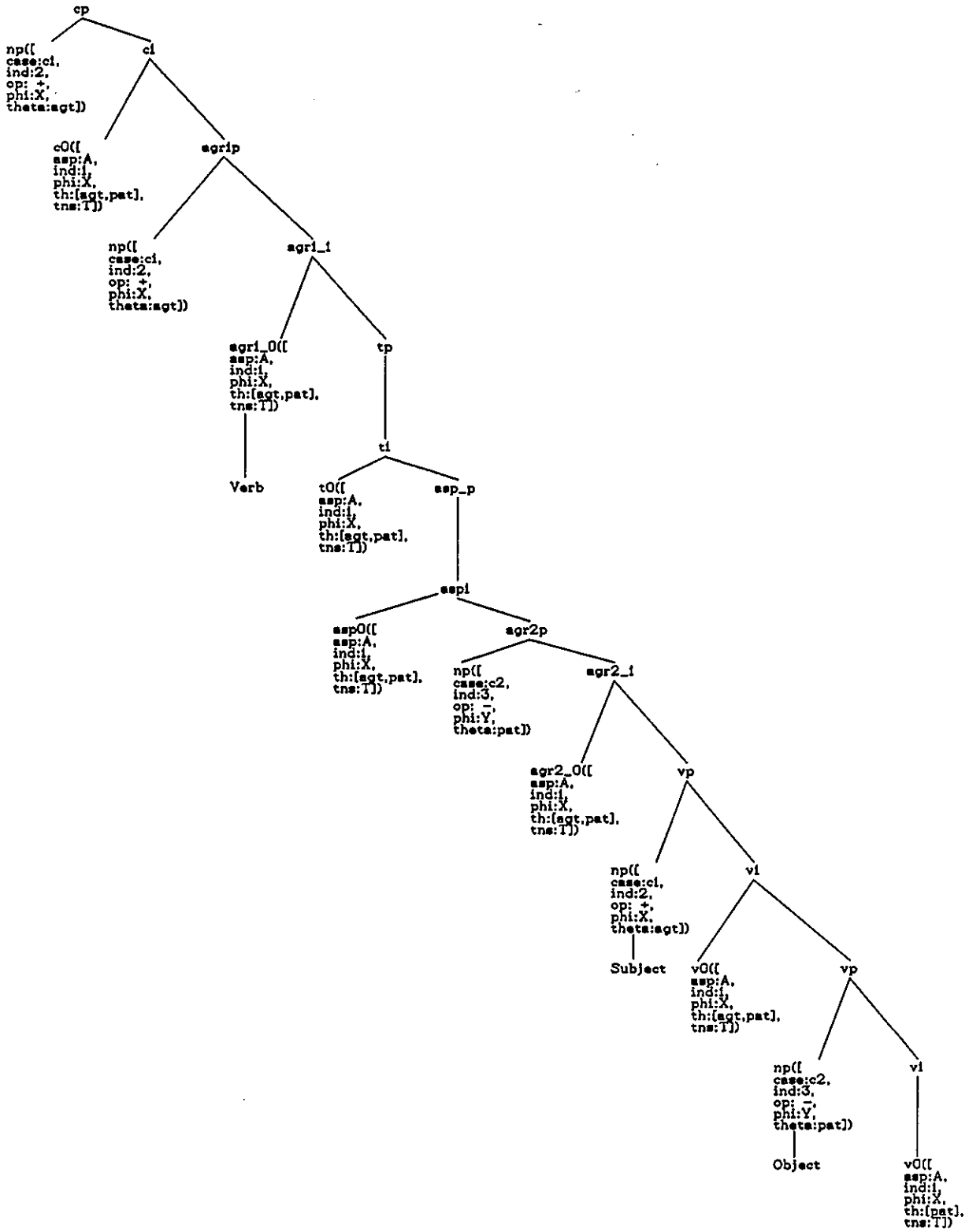
LF is of course not the only structure the parser has to build. When we talk about chains, at least two levels of structures must be involved, one represented by the heads of the chains and one by the tails. In terms of a chain resulting from LF movement, the head is the LF position of a constituent and the tail is the base position. The base position is the position where the constituent is generated through lexical projection. All the "content words" in our system are base-generated

VP-internally. It is roughly the D-structure (DS) position in traditional terminology. Since every LF movement involves moving a constituent from its base position to the LF position, the chain formed by this movement is a simultaneous representation of both positions. When we say that the chains are universal, we in fact mean that both the LF positions and base positions are invariant. By saying that the chains can be built uniformly, we have actually concluded that the LF and "DS" representations of all languages can be constructed through a uniform parsing procedure. In other words, no fine tuning in the parser will be necessary for parsing different languages if LF and DS information alone needs to be represented in the parse tree.

However, LF and DS are not the only representations at which the grammaticality of a sentence can be determined. We also have to look at the representation that is spelled out. In traditional terminology, this representation is called S-structure (SS). We can borrow this term and use it to refer to the structure which is fed into the PF component. The "S" thus stands for "Spell Out" rather than "surface". This is the structure which is subject to cross-linguistic variation. The source of variation in our model is the values of S(M)-parameters which determine which subset of LF movements must be performed before Spell-Out in a given language. The constituent involved in an LF movement appears at the head position of the chain at S-structure if it moves before Spell-Out. It appears at the tail position if it moves after Spell-Out. To find out if a sentence is grammatical in a given language, we have to take the S-structure positions of every constituent into consideration. In other words, the parse tree we build must represent SS in addition to LF and DS. This is not difficult to do. LF and DS position can as usual be represented by the chain, with its head invariably marking the LF position and the tail the DS position. The SS position can be indicated by the position of the moving constituent. It must be at the head position if it moves before Spell-Out and it must be at the tail position if it moves after Spell-Out. We can thus represent LF, DS and SS in a single parse tree. Examples of such parse trees are given in (197) and (198).


cp
np([case:c1, ind:2, op: +, phi:X, theta:agt])
c1
c0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
agrip
np([case:c1, ind:2, op: +, phi:X, theta:agt])
Subject
agr1_1
agr1_0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
tp
t1
t0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
asp_p
asp1
asp0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
agr2p
np([case:c2, ind:3, op:-, phi:Y, theta:pat])
Object
agr2_1
agr2_0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
vp
np([case:c1, ind:2, op: +, phi:X, theta:agt])
v1
v0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
Verb
vp
np([case:c2, ind:3, op:-, phi:Y, theta:pat])
v1
v0([asp:A, ind:i, phi:X, th:[pat], tns:T])

(197)


cp
np([case:ci, ind:2, op: +, phi:X, theta:agt])
ci
c0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
agrip
np([case:ci, ind:2, op: +, phi:X, theta:agt])
agri_i
agri_0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
tp
Verb
ti
t0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
asp_p
aspi
asp0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
agr2p
np([case:c2, ind:3, op: -, phi:Y, theta:pat])
agr2_i
agr2_0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
vp
np([case:ci, ind:2, op: +, phi:X, theta:agt])
Subject
vi
v0([asp:A, ind:i, phi:X, th:[agt,pat], tns:T])
vp
np([case:c2, ind:3, op: -, phi:Y, theta:pat])
Object
vi
v0([asp:A, ind:i, phi:X, th:[pat], tns:T])

(198)

The chains in those parse trees are represented through coindexing and feature-sharing. There are three chains in each of the trees.

(i) A V-chain linking the nodes C0, Agr1-0, T0, Asp0, Agr2-0 and V0. All those nodes are assigned the same index: "1". They also share the following feature values: phi:X, tns:T, asp:A, and th:[agt,pat]. This chain is formed by successive head movements from V0 to C0. The variables in the chain, X, T and A, can be instantiated to things like 3sm (3rd person singular masculine), pres (present tense), prog (progressive aspect) in an actual sentence.

(ii) An NP chain linking the nodes Cspec, Agr1spec and the first Vspec. These nodes are coindexed "2" and they share the following feature values: case:1, op:+, phi:X, and theta:agt. This chain is formed by two successive movements of the subject NP: an A-movement from Vspec to Agr1spec and an $\bar{A}$-movement from Agr1spec to Cspec. The fact that both the "phi" feature in this NP chain and the "phi" feature in the V-chain have the value "X" indicates that the subject and the verb must agree with each other. The "+" value of the "op" feature in this NP chain shows that this NP can be the topic or focus of the sentence, or a QP which receives a wide-scope reading.

(iii) Another NP chain linking the nodes Agr2spec and the second Vspec. The index for these nodes is "3" and the feature values that are shared between them are case:2, op:-, phi:Y, and theta:pat. This chain is formed by movement of the object NP from Vspec to Agr2spec.

As far as these chains are concerned, (197) and (198) are identical. This should be the case because these chains are formed by LF movements which are universal.[2] What differentiates (197) and (198) are the positions of the lexical items in these chains. The lexical items, which are actually pronounced, are represented by "Subject", "Object" and "Verb" in those trees. In a real sentence they will be real words like *he*, *him* and *likes*. In (197), Subject is in Agr1spec, Object in Agr1spec, and Verb in V0. In (198), Verb is in C0 and Subject and Object are in their VP-internal positions. These positions tell us where Subject, Object and Verb are at the time of Spell-Out. In (197), Subject and Object have moved overtly to their case positions and the surface word order is SOV. In (198), Verb has overtly moved to C0 and the surface order is VSO. We get the tree in (197) when the S(M)-parameters are set to [0, 0, 0, 0, 0, 1, 1, 0] and we get (198) when they are set to [1, 1, 1, 1, 0, 0, 0, 0]. It is clear that three levels of representation – LF, DS and SS – are merged into one in those parse trees. The LF and DS positions are identical in the two trees but the SS positions are different.

In building trees of this kind, the parser is constructing three levels of representations at the same time. There are three things that the parser must do. It must (i) build the tree; (ii) build the

[2]Both (197) and (198) have an alternative representation where Cspec is coindexed with the object NP which will then have the "+" value for the operator feature. This is possible when the object is understood to be the topic, focus, or wide-scope QP of the sentence.

chains; and (iii) decide where the lexical item appears in each chain.

Chain-building is universal, as we have seen. The parsing procedures which are responsible for chain-formation can therefore be invariant. It can simply be hard-wired into the parser, since it does not respond to language variation at all. So this part of the parsing mechanism can be totally innate. No learning or fine tuning is necessary.

Tree-building is universal in that the same set of nodes are built in a given type of sentence no matter what the language is. This has been illustrated in (197) and (198) which represent different languages but have the same nodes. Furthermore the same set of dominance relations holds between those nodes in every language. Agr1P is always dominated by C1, for example. However, linear precedence may vary according to the values of HD-parameters. When building CP and IP, the parser must be able to respond in two different ways. It has to build the phrase head-initially if the HD-parameter is set to "i" and build the phrase head-finally if the parameter is set to "f". These alternative actions can again be built-in. We can suppose that the parser is innately able to build a phrase in either way. In parsing a particular language, it has to receive instructions from the HD-parameters in order to decide which action to take. But such choices are made by the grammar rather than the parser. The parser does not have to adapt itself in this respect when parsing different languages. Once the parameters are set in a language, the parse will respond accordingly. We can thus conclude that no learning or fine tuning on the part of the parser is necessary as long as the parser is innately able to build a phrase in either way, head-initial or head-final.

The task of deciding where the lexical item appears in each chain consists of the following computation. The parser must determine for each terminal node in the tree whether this node should dominate a lexical item. In addition, it must make sure that only one node in each chain dominates a lexical item. Can these decisions again be made universally? In other words, is the parser able to handle all languages without learning or fine-tuning? The answer appears to be negative at first sight. Apparently, the action the parser takes at a given terminal node can vary cross-linguistically. In some languages it must find a lexical item to be dominated by this terminal node while in some other languages it must not find one. It seems that we may need to specify the parser action at each terminal for each different language. This does not have to be the case, however. In our model, where the lexical item appears in a chain is determined by the values of S(M)-parameters. These parameters decide how far each lexical item moves before Spell-Out. A terminal node must dominate a lexical item if this lexical item has moved exactly to that position at Spell-Out. The node should dominate nothing (i.e. empty) if the lexical item in that chain has either moved through this node to a higher position or has not moved to this node by the time of Spell-Out. Take T0 as an example. It must dominate a verb if S(M(tns)), S(M(asp)) and S(M(agr2)) are set to 1 while S(M(agr1)) is set to 0. In this case, the verb will overtly move to T0 but no further. T0 must be empty in either of the following situations: (a) S(M(tns)), S(M(asp)) and S(M(agr2)) are set to 1 and S(M(agr1)) is set to 1 as well (the verb moves through T0 to a higher position); (b)

S(M(tns)) is set to 0 (the verb does not move to T0 before Spell-Out). We assume that, whenever a terminal node is built, the parser is able to consult the S(M)-parameter values and decide whether the node should be empty or not. We can further assume that such decision-making capability of the parser is innate. There can be a built-in mechanism which can check the parameter values and take the right actions accordingly. If this is true, no learning or fine tuning is needed in this part of the parsing procedure, either. What has to be learned or tuned is the parameter setting which exists independently of the parser. Once the parameters are set for a given language, the parser will be able to process that language.

It turns out that, given a certain value combination of S(M)-parameters, the action that the parser must take at any given terminal node is unique. The following table shows the S(M)-parameter conditions under which a terminal node is to dominate a content word (V or NP).

(199)

V in C0	S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(1)))&S(M(agr2(1)))
V in Agr1-0	S(M(c(0)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(1)))&S(M(agr2(1)))
V in T0	S(M(agr1(0)))&S(M(tns(1)))&S(M(asp(1)))&S(M(agr2(1)))
V in Asp0	S(M(tns(0)))&S(M(asp(1)))&S(M(agr2(1)))
V in Agr2-0	S(M(asp(0)))&S(M(agr2(1)))
V in V0	S(M(agr2(0)))
NP in Cspec	S(M(cspec(1)))& (S(M(spec1(1))) or S(M(spec2(1))))
NP in Agr1spec	S(M(spec1(1)))
NP in Agr2spec	S(M(spec2(1)))
NP in Vspec1	S(M(spec1(0)))
NP in Vspec2	S(M(spec2(0)))

Notice that the conditions for an NP appearing Agr1spec or Agr2spec are necessary but not sufficient conditions. S(M(spec1(1))) is necessary for Agr1spec to dominate an lexical NP, but it is not sufficient. The NP may move further up to Cspec if S(M(cspec(1))) holds. However, we cannot state the condition as S(M(spec1(1)))&S(M(cspec(0))) because that would be too limiting. The subject NP may stay in Agr1spec with S(M(cspec)) set to 1 if the object NP moves to Cspec instead. By putting in the necessary conditions only, some kind of non-determinism is allowed for. But the non-determinism is local. In any given context only one choice can be correct and the wrong choice will eventually be ruled out. Let us take Agr1spec again as an example. Suppose both S(M(spec1)) and S(M(cspec)) are set to 1. Upon seeing S(M(spec2(1))), the parser may decide to put the subject NP under Agr1spec. This would be the correct choice if the object NP has moved to Cspec. If the object is not there, however, Cspec will be empty which is contradictory to S(M(cspec(1))). The parser will realize that a mistake has been made and it will try the other choice – putting the subject in Cspec and leaving Agr1spec empty – which is also permitted by the current parameter setting.

It should also be noted that the V0 in (199) is the head of the top layer of VP. The lower V0(s) are always empty. The assumption is that the verb always moves through all layers of VP to the top one before Spell-Out. There is no variation there.

In addition to deciding whether a terminal should contain a content word or not, the parser

also has to check, in cases where the terminal is not empty, what features are overtly expressed in the word. An NP may be overtly marked for case or not; a verb can be overtly marked for the predication, agreement, tense or aspect feature, or any combination of them. A sentence is accepted only if the morphological patterns are correct as well. These patterns are determined by the values of S(F)-parameters. An NP must be overtly marked for case if S(F(case)) is set to X-1; otherwise it must be set to X-0. A verb can be overtly marked for predication, agreement, tense or aspect if and only if S(F(pred)), S(F(agr)), S(F(tns)) or S(F(asp)) is set to 1; it must have no overt inflectional morphology if every S(F)-parameter is set to 0. The complete set of possible spell-out of S (Subject NP), O (object NP), and V (verb) and the S(F)-parameter values which are necessary and sufficient conditions for the spell-out are given in (200). As usual, we will indicate the overtness of a feature by placing it in the feature list of every symbol. For conciseness, we will use "v" as a cover term for both "iv" and "tv".

(200)

S	s-[]	S(F(case(X-0)))
	s-[c1]	S(F(case(X-1)))
O	o-[]	S(F(case(X-0)))
	o-[c2]	S(F(case(X-1)))
V	v-[]	S(F(pred(X-0)))&S(F(agr(X-0)))&S(F(tns(X-0)))&S(F(asp(X-0)))
	v-[pred]	S(F(pred(X-1)))&S(F(agr(X-0)))&S(F(tns(X-0)))&S(F(asp(X-0)))
	v-[agr]	S(F(pred(X-0)))&S(F(agr(X-1)))&S(F(tns(X-0)))&S(F(asp(X-0)))
	v-[tns]	S(F(pred(X-0)))&S(F(agr(X-0)))&S(F(tns(X-1)))&S(F(asp(X-0)))
	v-[asp]	S(F(pred(X-0)))&S(F(agr(X-0)))&S(F(tns(X-0)))&S(F(asp(X-1)))
	v-[pred,agr]	S(F(pred(X-1)))&S(F(agr(X-1)))&S(F(tns(X-0)))&S(F(asp(X-0)))
	v-[pred,tns]	S(F(pred(X-1)))&S(F(agr(X-0)))&S(F(tns(X-1)))&S(F(asp(X-0)))
	v-[pred,asp]	S(F(pred(X-1)))&S(F(agr(X-0)))&S(F(tns(X-0)))&S(F(asp(X-1)))
	v-[agr,tns]	S(F(pred(X-0)))&S(F(agr(X-1)))&S(F(tns(X-1)))&S(F(asp(X-0)))
	v-[agr,asp]	S(F(pred(X-0)))&S(F(agr(X-1)))&S(F(tns(X-0)))&S(F(asp(X-1)))
	v-[tns,asp]	S(F(pred(X-0)))&S(F(agr(X-0)))&S(F(tns(X-1)))&S(F(asp(X-1)))
	v-[pred,agr,tns]	S(F(pred(X-1)))&S(F(agr(X-1)))&S(F(tns(X-1)))&S(F(asp(X-0)))
	v-[pred,agr,asp]	S(F(pred(X-1)))&S(F(agr(X-1)))&S(F(tns(X-0)))&S(F(asp(X-1)))
	v-[pred,tns,asp]	S(F(pred(X-1)))&S(F(agr(X-0)))&S(F(tns(X-1)))&S(F(asp(X-1)))
	v-[agr,tns,asp]	S(F(pred(X-0)))&S(F(agr(X-1)))&S(F(tns(X-1)))&S(F(asp(X-1)))
	v-[pred,agr,tns,asp]	S(F(pred(X-1)))&S(F(agr(X-1)))&S(F(tns(X-1)))&S(F(asp(X-1)))

So far we have limited the lexical items that can appear in the tree to content words (verbs and NPs) only. There is another kind of lexical items that can be dominated by terminal nodes: auxiliaries/grammatical particles which we have represented as Aux. Whether a terminal node can dominate an Aux has to be determined by both S(M)-parameters and S(F)-parameters. Take T0 as an example. There are three situations where T0 can dominate an Aux.

(i) When S(M(tns(0)))&S(F(tns(1-X))&S(M(agr1(0))) is true. This condition requires that (a) no constituent move to T0, (b) the feature of T0 be spelled out as an Aux, and (c) this Aux not move up. In this case, the Aux appears as "aux-[tns]".

(ii) When S(M(asp(0)))&(F(asp(1-X)))&S(M(tns(1)))&S(M(agr1(0)))&S(F(tns(0-X))) is true. This condition requires that (a) no constituent move to Asp0, (b) the feature of Asp0 be spelled out as an Aux, (c) this Aux move to T0, (d) it move no furhter up after moving to T0, and (e) the feature of T0 not be spelled out. In this case, the Aux appears as "aux-[asp]".

(iii) When S(M(asp(0)))&(F(asp(1-X)))&S(M(tns(1)))&S(M(agr1(0)))&S(F(tns(1-X))) is true. This condition requires that (a) no constituent move to Asp0, (b) the feature of Asp0 be spelled out as an Aux, (c) this Aux move to T0, (d) it move no furhter up after moving to T0, and (e) the feature of T0 be spelled out as well. In this case, the Aux appears as "aux-[tns,asp]".

We assume that an auxiliary can be overtly inflected for a certain feature only if it has originated from or moved through the position in which this feature is found. An Aux can have "tns" in its feature list only if it is in T0, or has moved from or through T0. In order for an Aux to be spelled out as aux-[pred,agr,tns,asp], for instance, two conditions must be met. First, the Aux must have originated from Asp0 and moved all the way up to C0; second, S(F(pred)), S(F(agr)), S(F(tns)) and S(F(asp)) must all be set to 1-X. In short, the spell-out of Aux has to be determined by the values of both S(F)- and S(F)- parameters. The following table lists all the possible spell-out of Aux and the necessary and sufficient conditions for each possibility. Since an Aux can have a different set of spell-out possibilities in each different position, we have to consider them one by one.[3]

[3] Agr2-0 is being ignored here because we are restricting ourselves to situations where there is no object-verb agreement. We can easily extend it to object-verb agreement but we choose not to do so for the sake of avoiding unnecessary complications in our exposition.

(201)

Aux in C0	aux-[pred]	S(F(pred(1-X))) & S(M(c(0)))
	aux-[agr]	S(F(pred(0-X)))&S(F(agr(1-X))) & S(M(c(1)))&S(M(agr1(0)))
	aux-[tns]	S(F(pred(0-X)))&S(F(agr(0-X)))&S(F(tns(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[asp]	S(F(pred(0-X)))&S(F(agr(0-X)))&S(F(tns(0-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[pred,agr]	S(F(pred(1-X)))&S(F(agr(1-X)))& S(M(c(1)))&S(M(agr1(0)))
	aux-[pred,tns]	S(F(pred(1-X)))&S(F(agr(0-X)))&S(F(tns(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[pred,asp]	S(F(pred(1-X)))&S(F(agr(0-X)))&S(F(tns(0-X)))&S(F(asp(1-X))) S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[agr,tns]	S(F(pred(0-X)))&S(F(agr(1-X)))&S(F(tns(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[agr,asp]	S(F(pred(0-X)))&S(F(agr(1-X)))&S(F(tns(0-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[tns,asp]	S(F(pred(0-X)))&S(F(agr(0-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[pred,agr,tns]	S(F(pred(1-X)))&S(F(agr(1-X)))&S(F(tns(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[pred,agr,asp]	S(F(pred(1-X)))&S(F(agr(1-X)))&S(F(tns(0-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[pred,tns,asp]	S(F(pred(1-X)))&S(F(agr(0-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[agr,tns,asp]	S(F(pred(0-X)))&S(F(agr(1-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[pred,agr,tns,asp]	S(F(pred(1-X)))&S(F(agr(1-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(1)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
Aux in Agr1-0	aux-[agr]	S(F(agr(1-X)))&S(M(c(0)))&s(M(agr1(0)))
	aux-[tns]	S(F(agr(0-X)))&S(F(tns(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[asp]	S(F(agr(0-X)))&S(F(tns(0-X)))&S(F(asp(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[agr,tns]	S(F(agr(1-X)))&S(F(tns(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(0)))
	aux-[agr,asp]	S(F(agr(1-X)))&S(F(tns(0-X)))&S(F(asp(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[tns,asp]	S(F(agr(0-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[agr,tns,asp]	S(F(agr(1-X)))&S(F(tns(1-X)))&S(F(asp(1-X)))& S(M(c(0)))&S(M(agr1(1)))&S(M(tns(1)))&S(M(asp(0)))
Aux in T0	aux-[tns]	S(F(tns(1-X)))&S(M(agr1(0)))&S(M(tns(0)))
	aux-[asp]	S(F(tns(0-X)))&S(F(asp(1-X))) S(M(agr1(0)))&S(M(tns(1)))&S(M(asp(0)))
	aux-[tns,asp]	S(F(tns(1-X)))&S(F(asp(1-X))) S(M(agr1(0)))&S(M(tns(1)))&S(M(asp(0)))
Aux in Asp0	aux-[asp]	S(F(asp(1-X)))&S(M(tns(0)))&S(M(asp(0)))

Note that Aux can never have an empty list attached to it. This is so because Aux is the spell-out of some F-feature(s) and there can be no Aux if no feature is spelled out.

Using the decision tables in (199), (200) and (201), the parser can uniquely determine the status of each terminal. If the condition in (199) holds, the terminal must contain a content word (a verb or a lexical NP). It then uses (200) to determine the inflectional pattern of this word. If the condition in (201) holds, the terminal must dominate an Aux of a particular morphological make-up. Notice that there is no overlapping between the conditions in (199) and (201). If neither the conditions in (199) nor those in (201) are met, the terminal must be empty i.e. dominating no lexical item.

Now we sum up the characteristics of the parser in our model. Like all other principle-based parsers, the present parser has to do at least three things: it has to build a tree, it has to build chains, and it has to decide which terminals are empty and which terminals contain lexical items. What distinguishes this parser from all others is the degree of universality found in the parsing procedures. Our parser can be used to analyze any given language without a single change in its structure. All the cross-linguistic variations are encoded in the parameter values rather than the parser itself. The parser does not vary, but it is able to respond to different parameter settings and act accordingly. As we have seen, chain-building is universal and tree-building is universal aside from the limited variation in head direction. Most of the language-particular decisions are made at the terminal nodes and these decisions can always be made correctly by consulting the values of S-parameters. In the next section, we will look at a Prolog implementation of this parser. This will clarify the discussion in this section and add some concreteness to our understanding of the parsing algorithm.

6.2 A Prolog Implementation

In this section we look at a particular implementation of the parser described above. It is a top-down parser implemented in the DCG (definite clause grammar) format. The choice of presenting a top-down version of the parser in this thesis is motivated by the consideration that this seems to be simplest way to describe the underlying logic of the parser. It is not theoretically superior, nor is it the most efficient parser that can be built in the present model. We choose it in order to make the main characteristics of the parser clear without getting into the complications of parsing strategies. Once the logic is clear, we can turn it into any other kind of parser.

The top-down parser to be discussed is presented in Appendix A.7. We shall examine it by looking at the following sub-processes one by one: (a) tree-building; (b) feature-checking; and (c) leaf-attachment.

6.2.1 Tree-Building

The tree is built by `cp/3`, `c1/4`, `agr1p/5`, `agr1_1/4`, `tp/6`, `t1/6`, `asp_p/6`, `asp1/6`, `agr2p/6`, `agr2_1/5`, `vp/5` and `v1/5`.[4] These predicates implement the following phrase structure rules which are equivalent to the rules in (??) presented as part of our experimental grammar in Chapter 3.
(202)

$$
\begin{array}{ll}
CP \rightarrow XP\ \ C^1 & (i) \\
C^1 \rightarrow \{\ C^0,\ Agr1P\ \} & (ii) \\
Agr1P \rightarrow NP(1)\ \ Agr1^1 & (iii) \\
Agr1^1 \rightarrow \{Agr1^0,\ TP\ \} & (iv) \\
TP \rightarrow T^1 & (v) \\
T^1 \rightarrow often\ \ T^1 & (vi) \\
T^1 \rightarrow \{\ T^0,\ AspP\ \} & (vii) \\
AspP \rightarrow Asp^1 & (viii) \\
Asp^1 \rightarrow \{\ Asp^0,\ Agr2P\ \} & (ix) \\
Agr2P \rightarrow NP(2)\ \ Agr2^1 & (x) \\
Agr2P \rightarrow Agr2^1 & (xi) \\
Agr2^1 \rightarrow \{\ Agr2^0,\ VP(1)\ \} & (xii) \\
VP(1) \rightarrow NP(1)\ \ V^1(1) & (xii) \\
V^1(1) \rightarrow V^0\ \ VP(2) & (xiii) \\
V^1(1) \rightarrow V^0 & (xiv) \\
VP(2) \rightarrow NP(2)\ \ V^1(2) & (xv) \\
V^1(2) \rightarrow V^0 & (xvi)
\end{array}
$$

The XP in (i) can be an NP (subject or object) or an AdvP such as *often*. The first clause of `c/3` takes care of the case where Cspec is occupied by an NP. The second clause lets Cspec contain an AdvP whose instantiation is limited to *often* in the present system.

Some of the rules have their right-hand side enclosed in curly brackets. The symbols in these brackets are unspecified for linear order. Which symbol precedes the other depends on the values of HD-parameters. This is why `c1/4`, `agr1_1/4`, `t1/6`, `asp1/6` and `agr2_1/5` each have at least two clauses, one for the value "i" (head-initial) and one for "f" (head-final).

Two different NPs are distinguished in these rules: $NP(1)$ and $NP(2)$. $NP(1)$ is the subject NP which is assigned Case1 and $NP(2)$ is the object NP which has Case2. This distinction is implemented by the checking requirement `case(NF)===c1` or `case(NF)===c2` in `agr1p/5`, `agr2/6` and `vp/6`. (The Variables `NF`, `CF`, `Agr1F`, `Agr2F`, `TF`, `AspF`, `VF`, and `AdF` in the program represent the feature matrices of C, Agr1, Agr2, T, Asp, V and Adv respectively.)

Differentiation is also made between two VPs: $VP(1)$ and $VP(2)$. $VP(1)$ is the top layer of VP whose specifier is the subject NP. This layer of VP is always present in the structure, since every sentence must have at least one argument. $VP(2)$ only appears in transitive sentences and its specifier is the object NP. As we are restricting ourselves to sentences with no more than two

[4] The predicates are again referred to by the notation X/Y where X is the predicate name and Y is the number of arguments in the predicate. Notice that the two arguments representing difference lists are invisible due to the DCG format used here.

arguments for the time being, $VP(2)$ will be the bottom layer of VP. Hence the rules in (xv) and (xvi) which close the whole VP. $VP(1)$ may or may not also be the bottom layer, depending on whether the verb is transitive or not. This is why there are two different expansions of $V^1(1)$ ((xiii) and (xiv)), one taking a $VP(2)$ complement and one closing the VP. In the Prolog implementation, the distinction between $VP(1)$ and $VP(2)$ is made by theta-grid checking. The first clause of `vp/5`, corresponding to (xii), requires `[agt|Ths]` which shows that the theta-role assigned in this layer of VP is the agent role. The second clause corresponds to (xv). It requires `[pat]` which shows that the agent role has already been assigned in the layer above.

We notice that there are two alternative ways of expanding *Agr2P*: (x) and (xi). The rule (x) applies when the sentence is transitive and (xi) applies when it is intransitive. The distinction is again made by theta-grid checking. The specifier of *Agr2P* is generated only if the theta-grid contains two theta roles.

The rules in (vi) and (vii) permit zero or more *often* to be attached to T^1. Thus T^1 can be expanded recursively, allowing an infinite number of *often* to be adjoined to T^1. In our implementation, however, the recursion is interrupted so that at most one *often* can be generated. (This is done by adding a dummy argument to `t1/6` after *often* is attached.) We have to do this because the parser also functions as a generator in the system. The generator is used to generate all possible strings in a language. By limiting the recursion to Depth 1, we can prevent the generation from being infinite.

We find that the second clause of `cp/3`, the third clause of `agr1p/5` , and the third, fourth and fifth clauses of `t1/6` are commented out by "%" in the Prolog program. These clauses are needed only if we want *often* to appear in the strings.

6.2.2 Feature-Checking

There are two types of feature checking in the parsing process. One shows up as Spec-head agreement and the other involves movement. Both types of feature-checking are performed in the tree-building process. Spec-head agreement is checked whenever a specifier is built, and chains are formed as the tree grows. As we can see in the program, all the feature-checking operations are built into the phrasal expansions.

Let us look at spec-head agreement first. The tree we build has five Spec positions: Cspec, Agr1spec, Agr2spec, Vspec1 and Vspec2. The features checked in these five positions are different. The constituent in Cspec is assigned the "+" value of the operator feature. The assignment is done in `cp/3` by "`op(XF)==='+'`". The features checked through spec-head agreement in Agr1 are case and agreement features. The specifier of Agr1 is assigned Case1 (`case(NF)===c1` in `agr1p/5`). In addition, the NP in Agr1spec must agree with Agr1 in the values of phi-features (`phi(Agr1F)===phi(NF)` in `agr_1/6`). Similar checking is done in Agr2 where the NP in Agr2spec is assigned Case2.[5] The

[5] Verb-object agreement is being ignored in this program. This is why we do not find `phi(Agr2F)===phi(NF)` in

spec-head agreement in VPs are responsible for theta-role assignment. The NP in Vspec1 is assigned the first role in the theta grid (which is always the agent role in our restricted system) and the NP in Vspec2 is assigned the second theta-role (patient in our case). (See `theta(NF)===agt` and `theta(NF)===pat` in `vp/5`). All the feature checking operations are performed by "`===/2`" which is Johnson's (??ref) unification algorithm implemented by Ed Stabler. This algorithm (`johnson.pl`) is given in Appendix A.7 as well.

The chains resulting from movement are formed through feature-passing and feature-checking. Three types of chains are built in the parsing process: X0-chains, A-chains and Ā-chains. Each of the three is represented by a separate argument in the predicates. They are named `HC` (Head Chain), `AC` (A chain) and `ABC` (A-Bar Chain) respectively when appearing as variables. `HC` is the second argument (if any) in each predicate. Its content is `x0(HF,Th)` where "`HF`" is the feature matrix being passed on in the chain and "`Th`" the theta-grid. `AC` is the argument following `HC` (if any). It is a list because there can be more than one A-chains being formed at the same time. Each member of this list is an "`np(NF)`" where "`NF`" consists of the NP features of the chain. The argument following `AC` is `ABC` (if any) which is represented as a list not because there is more than one Ā-chain but because we want to distinguish between empty and non-empty lists. In our system, `ABC` may contain an NP (NP1 or NP2) or an AdvP depending on which constituent has moved to Cspec. A chain starts when the head of that chain is encountered and terminates when the tail of that chain is reached.

Since a verb moves successively from V0 to Agr2-0, Asp0, T0, Agr1-0 and finally to C0, the head of the X0-chain (`HC`) is C0 and the tail is V0. The formation of the chain starts in `c1/4` where an extra argument is created to hold the chain. It goes through `agr1p/5`, `agr1_1/4`, `tp/6`, `t1/6`, `asp_p/6`, `asp1/6`, `agr2p/6`, `agr2_1/5`, `vp/5` and ends in `v1/5` where the chain terminates. The features of this chain are checked at each link. The checking is done through `check_v_features/2` at `c1/4`, `agr1_1/4`, `t1/6`, `asp1/6`, `agr2_1/5` and `v1/5`. A new head is found in each of these steps and the features of the new head is unified with those of the chain. The features that are checked in the present program are index, tense, aspect and phi-features.

The A-chains are created by NP-movement. In a transitive sentence, there are two A-chains, one for the subject and one for the object. The subject A-chain has its head in Agr1spec and its tail in Vspec1. The object chain starts in Agr2spec and ends in Vspec2. By the time we come to VP(1), `AC` contains two NPs. The subject NP, whose case feature is instantiated to c1, is selected and unified with the NP in Vspec1. The other NP is passed on until it is unified with the NP in Vspec2. The unification is performd by `check_np_features/2` which checks the index, operator, theta, case and phi-features.

The Ā-chain can consist of the subject NP, the object NP or an AdvP like *often*. The chain starts in Cspec. The first clause of `cp/3` deals with the cases where an NP moves to Cspec. An "`np(NF)`" is thus put in `ABC`. The second clause is used when Cspec is occupied by an AdvP. In this case `ABC`

`agr2_1/5`.

will contain `advp(AdF)`. The Ā-chain is passed on and different actions are taken depending on what XP is in the chain. If Cspec is filled by the subject NP, this chain will terminate at Agr1spec in which case the first clause of `agr1p/5` will apply. (`ABC` becomes empty after that.) If the object is in Cspec, the chain will be passed on through Agr1P (second clause of agr1p/5), TP and AspP until it comes to Agr2P where the tail of the chain is found in Agr2spec (first clause of `agr2p/6`). If the constituent that has moved to Cspec is an AdvP, `ABC` will contain an AdvP instead of an NP. It passes through Agr1P (third clause of `agr1p/5`) and terminates in TP where an empty AdvP is adjoined to T1 and this AdvP is unified with the AdvP in `ABC` (fourth clause of `t1/6`).

Besides building the chains, we also have to make sure that each chain contains exactly one lexical head (a pronounced NP or verb). This checking is done through indexing. The value of the index feature starts as a variable. When a visible NP or verb is attached to a terminal, the variable becomes instantiated. In this particular implementation, the verb always receives the index 1, the subject NP 2, the object NP 3 and and the AdvP 4. Aux is not considered a full lexical item, so an X0-chain can seemingly contain more than one lexical item: a verb and one or more auxiliaries. For this reason, an Aux shares the index of the verb instead of having one of its own. Once the index feature of a chain has been instantiated, no other visible NPs or verbs can be put into the chain. This is achieved through `index/2` which is applied whenever a visible head is found. It requires that the value of "index" be a variable and it will refuse to incorporate a lexical item into a chain if the value is already a constant. This prevents a chain from having more than one lexical head. In addition, we use `lexical/1` to make sure that each chain does have a lexical head, in which case the value of "index" should be a constant rather than a variable. This predicate is applied after each chain terminates, by which time every chain is supposed to have found a lexical head. The checking of NP chains is done in `vp/5` after the termination of each chain in Vspec1 or Vspec2. The X0-chain is checked after the parse tree is complete. The checking cannot be done earlier, say, when V0 is reached, because the chain will not be complete at that time if CP or IP is head-final. The joint effect of `index/2` and `lexical/1` ensures that each chain has exactly one lexical head.

6.2.3 Leaf-Attachment

This is the part of the parsing procedure which deals with cross-linguistic variation due to different value combinations of S-parameters. It is applied whenever a terminal node is created. The procedure determines, on the basis of the current setting of S-parameters, whether the terminal node should dominate a content word, an Aux, or be empty. How such decisions are made has been discussed in 6.1. The particular actions to take in all individual situations have been summed up in the decision tables in (199), (200) and (201). They are directly coded in our Prolog program as `c0/5`, `agr1_0/5`, `t0/5`, `asp0/5`, `agr2_0/5`, `v0/5`, `np/5`, `subject/4`, `object/4`, `verb/5` and `aux/5`.

NPs in different positions are differentiated by the second argument in `np/5`. In each case the parser looks at S(M(cspec)), S(M(spec1)) and S(M(spec2)) to decide whether the terminal NP

should contain a subject, object or nothing. If the terminal node must be non-empty, a "word" (Subject/Object) will be taken from the input string and attached to the node as a leaf. If a subject NP is to be attached, `subject/4` is called to determine whether this NP must be overtly marked for case. `Object/4` is called when an object NP is to be attached. In cases where there is overt case, the morphological case of the overt NP must get unified with the case feature of the NP chain of which the terminal is a link. Indexing is also done at this point.

The leaf to be attached to X0 can be a verb, an Aux, or nothing. The computation involved here is more complex because a three-way decision has to be made. Each of the three possibilities is handled by a separate clause in `c0/5`, `agr1_0/5`, `t0/5`, `asp0/5`, `agr2_0/5` and `v0/5`. The first clause finds out if a verb can be attached here. If so, `verb/5` (which implements the decision table in (200)) is called to process the morphology of the verb to be attached. If not, the second clause is applied to find out if an Aux can be attached here. The real work is done by `aux/5` where the decision table in (201) is implemented. This predicate determines not only the presence/absence of an Aux in a given position but the specific morphological make-up of the Aux as well. The morphological information of the Aux to be attached is incorporated into the X0-chain using `code-features/2`. If neither the first clause nor the second applies, the third clause will be used and the terminal will be empty.

Since the terminal nodes are encountered strictly from left to right and every node is checked for its status as soon as it is encountered, the input string will not be accepted by the parser unless it has the required word order. The string will also be unacceptable if some symbols in the string do not have the correct morphological pattern.

6.2.4 The Parser in Action

The parser described above can be used in several different ways. We can call `parse/0` to get all the strings in a language and graphically displays their parse trees. We have seen examples of such parse trees in (197) and (198). We can also use `parse/1` to process a string without showing the tree and `parse/2` to get both the string and the tree.

Before a sentence is parsed, the parameters must be set to a particular value combination. The setting can be done on-line using `reset/0`. The following are two Prolog sessions illustrating the use of `reset/0` and `parse/1`. The clauses concerning *often* were commented out while running the first session but were included when the second session was run. The parse tree is printed out for each string that is generated. To save space, I omitted all the parse trees except for the last string in each setting. Furthermore, only the category label for each node is printed out, with all the other features suppressed in the tree printing.

(203) Session 1:

```
| ?- reset.
New setting: [1,1,1,1,0,1,1,0,i,i,0-1,0-0,0-1,0-1,0-1].          %1
```

```
yes
| ?- parse(S).

S = [s-[c1],iv-[agr,tns,asp]] ;

S = [s-[c1],tv-[agr,tns,asp],o-[c2]] ;

cp
    np
    c1
        c0
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0  Verb-[agr,tns,asp]
                tp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np  Obj-[c2]
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0
                                                vp
                                                    np
                                                    v1  V0
S = [s-[c1],tv-[agr,tns,asp],o-[c2]] ;

no
| ?- reset.
New setting: [0,0,0,0,0,0,0,0,f,f,0-0,1-0,0-0,1-0,0-1].        %2

yes
| ?- parse(S).

S = [s-[],tv-[asp],o-[],aux-[tns],aux-[pred]] ;

S = [s-[],iv-[asp],aux-[tns],aux-[pred]] ;

cp
    np
    c1
        agr1p
            np
            agr1_1
                tp
```

```
                    t1
                        asp_p
                            asp1
                                agr2p
                                    np
                                    agr2_1
                                        vp
                                            np  Subj-[]
                                            v1
                                                v0  Verb-[asp]
                                                vp
                                                    np  Obj-[]
                                                    v1  v0
                                        agr2_0
                                asp0
                        t0  Aux-[tns]
                agr1_0
        c0  Aux-[pred]

S = [s-[],tv-[asp],o-[],aux-[tns],aux-[pred]] ;

no
| ?- reset.
New setting: [1,1,1,1,1,1,1,0,0,i,i,0-1,0-0,0-1,0-1,0-1].          %3

yes
| ?- parse(S).

S = [iv-[agr,tns,asp],s-[c1]] ;

S = [tv-[agr,tns,asp],s-[c1],o-[c2]] ;

cp
    np
    c1
        c0  Verb-[agr,tns,asp]
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0
                tp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
```

```
                                                v0
                                                vp
                                                    np  Obj-[c2]
                                                    v1  v0
S = [tv-[agr,tns,asp],s-[c1],o-[c2]] ;

no
| ?- reset.
New setting: [0,0,0,1,0,1,0,0,i,i,0-1,0-0,1-0,1-0,0-1].        %4

yes
| ?- parse(S).

S = [s-[c1],aux-[agr,tns],tv-[asp],o-[c2]] ;

S = [s-[c1],aux-[agr,tns],iv-[asp]] ;

cp
    np
    c1
        c0
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0  Aux-[agr,tns]
                tp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0  Verb-[asp]
                                                vp
                                                    np  Obj-[c2]
                                                    v1  v0
S = [s-[c1],aux-[agr,tns],tv-[asp],o-[c2]] ;

no
| ?- reset.
New setting: [1,1,0,1,1,1,1,1,i,f,0-1,0-0,1-0,1-0,0-1].        %5

yes
| ?- parse(S).

S = [s-[c1],aux-[agr,tns],o-[c2],tv-[asp]] ;
```

```
S = [s-[c1],aux-[agr,tns],iv-[asp]] ;

cp
    np  Obj-[c2]
    c1
        c0  Aux-[agr,tns]
        agr1p
            np  Subj-[c1]
            agr1_1
                tp
                    t1
                        asp_p
                            asp1
                                agr2p
                                    np
                                    agr2_1
                                        vp
                                            np
                                            v1
                                                v0
                                                vp
                                                    np
                                                    v1  v0
                                        agr2_0
                                asp0  Verb-[asp]
                        t0
                agr1_0

S = [o-[c2],aux-[agr,tns],s-[c1],tv-[asp]] ;

no
| ?- reset.
New setting: [0,0,1,0,1,1,1,0,f,i,0-0,1-0,1-0,1-0,1-0].        %6

yes
| ?- parse(S).

S = [s-[],aux-[tns,asp],o-[],tv-[],aux-[pred,agr]] ;

S = [s-[],aux-[tns,asp],iv-[],aux-[pred,agr]] ;

cp
    np
    c1
        agr1p
            np  Subj-[]
            agr1_1
                agr1_0
                tp
                    t1
                        t0  Aux-[tns,asp]
                        asp_p
                            asp1
```

```
                                asp0
                                agr2p
                                    np  Obj-[]
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0  Verb-[]
                                                vp
                                                    np
                                                    v1  v0
        c0  Aux-[pred,agr]

S = [s-[],aux-[tns,asp],o-[],tv-[],aux-[pred,agr]] ;

no
| ?- reset.
New setting: [0,0,1,1,1,1,1,0,i,i,0-1,1-0,1-0,1-0,1-0].        %7

yes
| ?- parse(S).

S = [aux-[pred,agr,tns,asp],s-[c1],o-[c2],tv-[]] ;

S = [aux-[pred,agr,tns,asp],s-[c1],iv-[]] ;

cp
    np
    c1
        c0  Aux-[pred,agr,tns,asp]
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0
                tp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np  Obj-[c2]
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0  Verb-[]
                                                vp
                                                    np
                                                    v1  v0
```

```
S = [aux-[pred,agr,tns,asp],s-[c1],o-[c2],tv-[]] ;

no
| ?- reset.
New setting: [0,0,1,1,1,1,1,0,i,i,0-0,1-1,1-1,1-1,1-1].        %8

yes
| ?- parse(S).

S = [aux-[pred,agr,tns,asp],s-[],o-[],tv-[pred,agr,tns,asp]] ;

S = [aux-[pred,agr,tns,asp],s-[],iv-[pred,agr,tns,asp]] ;

cp
    np
    c1
        c0  Aux-[pred,agr,tns,asp]
        agr1p
            np  Subj-[]
            agr1_1
                agr1_0
                tp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np  Obj-[]
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0  Verb-[pred,agr,tns,asp]
                                                vp
                                                    np
                                                    v1  v0

S = [aux-[pred,agr,tns,asp],s-[],o-[],tv-[pred,agr,tns,asp]] ;

no
| ?-
```

(204) Session 2:

```
| ?- reset.
New setting: [1,1,1,0,0,1,0,1,i,i,0-1,0-0,0-1,0-1,0-0].        %1

yes
| ?- parse(S).

S = [s-[c1],iv-[agr,tns]] ;
```

```
S = [s-[c1],tv-[agr,tns],o-[c2]] ;

S = [s-[c1],often,iv-[agr,tns]] ;

S = [s-[c1],often,tv-[agr,tns],o-[c2]] ;

S = [often,s-[c1],iv-[agr,tns]] ;

cp
    advp  often
    c1
        c0
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0  Verb-[agr,tns]
                tp
                    t1
                        advp
                        t1
                            t0
                            asp_p
                                asp1
                                    asp0
                                    agr2p
                                        np
                                        agr2_1
                                            agr2_0
                                            vp
                                                np
                                                v1
                                                    v0
                                                    vp
                                                        np  Obj-[c2]
                                                        v1  v0
S = [often,s-[c1],tv-[agr,tns],o-[c2]] ;

no
| ?- reset.
New setting: [1,1,1,1,0,1,0,1,i,i,0-1,0-0,0-1,0-1,0-0].         %2

yes
| ?- parse(S).

S = [s-[c1],iv-[agr,tns]] ;

S = [s-[c1],iv-[agr,tns],often] ;

S = [s-[c1],tv-[agr,tns],o-[c2]] ;

S = [s-[c1],tv-[agr,tns],often,o-[c2]] ;

S = [often,s-[c1],iv-[agr,tns]] ;
```

```
cp
    advp  often
    c1
        c0
        agr1p
            np  Subj-[c1]
            agr1_1
                agr1_0  Verb-[agr,tns]
                tp
                    t1
                        advp
                        t1
                            t0
                            asp_p
                                asp1
                                    asp0
                                    agr2p
                                        np
                                        agr2_1
                                            agr2_0
                                            vp
                                                np
                                                v1
                                                    v0
                                                    vp
                                                        np  Obj-[c2]
                                                        v1  v0
S = [often,s-[c1],tv-[agr,tns],o-[c2]] ;

no
| ?- reset.
New setting: [1,1,1,1,1,1,1,1,1,i,i,0-0,0-0,0-0,0-0,0-0,0-0].        %3

yes
| ?- parse(S).

S = [s-[],iv-[]] ;

S = [s-[],iv-[],often] ;

S = [s-[],tv-[],o-[]] ;

S = [s-[],tv-[],often,o-[]] ;

S = [o-[],tv-[],s-[]] ;

S = [o-[],tv-[],s-[],often] ;

S = [often,iv-[],s-[]] ;

cp
    advp  often
```

```
c1
    c0  Verb-[]
    agr1p
        np  Subj-[]
        agr1_1
            agr1_0
            tp
                t1
                    advp
                    t1
                        t0
                        asp_p
                            asp1
                                asp0
                                agr2p
                                    np  Obj-[]
                                    agr2_1
                                        agr2_0
                                        vp
                                            np
                                            v1
                                                v0
                                                vp
                                                    np
                                                    v1  v0
S = [often,tv-[],s-[],o-[]] ;

no
| ?-
```

Every time a new setting is entered, `parse/1` is run exhaustively to find out all the strings that can be parsed with this setting. In Session 1, the number of strings which are accepted by each setting is always three: one intransitive sentence and two transitive sentences. The two transitive sentences are different from each other in that, at LF, Cspec is occupied by the subject NP in the first one while it is occupied by the object NP in the second. In other words, the two sentences differ as to whether the subject or object is the topic/focus of the sentence. This difference does not show up in the surface strings if there is no overt movement to Cspec. But it does show up in the parse tree. It shows up in the surface string when S(M(cspec)) is set to 1, as we can see in the fifth setting. In Session 2, more strings are generated with each setting because of the optional appearance of *often*. The number of strings generated is six if there is no overt movement to Cspec and eight if overt movement to Cspec occurs. There are two additional strings in the latter case because *often* itself can move to Cspec.

6.2.5 Alternative Implementations

What we have seen here is just one possible way of implementing the parser. There are many other alternatives of which we will briefly mention two. The alternative implementations may differ from

our present parser with regard to how the tree is built and how the chains are formed. They will not be much different, however, in terms of the way in which the leaves are attached.

We can build a head-driven parser which is like any other head-driven parser except the following: when a phrase is projected from a head, we must decide whether it is to be projected from a content word, a function word such as Aux, or from an empty head. This can be done in exactly the same way as we have done in the top-down parser presented here. The fact that we can find out precisely when a head is to be empty may help us solve a major problem in head-driven parsing: the problem of empty categories. With the decision tables in (199), (200) and (201), we never have to guess when to project a phrase from an empty head. Consequently, spurious projections can be avoided in spite of the existence of empty categories.

We can even construct a parser where all the possible trees and chains are precompiled. When parsing a sentence, we can use the subcategorization information of the verb to activate the relevant tree. Once a tree is activated, what remains to be done is just leaf attachment which can again be performed using the decision tables in (199), (200) and (201). The number of trees that need to be precompiled is not likely to be very big. Presumably, verbs of different types may project different trees. So there have to be at least as many trees as verb types. In addition, each verb type will need four trees, corresponding to the four different value combinations of HD-parameters. Therefore, if the number of verb types is n, the number of trees we have to precompile is $4n$. This is feasible if n is finite with a low bound. In traditional models where there are both head-specifier and head-complement parameters, the number of possible trees for each verb type would be greater and the approach of precompiltion would be less feasible.

The parsers mentioned above, the head-driven parser and the parser using precompiled trees, have been implemented in Prolog as well. They are available upon request.

6.2.6 Universal vs. Language-Particular Parsers

As we have seen, the parser presented here is universal in that it can process strings from any language by consulting the values of parameters. The reason why it is capable of doing so is that the parser is constructed in such a way that it can handle all possible cases. As a matter of fact, it is the union of parsers for all individual languages. When the parameters are set in a certain way so that just one language can be accepted, only a subpart of the parser is used. When the HD-parameters are set to "i", for instance, the clauses which require that the parameters be set to "f" will not be activated. There are three clauses for `c0/5`, `agr1_0/5`, `t0/5`, and `asp0/5`, fifteen clauses for `verb/5`, and twenty-six clauses for `aux/5`. Once the S-parameters are set, however, only one of them will be used. Which one is used depends on the parameter values. Therefore, although the program in Appendix A.7 is fairly big with many disjunctions, the parser for any particular language can be reasonably small. In fact, we can obtain any language-particular parser by removing all the clauses which will not be used with the given parameter setting. Once these clauses are removed, all the

choice points where parameter values are consulted no longer exist. As a result, we can remove all the calls to parameter values as well. The resulting parser can parse one language only but it will be more efficient because the computation involving parameter values is not necessary any more. An example of such a parser is given in Appendix A.8. This parser can only accept the language generated with the following parameter setting: [1, 1, 0, 1, 1, 1, 1, 0, i, i, 1-0, 0-0, 1-0, 1-0, 0-1]. Compared with the parser in Appendix A.7, this parser is much smaller, being a subset of the universal parser. Generally speaking, we can get such small parsers for all particular languages by taking different subsets out of the original parser.

This relationship between the universal parser and particular parsers is suggestive of a certain hypothesis on learning. We can speculate that children are born with the universal parser. This parser is used in setting the parameters. Once the parameters are set, however, there might be no longer the need to keep every part of the original parser. Some parts of the parser are never used and these parts may eventually be pruned away. The pruning results in the removal of all the choice points concerning parameter values. The resulting "adult parser" can therefore be used without reference to parameter values. Consequently the parameters can be removed as well. The removal of the unused parts of the parser can save time and space in parsing a particular language, but it can make it very difficult, if not impossible, to learn other languages. Whether this actually happens in human learning and whether this is related to the difficulty associated with second language acquisition are open to discussion.

6.3 Summary

In this chapter we have seen how parsing might work in our present model. Our parser differs from other parsers in that the procedures for tree-building and chain-building are almost invariable across languages. Differences between different languages show up mostly in how the leaves are attached to the tree. It is found that, given a particular setting of S-parameters, there is a unique way to attach the leaves. The parser can consult the parameter values and attach the leaves accordingly. It is universal in the sense that it can parse any language in the parameter space without a single change in the parser itself. A Prolog implementation of the parser is presented and alternative implementations are discussed. The presentation and the discussion show that our present syntactic model might have advantages over traditional models in terms of parsing.

Chapter 7

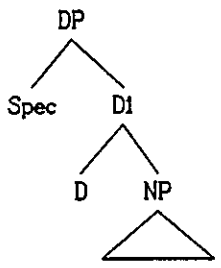
Final Discussion

This thesis has been a syntactic experiment in the field of Principles and Parameters theory. We have explored a parametric syntactic model which is based on the notion of Spell-Out in the Minimalist framework. A specific grammar was proposed and this grammar has been put to the test of language typology, language acquisition, and language processing. We are now in a better position to see the potentials and limitations of this model. In this chapter, I will first consider some possible extensions of the model and then discuss the validity of the present model in general.

7.1 Possible Extensions

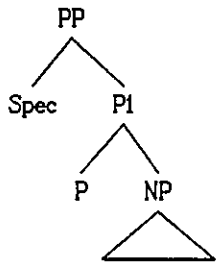
The experimental grammar we have examined in detail here is a very restricted one. Among those things that were left out are the internal structures of DP/NP and PP. We have put these phrases aside in order to limit our experimental parameter space to a manageable size. There is no principled reason why the present approach cannot be applied to the internal word orders of these constituents as well. As a matter of fact, a great deal of research has already done in this direction. The parallels between CP/IP and DP have been discussed by many people (Abney (1987), Ritter (1988,1990), Tellier (1988), Stowell (1989), Szabolcsi (1990), Carstens (1991,1993), Valois (1991), Mitchell (1993), Campbell (1993)). I will not get into a full discussion of non-verbal projections, but it is fairly obvious how the S(M)-parameters can work there. We can use a very simple DP structure to illustrate this. Suppose lexical projection and GT operations universally generate the following tree:

(205)



Suppose also that the NP in this tree must move to the Spec of DP by LF in order to have its case and ϕ-features checked against those of the determiner. (The fact that the noun has to agree with the determiner in many languages suggests that this checking operation is plausible.) If this checking movement has an S(M)-parameter associated with it, then the determiner will get spelled out in a pre-nominal position if this parameter is set to 0 and in a post-nominal position if the parameter is set to 1. The word order inside a PP can be derived in a similar way. Let us suppose that PP also has a Spec position as shown in (206).

(206)



There could be an LF requirement that the prepositional object must move to the Spec of PP to have its case features checked. If so, at Spell-Out the P will precede its object NP when the movement is covert and follow the object when the movement is overt.

Extensions of our current approach can also be made with respect to the notion of feature spell-out. Take PP again as an example. In all the cases where a preposition takes an NP object, we can treat the P as an overt realization of the case feature of this NP. In other words, we can let the case feature of this NP be associated with an S(F)-parameter. We see a preposition when this parameter is set to 1. This idea is by no means my own invention. It has been proposed recently (Ref??) that every NP has an abstract P (whether overt or covert) associated with it. The abstract P may well be a case feature which is spelled out as a preposition in certain cases. If this is true, we will not even need the tree in (206) and the case-checking movement to derive both prepositional and postpositional structures. The P is simply a case-marker which can appear either as a prefix or suffix. What we will have to explain then is why the the case marker can have different physical realizations on different NPs in the sentence, sometimes as an integral part of an NP/DP and sometimes as an more independent element such as a preposition.

In many languages, including English and Chinese, there exist both NPs carrying no case marker and prepositional NPs. If we regard P as a case marker, we face the question of why some NPs have to be overtly marked for case (by a P) and some do not. Here is a tentative answer to this question. As a working hypothesis, we can assume that any NP whose case feature is not checked in the Spec of IP (Agr1spec or Agr2spec) must be spelled out as a preposition. Consider the grammatical model used in our experiment. There are two Agr projections in an IP: Agr1P and Agr2P. Usually the subject NP can have its case checked in Agr1spec and the object NP can have its case checked in

Agr2P. This is probably why the subject and object NPs almost never need a preposition. If there are other NPs in a sentence, however, there will be no more Agrspecs for these NPs to move to in order to have their cass features checked. This can happen in many situations. One situation is where the sentence has an adjunct modifier, such as in (207).

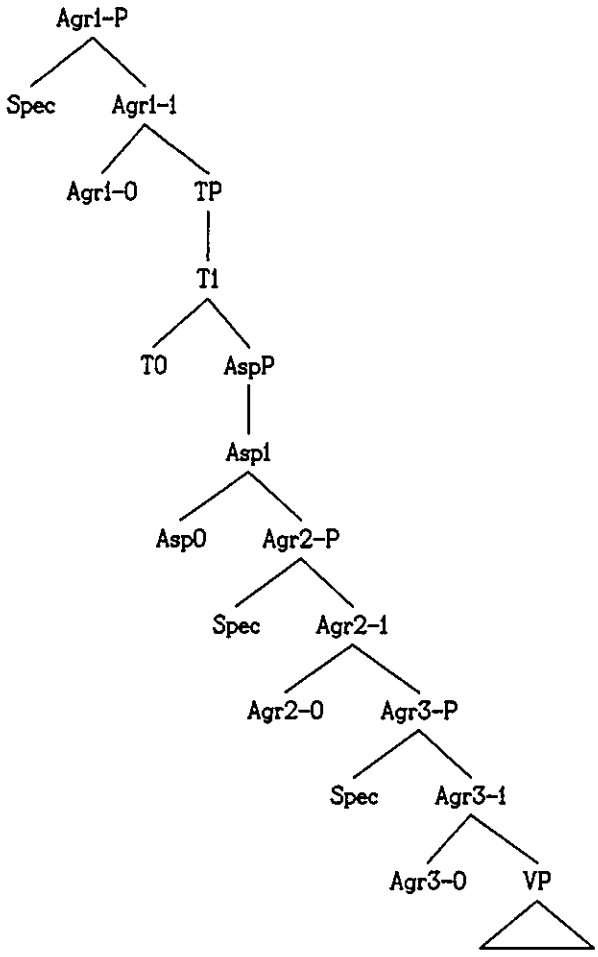
(207) *The girl met the boy in the garden.*

The subject and object NPs in this sentence, *the girl* and *the boy*, can obviously have their case features checked in Agr1spec and Agr2spec respectively. The third NP *the garden*, however, cannot move to any Spec of AgrP. It must therefore have its case feature spelled out as a P, as predicted by the hypothesis suggested above. This hypothesis may also explain why the subject NP in a passive sentence has to appear in a by-phrase. In passivization, the subject θ-role is absorbed. The object NP is "promoted" and can thus move to Agr1spec to have its case checked. If we want to mention the subject NP in a passive sentence, this NP can not move to Agr1spec which has already been occupied by the object NP. It cannot move to Agr2spec, either, because its case feature and the feature in Agr2spec will clash. As a result, it must have its case feature spelled out as a preposition, namely, *by*. Another way to look at it is by treating the by-phrase as an adjunct which, like *in the garden*, must appear as a PP.

The fact that we have assumed two Agreement projections in IP in our experimental model does not mean that there cannot be a third AgrP in IP. Certain verbs may project a triple-Agr IP. One such verb might be *give* which can be used in a double-object construction such as (208).

(208) *The girl gave the boy a book.*

The IP projected by *give* may look like the following.



(209)

In a sentence like (208), each of the three NPs can have its case features checked in one of the Agrspecs. There is therefore no need of a preposition.

An obvious question that arises here is why we need a preposition in (210).

(210) *The girl gave a book to the boy.*

This sentence contains exactly the same number of NPs as in (208), but one of them has to take a preposition. One way to tackle this problem is to assume that the verb *give* is syntactically ambiguous. It may project either a triple-Agr IP or a double-Agr IP. When a double-Agr IP is projected, the third NP in the sentence must be an adjunct which has to be licensed by an overt case feature manifested in a P.

The present model can also be extended to cover both nominative-accusative languages and ergative-absolutive languages. We have assumed that an IP contains two Agr projections even in an

intransitive sentence. As a result, there are two potential Agrspecs that the sole NP in an intransitive sentence can move to. Let us assume that the case checked in Agr1spec is nominative/ergative and the one checked in Agr2spec is accusative/absolutive. Then we will have an nominative/accusative language if this NP chooses to move to Agr1spec; we get an ergative/absolutive language if this NP moves to Agr2spec. We can then propose a parameter which determines which Agrspec an NP moves to in an intransitive sentence. This is again not my own invention. Similar approaches have been taken by Bobaljik (1992), Chomsky (1992), Laka (1992), etc. They actually have a name for this parameter: the *Obligatory Case Parameter*. A potential problem we can have with the particular structure assumed in this thesis is word order. In our IP projection, TP and AspectP come between Agr1P and Agr2P. In a language where the verb moves to T, we can have two different word orders in an intransitive sentence depending on which Agrspec the NP moves to. The order is NP-Verb if it moves to Agr1spec and Verb-NP if it moves to Agr2spec. In an ergative language where the verb moves to T, a transitive sentence will have the order NP-Verb-NP and an intransitive sentence will be Verb-NP. To account for an ergative language which is NP-Verb-NP when transitive and NP-Verb when intransitive, we have to assume that the verb can never move beyond Agr2 in this ergative language. This assumption will almost certainly turn out to be wrong. To avoid this problem, we can try an alternative model where there is only one Agr projection in IP when a sentence is intransitive. But this AgrP can have different case features in different languages. The obligatory case parameter then determines which case the AgrP has. We have a nominative/accusative language if it is Case 1 and an ergative/absolutive language if it is Case 2.

All the extensions proposed above can make our model more complete, but they have not been worked out in detail. A lot more research has to be done before we can incorporate them into our theory.

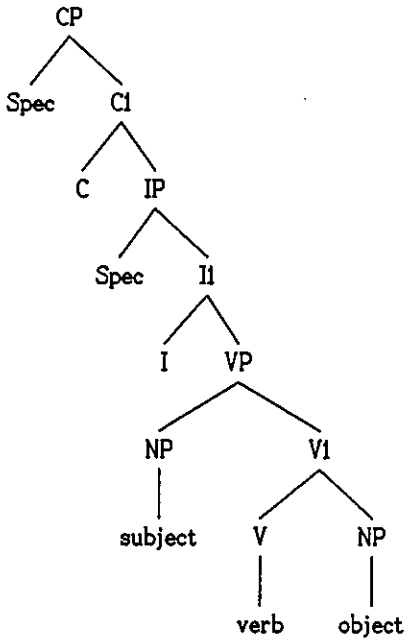
7.2 Potential Problems

The present model is not perfect and it can be challenged in many different ways. There are at least two kinds of argument that can be made against our approach. First of all, this model may seem too theory-dependent. We may wonder what will happen if some of the specific assumptions in our grammar turn out to be incorrect. Secondly, one may worry about the number of parameters we may need in a full version of the theory. It may seem that, as the grammar is expanded to cover more and more constructions, the parameters will become so many that learnability can become a problem. We will address these two potential problems in this section. We will see that our general approach can remain plausible even if many of the specifics of this theory are thrown out.

Almost every present-day syntactic model is theory-dependent to a certain extent. Any approach in the Principles and Parameters paradigm has to start from some basic assumptions of this theory, such as the existence of Universal Grammar. Our current approach is built upon some hypotheses in

the Minimalist Framework. One of those hypotheses is the notion of Spell-Out. All the experiments we have done in this thesis will be pointless if this basic notion turns out to be fallacious. However, what we have to worry more about is not whether a model is theory-dependent but the degree of such dependency. It is acceptable for a model to depend on certain theoretical assumptions, but these assumptions should be as general as possible. It is not desirable to have a model whose success hinges on some very specific assumptions which have not been generally accepted. One of the assumptions in our model which we may find suspicious is the structure of IP. It has been assumed in our model that the IP consists of a TP, an AspectP and two AgrPs. In addition, these phrasal projections must be arranged in a certain structural configuration. We may wonder what will happen if we replace this more articulated IP with a traditional non-split IP structure, such as the one in (211).

(211)



Let us assume this base structure and the following LF movements:

A. The verb must move to I to have its tense, aspect and agreement features checked.

B. After moving to I, the verb must move to C to have its predication features checked.

C. The subject NP must move to the Spec of IP to have its case and agreement features checked.

D. Either the subject NP or the Object NP must move to the Spec of CP to have its operator features checked.

We can let each of these movements be associated with an S(M)-parameter and let CP and IP each have an HD-parameter, as we have done before. Then we can derive the following word orders by

varying the parameter values (only one of the possibilities is given below for each order):

S V O if IP is head-initial, the subject NP moves overtly to Spec of IP and the verb moves to I;

S O V if IP is head-final, the subject NP moves overtly to Spec of IP and the verb moves to I;

V S O if CP is head-initial, the subject NP moves overtly to Spec of IP and the verb moves to C;

V2 if CP is head-initial, the verb moves overtly to C, and either the subject or object NP moves to the Spec of CP;

O S V if the object moves to Spec of CP and both the subject and the verb remain in situ;

O V S if CP is head-initial, the object NP moves overtly to Spec of CP, the verb moves to C, and the subject NP remains in situ.

We notice that the V O S order cannot be derived unless we allow the object NP to move to the Spec of IP or allow the Spec of IP to appear on the right. We will also lose many of the scrambling orders. What this shows is mixed. On the one hand, our model does seem too theory-dependent, since it misses certain word orders once the Split-Infl hypothesis is removed; on the other hand, we can still get most of the basic word orders even if the IP is non-split. In any case, our model *is* dependent on the Split-Infl hypothesis, at least to a certain degree.

Our specific theory also depends on the VP-internal Subject hypothesis. Once this hypothesis is dismissed, many movements will not be necessary any more. The word order variation we can derive from movement will be very limited. What does all this show? It may mean that the Split-Infl hypothesis and VP-internal Subject hypothesis are correct, as they can provide us with more explanatory power. But let us consider the worst case. Suppose that both of these two hypotheses are proven incorrect in the end. Can the model proposed in this thesis still exist? The answer can be "yes" or "no". The specific grammar used in this thesis can of course no longer exist. The movement patterns will have changed and so will the S(M)-parameters. All the experimental results in the thesis will need to be reconsidered. However, the general approach we are taking here can remain valid even in such a situation. We can proceed in this direction as long as the following are true:

(i) The grammar has X-bar structures and movement operations;

(ii) The X-bar structures are universal modulo head directions;

(iii) The movement operations are universal modulo the timing of Spell-Out;

(iv) Different head-directions and different spell-out of movements result in word order variation;

(v) The head directions of X-bar structures can be parameterized;

(vi) The spell-out of movement can be parameterized.

If these assumptions hold, we can build a model of the present sort no matter how the other specific assumptions change. The general picture of word order typology described in this thesis will not change; the learning algorithm presented here will still be applicable; and parsing can still proceed in the way presented in this thesis. The basic problem this thesis has addressed is how to handle word order in a syntactic model where cross-linguistic variation can result from both X-bar structure *and* movement. We have found a way to describe a word order typology in terms of both head-direction and movement. We have also discovered a learning strategy which the learner can use to converge on any particular grammar by simultaneously setting two different types of parameters: X-bar parameters and movement parameters. This is a problem that has to be addressed by any acquisition theory which accepts the view that word order can be determined by both phrase structure and movement. Finally, we have seen the possibility of a more universal parser which can parse different languages by looking at the parameter settings of those languages.

Now let us consider the potential problem of "parameter explosion". The model we have been working with is minimal, but the number of parameters we have assumed does not seem too small. One may wonder how many parameters we would eventually need when the model is expanded to include more constructions. There seem to be many ways in which the number of parameters may grow. Here are a few of them:

(212) **a.** In order to account for word orders within other constructions, such as DP/NP and PP, more S(M)-parameters and HD-parameters may be needed to control the movements and head directions internal to these constituents.

b. Since even a single language may have different word orders in statements and questions, in main clauses and embedded clauses, etc., we seem in need of different parameters in different types of clauses.

c. To handle the full range of inflectional morphology in world's languages, a greater number of features may need to be taken into consideration. As a result, the number of S(F)-parameters may increase.

It looks as if the parameter space could be much bigger than the one we have dealt with. The amount of search involved in learning and parsing could then be so great that language acquisition and language processing might become a problem.

Are the problems in (212) real problems? Let us examine them one by one.

The problem in (212(a)) exists only if the internal structures of DP/NP and PP are totally unrelated to those of CP/IP. This does not seem to be the case. There are more and more studies showing that DP/NP parallels CP/IP in many ways. It is very likely that these phrases are similar not only in X-bar terms, but in terms of movement as well. It could well be the case that a movement

in CP has a counterpart in DP. Moreover, the corresponding movements could be similar in their overtness, i.e. they might both occur before Spell-Out or both after Spell-Out. If so, we will not need two separate S(M)-parameters. The two movements could be considered different instances of a single type of movement whose spell-out is controlled by a single S(M)-parameter. Should this be true, the number of parameters will not increase as much as we might expect. Language acquisition will therefore not be a problem. As a matter of a fact, parameter setting could be easier, since the learner can get evidence for a parameter value from both CP/IP and DP/NP (Koopman ??)

The problem described in (212(b)) can be a real problem only if we adopt the assumption that HD and S(M)-parameters are the only determinants of word order. This assumption seems to be false. There are obviously other factors which can influence the word order of a language. When an S(M)-parameter has the value 1/0, for instance, whether the relevant movement occurs before Spell-Out depends on things other than the parameter values. The Principle of Procrastinate dictates that the movement should be after Spell-Out in this case, but this principle can be overridden if some other factors call for overt movement. When a language has different word orders in statements and questions, or in main clauses and embedded clauses, the difference can often be explained by the overtness of one or two movements. It is definitely not the case that different clauses have totally different parameters or parameter values. In English, statements are S-Aux-V-O and yes-no questions are Aux-S-V-O. A simple explanation for this fact is that the auxiliary moves to C in questions but not in statements. We do not need additional parameters to account for this if we assume that the S(M)-parameter for I-to-C movement is set to 1/0 in English. The real question we have to answer then is what overrides the Principle of Procrastinate in interrogative sentences to make the movement overt. This is a question that has to be addressed in any linguistic theory regardless of the existence of S-parameters.

A similar argument can be made for German which has the V2 order in main clauses and SOV orders in subordinate clauses. Assuming that CP is head-initial and IP is head-final in German, we can account for the word order difference by supposing that the S(M)-parameters for I-to-C movement and XP-movement to Cspec are both set to 1/0 in German. In subordinate clauses, these movements are covert due to the Principle of Procrastinate. In matrix clauses, however, the movements are made overt by some other factors. What these factors are remain the topics of current linguistic research. The important point is that we do not need different parameters for questions or embedded clauses. Once the module of linguistic theory we have studied here is interfaced with other modules, the correct word order in each type of clauses will emerge. The success of our model therefore depends on the research in those other modules.

Finally we address the problem in (212(c)). The number of S(F)-parameters required depends on the number of features required by the grammar. As long as the set of features is finitely small, there will not be too many S(F)-parameters. The question is how many features have to be there in our system. There is no definite answers here, but the number should be finite. This may seem false

in view of the fact that morphological variation in world's languages is so rich. But the seemingly infinite variation in inflectional morphology does not have to imply that the number of morphological features is infinite. We can get tens of thousands of surface morphological paradigms from a small set of features because of the following:

i. Different languages can have different subsets of features spelled out;

ii. Different combinations of features can have different phonological realizations;

iii. The phonological realization of a certain feature or a combination of features can be arbitrary.

Therefore, while we may need more features than we already have in the system, there is very little indication that the required set of features must be infinite.

In conclusion, the likelihood of an explosion of S(M)-parameters and S(F)-parameters is very small. We probably need more parameters but the increase will not be dramatic. As long as the number of parameters is finite and reasonably small, parameter-counting is not particularly meaningful. Given two grammars which account for the same range of linguistic phenomena, the one with fewer parameters is of course preferable. However, there is no principled reason why the number of parameters should be less than 20 or less than 30. As long as there is a learning algorithm whereby those parameters can be correctly set, the exact number of parameters should not be an issue. In fact, a small increase in parameters is welcome if this can result in a simplification of the principles of our grammar.

7.3 Concluding Remarks

In this thesis, we have studied a particular model of the Principles and Parameters theory. By fully exploiting the notion of Spell-Out, we have set up a grammar where cross-linguistic variation in word order and inflectional morphology is mainly determined by a set of S-parameters. The new parametric system, though still in its preliminary form, has been found to possess some desirable properties in terms of language typology, language acquisition and language processing. The experiments we have performed in this thesis are far from complete, but the initial results are encouraging. There is reason to believe that this line of research is at least worth pursuing, though a great deal of future work is needed to get this model closer to the truth.

Appendix A

Prolog Programs

A.1

```
% File: pspace.pl
% Author: Andi Wu
% Date: July 15, 1993
% Purpose: Find all values combinations of S(M)-parameters, S(F)-parameters,
%       HD-parameters and AA-parameter.  Try generating some language
%       (possibly empty) with each value combination and collect the set of
%       all non-empty languages that are generated and their corresponding
%       parameter settings.

:- ensure_loaded(library(basics)).
:- ensure_loaded(generator). % The parser/generator used here is in a separate file.

:- dynamic s/1, hd1/1, hd2/1, aa/1, lang/1.

%% pspace(A list of all the setting-language pairs in the parameter space,
%%        each containing one possible parameter setting and the language
%%        it generates)
pspace(Ps) :- setof(P,sl_pair(P),Ps).

%% pspace1(A list of all the setting-language pairs in the parameter space,
%%         each containing one or more possible parameter settings and the
%%         single language they generate)
% (The settings in a single pair all generate the same language.)
pspace1(Ps) :-
        setof(P,sl_pair(P),Ps0),
        group_settings(Ps0,Ps).



%% pspace2(A list of all the distinct languages that can be generated in
          the parameter space)
pspace2(Ls) :-
setof(P,sl_pair(P),Ps),
collect_langs(Ps,Ls).
```

```
%% sl_pair([Setting,Language]).
sl_pair([S,L]):-
        get_setting(S),
        setof(L,sl_pair1(S,L),Ls),
        merge_l(Ls,L).


%% sl_pair1(Setting, A list of setting-language pairs, each with a different
%%          variable instantiation of the setting)
sl_pair1(S,L) :-
        instantiate_var(S,S1),
        generate_all(S1,L).

% Find all strings that can be generated from a given (fully instantiated)
% value combination
generate_all(Setting,Strings) :-
        retract_old_setting,
        assert_new_setting(Setting),
        setof(String,generate(String),Strings).

%% instantiate_var(Setting,Particular_Instantiation_of_Setting)
% (It has no effect on settings that do not contain variables.)
% Note: s(m(spec1)), s(m(spec2)) and s(m(cspec)) may be set to 1/0 which,
% being a variable, can be instantiated to either 1 or 0 in a particular
% parse.  The language generated by a setting containing such variable(s)
% is the union of the languages generated with each particular instantiation.
% (It only has effect on settings containing variable values.)
instantiate_var([1/0|Vs1],[V|Vs2]) :- !,
        (V=0; V=1),
        instantiate_var(Vs1,Vs2).
instantiate_var([V|Vs1],[V|Vs2]) :-
        instantiate_var(Vs1,Vs2).
instantiate_var([],[]).

%% merge_l(Sets_of_Strings,Union_of_Sets_of_Strings)
% Merge languages generated with different instantiations of a setting
merge_l([L1,L2|Ls],L):-
        merge(L1,L2,L3),
        merge_l([L3|Ls],L).
merge_l(L,L).

%% group_settings
%%      (A list of setting-language pairs,
%%       A list of setting(s)-language pairs
%%      ).
% Group together settings that generate identical languages.
group_settings(SL_Pairs,SL_Pairs1) :-
        retractall(lang(_,_)),
        pack_pairs(SL_Pairs),
        collect_pairs(SL_Pairs1).

% Assert all languages and group together the settings for each of them
```

```
pack_pairs([[S,L]|Pairs]) :-
        lang(S1,L1),            % the present language has already
        same_set(L,L1),!,       % been asserted.
        retract(lang(S1,L1)),   % add the present setting to the
        assert(lang([S|S1],L)), % settings for this language.
        pack_pairs(Pairs).
pack_pairs([[S,L]|Pairs]) :-    % the present language has not been asserted.
        assert(lang([S],L)),    % assert this new language
        pack_pairs(Pairs).
pack_pairs([]).

% Collect all languages and their settings.
collect_pairs(Ps) :-
        collect_pairs([],Ps).

collect_pairs(Ps0,Ps) :-
        retract(lang(Ss,L)),
        collect_pairs([[Ss,L]|Ps0],Ps).
collect_pairs(Ps,Ps).

collect_langs(Ps,Ls) :-
collect_langs(Ps,[],Ls).

collect_langs([[_,L]|Ps],Ls_So_Far,Ls) :-
member1(L,Ls_So_Far),!,
collect_langs(Ps,Ls_So_Far,Ls).
collect_langs([[_,L]|Ps],Ls_So_Far,Ls) :-
collect_langs(Ps,[L|Ls_So_Far],Ls).
collect_langs([],Ls,Ls).

% Take union of two or more sets of strings.
merge([S|Ss],S2,New_S) :-
        member(S,S2),!,
        merge(Ss,S2,New_S).

merge([S|Ss],S2,New_S) :-
        merge(Ss,[S|S2],New_S).
merge([],S,S).

same_set([],[]).
same_set([A|As],B) :-
        member(A,B),
        select(A,B,Bs),
        same_set(As,Bs).

member1(A,[B|_]) :- same_set(A,B).
member1(A,[_|Bs]) :-
member1(A,Bs).

select(A,[A|As],As).
select(A,[B|Bs],[B|Cs]) :-
select(A,Bs,Cs).
```

```
retract_old_setting :-
        retractall(s(_)),
        retractall(hd1(_)),
        retractall(hd2(_)),
        retractall(aa(_)).

assert_new_setting([A,B,C,D,E,F,G,H,HD1,HD2,Case,Agr,T,Asp,Pred,Op,AA]) :-
        assert(s(m(agr2(A)))),          assert(s(m(asp(B)))),
        assert(s(m(tns(C)))),           assert(s(m(agr1(D)))),
        assert(s(m(c(E)))),             assert(s(m(spec1(F)))),
        assert(s(m(spec2(G)))),         assert(s(m(cspec(H)))),
        assert(hd1(HD1)),               assert(hd2(HD2)),
        assert(s(f(case(Case)))),       assert(s(f(agr(Agr)))),
        assert(s(f(tns(T)))),           assert(s(f(asp(Asp)))),
        assert(s(f(pred(Pred)))),       assert(s(f(op(Op)))).

get_setting([A,B,C,D,E,F,G,H,D1,D2,Case,Agr,T,Asp,Pred,Op,AA]):-
        sp(m(agr2(A))),sp(m(asp(B))),sp(m(tns(C))),sp(m(agr1(D))),
        sp(m(c(E))),sp(m(spec1(F))),sp(m(spec2(G))),sp(m(cspec(H))),
        sp(f(case(Case))),sp(f(agr(Agr))),sp(f(tns(T))),
        sp(f(asp(Asp))),sp(f(pred(Pred))),sp(f(op(Op))),
        c_head(D1),i_head(D2).

sp(m(agr2(0))).         sp(m(agr2(1))).
sp(m(asp(0))).          sp(m(asp(1))).
sp(m(tns(0))).          sp(m(tns(1))).
sp(m(agr1(0))).         sp(m(agr1(1))).
sp(m(c(0))).            sp(m(c(1))).
sp(m(spec1(0))).        sp(m(spec1(1))).        sp(m(spec1(1/0))).
sp(m(spec2(0))).        sp(m(spec2(1))).        sp(m(spec2(1/0))).
sp(m(cspec(0))).        sp(m(cspec(1))).        sp(m(cspec(1/0))).

sp(f(case(0-0))).       sp(f(case(0-1))).
sp(f(case(1-0))).       sp(f(case(1-1))).
sp(f(agr(0-0))).        sp(f(agr(0-1))).
sp(f(agr(1-0))).        sp(f(agr(1-1))).
sp(f(tns(0-0))).        sp(f(tns(0-1))).
sp(f(tns(1-0))).        sp(f(tns(1-1))).
sp(f(asp(0-0))).        sp(f(asp(0-1))).
sp(f(asp(1-0))).        sp(f(asp(1-1))).
sp(f(pred(0-0))).       sp(f(pred(0-1))).
sp(f(pred(1-0))).       sp(f(pred(1-1))).
sp(f(op(0-0))).         sp(f(op(0-1))).
sp(f(op(1-0))).         sp(f(op(1-1))).

c_head(i).      c_head(f).
i_head(i).      i_head(f).
```

A.2

```
% File: sets.pl
% Author: Andi Wu
```

```
% Update: July 16, 1993
% Purpose: Compute the set-theoretic relations between languages in a given
%        parameter space.

:- ensure_loaded(pspace0).
:- ensure_loaded(generator0).

%% disjoint_pairs(A list consisting of pairs of languages in the parameter
%%                space which are disjoint with each other)
% (Each distinct language represented by a distinct number in the output)
disjoint_pairs(Ps) :-
        pspace2(Ls),
        retractall(language(_)),
        assert_languages(Ls,1),!,
        setof(P,disjoint_pair(P),Ps).

disjoint_pair([N1,N2]) :-
        language(N1,A),
        language(N2,B),
        disjoint(A,B).



%% intersecting_pairs(A list consisting of pairs of languages in the parameter
%%                    space which intersect each other)
% (Each distinct language represented by a distinct number in the output)
intersecting_pairs(Ps) :-
        pspace2(Ls),
        retractall(language(_)),
        assert_languages(Ls,1),!,
        setof(P,intersecting_pair(P),Ps).

intersecting_pair([N1,N2]) :-
        language(N1,A),
        language(N2,B),
        intersecting(A,B).

%% proper_inclusions(A list consisting of pairs of languages in the parameter
%%                   space where the first member of each pair is a proper
%%                   subset of the second member)
% (Each distinct language represented by a distinct number in the output)
proper_inclusions(Ps) :-
        pspace2(Ls),
        retractall(language(_)),
        assert_languages(Ls,1),!,
        setof(P, properly_included(P),Ps).

properly_included([N1,N2]) :-
        language(N1,A),
        language(N2,B),
        properly_includes(B,A).

%% Find out the set-theoretic relation between any two languages.
```

```
set_relation(L1,L2) :-
        (       identical(L1,L2),
                write(L1),nl, write(and),nl, write(L2),nl,
                write('are identical.')
        ;       disjoint(L1,L2),
                write(L1),nl, write(and),nl, write(L2),nl,
                write('are disjoint.')
        ;       intersect(L1,L2),
                write(L1),nl, write(and),nl, write(L2),nl,
                write('are intersecting.')
        ;       properly_includes(L1,L2),
                write(L2),nl,
                write('is a proper subset of'),nl,
                write(L1)
        ),nl.


identical([A|As],B) :-
        member(A,B),
        select(A,B,Bs),
        identical(As,Bs).
identical([],[]).

disjoint(A,B) :-
        \+ co_member(A,B).

intersect(A,B) :-
        co_member(A,B),
        unique_member(A,B),
        unique_member(B,A),!.

properly_includes(A,B) :-
        subset(B,A),
        unique_member(A,B),!.

subset([],_).
subset([A|As],B) :-
        member(A,B),
        subset(As,B).

co_member([A|_],B) :-
        member(A,B).
co_member([_|As],B) :-
        co_member(As,B).

unique_member([A|_],B) :-
        \+ member(A,B).
unique_member([_|As],B) :-
        unique_member(As,B).

assert_languages([L|Ls],N) :-
        assert(language(N,L)),
        N1 is N+1,
```

```
        assert_languages(Ls,N1).
assert_languages([],_).
```

A.3

```
% File: order.pl
% Author: Andi Wu
% Date: August 8, 1993
% Purpose: Order the settings in a given
%        parameter space in the spirit of
%        the Principle of Procrastinate

order_settings(S,S1) :-
        quicksort(S,S1).

quicksort(List,Sorted) :-
        quicksort2(List,Sorted-[]).

quicksort2([],Z-Z).

quicksort2([X|Tail],A1-Z2) :-
        split(X,Tail,Small,Big),
        quicksort2(Small,A1-[X|A2]),
        quicksort2(Big,A2-Z2).

split(_X,[],[],[]).
split(X,[Y|Tail],[Y|Small],Big) :-
        verify(precedes(Y,X)),!,
        split(X,Tail,Small,Big).

split(X,[Y|Tail],Small,[Y|Big]) :-
        split(X,Tail,Small,Big).

precedes(P1,P2) :-
        fewer_opt_mvnt_than(P1,P2),!.
precedes(P1,P2) :-
        \+ has_abar_mvnt(P1),
        has_abar_mvnt(P2),!.
precedes(P1,P2) :-
        fewer_mvnt_than(P1,P2).

fewer_opt_mvnt_than(P1,P2) :-
        num_of_opt_mvnt(P1,N1),
        num_of_opt_mvnt(P2,N2),
        N1<N2.

num_of_opt_mvnt(P,N) :-
        num_of_opt_mvnt1(P,0,N).

num_of_opt_mvnt1([],N,N).
num_of_opt_mvnt1([P|Ps],N0,N) :-
        var(P),!,
        N1 is N0+1,
```

```
        num_of_opt_mvnt1(Ps,N1,N).
num_of_opt_mvnt1([_|Ps],N0,N) :-
        num_of_opt_mvnt1(Ps,N0,N).

has_abar_mvnt([_,_,_,_,_,_,_,_,1|_]).

fewer_mvnt_than(P1,P2) :-
        num_of_mvnt(P1,N1),
        num_of_mvnt(P2,N2),
        N1<N2.

num_of_mvnt(P,N) :-
        num_of_mvnt1(P,0,N).

num_of_mvnt1([],N,N).
num_of_mvnt1([1|Ps],N0,N) :- !,
        N1 is N0+1,
        num_of_mvnt1(Ps,N1,N).
num_of_mvnt1([_|Ps],N0,N) :-
        num_of_mvnt1(Ps,N0,N).

verify(P) :- \+(\+P).
```

A.4

```
% File: sp.pl
% Author: Andi Wu
% Date: August 3, 1993
% Purpose: Acquiring word orders by setting S-parameters.

:- ensure_loaded(library(basics)).
:- ensure_loaded(sets).
:- ensure_loaded(parser).
:- ensure_loaded(order).
:- ensure_loaded(sputil).

:- dynamic settings/1.

get_pspace :-
        get_settings,
        get_languages.

get_settings :-
        setof(S,get_setting(S),Ss),
        order_settings(Ss,Ss1),
        assert(settings0(Ss1)).

get_languages :-
        setof(L,get_language(L),Ls),
        assert(langs(Ls)).


get_setting([A,B,C,D,E,F,G,H]) :-
```

```
        sp(m(agr2(A))),sp(m(asp(B))),sp(m(tns(C))),sp(m(agr1(D))),
        sp(m(c(E))),sp(m(spec1(F))),sp(m(spec2(G))),sp(m(cspec(H))).

get_language(L) :-
        get_setting(Setting),
        setof(String,generate(Setting,String),L).

sp :- initialize,
        write('The initial setting is '),
        current_setting,
        sp1.

sp1 :-  next_input(S),
        (       S=initialize -> sp
        ;       S=bye -> true
        ;       S=generate -> generate,sp1
        ;       S=current_setting -> current_setting,sp1
        ;       process(S),!,
                write('Current setting remains unchanged.'),nl,
                sp1
        ;       write('Unable to parse '),
                write(S),nl,
                write('Resetting the parameters ... '),nl,nl,
                reset_to_process(S),
                sp1
        ).

reset_to_process(S) :-
        try_next_setting,!,
        (       process(S),!,
                write('Parameters reset to: '),
                current_setting
        ;       reset_to_process(S)
        ).

learn_all_langs :-
        langs(Ls),
        learn_all(Ls).

learn_all([L|Ls]) :-
        learn1(L),!,
        learn_all(Ls).
learn_all([]).



learn1(L) :-
        write('Trying to learn '),
        write(L),write(' ...'),nl,
        initialize,
        learn(L).

learn(L) :-
```

```
        process_all(L),!,
        write('Final setting: '),
        current_setting,
        generate(L1),
        write('Language generated: '),
        write(L1),nl,
        (       identical(L,L1),!,
                write('The language '),
                write(L),
                write(' is learnable.'),nl
        ;       write(' which is a superset of '),
                write(L),nl,
                write('The language '),
                write(L),
                write(' is NOT learnable.'),nl,nl
        ),nl.
learn(L) :-
        try_next_setting,!,
        learn(L).

initialize :-
        retractall(current_setting(_)),
        retractall(s(_)),
        retractall(settings(_)),
        settings0([S|Ss]),
        assert(current_setting(S)),
        assert(settings(Ss)).

process_all([S|Ss]) :-
        process(S),
        process_all(Ss).
process_all([]).

process(S) :-
        current_setting(P),
        instantiate_var(P,P1),
        retract_old_value,
        assert_new_value(P1),
        parse(S).

try_next_setting :-
        retractall(current_setting(_)),
        retract(settings([S|Ss])),
        assert(current_setting(S)),
        assert(settings(Ss)).

generate :-
        current_setting(P),
        setof(S,generate(P,S),Ss),
        write('Language generated with current setting: '),nl,
        write(Ss),nl.

generate(Ss) :-
```

```
        current_setting(P),
        setof(S,generate(P,S),Ss).

generate(P,S) :-
        instantiate_var(P,P1),
        retract_old_value,
        assert_new_value(P1),
        parse(S).

current_setting :-
        current_setting(P),
        write_values(P).

retract_old_value :-
        retractall(s(m(_))).

assert_new_value([A,B,C,D,E,F,G,H]) :-
        assert(s(m(agr2(A)))),
        assert(s(m(asp(B)))),
        assert(s(m(tns(C)))),
        assert(s(m(agr1(D)))),
        assert(s(m(c(E)))),
        assert(s(m(spec1(F)))),
        assert(s(m(spec2(G)))),
        assert(s(m(cspec(H)))).

sp(m(agr2(0))).         sp(m(agr2(1))).
sp(m(asp(0))).          sp(m(asp(1))).
sp(m(tns(0))).          sp(m(tns(1))).
sp(m(agr1(0))).         sp(m(agr1(1))).
sp(m(c(0))).            sp(m(c(1))).
sp(m(spec1(0))).        sp(m(spec1(1))).        sp(m(spec1(_))).
sp(m(spec2(0))).        sp(m(spec2(1))).        sp(m(spec2(_))).
sp(m(cspec(0))).        sp(m(cspec(1))).        sp(m(cspec(_))).
```

A.5

```
% File: spuntil.ps
% Author: Andi Wu
% Date: August 15, 1993
% Purpose: Tools used in sp.pl

write_values(Vs) :-
        write('['),
        write_values1(Vs),
        write(']'),nl.

write_values1([]).
write_values1([V|Vs]) :-
        var(V),!,
        write('1/0'), tab(2),
        write_values1(Vs).
write_values1([V|Vs]) :-
```

```
        write(V), tab(2),
        write_values1(Vs).

write1([S|Ss]) :-
        write(S),
        write(Ss).
write1([]).

next_input(Input) :-
        repeat,
        write('Next? '),
        read(Input).

instantiate_var([V1|Vs1],[V2|Vs2]) :-
        var(V1),!, (V2=0; V2=1),
        instantiate_var(Vs1,Vs2).
instantiate_var([V|Vs1],[V|Vs2]) :-
        instantiate_var(Vs1,Vs2).
instantiate_var([],[]).
```

A.6

```
% File: sp2.pl
% Author: Andi Wu
% Date: August 3, 1993
% Purpose: Acquiring word orders and inflectional morphology by setting
% S(M)-parameters, HD-parameters and S(F)-parameters.

:- ensure_loaded(library(basics)).
:- ensure_loaded(sets).
:- ensure_loaded(parser).
:- ensure_loaded(order).
:- ensure_loaded(sputil).
:- dynamic settings/1.

get_pspace :-
        get_settings,
        get_languages.

get_settings :-
        setof(S,get_setting(S),Ss),
        order_settings(Ss,Ss1),
        assert(settings0(Ss1)).

get_languages :-
        setof(L,get_language(L),Ls),
        assert(langs(Ls)).

get_setting([A,B,C,D,E,F,G,H,HD1,HD2]) :-
        sp(m(agr2(A))),sp(m(asp(B))),sp(m(tns(C))),sp(m(agr1(D))),
        sp(m(c(E))),sp(m(spec1(F))),sp(m(spec2(G))),sp(m(cspec(H))),
        c_head(HD1),i_head(HD2).
```

```
get_setting1([A,B,C,D,E,F,G,H,HD1,HD2,Case,Pred,Agr,Tns,Asp]) :-
        sp(m(agr2(A))),sp(m(asp(B))),sp(m(tns(C))),sp(m(agr1(D))),
        sp(m(c(E))),sp(m(spec1(F))),sp(m(spec2(G))),sp(m(cspec(H))),
        c_head(HD1),i_head(HD2),
        sp(f(case(Case))),sp(f(pred(Pred))),sp(f(agr(Agr))),
        sp(f(tns(Tns))),sp(f(asp(Asp))).

get_language(L) :-
        get_setting1(Setting),
        setof(String,generate1(Setting,String),L).

sp :- initialize,
        write('The initial setting is '),nl,
        current_setting,
        sp1.

sp1 :- next_input(S),
        (       S=initialize -> sp
        ;       S=bye -> true
        ;       S=generate -> generate,sp1
        ;       S=current_setting -> current_setting,sp1
        ;       process(S),!,
                write('Current setting remains unchanged.'),nl,
                sp1
        ;       write('Unable to parse '),
                write(S),nl,
                write('Resetting the parameters ... '),nl,nl,
                reset_to_process(S),
                sp1
        ).

reset_to_process(S) :-
        (       reset_sfp(S),
                process(S),!,
                write('Successful parse.'),nl
        ;       try_next_setting(V),
                process(S),!,
                write('Word order parameters reset to: '),
                write_values(V),nl,
                write('Successful parse.'),nl
        ;       !,reset_to_process(S)
        ).

learn_all_langs :-
        langs(Ls),
        learn_all(Ls).

learn_all([L|Ls]) :-
        learn1(L),!,
        learn_all(Ls).
learn_all([]).

learn1(L) :-
```

```
        write('Trying to learn '),
        write(L),write(' ...'),nl,
        initialize,
        set_sfp(L),
        learn(L).

learn(L) :-
        process_all(L),!,
        write('Final setting: '),
        current_setting,
        generate(L1),
        write('Language generated: '),
        write(L1),nl,
        (       identical(L,L1),!,
                write('The language '),
                write(L),
                write(' is learnable.'),nl
        ;       write(' which is a superset of '),
                write(L),nl,
                write('The language '),
                write(L),
                write(' is NOT learnable.'),nl,nl
        ),nl.
learn(L) :-
        try_next_setting(_),!,
        learn(L).

initialize :-
        retractall(current_setting(_)),
        retractall(s(_)),
        retractall(hd1(_)),
        retractall(hd2(_)),
        retractall(settings(_)),
        initial_setting([S|Ss]),
        assert(current_setting(S)),
        assert(current_setting(S)),
        assert(settings(Ss)),
        assert(s(f(case(0-0)))),
        assert(s(f(pred(0-0)))),
        assert(s(f(agr(0-0)))),
        assert(s(f(tns(0-0)))),
        assert(s(f(asp(0-0)))).

process_all([S|Ss]) :-
        process(S),
        process_all(Ss).
process_all([]).


process(S) :-
        current_setting(P),
        instantiate_var(P,P1),
        retract_old_values,
```

```
        assert_new_values(P1),
        parse(S).

try_next_setting(S) :-
        retractall(current_setting(_)),
        retract(settings([S|Ss])),
        assert(current_setting(S)),
        assert(settings(Ss)).


set_sfp([S|Ss]) :-
        reset_sfp(S),
        set_sfp(Ss).
set_sfp([]).

reset_sfp([W|Ws]) :-
        check_sfp(W),
        reset_sfp(Ws).
reset_sfp([]).

check_sfp(_-[]).
check_sfp(W-[F|Fs]) :-
        check_sfp1(W-[F]),
        check_sfp(W-Fs).
check_sfp(often).

check_sfp1(aux-[F]) :- !,
        check_f_feature(F).
check_sfp1(_-[F]) :-
        check_l_feature(F).

check_f_feature(pred) :-
        (       s(f(pred(1-_))),!
        ;       retract(s(f(pred(_-L)))),
                assert(s(f(pred(1-L)))),
                write('s(f(pred)) is reset to '),
                write(1-L),nl
        ).
check_f_feature(agr) :-
        (       s(f(agr(1-_))),!
        ;       retract(s(f(agr(_-L)))),
                assert(s(f(agr(1-L)))),
                write('s(f(agr)) is reset to '),
                write(1-L),nl
        ).
check_f_feature(tns) :-
        (       s(f(tns(1-_))),!
        ;       retract(s(f(tns(_-L)))),
                assert(s(f(tns(1-L)))),
                write('s(f(tns)) is reset to '),
                write(1-L),nl
        ).
```

```
check_f_feature(asp) :-
        (       s(f(asp(1-_))),!
        ;       retract(s(f(asp(_-L)))),
                assert(s(f(asp(1-L)))),
                write('s(f(asp)) is reset to '),
                write(1-L),nl
        ).

check_l_feature(Ftr) :-
        ( Ftr=c1; Ftr=c2),
        (       s(f(case(_-1))),!
        ;       retract(s(f(case(F-_)))),
                assert(s(f(case(F-1)))),
                write('s(f(case)) is reset to '),
                write(F-1),nl
        ).
check_l_feature(pred) :-
        (       s(f(pred(_-1))),!
        ;       retract(s(f(pred(F-_)))),
                assert(s(f(pred(F-1)))),
                write('s(f(pred)) is reset to '),
                write(F-1),nl
        ).
check_l_feature(agr) :-
        (       s(f(agr(_-1))),!
        ;       retract(s(f(agr(F-_)))),
                assert(s(f(agr(F-1)))),
                write('s(f(agr)) is reset to '),
                write(F-1),nl
        ).
check_l_feature(tns) :-
        (       s(f(tns(_-1))),!
        ;       retract(s(f(tns(F-_)))),
                assert(s(f(tns(F-1)))),
                write('s(f(tns)) is reset to '),
                write(F-1),nl
        ).
check_l_feature(asp) :-
        (       s(f(asp(_-1))),!
        ;       retract(s(f(asp(F-_)))),
                assert(s(f(asp(F-1)))),
                write('s(f(asp)) is reset to '),
                write(F-1),nl
        ).

generate :-
        current_setting(P),
        setof(S,generate(P,S),Ss),
        write('Language generated with current setting: '),nl,
        writel(Ss),nl.


generate(Ss) :-
```

```
        current_setting(P),
        setof(S,generate(P,S),Ss).

generate(P,S) :-
        instantiate_var(P,P1),
        retract_old_values,
        assert_new_values(P1),
        parse(S).

generate1(P,S) :-
        instantiate_var(P,P1),
        retract_old_values1,
        assert_new_values1(P1),
        parse(S).

current_setting :-
        current_setting(P),
        setof(V,s(f(V)),Vs),
        write('['), write_values(P),nl,tab(1),
        write_values(Vs),write(']'),nl.

retract_old_values :-
        retractall(s(m(_))),
        retractall(hd1(_)),
        retractall(hd2(_)).

assert_new_values([A,B,C,D,E,F,G,H,HD1,HD2]) :-
        assert_smp([A,B,C,D,E,F,G,H]),
        assert_hdp([HD1,HD2]).

retract_old_values1 :-
        retractall(s(_)),
        retractall(hd1(_)),
        retractall(hd2(_)).

assert_new_values1([A,B,C,D,E,F,G,H,HD1,HD2,Case,Pred,Agr,Tns,Asp]) :-
        assert_smp([A,B,C,D,E,F,G,H]),
        assert_hdp([HD1,HD2]),
        assert_sfp([Case,Pred,Agr,Tns,Asp]).

assert_smp([A,B,C,D,E,F,G,H]) :-
        assert(s(m(agr2(A)))),
        assert(s(m(asp(B)))),
        assert(s(m(tns(C)))),
        assert(s(m(agr1(D)))),
        assert(s(m(c(E)))),
        assert(s(m(spec1(F)))),
        assert(s(m(spec2(G)))),
        assert(s(m(cspec(H)))).

assert_hdp([HD1,HD2]) :-
        assert(hd1(HD1)),
        assert(hd2(HD2)).
```

```
assert_sfp([Case,Pred,Agr,Tns,Asp]) :-
        assert(s(f(case(Case)))),
        assert(s(f(pred(Pred)))),
        assert(s(f(agr(Agr)))),
        assert(s(f(tns(Tns)))),
        assert(s(f(asp(Asp)))).

sp(m(agr2(0))).         sp(m(agr2(1))).
sp(m(asp(0))).          sp(m(asp(1))).
sp(m(tns(0))).          sp(m(tns(1))).
sp(m(agr1(0))).         sp(m(agr1(1))).
sp(m(c(0))).            sp(m(c(1))).
sp(m(spec1(0))).        sp(m(spec1(1))).        sp(m(spec1(_))).
sp(m(spec2(0))).        sp(m(spec2(1))).        sp(m(spec2(_))).
sp(m(cspec(0))).        sp(m(cspec(1))).        sp(m(cspec(_))).

sp(f(F-L)) :-
        f_feature(F),
        l_feature(L).

f_feature(0).       f_feature(1).
l_feature(0).       l_feature(1).

c_head(i).      c_head(f).
i_head(i).      i_head(f).
```

A.7

```
% File: parser.pl
% Author: Andi Wu
% Updated: August 20, 1993
% Purpose: A top-down parser implementing the S-parameter model.

:- ensure_loaded(johnson).
:- ensure_loaded(tree).

:- dynamic s/1,hd1/1,hd2/1.

parse :- cp(Tree,_,[]), d(Tree),fail.
parse :- write('No more parse.').

parse(S) :- cp(_,S,[]).

parse(S,T) :- cp(T,S,[]).

cp(cp/[np(NF)/NP,C1]) -->
        {op(NF)===’+’},
        np(NP,cspec,NF),
        c1(C1,[np(NF)]).
cp(cp/[advp(AdF)/[often],C1]) --> [often],
        {s(m(cspec(1))),
         op(AdF)===’+’,
```

```
         index(AdF)===4
        },
        c1(C1,[advp(AdF)]).

c1(c1/[c0(CF,Th)/C,Agr1P],ABC) -->
        {hd1(i)},
        c0(C,CF,Th),
        agr1p(Agr1P,x0(CF,Th),ABC),
        {lexical(CF)}.
c1(c1/[Agr1P,c0(CF,Th)/C],ABC) -->
        {hd1(f)},
        agr1p(Agr1P,x0(CF,Th),ABC),
        c0(C,CF,Th),
        {lexical(CF)}.


agr1p(agr1p/[np(NF)/NP,Agr1_1],HC,[np(NF1)]) -->
        {case(NF)===c1,
         check_np_features(NF,NF1)
        },
        np(NP,agr1spec,NF),
        agr1_1(Agr1_1,HC,[np(NF)],[]).
agr1p(agr1p/[np(NF)/NP,Agr1_1],x0(HF,Th),[np(NF1)]) -->
        {Th=[_,_],
         case(NF)===c1,
         case(NF)=/=case(NF1)
        },
        np(NP,agr1spec,NF),
        agr1_1(Agr1_1,x0(HF,Th),[np(NF)],[np(NF1)]).
agr1p(agr1p/[np(NF)/NP,Agr1_1],x0(HF,Th),[advp(AdF)]) -->
        np(NP,agr1spec,NF),
        agr1_1(Agr1_1,x0(HF,Th),[np(NF)],[advp(AdF)]).


agr1_1(agr1_1/[agr1_0(Agr1F,Th)/Agr1,TP],x0(HF,Th),[np(NF)],ABC) -->
        {hd2(i),
         phi(Agr1F)===phi(NF),
         check_v_features(Agr1F,HF)
        },
        agr1_0(Agr1,Agr1F,Th),
        tp(TP,x0(HF,Th),[np(NF)],ABC).
agr1_1(agr1_1/[TP,agr1_0(Agr1F,Th)/Agr1],x0(HF,Th),[np(NF)],ABC) -->
        {hd2(f),
         phi(Agr1F)===phi(NF),
         check_v_features(Agr1F,HF)
        },
        tp(TP,x0(HF,Th),[np(NF)],ABC),
        agr1_0(Agr1,Agr1F,Th).

tp(tp/[T1],HC,AC,ABC) -->
        t1(T1,HC,AC,ABC).

t1(t1/[t0(TF,Th)/T,AspP],x0(HF,Th),AC,ABC) -->
```

```
        {hd2(i),
         check_v_features(TF,HF),
         \+ABC=[advp(_)]
        },
        t0(T,TF,Th),
        asp_p(AspP,x0(HF,Th),AC,ABC).
t1(t1/[AspP,t0(TF,Th)/T],x0(HF,Th),AC,ABC) -->
        {hd2(f),
         check_v_features(TF,HF),
         \+ABC=[advp(_)]
        },
        asp_p(AspP,x0(HF,Th),AC,ABC),
        t0(T,TF,Th).

t1(t1/[advp/[often],T1],x0(HF,Th),AC,ABC) --> [often],
        {\+ABC=[advp(_)]},
        t1(T1,x0(HF,Th),AC,ABC,_).
t1(t1/[advp(AdF)/[],T1],x0(HF,Th),AC,[advp(AdF)]) -->
        t1(T1,x0(HF,Th),AC,[],_).
t1(t1/[t0(TF,Th)/T,AspP],x0(HF,Th),AC,ABC,_) -->
        {check_v_features(TF,HF)},
        t0(T,TF,Th),
        asp_p(AspP,x0(HF,Th),AC,ABC).

asp_p(asp_p/[Asp1],HC,AC,ABC) -->
        asp1(Asp1,HC,AC,ABC).


asp1(asp1/[asp0(AspF,Th)/Asp,Agr2P],x0(HF,Th),AC,ABC) -->
        {hd2(i),
         check_v_features(AspF,HF)
        },
        asp0(Asp,AspF,Th),
        agr2p(Agr2P,x0(HF,Th),AC,ABC).
asp1(asp1/[Agr2P,asp0(AspF,Th)/Asp],x0(HF,Th),AC,ABC) -->
        {hd2(f),
         check_v_features(AspF,HF)
        },
        agr2p(Agr2P,x0(HF,Th),AC,ABC),
        asp0(Asp,AspF,Th).

agr2p(agr2p/[np(NF)/NP,Agr2_1],x0(HF,Th),AC,[np(NF1)]) -->
        {Th=[_,_],
         case(NF)===c2,
         check_np_features(NF,NF1)
        },
        np(NP,agr2spec,NF),
        agr2_1(Agr2_1,x0(HF,Th),[np(NF)|AC]).
agr2p(agr2p/[np(NF)/NP,Agr2_1],x0(HF,Th),AC,[]) -->
        {Th=[_,_],
         case(NF)===c2
        },
        np(NP,agr2spec,NF),
```

```
        agr2_1(Agr2_1,x0(HF,Th),[np(NF)|AC]).
agr2p(agr2p/[Agr2_1],x0(HF,Th),AC,[]) -->
        {Th=[_]},
        agr2_1(Agr2_1,x0(HF,Th),AC).

agr2_1(agr2_1/[agr2_0(Agr2F,Th)/Agr2,VP],x0(HF,Th),[np(NF)|NPs]) -->
        {hd2(i),
         check_v_features(Agr2F,HF)
        },
        agr2_0(Agr2,Agr2F,Th),
        vp(VP,x0(HF,Th),[np(NF)|NPs]).
agr2_1(agr2_1/[VP,agr2_0(Agr2F,Th)/Agr2],x0(HF,Th),[np(NF)|NPs]) -->
        {hd2(f),
         check_v_features(Agr2F,HF)
        },
        vp(VP,x0(HF,Th),[np(NF)|NPs]),
        agr2_0(Agr2,Agr2F,Th).

vp(vp/[np(NF)/NP,V1],x0(HF,[agt|Ths]),AC) -->
        {theta(NF)===agt,
         select(AC,np(NF1),AC1),
         case(NF1)===c1,
         check_np_features(NF,NF1)
        },
        np(NP,vspec1,NF),
        {lexical(NF)},
        v1(V1,x0(HF,[agt|Ths]),AC1).
vp(vp/[np(NF)/NP,V1],x0(HF,[pat]),AC) -->
        {theta(NF)===pat,
         select(AC,np(NF1),AC1),
         case(NF1)===c2,
         check_np_features(NF,NF1)
        },
        np(NP,vspec2,NF),
        {lexical(NF)},
        v1(V1,x0(HF,[pat]),AC1).

v1(v1/[v0(VF,[Th1|Ths])/V,VP],x0(HF,[Th1|Ths]),AC) -->
        {check_v_features(VF,HF)},
        v0(V,VF,[Th1|Ths]),
        vp(VP,x0(HF,Ths),AC).
v1(v1/[v0(VF,[Th])/V],x0(HF,[Th]),_AC) -->
        {check_v_features(VF,HF)},
        v0(V,VF,[Th]).

c0(Verb,CF,Th) -->
        {v_to_c},
        verb(Verb,CF,Th).
c0(Aux,CF,_Th) -->
        aux(Aux,c,CF).
c0([],_,_) --> [],
        {\+v_to_c,
         \+aux(_,c,_,_,_)
```

```
        }.

agr1_0(V,Agr1F,Th) -->
        {v_to_agr1},
        verb(V,Agr1F,Th).
agr1_0(Aux,Agr1F,_Th) -->
        aux(Aux,agr1,Agr1F).
agr1_0([],_,_) --> [],
        {\+v_to_agr1,
         \+aux(_,agr1,_,_,_)
        }.

t0(V,TF,Th) -->
        {v_to_t},
        verb(V,TF,Th).

t0(Aux,TF,_Th) -->
        aux(Aux,t,TF).
t0([],_,_) --> [],
        {\+v_to_t,
         \+aux(_,t,_,_,_)
        }.
asp0(V,AspF,Th) -->
        {v_to_asp},
        verb(V,AspF,Th).
asp0(Aux,AspF,_Th) -->
        aux(Aux,asp,AspF).
asp0([],_,_) --> [],
        {\+v_to_asp,
         \+aux(_,asp,_,_,_)
        }.

agr2_0(V,Agr2F,Th) -->
        {v_to_agr2},
        verb(V,Agr2F,Th).
agr2_0([],_,_) --> [].

v0(V,VF,[agt|Ths]) -->
        {s(m(agr2(0)))},
        verb(V,VF,[agt|Ths]).
v0([],_,[agt|_]) --> [],
        {s(m(agr2(1)))}.
v0([],_,[Th|_]) --> [],
        {\+Th=agt}.

np(Subj,cspec,NF) -->
        {s(m(cspec(1))), s(m(spec1(1)))},
        subject(Subj,NF).
np(Obj,cspec,NF) -->
        {s(m(cspec(1))), s(m(spec2(1)))},
        object(Obj,NF).
np([],cspec,_) --> [],
        {s(m(cspec(0)))}.
```

```
np(Subj,agr1spec,NF) -->
        {s(m(spec1(1)))},
        subject(Subj,NF).
np([],agr1spec,_) --> [].

np(Obj,agr2spec,NF) -->
        {s(m(spec2(1)))},
        object(Obj,NF).
np([],agr2spec,_) --> [].

np(Subj,vspec1,NF) -->
        {s(m(spec1(0)))},
        subject(Subj,NF).
np([],vspec1,_) --> [].

np(Obj,vspec2,NF) -->
        {s(m(spec2(0)))},
        object(Obj,NF).
np([],vspec2,_) --> [].

subject(['Subj-[]'/[]],NF) --> [s-[]],
        {s(f(case(0-0))),
         case(NF)===c1,
         index(NF,s)
        }.
subject(['Subj-[c1]'/[]],NF) --> [s-[c1]],
        {s(f(case(0-1))),
         case(NF)===c1,
         index(NF,s)
        }.

object(['Obj-[]'/[]],NF) --> [o-[]],
        {s(f(case(0-0))),
         case(NF)===c2,
         index(NF,o)
        }.
object(['Obj-[c2]'/[]],NF) --> [o-[c2]],
        {s(f(case(0-1))),
         case(NF)===c2,
         index(NF,o)
        }.

verb(['Verb-[]'/[]],VF,Th) --> [V-[]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-0))),s(f(tns(_-0))),s(f(asp(_-0))),
         code_features([],VF),
         index(VF,V)
        }.
verb(['Verb-[pred]'/[]],VF,Th) --> [V-[pred]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-0))),s(f(tns(_-0))),s(f(asp(_-0))),
         code_features([pred],VF),
```

```
         index(VF,V)
        }.
verb(['Verb-[agr]'/[]],VF,Th) --> [V-[agr]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-1))),s(f(tns(_-0))),s(f(asp(_-0))),
         code_features([agr],VF),
         index(VF,V)
        }.
verb(['Verb-[tns]'/[]],VF,Th) --> [V-[tns]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-0))),s(f(tns(_-1))),s(f(asp(_-0))),
         code_features([tns],VF),
         index(VF,V)
        }.
verb(['Verb-[asp]'/[]],VF,Th) --> [V-[asp]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-0))),s(f(tns(_-0))),s(f(asp(_-1))),
         code_features([asp],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,agr]'/[]],VF,Th) --> [V-[pred,agr]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-1))),s(f(tns(_-0))),s(f(asp(_-0))),
         code_features([pred,agr],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,tns]'/[]],VF,Th) --> [V-[pred,tns]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-0))),s(f(tns(_-1))),s(f(asp(_-0))),
         code_features([pred,tns],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,asp]'/[]],VF,Th) --> [V-[pred,asp]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-0))),s(f(tns(_-0))),s(f(asp(_-1))),
         code_features([pred,asp],VF),
         index(VF,V)
        }.
verb(['Verb-[agr,tns]'/[]],VF,Th) --> [V-[agr,tns]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-1))),s(f(tns(_-1))),s(f(asp(_-0))),
         code_features([agr,tns],VF),
         index(VF,V)
        }.
verb(['Verb-[agr,asp]'/[]],VF,Th) --> [V-[agr,asp]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-1))),s(f(tns(_-0))),s(f(asp(_-1))),
         code_features([agr,asp],VF),
         index(VF,V)
        }.
verb(['Verb-[tns,asp]'/[]],VF,Th) --> [V-[tns,asp]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-0))),s(f(tns(_-1))),s(f(asp(_-1))),
```

```
         code_features([tns,asp],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,agr,tns]'/[]],VF,Th) --> [V-[pred,agr,tns]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-1))),s(f(tns(_-1))),s(f(asp(_-0))),
         code_features([pred,agr,tns],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,agr,asp]'/[]],VF,Th) --> [V-[pred,agr,asp]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-1))),s(f(tns(_-0))),s(f(asp(_-1))),
         code_features([pred,agr,asp],VF),
         index(VF,V)
        }.
verb(['Verb-[agr,tns,asp]'/[]],VF,Th) --> [V-[agr,tns,asp]],
        {th_grid(V,Th),
         s(f(pred(_-0))),s(f(agr(_-1))),s(f(tns(_-1))),s(f(asp(_-1))),
         code_features([agr,tns,asp],VF),
         index(VF,V)
        }.
verb(['Verb-[pred,agr,tns,asp]'/[]],VF,Th) --> [V-[pred,agr,tns,asp]],
        {th_grid(V,Th),
         s(f(pred(_-1))),s(f(agr(_-1))),s(f(tns(_-1))),s(f(asp(_-1))),
         code_features([pred,agr,tns,asp],VF),
         index(VF,V)
        }.

aux(['Aux-[pred]'/[]],c,CF) --> [aux-[pred]],
        {s(f(pred(1-_))),
         s(m(c(0))),
         code_features([pred],CF)
        }.
aux(['Aux-[agr]'/[]],c,CF) --> [aux-[agr]],
        {s(f(pred(0-_))),s(f(agr(1-_))),
         s(m(c(1))),s(m(agr1(0))),
         code_features([agr],CF)
        }.
aux(['Aux-[tns]'/[]],c,CF) --> [aux-[tns]],
        {s(f(pred(0-_))),s(f(agr(0-_))),s(f(tns(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(0))),
         code_features([tns],CF)
        }.
aux(['Aux-[asp]'/[]],c,CF) --> [aux-[asp]],
        {s(f(pred(0-_))),s(f(agr(0-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([asp],CF)
        }.
aux(['Aux-[pred,agr]'/[]],c,CF) --> [aux-[pred,agr]],
        {s(f(pred(1-_))),s(f(agr(1-_))),
         s(m(c(1))),s(m(agr1(0))),
         code_features([pred,agr],CF)
        }.
```

```
aux(['Aux-[pred,tns]'/[]],c,CF) --> [aux-[pred,tns]],
        {s(f(pred(1-_))),s(f(agr(0-_))),s(f(tns(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(0))),
         code_features([pred,tns],CF)
        }.
aux(['Aux-[pred,asp]'/[]],c,CF) --> [aux-[pred,asp]],
        {s(f(pred(1-_))),s(f(agr(0-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([pred,asp],CF)
        }.
aux(['Aux-[agr,tns]'/[]],c,CF) --> [aux-[agr,tns]],
        {s(f(pred(0-_))),s(f(agr(1-_))),s(f(tns(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(0))),
         code_features([agr,tns],CF)
        }.
aux(['Aux-[agr,asp]'/[]],c,CF) --> [aux-[agr,asp]],
        {s(f(pred(0-_))),s(f(agr(1-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([agr,asp],CF)
        }.
aux(['Aux-[tns,asp]'/[]],c,CF) --> [aux-[tns,asp]],
        {s(f(pred(0-_))),s(f(agr(0-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([tns,asp],CF)
        }.
aux(['Aux-[pred,agr,tns]'/[]],c,CF) --> [aux-[pred,agr,tns]],
        {s(f(pred(1-_))),s(f(agr(1-_))),s(f(tns(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(0))),
         code_features([pred,agr,tns],CF)
        }.
aux(['Aux-[pred,agr,asp]'/[]],c,CF) --> [aux-[pred,agr,asp]],
        {s(f(pred(1-_))),s(f(agr(1-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([pred,agr,asp],CF)
        }.
aux(['Aux-[pred,tns,asp]'/[]],c,CF) --> [aux-[pred,tns,asp]],
        {s(f(pred(1-_))),s(f(agr(0-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([pred,tns,asp],CF)
        }.
aux(['Aux-[agr,tns,asp]'/[]],c,CF) --> [aux-[agr,tns,asp]],
        {s(f(pred(0-_))),s(f(agr(1-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([agr,tns,asp],CF)
        }.
aux(['Aux-[pred,agr,tns,asp]'/[]],c,CF) --> [aux-[pred,agr,tns,asp]],
        {s(f(pred(1-_))),s(f(agr(1-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([pred,agr,tns,asp],CF)
        }.
aux(['Aux-[agr]'/[]],agr1,Agr1F) --> [aux-[agr]],
        {s(f(agr(1-_))),
         s(m(c(0))),s(m(agr1(0))),
```

```
         code_features([agr],Agr1F)
        }.
aux(['Aux-[tns]'/[]],agr1,Agr1F) --> [aux-[tns]],
        {s(f(agr(0-_))),s(f(tns(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(0))),
         code_features([tns],Agr1F)
        }.
aux(['Aux-[asp]'/[]],agr1,Agr1F) --> [aux-[asp]],
        {s(f(agr(0-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([asp],Agr1F)
        }.
aux(['Aux-[agr,tns]'/[]],agr1,Agr1F) --> [aux-[agr,tns]],
        {s(f(agr(1-_))),s(f(tns(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(0))),
         code_features([agr,tns],Agr1F)
        }.
aux(['Aux-[agr,asp]'/[]],agr1,Agr1F) --> [aux-[agr,asp]],
        {s(f(agr(1-_))),s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([agr,asp],Agr1F)
        }.
aux(['Aux-[tns,asp]'/[]],agr1,Agr1F) --> [aux-[tns,asp]],
        {s(f(agr(0-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([tns,asp],Agr1F)
        }.
aux(['Aux-[agr,tns,asp]'/[]],agr1,Agr1F) --> [aux-[agr,tns,asp]],
        {s(f(agr(1-_))),s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(c(0))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(0))),
         code_features([agr,tns,asp],Agr1F)
        }.
aux(['Aux-[tns]'/[]],t,TF) --> [aux-[tns]],
        {s(f(tns(1-_))),
         s(m(agr1(0))),s(m(tns(0))),
         code_features([tns],TF)
        }.
aux(['Aux-[asp]'/[]],t,TF) --> [aux-[asp]],
        {s(f(tns(0-_))),s(f(asp(1-_))),
         s(m(agr1(0))),s(m(tns(1))),s(m(asp(0))),
         code_features([asp],TF)
        }.
aux(['Aux-[tns,asp]'/[]],t,TF) --> [aux-[tns,asp]],
        {s(f(tns(1-_))),s(f(asp(1-_))),
         s(m(agr1(0))),s(m(tns(1))),s(m(asp(0))),
         code_features([tns,asp],TF)
        }.
aux(['Aux-[asp]'/[]],asp,AspF) --> [aux-[asp]],
        {s(f(asp(1-_))),
         s(m(tns(0))),s(m(asp(0))),
         code_features([asp],AspF)
        }.
```

```
check_v_features(HF1,HF2) :-
        ind(HF1)===ind(HF2),
        phi(HF1)===phi(HF2),
        tns(HF1)===tns(HF2),
        asp(HF1)===asp(HF2).

check_np_features(NF1,NF2) :-
        ind(NF1)===ind(NF2),
        theta(NF1)===theta(NF2),
        case(NF1)===case(NF2),
        phi(NF1)===phi(NF2),
        op(NF1)===op(NF2).

th_grid(V,Th) :-
        (       V=iv, Th=[agt]
        ;       V=tv, Th=[agt,pat]
        ).

code_features([F|Fs],VF) :-
        code_feature(F,VF),
        code_features(Fs,VF).
code_features([],_).

code_feature(F,VF) :-
        Feature=..[F,VF],
        Feature === @(F1,F1,_),
        F=F1.

index(Chain,Cat) :-
        ind(Chain) === @(I,I,_),var(I),
        (       (Cat=iv;Cat=tv), I=1
        ;       Cat=s, I=2
        ;       Cat=o, I=3
        ).

lexical(Chain) :-
        ind(Chain)=== @(I,_,_),nonvar(I).

v_to_c :- s(m(c(1))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(1))),s(m(agr2(1))).
v_to_agr1 :- s(m(c(0))),s(m(agr1(1))),s(m(tns(1))),s(m(asp(1))),s(m(agr2(1))).
v_to_t :- s(m(agr1(0))),s(m(tns(1))),s(m(asp(1))),s(m(agr2(1))).
v_to_asp :- s(m(tns(0))),s(m(asp(1))),s(m(agr2(1))).
v_to_agr2 :- s(m(asp(0))),s(m(agr2(1))).

reset :-
        write('New setting: '),
        read([A,B,C,D,E,F,G,H,HD1,HD2,Case,Pred,Agr,Tns,Asp]),
        retractall(s(_)), retractall(hd1(_)), retractall(hd2(_)),
        assert(s(m(agr2(A)))),      assert(s(m(asp(B)))),
        assert(s(m(tns(C)))),       assert(s(m(agr1(D)))),
        assert(s(m(c(E)))),         assert(s(m(spec1(F)))),
        assert(s(m(spec2(G)))),     assert(s(m(cspec(H)))),
        assert(hd1(HD1)),           assert(hd2(HD2)),
```

```
        assert(s(f(case(Case)))),    assert(s(f(pred(Pred)))),
        assert(s(f(agr(Agr)))),      assert(s(f(tns(Tns)))),
        assert(s(f(asp(Asp)))).
```

A.8

```
% File: parser1.pl
% Author: Andi Wu
% Updated: August 24, 1993
% Purpose: A top-down parser implementing the S-parameter model.

:- ensure_loaded(johnson).
:- ensure_loaded(tree).

parse :- cp(Tree,_,[]), d(Tree),fail.
parse :- write('No more parse.').

parse(S) :- cp(_,S,[]).

parse(S,T) :- cp(T,S,[]).

cp(cp/[np(NF)/NP,C1]) -->
        {op(NF)===':+'},
        np(NP,cspec,NF),
        c1(C1,[np(NF)]).

c1(c1/[c0(CF,Th)/C,Agr1P],ABC) -->
        c0(C,CF,Th),
        agr1p(Agr1P,x0(CF,Th),ABC),
        {lexical(CF)}.

agr1p(agr1p/[np(NF)/NP,Agr1_1],HC,[np(NF1)]) -->
        {case(NF)===c1,
         check_np_features(NF,NF1)
        },
        np(NP,agr1spec,NF),
        agr1_1(Agr1_1,HC,[np(NF)],[]).
agr1p(agr1p/[np(NF)/NP,Agr1_1],x0(HF,Th),[np(NF1)]) -->
        {Th=[_,_],
         case(NF)===c1,
         case(NF)=/=case(NF1)
        },
        np(NP,agr1spec,NF),
        agr1_1(Agr1_1,x0(HF,Th),[np(NF)],[np(NF1)]).

agr1_1(agr1_1/[agr1_0(Agr1F,Th)/Agr1,TP],x0(HF,Th),[np(NF)],ABC) -->
        { phi(Agr1F)===phi(NF),
         check_v_features(Agr1F,HF)
        },
        agr1_0(Agr1,Agr1F,Th),
        tp(TP,x0(HF,Th),[np(NF)],ABC).

tp(tp/[T1],HC,AC,ABC) -->
```

```
        t1(T1,HC,AC,ABC).

t1(t1/[t0(TF,Th)/T,AspP],x0(HF,Th),AC,ABC) -->
        { check_v_features(TF,HF),
         \+ABC=[advp(_)]
        },
        t0(T,TF,Th),
        asp_p(AspP,x0(HF,Th),AC,ABC).

t1(t1/[advp/[often],T1],x0(HF,Th),AC,ABC) --> [often],
        {\+ABC=[advp(_)]},
        t1(T1,x0(HF,Th),AC,ABC,_).
t1(t1/[t0(TF,Th)/T,AspP],x0(HF,Th),AC,ABC,_) -->
        {check_v_features(TF,HF)},
        t0(T,TF,Th),
        asp_p(AspP,x0(HF,Th),AC,ABC).

asp_p(asp_p/[Asp1],HC,AC,ABC) -->
        asp1(Asp1,HC,AC,ABC).

asp1(asp1/[asp0(AspF,Th)/Asp,Agr2P],x0(HF,Th),AC,ABC) -->
        { check_v_features(AspF,HF)
        },
        asp0(Asp,AspF,Th),
        agr2p(Agr2P,x0(HF,Th),AC,ABC).

agr2p(agr2p/[np(NF)/NP,Agr2_1],x0(HF,Th),AC,[np(NF1)]) -->
        {Th=[_,_],
         case(NF)===c2,
         check_np_features(NF,NF1)
        },
        np(NP,agr2spec,NF),
        agr2_1(Agr2_1,x0(HF,Th),[np(NF)|AC]).
agr2p(agr2p/[np(NF)/NP,Agr2_1],x0(HF,Th),AC,[]) -->
        {Th=[_,_],
         case(NF)===c2
        },
        np(NP,agr2spec,NF),
        agr2_1(Agr2_1,x0(HF,Th),[np(NF)|AC]).
agr2p(agr2p/[Agr2_1],x0(HF,Th),AC,[]) -->
        {Th=[_]},
        agr2_1(Agr2_1,x0(HF,Th),AC).

agr2_1(agr2_1/[agr2_0(Agr2F,Th)/Agr2,VP],x0(HF,Th),[np(NF)|NPs]) -->
        { check_v_features(Agr2F,HF)
        },
        agr2_0(Agr2,Agr2F,Th),
        vp(VP,x0(HF,Th),[np(NF)|NPs]).

vp(vp/[np(NF)/NP,V1],x0(HF,[agt|Ths]),AC) -->
        {theta(NF)===agt,
         select(AC,np(NF1),AC1),
         case(NF1)===c1,
```

```
         check_np_features(NF,NF1)
        },
        np(NP,vspec1,NF),
        {lexical(NF)},
        v1(V1,x0(HF,[agt|Ths]),AC1).
vp(vp/[np(NF)/NP,V1],x0(HF,[pat]),AC) -->
        {theta(NF)===pat,
         select(AC,np(NF1),AC1),
         case(NF1)===c2,
         check_np_features(NF,NF1)
        },
        np(NP,vspec2,NF),
        {lexical(NF)},
        v1(V1,x0(HF,[pat]),AC1).

v1(v1/[v0(VF,[Th1|Ths])/V,VP],x0(HF,[Th1|Ths]),AC) -->
        {check_v_features(VF,HF)},
        v0(V,VF,[Th1|Ths]),
        vp(VP,x0(HF,Ths),AC).
v1(v1/[v0(VF,[Th])/V],x0(HF,[Th]),_AC) -->
        {check_v_features(VF,HF)},
        v0(V,VF,[Th]).

c0([],_,_) --> [].

agr1_0(Aux,Agr1F,_Th) -->
        aux(Aux,agr1,Agr1F).

t0([],_,_) --> [].

asp0(V,AspF,Th) -->
        verb(V,AspF,Th).

agr2_0([],_,_) --> [].

v0([],_,_) --> [].

np([],cspec,_) --> [].

np(Subj,agr1spec,NF) -->
        subject(Subj,NF).

np(Obj,agr2spec,NF) -->
        object(Obj,NF).

np([],vspec1,_) --> [].

np([],vspec2,_) --> [].

subject(['Subj-[c1]'/[]],NF) --> [s-[c1]],
        { case(NF)===c1,
         index(NF,s)
        }.
```

```
object(['Obj-[c2]'/[]],NF) --> [o-[c2]],
        { case(NF)===c2,
         index(NF,o)
        }.

verb(['Verb-[asp]'/[]],VF,Th) --> [V-[asp]],
        {th_grid(V,Th),
         code_features([asp],VF),
         index(VF,V)
        }.

aux(['Aux-[agr,tns]'/[]],agr1,Agr1F) --> [aux-[agr,tns]],
        {code_features([agr,tns],Agr1F)}.

check_v_features(HF1,HF2) :-
        ind(HF1)===ind(HF2),        phi(HF1)===phi(HF2),
        tns(HF1)===tns(HF2),        asp(HF1)===asp(HF2).

check_np_features(NF1,NF2) :-
        ind(NF1)===ind(NF2),        theta(NF1)===theta(NF2),
        case(NF1)===case(NF2),      phi(NF1)===phi(NF2),
        op(NF1)===op(NF2).

th_grid(V,Th) :-
        (       V=iv, Th=[agt]
        ;       V=tv, Th=[agt,pat]
        ).

code_features([F|Fs],VF) :-
        code_feature(F,VF),
        code_features(Fs,VF).
code_features([],_).

code_feature(F,VF) :-
        Feature=..[F,VF],
        Feature === @(F1,F1,_),
        F=F1.

index(Chain,Cat) :-
        ind(Chain) === @(I,I,_),var(I),
        (       (Cat=iv;Cat=tv), I=1
        ;       Cat=s, I=2
        ;       Cat=o, I=3
        ).

lexical(Chain) :-
        ind(Chain)=== @(I,_,_),nonvar(I).
```

Appendix B

Parameter Spaces

B.1

#1 0 0 0 0 0 0 0 0
[s v, s v o]

#2 0 0 0 0 0 0 1 0
[o s v, s v]

#3 0 0 0 0 0 1 0 0
[s v, s v o]

#4 1 0 0 0 0 0 0 0
[v s, v s o]

#5 0 0 0 0 0 0 1 1
[o s v]

#6 0 0 0 0 0 1 0 1
[s v, s v o]

#7 0 0 0 0 0 1 1 0
[s v, s o v]

#8 1 0 0 0 0 0 1 0
[v s, o v s]

#9 1 0 0 0 0 1 0 0
[s v, s v o]

#10 1 1 0 0 0 0 0 0
[v s, v s o]

#11 0 0 0 0 0 1 1 1
[o s v, s v, s o v]

#12 1 0 0 0 0 0 1 1
[o v s]

#13 1 0 0 0 0 1 0 1
[s v, s v o]

#14 1 0 0 0 0 1 1 0
[s v, s o v]

#15 1 1 0 0 0 0 1 0
[v s, v o s]

#16 1 1 0 0 0 1 0 0
[s v, s v o]

#17 1 1 1 0 0 0 0 0
[v s, v s o]

#18 1 0 0 0 0 1 1 1
[o s v, s v, s o v]

#19 1 1 0 0 0 0 1 1
[o v s]

#20 1 1 0 0 0 1 0 1
[s v, s v o]

#21 1 1 0 0 0 1 1 0
[s v, s v o]

#22 1 1 1 0 0 0 1 0
[v s, v o s]

#23 1 1 1 0 0 1 0 0
[s v, s v o]

#24 1 1 1 1 0 0 0 0
[v s, v s o]

#25 1 1 0 0 0 1 1 1
[o s v, s v, s v o]

#26 1 1 1 0 0 0 1 1
[o v s]

#27 1 1 1 0 0 1 0 1
[s v, s v o]

#28 1 1 1 0 0 1 1 0
[s v, s v o]

#29 1 1 1 1 0 0 1 0
[v s, v o s]

#30 1 1 1 1 0 1 0 0

[s v, s v o]

#31 1 1 1 1 1 0 0 0
[v s, v s o]

#32 1 1 1 0 0 1 1 1
[o s v, s v, s v o]

#33 1 1 1 1 0 0 1 1
[o v s]

#34 1 1 1 1 0 1 0 1
[s v, s v o]

#35 1 1 1 1 0 1 1 0
[s v, s v o]

#36 1 1 1 1 1 0 1 0
[v s, v o s]

#37 1 1 1 1 1 1 0 0
[v s, v s o]

#38 1 1 1 1 0 1 1 1
[o s v, s v, s v o]

#39 1 1 1 1 1 0 1 1
[o v s]

#40 1 1 1 1 1 1 0 1
[s v, s v o]

#41 1 1 1 1 1 1 1 0
[v s, v s o]

#42 1 1 1 1 1 1 1 1
[o v s, s v, s v o]

#43 0 0 0 0 0 1/0 0 0
[s v, s v o]

#44 0 0 0 0 0 0 1/0 0
[o s v, s v, s v o]

#45 0 0 0 0 0 0 0 1/0
[s v, s v o]

#46 0 0 0 0 0 1/0 1 0
[s o v, o s v, s v]

#47 0 0 0 0 0 1/0 0 1
[s v, s v o]

#48 0 0 0 0 0 0 1/0 1
[o s v]

#49 0 0 0 0 0 0 1 1/0
[o s v, s v]

#50 0 0 0 0 0 1 1/0 0
[s o v, s v, s v o]

#51 0 0 0 0 0 1 0 1/0
[s v, s v o]

#52 1 0 0 0 0 1/0 0 0
[s v o, s v, v s, v s o]

#53 1 0 0 0 0 0 1/0 0
[o v s, v s, v s o]

#54 1 0 0 0 0 0 0 1/0
[v s, v s o]

#55 0 0 0 0 0 1/0 1 1
[s o v, s v, o s v]

#56 0 0 0 0 0 1 1/0 1
[s o v, o s v, s v, s v o]

#57 0 0 0 0 0 1 1 1/0
[o s v, s v, s o v]

#58 1 0 0 0 0 1/0 1 0
[s o v, s v, v s, o v s]

#59 1 0 0 0 0 1/0 0 1
[s v, s v o]

#60 1 0 0 0 0 0 1/0 1
[o v s]

#61 1 0 0 0 0 0 1 1/0
[v s, o v s]

#62 1 0 0 0 0 1 1/0 0
[s o v, s v, s v o]

#63 1 0 0 0 0 1 0 1/0
[s v, s v o]

#64 1 1 0 0 0 1/0 0 0
[s v o, s v, v s, v s o]

#65 1 1 0 0 0 0 1/0 0
[v o s, v s, v s o]

#66 1 1 0 0 0 0 0 1/0
[v s, v s o]

#67 1 0 0 0 0 1/0 1 1
[s o v, s v, o s v, o v s]

#68 1 0 0 0 0 1 1/0 1
[s o v, o s v, s v, s v o]

#69 1 0 0 0 0 1 1 1/0
[o s v, s v, s o v]

#70 1 1 0 0 0 1/0 1 0
[s v o, s v, v s, v o s]

#71 1 1 0 0 0 1/0 0 1
[s v, s v o]

```
-------------------------------
#72   1 1 0 0 0 0 1/0 1
[o v s]
-------------------------------
#73   1 1 0 0 0 0 1 1/0
[o v s,  v s,  v o s]
-------------------------------
#74   1 1 0 0 0 1 1/0 0
[s v,  s v o]
-------------------------------
#75   1 1 0 0 0 1 0 1/0
[s v,  s v o]
-------------------------------
#76   1 1 1 0 0 1/0 0 0
[s v o,  s v,  v s,  v s o]
-------------------------------
#77   1 1 1 0 0 0 1/0 0
[v o s,  v s,  v s o]
-------------------------------
#78   1 1 1 0 0 0 0 1/0
[v s,  v s o]
-------------------------------
#79   1 1 0 0 0 1/0 1 1
[s v o,  s v,  o s v,  o v s]
-------------------------------
#80   1 1 0 0 0 1 1/0 1
[o s v,  s v,  s v o]
-------------------------------
#81   1 1 0 0 0 1 1 1/0
[o s v,  s v,  s v o]
-------------------------------
#82   1 1 1 0 0 1/0 1 0
[s v o,  s v,  v s,  v o s]
-------------------------------
#83   1 1 1 0 0 1/0 0 1
[s v,  s v o]
-------------------------------
#84   1 1 1 0 0 0 1/0 1
[o v s]
-------------------------------
#85   1 1 1 0 0 0 1 1/0
[o v s,  v s,  v o s]
-------------------------------
#86   1 1 1 0 0 1 1/0 0
[s v,  s v o]
-------------------------------
#87   1 1 1 0 0 1 0 1/0
[s v,  s v o]
-------------------------------
#88   1 1 1 1 0 1/0 0 0
[s v o,  s v,  v s,  v s o]
-------------------------------
#89   1 1 1 1 0 0 1/0 0
[v o s,  v s,  v s o]
-------------------------------
#90   1 1 1 1 0 0 0 1/0
[v s,  v s o]
-------------------------------
#91   1 1 1 0 0 1/0 1 1
[s v o,  s v,  o s v,  o v s]
-------------------------------
#92   1 1 1 0 0 1 1/0 1
[o s v,  s v,  s v o]
-------------------------------
#93   1 1 1 1 0 0 1 1 1/0
[o s v,  s v,  s v o]
-------------------------------
#94   1 1 1 1 1 0 1/0 1 0
[s v o,  s v,  v s,  v o s]
-------------------------------
#95   1 1 1 1 1 0 1/0 0 1
[s v,  s v o]
-------------------------------
#96   1 1 1 1 1 0 0 1/0 1
[o v s]
-------------------------------
#97   1 1 1 1 1 0 0 1 1/0
[o v s,  v s,  v o s]
-------------------------------
#98   1 1 1 1 1 0 1 1/0 0
[s v,  s v o]
-------------------------------
#99   1 1 1 1 1 0 1 0 1/0
[s v,  s v o]
-------------------------------
#100  1 1 1 1 1 1 1/0 0 0
[v s,  v s o]
-------------------------------
#101  1 1 1 1 1 1 0 1/0 0
[v o s,  v s,  v s o]
-------------------------------
#102  1 1 1 1 1 1 0 0 1/0
[v s,  v s o]
-------------------------------
#103  1 1 1 1 1 0 1/0 1 1
[s v o,  s v,  o s v,  o v s]
-------------------------------
#104  1 1 1 1 1 0 1 1/0 1
[o s v,  s v,  s v o]
-------------------------------
#105  1 1 1 1 1 0 1 1 1/0
[o s v,  s v,  s v o]
-------------------------------
#106  1 1 1 1 1 1 1/0 1 0
[v s o,  v s,  v o s]
-------------------------------
#107  1 1 1 1 1 1 1/0 0 1
[s v,  s v o]
-------------------------------
#108  1 1 1 1 1 1 0 1/0 1
[o v s]
-------------------------------
#109  1 1 1 1 1 1 0 1 1/0
[o v s,  v s,  v o s]
-------------------------------
#110  1 1 1 1 1 1 1 1/0 0
[v s,  v s o]
-------------------------------
#111  1 1 1 1 1 1 1 0 1/0
[s v o,  s v,  v s,  v s o]
-------------------------------
#112  1 1 1 1 1 1 1/0 1 1
[s v o,  s v,  o v s]
-------------------------------
```

#113 1 1 1 1 1 1 1/0 1
[o v s, s v, s v o]

#114 1 1 1 1 1 1 1 1/0
[s v o, s v, o v s, v s, v s o]

#115 0 0 0 0 0 1/0 1/0 0
[o s v, s o v, s v, s v o]

#116 0 0 0 0 0 1/0 0 1/0
[s v, s v o]

#117 0 0 0 0 0 0 1/0 1/0
[o s v, s v, s v o]

#118 0 0 0 0 0 1/0 1/0 1
[s v o, s v, s o v, o s v]

#119 0 0 0 0 0 1/0 1 1/0
[s o v, o s v, s v]

#120 0 0 0 0 0 1 1/0 1/0
[o s v, s o v, s v, s v o]

#121 1 0 0 0 0 1/0 1/0 0
[o v s, s o v, s v, s v o, v s,
v s o]

#122 1 0 0 0 0 1/0 0 1/0
[s v o, s v, v s, v s o]

#123 1 0 0 0 0 0 0 1/0 1/0
[o v s, v s, v s o]

#124 1 0 0 0 0 1/0 1/0 1
[s v o, s v, o s v, s o v, o v s]

#125 1 0 0 0 0 1/0 1 1/0
[o s v, s v, s o v, v s, o v s]

#126 1 0 0 0 0 1 1/0 1/0
[o s v, s o v, s v, s v o]

#127 1 1 0 0 0 1/0 1/0 0
[v o s, s v, s v o, v s, v s o]

#128 1 1 0 0 0 1/0 0 1/0
[s v o, s v, v s, v s o]

#129 1 1 0 0 0 0 1/0 1/0
[v o s, o v s, v s, v s o]

#130 1 1 0 0 0 1/0 1/0 1
[s v o, s v, o s v, o v s]

#131 1 1 0 0 0 1/0 1 1/0
[o v s, o s v, s v, s v o, v s,
v o s]

#132 1 1 0 0 0 1 1/0 1/0
[o s v, s v, s v o]

#133 1 1 1 0 0 1/0 1/0 0
[v o s, s v, s v o, v s, v s o]

#134 1 1 1 0 0 1/0 0 1/0
[s v o, s v, v s, v s o]

#135 1 1 1 0 0 0 1/0 1/0
[v o s, o v s, v s, v s o]

#136 1 1 1 0 0 1/0 1/0 1
[s v o, s v, o s v, o v s]

#137 1 1 1 0 0 1/0 1 1/0
[o v s, o s v, s v, s v o, v s,
v o s]

#138 1 1 1 0 0 1 1/0 1/0
[o s v, s v, s v o]

#139 1 1 1 1 0 1/0 1/0 0
[v o s, s v, s v o, v s, v s o]

#140 1 1 1 1 0 1/0 0 1/0
[s v o, s v, v s, v s o]

#141 1 1 1 1 0 0 1/0 1/0
[v o s, o v s, v s, v s o]

#142 1 1 1 1 0 1/0 1/0 1
[s v o, s v, o s v, o v s]

#143 1 1 1 1 0 1/0 1 1/0
[o v s, o s v, s v, s v o, v s,
v o s]

#144 1 1 1 1 0 1 1/0 1/0
[o s v, s v, s v o]

#145 1 1 1 1 1 1/0 1/0 0
[v o s, v s, v s o]

#146 1 1 1 1 1 1/0 0 1/0
[s v, s v o, v s, v s o]

#147 1 1 1 1 1 0 1/0 1/0
[v o s, o v s, v s, v s o]

#148 1 1 1 1 1 1/0 1/0 1
[s v o, s v, o v s]

#149 1 1 1 1 1 1/0 1 1/0
[o v s, s v o, s v, v s o, v s,
v o s]

#150 1 1 1 1 1 1 1/0 1/0
[s v o, s v, o v s, v s, v s o]

#151 0 0 0 0 0 1/0 1/0 1/0
[o s v, s o v, s v, s v o]

#152 1 0 0 0 0 1/0 1/0 1/0
[o v s, s v o, s v, s o v, o s v,

```
 v s,  v s o]
--------------------------------
#153   1  1  0  0  0  1/0  1/0  1/0
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
#154   1  1  1  0  0  1/0  1/0  1/0
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
#155   1  1  1  1  0  1/0  1/0  1/0
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
#156   1  1  1  1  1  1/0  1/0  1/0
[v o s,  s v,  s v o,  o v s,  v s,
 v s o]
--------------------------------
```

B.2

```
#1     0  0  0  0  0  0  0  0
       0  0  0  0  0  1  0  0
       1  0  0  0  0  1  0  0
       0  0  0  0  0  1  0  1
       1  1  0  0  0  1  0  0
       1  0  0  0  0  1  0  1
       1  1  1  0  0  1  0  0
       1  1  0  0  0  1  1  0
       1  1  0  0  0  1  0  1
       1  1  1  1  0  1  0  0
       1  1  1  0  0  1  1  0
       1  1  1  0  0  1  0  1
       1  1  1  1  0  1  1  0
       1  1  1  1  0  1  0  1
       1  1  1  1  1  1  0  1
       0  0  0  0  0  0  0  1/0
       0  0  0  0  0  1/0  0  0
       0  0  0  0  0  1  0  1/0
       0  0  0  0  0  1/0  0  1
       1  0  0  0  0  1  0  1/0
       1  0  0  0  0  1/0  0  1
       1  1  0  0  0  1  0  1/0
       1  1  0  0  0  1  1/0  0
       1  1  0  0  0  1/0  0  1
       1  1  1  0  0  1  0  1/0
       1  1  1  0  0  1  1/0  0
       1  1  1  0  0  1/0  0  1
       1  1  1  1  0  1  0  1/0
       1  1  1  1  0  1  1/0  0
       1  1  1  1  0  1/0  0  1
       1  1  1  1  1  1/0  0  1
       0  0  0  0  0  1/0  0  1/0
[s v,  s v o]
--------------------------------
#2     1  0  0  0  0  0  0  0
       1  1  0  0  0  0  0  0
       1  1  1  0  0  0  0  0
       1  1  1  1  0  0  0  0
       1  1  1  1  1  0  0  0
       1  1  1  1  1  1  0  0
       1  1  1  1  1  1  1  0
       1  0  0  0  0  0  0  1/0
       1  1  0  0  0  0  0  1/0
       1  1  1  0  0  0  0  1/0
       1  1  1  1  0  0  0  1/0
       1  1  1  1  1  0  0  1/0
       1  1  1  1  1  1/0  0  0
       1  1  1  1  1  1  1/0  0
[v s,  v s o]
--------------------------------
#3     0  0  0  0  0  0  1  0
       0  0  0  0  0  0  1  1/0
[o s v,  s v]
--------------------------------
#4     0  0  0  0  0  1  1  0
       1  0  0  0  0  1  1  0
[s v,  s o v]
--------------------------------
#5     1  0  0  0  0  0  1  0
       1  0  0  0  0  0  1  1/0
[v s,  o v s]
--------------------------------
#6     0  0  0  0  0  0  1  1
       0  0  0  0  0  0  1/0  1
[o s v]
--------------------------------
#7     1  0  0  0  0  0  1  1
       1  1  0  0  0  0  1  1
       1  1  1  0  0  0  1  1
       1  1  1  1  0  0  1  1
       1  1  1  1  1  0  1  1
       1  0  0  0  0  0  1/0  1
       1  1  0  0  0  0  1/0  1
       1  1  1  0  0  0  1/0  1
       1  1  1  1  0  0  1/0  1
       1  1  1  1  1  0  1/0  1
[o v s]
--------------------------------
#8     1  1  0  0  0  0  1  0
       1  1  1  0  0  0  1  0
       1  1  1  1  0  0  1  0
       1  1  1  1  1  0  1  0
[v s,  v o s]
--------------------------------
#9     0  0  0  0  0  1  1  1
       1  0  0  0  0  1  1  1
       0  0  0  0  0  1/0  1  0
       0  0  0  0  0  1  1  1/0
       0  0  0  0  0  1/0  1  1
       1  0  0  0  0  1  1  1/0
       0  0  0  0  0  1/0  1  1/0
[s o v,  o s v,  s v]
--------------------------------
#10    1  1  0  0  0  1  1  1
       1  1  1  0  0  1  1  1
       1  1  1  1  0  1  1  1
       0  0  0  0  0  0  1/0  0
       1  1  0  0  0  1  1  1/0
       1  1  0  0  0  1  1/0  1
       1  1  1  0  0  1  1  1/0
       1  1  1  0  0  1  1/0  1
       1  1  1  1  0  1  1  1/0
```

```
      1  1  1  1  0  1  1/0  1
      0  0  0  0  0  0  1/0  1/0
      1  1  0  0  0  1  1/0  1/0
      1  1  1  0  0  1  1/0  1/0
      1  1  1  1  0  1  1/0  1/0
[o s v,  s v,  s v o]
--------------------------------
#11   1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1/0  1
      1  1  1  1  1  1/0  1  1
      1  1  1  1  1  1/0  1/0  1
[s v o,  s v,  o v s]
--------------------------------
#12   1  0  0  0  0  1/0  0  0
      1  1  0  0  0  1/0  0  0
      1  1  1  0  0  1/0  0  0
      1  1  1  1  0  1/0  0  0
      1  1  1  1  1  1  0  1/0
      1  0  0  0  0  1/0  0  1/0
      1  1  0  0  0  1/0  0  1/0
      1  1  1  0  0  1/0  0  1/0
      1  1  1  1  0  1/0  0  1/0
      1  1  1  1  1  1/0  0  1/0
[s v o,  s v,  v s,  v s o]
--------------------------------
#13   0  0  0  0  0  1  1/0  0
      1  0  0  0  0  1  1/0  0
[s o v,  s v,  s v o]
--------------------------------
#14   1  0  0  0  0  0  1/0  0
      1  0  0  0  0  0  1/0  1/0
[o v s,  v s,  v s o]
--------------------------------
#15   1  1  0  0  0  0  1/0  0
      1  1  1  0  0  0  1/0  0
      1  1  1  1  0  0  1/0  0
      1  1  1  1  1  0  1/0  0
      1  1  1  1  1  1/0  1  0
      1  1  1  1  1  1/0  1/0  0
[v o s,  v s,  v s o]
--------------------------------
#16   0  0  0  0  0  1  1/0  1
      1  0  0  0  0  1  1/0  1
      0  0  0  0  0  1/0  1/0  0
      0  0  0  0  0  1  1/0  1/0
      0  0  0  0  0  1/0  1/0  1
      1  0  0  0  0  1  1/0  1/0
      0  0  0  0  0  1/0  1/0  1/0
[o s v,  s o v,  s v,  s v o]
--------------------------------
#17   1  0  0  0  0  1/0  1  0
[s o v,  s v,  v s,  o v s]
--------------------------------
#18   1  1  0  0  0  0  1  1/0
      1  1  1  0  0  0  1  1/0
      1  1  1  1  0  0  1  1/0
      1  1  1  1  1  0  1  1/0
[o v s,  v s,  v o s]
--------------------------------
#19   1  1  0  0  0  1/0  1  0
      1  1  1  0  0  1/0  1  0
      1  1  1  1  0  1/0  1  0
[s v o,  s v,  v s,  v o s]
--------------------------------
#20   1  0  0  0  0  1/0  1  1
[s o v,  s v,  o s v,  o v s]
--------------------------------
#21   1  1  0  0  0  1/0  1  1
      1  1  1  0  0  1/0  1  1
      1  1  1  1  0  1/0  1  1
      1  1  0  0  0  1/0  1/0  1
      1  1  1  0  0  1/0  1/0  1
      1  1  1  1  0  1/0  1/0  1
[s v o,  s v,  o s v,  o v s]
--------------------------------
#22   1  1  1  1  1  1  1  1/0
      1  1  1  1  1  1  1/0  1/0
[s v o,  s v,  o v s,  v s,  v s o]
--------------------------------
#23   1  0  0  0  0  1/0  1/0  0
[o v s,  s o v,  s v,  s v o,  v s,
 v s o]
--------------------------------
#24   1  1  0  0  0  0  1/0  1/0
      1  1  1  0  0  0  1/0  1/0
      1  1  1  1  0  0  1/0  1/0
      1  1  1  1  1  0  1/0  1/0
[v o s,  o v s,  v s,  v s o]
--------------------------------
#25   1  1  0  0  0  1/0  1/0  0
      1  1  1  0  0  1/0  1/0  0
      1  1  1  1  0  1/0  1/0  0
[v o s,  s v,  s v o,  v s,  v s o]
--------------------------------
#26   1  0  0  0  0  1/0  1  1/0
[o s v,  s v,  s o v,  v s,  o v s]
--------------------------------
#27   1  0  0  0  0  1/0  1/0  1
[s v o,  s v,  o s v,  s o v,  o v s]
--------------------------------
#28   1  1  0  0  0  1/0  1  1/0
      1  1  1  0  0  1/0  1  1/0
      1  1  1  1  0  1/0  1  1/0
[o v s,  o s v,  s v,  s v o,  v s,
 v o s]
--------------------------------
#29   1  1  1  1  1  1/0  1  1/0
      1  1  1  1  1  1/0  1/0  1/0
[v o s,  s v,  s v o,  o v s,  v s,
 v s o]
--------------------------------
#30   1  0  0  0  0  1/0  1/0  1/0
[o v s,  s v o,  s v,  s o v,  o s v,
 v s,  v s o]
--------------------------------
#31   1  1  0  0  0  1/0  1/0  1/0
      1  1  1  0  0  1/0  1/0  1/0
      1  1  1  1  0  1/0  1/0  1/0
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
```

B.3

#1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1/0
[(often) s v, (often) s v o]

#2 0 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
0 0 0 0 0 1 0 1
1 1 0 0 0 1 0 0
1 0 0 0 0 1 0 1
1 1 1 0 0 1 0 0
1 1 0 0 0 1 1 0
1 1 0 0 0 1 0 1
1 1 1 0 0 1 1 0
1 1 1 0 0 1 0 1
0 0 0 0 0 1 0 1/0
0 0 0 0 0 1/0 0 1
1 0 0 0 0 1 0 1/0
1 0 0 0 0 1/0 0 1
1 1 0 0 0 1 0 1/0
1 1 0 0 0 1 1/0 0
1 1 0 0 0 1/0 0 1
1 1 1 0 0 1 0 1/0
1 1 1 0 0 1 1/0 0
1 1 1 0 0 1/0 0 1
[s (often) v, s (often) v o]

#3 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 1/0
1 1 0 0 0 0 0 1/0
1 1 1 0 0 0 0 1/0
[(often) v s, (often) v s o]

#4 0 0 0 0 0 0 1 0
[(often) o s v, (often) s v]

#5 0 0 0 0 0 1 1 0
1 0 0 0 0 1 1 0
[s (often) v, s (often) o v]

#6 1 0 0 0 0 0 1 0
[(often) v s, (often) o v s]

#7 0 0 0 0 0 0 1 1
0 0 0 0 0 0 1/0 1
[o (often) s v]

#8 1 0 0 0 0 0 1 1
1 1 0 0 0 0 1 1
1 1 1 0 0 0 1 1
1 0 0 0 0 0 1/0 1
1 1 0 0 0 0 1/0 1
1 1 1 0 0 0 1/0 1
[o (often) v s]

#9 1 1 0 0 0 0 1 0
1 1 1 0 0 0 1 0
[(often) v s, (often) v o s]

#10 0 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1
0 0 0 0 0 1 1 1/0
1 0 0 0 0 1 1 1/0
[o s (often) v, s (often) v,
s (often) o v]

#11 1 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0
1 1 1 1 0 0 0 1/0
1 1 1 1 1 0 0 1/0
[v (often) s, v (often) s o]

#12 1 1 1 1 0 1 0 0
1 1 1 1 0 1 1 0
1 1 1 1 0 1 0 1
1 1 1 1 1 1 0 1
1 1 1 1 0 1 0 1/0
1 1 1 1 0 1 1/0 0
1 1 1 1 0 1/0 0 1
1 1 1 1 1 1/0 0 1
[s v (often), s v (often) o]

#13 1 1 1 1 0 0 1 0
1 1 1 1 1 0 1 0
[v (often) s, v (often) o s]

#14 1 1 0 0 0 1 1 1
1 1 1 0 0 1 1 1
1 1 0 0 0 1 1 1/0
1 1 0 0 0 1 1/0 1
1 1 1 0 0 1 1 1/0
1 1 1 0 0 1 1/0 1
1 1 0 0 0 1 1/0 1/0
1 1 1 0 0 1 1/0 1/0
[o s (often) v, s (often) v,
s (often) v o]

#15 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1/0 0
[v s (often), v s (often) o]

#16 1 1 1 1 0 0 1 1
1 1 1 1 1 0 1 1
1 1 1 1 0 0 1/0 1
1 1 1 1 1 0 1/0 1
[o v (often) s]

#17 1 1 1 1 0 1 1 1
1 1 1 1 0 1 1 1/0
1 1 1 1 0 1 1/0 1
1 1 1 1 0 1 1/0 1/0
[o s v (often), s v (often),
s v (often) o]

#18 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1/0 1
[o v s (often), s v (often),
s v (often) o]

#19 0 0 0 0 0 0 1/0 0

```
[(often) o s v,  (often) s v,
 (often) s v o]
--------------------------------
#20   0  0  0  0  0  1/0  0  0
      0  0  0  0  0  1/0  0  1/0
[s (often) v o,  s (often) v,
 (often) s v,  (often) s v o]
--------------------------------
#21   1  0  0  0  0  1/0  0  0
      1  1  0  0  0  1/0  0  0
      1  1  1  0  0  1/0  0  0
      1  0  0  0  0  1/0  0  1/0
      1  1  0  0  0  1/0  0  1/0
      1  1  1  0  0  1/0  0  1/0
[s (often) v o,  s (often) v,
 (often) v s,  (often) v s o]
--------------------------------
#22   0  0  0  0  0  1  1/0  0
      1  0  0  0  0  1  1/0  0
[s (often) o v,  s (often) v,
 s (often) v o]
--------------------------------
#23   1  0  0  0  0  0  1/0  0
[(often) o v s,  (often) v s,
 (often) v s o]
--------------------------------
#24   0  0  0  0  0  0  1  1/0
[o (often) s v,  (often) o s v,
 (often) s v]
--------------------------------
#25   0  0  0  0  0  1/0  1  0
[s (often) o v,  s (often) v,
 (often) o s v,  (often) s v]
--------------------------------
#26   1  1  0  0  0  0  1/0  0
      1  1  1  0  0  0  1/0  0
[(often) v o s,  (often) v s,
 (often) v s o]
--------------------------------
#27   0  0  0  0  0  1  1/0  1
      1  0  0  0  0  1  1/0  1
      0  0  0  0  0  1  1/0  1/0
      1  0  0  0  0  1  1/0  1/0
[o s (often) v,  s (often) o v,
 s (often) v,  s (often) v o]
--------------------------------
#28   1  0  0  0  0  0  1  1/0
[o (often) v s,  (often) v s,
 (often) o v s]
--------------------------------
#29   1  0  0  0  0  1/0  1  0
[s (often) o v,  s (often) v,
 (often) v s,  (often) o v s]
--------------------------------
#30   0  0  0  0  0  1/0  1  1
[s (often) o v,  s (often) v,
 o s (often) v,  o (often) s v]
--------------------------------
#31   1  1  0  0  0  0  1  1/0
      1  1  1  0  0  0  1  1/0
[o (often) v s,  (often) v s,
 (often) v o s]
--------------------------------
#32   1  1  0  0  0  1/0  1  0
      1  1  1  0  0  1/0  1  0
[s (often) v o,  s (often) v,
 (often) v s,  (often) v o s]
--------------------------------
#33   1  0  0  0  0  1/0  1  1
[s (often) o v,  s (often) v,
 o s (often) v,  o (often) v s]
--------------------------------
#34   1  1  1  1  0  0  1/0  0
      1  1  1  1  1  0  1/0  0
[v (often) o s,  v (often) s,
 v (often) s o]
--------------------------------
#35   1  1  1  1  0  1/0  0  0
      1  1  1  1  0  1/0  0  1/0
[s v (often) o,  s v (often),
 v (often) s,  v (often) s o]
--------------------------------
#36   1  1  0  0  0  1/0  1  1
      1  1  1  0  0  1/0  1  1
      1  1  0  0  0  1/0  1/0  1
      1  1  1  0  0  1/0  1/0  1
[s (often) v o,  s (often) v,
 o s (often) v,  o (often) v s]
--------------------------------
#37   1  1  1  1  0  0  1  1/0
      1  1  1  1  1  0  1  1/0
[o v (often) s,  v (often) s,
 v (often) o s]
--------------------------------
#38   1  1  1  1  1  1/0  0  0
[v s (often) o,  v s (often),
 v (often) s,  v (often) s o]
--------------------------------
#39   1  1  1  1  0  1/0  1  0
[s v (often) o,  s v (often),
 v (often) s,  v (often) o s]
--------------------------------
#40   1  1  1  1  1  1  0  1/0
[s v (often) o,  s v (often),
 v s (often),  v s (often) o]
--------------------------------
#41   1  1  1  1  1  1/0  1  0
[v s (often) o,  v s (often),
 v (often) s,  v (often) o s]
--------------------------------
#42   1  1  1  1  0  1/0  1  1
      1  1  1  1  0  1/0  1/0  1
[s v (often) o,  s v (often),
 o s v (often),  o v (often) s]
--------------------------------
#43   1  1  1  1  1  1  1  1/0
      1  1  1  1  1  1  1/0  1/0
[s v (often) o,  s v (often),
 o v s (often),  v s (often),
 v s (often) o]
--------------------------------
#44   1  1  1  1  1  1/0  1  1
      1  1  1  1  1  1/0  1/0  1
[s v (often) o,  s v (often),
```

```
 o v s (often),  o v (often) s]
---------------------------------
#45   0  0  0  0  0  0  1/0  1/0
[(often) o s v,  o (often) s v,
 (often) s v,  (often) s v o]
---------------------------------
#46   0  0  0  0  0  1/0  1/0  0
[(often) o s v,  s (often) o v,
 s (often) v,  s (often) v o,
 (often) s v,  (often) s v o]
---------------------------------
#47   1  0  0  0  0  0  1/0  1/0
[(often) o v s,  o (often) v s,
 (often) v s,  (often) v s o]
---------------------------------
#48   1  0  0  0  0  1/0  1/0  0
[(often) o v s,  s (often) o v,
 s (often) v,  s (often) v o,
 (often) v s,  (often) v s o]
---------------------------------
#49   0  0  0  0  0  1/0  1  1/0
[o (often) s v,  o s (often) v,
 s (often) v,  s (often) o v,
 (often) o s v,  (often) s v]
---------------------------------
#50   0  0  0  0  0  1/0  1/0  1
[s (often) v o,  s (often) v,
 o s (often) v,  s (often) o v,
 o (often) s v]
---------------------------------
#51   1  1  0  0  0  0  1/0  1/0
      1  1  1  0  0  0  1/0  1/0
[(often) v o s,  o (often) v s,
 (often) v s,  (often) v s o]
---------------------------------
#52   1  1  0  0  0  1/0  1/0  0
      1  1  1  0  0  1/0  1/0  0
[(often) v o s,  s (often) v,
 s (often) v o,  (often) v s,
 (often) v s o]
---------------------------------
#53   1  0  0  0  0  1/0  1  1/0
[o (often) v s,  o s (often) v,
 s (often) v,  s (often) o v,
 (often) v s,  (often) o v s]
---------------------------------
#54   1  0  0  0  0  1/0  1/0  1
[s (often) v o,  s (often) v,
 o s (often) v,  s (often) o v,
 o (often) v s]
---------------------------------
#55   1  1  0  0  0  1/0  1  1/0
      1  1  1  0  0  1/0  1  1/0
[o (often) v s,  o s (often) v,
 s (often) v,  s (often) v o,
 (often) v s,  (often) v o s]
---------------------------------
#56   1  1  1  1  0  0  1/0  1/0
      1  1  1  1  1  0  1/0  1/0
[v (often) o s,  o v (often) s,
 v (often) s,  v (often) s o]
---------------------------------
#57   1  1  1  1  0  1/0  1/0  0
[v (often) o s,  s v (often),
 s v (often) o,  v (often) s,
 v (often) s o]
---------------------------------
#58   1  1  1  1  1  1/0  0  1/0
[v s (often) o,  v s (often),
 s v (often),  s v (often) o,
 v (often) s,  v (often) s o]
---------------------------------
#59   1  1  1  1  1  1/0  1/0  0
[v (often) o s,  v s (often),
 v s (often) o,  v (often) s,
 v (often) s o]
---------------------------------
#60   1  1  1  1  0  1/0  1  1/0
[o v (often) s,  o s v (often),
 s v (often),  s v (often) o,
 v (often) s,  v (often) o s]
---------------------------------
#61   1  1  1  1  1  1/0  1  1/0
[o v (often) s,  s v (often) o,
 s v (often),  o v s (often),
 v s (often),  v s (often) o,
 v (often) s,  v (often) o s]
---------------------------------
#62   0  0  0  0  0  1/0  1/0  1/0
[(often) o s v,  s (often) v o,
 s (often) v,  s (often) o v,
 o s (often) v,  o (often) s v,
 (often) s v,  (often) s v o]
---------------------------------
#63   1  0  0  0  0  1/0  1/0  1/0
[(often) o v s,  s (often) v o,
 s (often) v,  s (often) o v,
 o s (often) v,  o (often) v s,
 (often) v s,  (often) v s o]
---------------------------------
#64   1  1  0  0  0  1/0  1/0  1/0
      1  1  1  0  0  1/0  1/0  1/0
[(often) v o s,  s (often) v o,
 s (often) v,  o s (often) v,
 o (often) v s,  (often) v s,
 (often) v s o]
---------------------------------
#65   1  1  1  1  0  1/0  1/0  1/0
[v (often) o s,  s v (often) o,
 s v (often),  o s v (often),
 o v (often) s,  v (often) s,
 v (often) s o]
---------------------------------
#66   1  1  1  1  1  1/0  1/0  1/0
[v (often) o s,  v s (often) o,
 v s (often),  o v s (often),
 s v (often),  s v (often) o,
 o v (often) s,  v (often) s,
 v (often) s o]
---------------------------------
```

B.4

```
#1
0  0  0  0  0  0  0  0 , i  i
0  0  0  0  0  0  0  0 , i  f
0  0  0  0  0  0  0  0 , f  i
0  0  0  0  0  0  0  0 , f  f
0  0  0  0  0  1  0  0 , i  i
0  0  0  0  0  1  0  0 , i  f
0  0  0  0  0  1  0  0 , f  i
0  0  0  0  0  1  0  0 , f  f
1  0  0  0  0  1  0  0 , i  i
1  0  0  0  0  1  0  0 , f  i
0  0  0  0  0  1  0  1 , i  i
0  0  0  0  0  1  0  1 , i  f
0  0  0  0  0  1  0  1 , f  i
0  0  0  0  0  1  0  1 , f  f
1  1  0  0  0  1  0  0 , i  i
1  1  0  0  0  1  0  0 , f  i
1  0  0  0  0  1  0  1 , i  i
1  0  0  0  0  1  0  1 , f  i
1  1  1  0  0  1  0  0 , i  i
1  1  1  0  0  1  0  0 , f  i
1  1  0  0  0  1  1  0 , i  i
1  1  0  0  0  1  1  0 , f  i
1  1  0  0  0  1  0  1 , i  i
1  1  0  0  0  1  0  1 , f  i
1  1  1  1  0  1  0  0 , i  i
1  1  1  1  0  1  0  0 , f  i
1  1  1  0  0  1  1  0 , i  i
1  1  1  0  0  1  1  0 , f  i
1  1  1  0  0  1  0  1 , i  i
1  1  1  0  0  1  0  1 , f  i
1  1  1  1  0  1  1  0 , i  i
1  1  1  1  0  1  1  0 , f  i
1  1  1  1  0  1  0  1 , i  i
1  1  1  1  0  1  0  1 , f  i
1  1  1  1  1  1  0  1 , i  i
1  1  1  1  1  1  0  1 , i  f
0  0  0  0  0  0  0 1/0 , i  i
0  0  0  0  0  0  0 1/0 , i  f
0  0  0  0  0  0  0 1/0 , f  i
0  0  0  0  0  0  0 1/0 , f  f
0  0  0  0  0  1/0  0  0 , i  i
0  0  0  0  0  1/0  0  0 , i  f
0  0  0  0  0  1/0  0  0 , f  i
0  0  0  0  0  1/0  0  0 , f  f
0  0  0  0  0  1  0 1/0 , i  i
0  0  0  0  0  1  0 1/0 , i  f
0  0  0  0  0  1  0 1/0 , f  i
0  0  0  0  0  1  0 1/0 , f  f
0  0  0  0  0  1/0  0  1 , i  i
0  0  0  0  0  1/0  0  1 , i  f
0  0  0  0  0  1/0  0  1 , f  i
0  0  0  0  0  1/0  0  1 , f  f
1  0  0  0  0  1  0 1/0 , i  i
1  0  0  0  0  1  0 1/0 , f  i
1  0  0  0  0  1/0  0  1 , i  i
1  0  0  0  0  1/0  0  1 , f  i
1  1  0  0  0  1  0 1/0 , i  i
1  1  0  0  0  1  0 1/0 , f  i
1  1  0  0  0  1  1/0  0 , i  i
1  1  0  0  0  1  1/0  0 , f  i
1  1  0  0  0  1/0  0  1 , i  i
1  1  0  0  0  1/0  0  1 , i  f
1  1  0  0  0  1/0  0  1 , f  i
1  1  0  0  0  1/0  0  1 , f  f
1  1  1  0  0  1  0 1/0 , i  i
1  1  1  0  0  1  0 1/0 , f  i
1  1  1  0  0  1  1/0  0 , i  i
1  1  1  0  0  1  1/0  0 , f  i
1  1  1  0  0  1/0  0  1 , i  i
1  1  1  0  0  1/0  0  1 , i  f
1  1  1  0  0  1/0  0  1 , f  i
1  1  1  0  0  1/0  0  1 , f  f
1  1  1  1  0  1  0 1/0 , i  i
1  1  1  1  0  1  0 1/0 , f  i
1  1  1  1  0  1  1/0  0 , i  i
1  1  1  1  0  1  1/0  0 , f  i
1  1  1  1  0  1/0  0  1 , i  i
1  1  1  1  0  1/0  0  1 , f  i
1  1  1  1  1  1/0  0  1 , i  i
1  1  1  1  1  1/0  0  1 , i  f
0  0  0  0  0  1/0  0 1/0 , i  i
0  0  0  0  0  1/0  0 1/0 , i  f
0  0  0  0  0  1/0  0 1/0 , f  i
0  0  0  0  0  1/0  0 1/0 , f  f
[s v,  s v o]
--------------------------------
#2
1  0  0  0  0  0  0  0 , i  i
1  0  0  0  0  0  0  0 , f  i
1  1  0  0  0  0  0  0 , i  i
1  1  0  0  0  0  0  0 , f  i
1  1  1  0  0  0  0  0 , i  i
1  1  1  0  0  0  0  0 , f  i
1  1  1  1  0  0  0  0 , i  i
1  1  1  1  0  0  0  0 , f  i
1  1  1  1  1  0  0  0 , i  i
1  1  1  1  1  0  0  0 , i  f
1  1  1  1  1  1  0  0 , i  i
1  1  1  1  1  1  0  0 , i  f
1  1  1  1  1  1  1  0 , i  i
1  1  1  1  1  1  1  0 , i  f
1  0  0  0  0  0  0 1/0 , i  i
1  0  0  0  0  0  0 1/0 , f  i
1  1  0  0  0  0  0 1/0 , i  i
1  1  0  0  0  0  0 1/0 , f  i
1  1  1  0  0  0  0 1/0 , i  i
1  1  1  0  0  0  0 1/0 , f  i
1  1  1  1  0  0  0 1/0 , i  i
1  1  1  1  0  0  0 1/0 , f  i
1  1  1  1  1  0  0 1/0 , i  i
1  1  1  1  1  0  0 1/0 , i  f
1  1  1  1  1  1/0  0  0 , i  i
1  1  1  1  1  1/0  0  0 , i  f
1  1  1  1  1  1  1/0  0 , i  i
1  1  1  1  1  1  1/0  0 , i  f
[v s,  v s o]
--------------------------------
#3
1  0  0  0  0  0  0  0 , i  f
1  0  0  0  0  0  0  0 , f  f
1  1  0  0  0  0  0  0 , i  f
1  1  0  0  0  0  0  0 , f  f
1  0  0  0  0  1  0  0 , i  f
1  0  0  0  0  1  0  0 , f  f
0  0  0  0  0  1  1  0 , i  i
```

```
0 0 0 0 0 1 1 0 , i f
0 0 0 0 0 1 1 0 , f i
0 0 0 0 0 1 1 0 , f f
1 1 1 0 0 0 0 0 , i f
1 1 1 0 0 0 0 0 , f f
1 1 0 0 0 1 0 0 , i f
1 1 0 0 0 1 0 0 , f f
1 0 0 0 0 1 1 0 , i i
1 0 0 0 0 1 1 0 , i f
1 0 0 0 0 1 1 0 , f i
1 0 0 0 0 1 1 0 , f f
1 0 0 0 0 1 0 1 , i f
1 0 0 0 0 1 0 1 , f f
1 1 1 1 0 0 0 0 , i f
1 1 1 1 0 0 0 0 , f f
1 1 1 0 0 1 0 0 , i f
1 1 1 0 0 1 0 0 , f f
1 1 0 0 0 1 1 0 , i f
1 1 0 0 0 1 1 0 , f f
1 1 0 0 0 1 0 1 , i f
1 1 0 0 0 1 0 1 , f f
1 1 1 1 1 0 0 0 , f i
1 1 1 1 1 0 0 0 , f f
1 1 1 1 0 1 0 0 , i f
1 1 1 1 0 1 0 0 , f f
1 1 1 0 0 1 1 0 , i f
1 1 1 0 0 1 1 0 , f f
1 1 1 0 0 1 0 1 , i f
1 1 1 0 0 1 0 1 , f f
1 1 1 1 1 1 0 0 , f i
1 1 1 1 1 1 0 0 , f f
1 1 1 1 0 1 1 0 , i f
1 1 1 1 0 1 1 0 , f f
1 1 1 1 0 1 0 1 , i f
1 1 1 1 0 1 0 1 , f f
1 1 1 1 1 1 1 0 , f i
1 1 1 1 1 1 1 0 , f f
1 1 1 1 1 1 0 1 , f i
1 1 1 1 1 1 0 1 , f f
1 0 0 0 0 0 0 1/0 , i f
1 0 0 0 0 0 0 1/0 , f f
1 0 0 0 0 1/0 0 0 , i f
1 0 0 0 0 1/0 0 0 , f f
1 1 0 0 0 0 0 1/0 , i f
1 1 0 0 0 0 0 1/0 , f f
1 1 0 0 0 1/0 0 0 , i f
1 1 0 0 0 1/0 0 0 , f f
1 0 0 0 0 1 0 1/0 , i f
1 0 0 0 0 1 0 1/0 , f f
1 0 0 0 0 1 1/0 0 , i f
1 0 0 0 0 1 1/0 0 , f f
1 0 0 0 0 1/0 0 1 , i f
1 0 0 0 0 1/0 0 1 , f f
1 1 1 0 0 0 0 1/0 , i f
1 1 1 0 0 0 0 1/0 , f f
1 1 1 0 0 1/0 0 0 , i f
1 1 1 0 0 1/0 0 0 , f f
1 1 0 0 0 1 0 1/0 , i f
1 1 0 0 0 1 0 1/0 , f f
1 1 0 0 0 1 1/0 0 , i f
1 1 0 0 0 1 1/0 0 , f f
1 1 0 0 0 1/0 0 1 , i f
1 1 0 0 0 1/0 0 1 , f f
1 1 1 1 0 0 0 1/0 , i f
1 1 1 1 0 0 0 1/0 , f f
1 1 1 1 0 1/0 0 0 , i f
1 1 1 1 0 1/0 0 0 , f f
1 1 1 0 0 1 0 1/0 , i f
1 1 1 0 0 1 0 1/0 , f f
1 1 1 0 0 1 1/0 0 , i f
1 1 1 0 0 1 1/0 0 , f f
1 1 1 0 0 1/0 0 1 , i f
1 1 1 0 0 1/0 0 1 , f f
1 1 1 1 1 0 0 1/0 , f i
1 1 1 1 1 0 0 1/0 , f f
1 1 1 1 1 1/0 0 0 , f i
1 1 1 1 1 1/0 0 0 , f f
1 1 1 1 0 1 0 1/0 , i f
1 1 1 1 0 1 0 1/0 , f f
1 1 1 1 0 1 1/0 0 , i f
1 1 1 1 0 1 1/0 0 , f f
1 1 1 1 0 1/0 0 1 , i f
1 1 1 1 0 1/0 0 1 , f f
1 1 1 1 1 1 0 1/0 , f i
1 1 1 1 1 1 0 1/0 , f f
1 1 1 1 1 1 1/0 0 , f i
1 1 1 1 1 1 1/0 0 , f f
1 1 1 1 1 1/0 0 1 , f i
1 1 1 1 1 1/0 0 1 , f f
1 0 0 0 0 1/0 0 1/0 , i f
1 0 0 0 0 1/0 0 1/0 , f f
1 1 0 0 0 1/0 0 1/0 , i f
1 1 0 0 0 1/0 0 1/0 , f f
1 1 1 0 0 1/0 0 1/0 , i f
1 1 1 0 0 1/0 0 1/0 , f f
1 1 1 1 0 1/0 0 1/0 , i f
1 1 1 1 0 1/0 0 1/0 , f f
1 1 1 1 1 1/0 0 1/0 , f i
1 1 1 1 1 1/0 0 1/0 , f f
[s v, s o v]
--------------------------------
#4
0 0 0 0 0 0 1 0 , i i
0 0 0 0 0 0 1 0 , i f
0 0 0 0 0 0 1 0 , f i
0 0 0 0 0 0 1 0 , f f
1 0 0 0 0 0 1 0 , i f
1 0 0 0 0 0 1 0 , f f
1 1 0 0 0 0 1 0 , i f
1 1 0 0 0 0 1 0 , f f
1 1 1 0 0 0 1 0 , i f
1 1 1 0 0 0 1 0 , f f
1 1 1 1 0 0 1 0 , i f
1 1 1 1 0 0 1 0 , f f
1 1 1 1 1 0 1 0 , f i
1 1 1 1 1 0 1 0 , f f
0 0 0 0 0 0 1 1/0 , i i
0 0 0 0 0 0 1 1/0 , i f
0 0 0 0 0 0 1 1/0 , f i
0 0 0 0 0 0 1 1/0 , f f
1 0 0 0 0 0 1 1/0 , i f
1 0 0 0 0 0 1 1/0 , f f
1 1 0 0 0 0 1 1/0 , i f
1 1 0 0 0 0 1 1/0 , f f
```

```
1  1  1  0  0  0  1  1/0 , i  f
1  1  1  0  0  0  1  1/0 , f  f
1  1  1  1  0  0  1  1/0 , i  f
1  1  1  1  0  0  1  1/0 , f  f
1  1  1  1  1  0  1  1/0 , f  i
1  1  1  1  1  0  1  1/0 , f  f
[o s v,  s v]
--------------------------------
#5
0  0  0  0  0  0  1  1 , i  i
0  0  0  0  0  0  1  1 , i  f
0  0  0  0  0  0  1  1 , f  i
0  0  0  0  0  0  1  1 , f  f
1  0  0  0  0  0  1  1 , i  f
1  0  0  0  0  0  1  1 , f  f
1  1  0  0  0  0  1  1 , i  f
1  1  0  0  0  0  1  1 , f  f
1  1  1  0  0  0  1  1 , i  f
1  1  1  0  0  0  1  1 , f  f
1  1  1  1  0  0  1  1 , i  f
1  1  1  1  0  0  1  1 , f  f
1  1  1  1  1  0  1  1 , f  i
1  1  1  1  1  0  1  1 , f  f
0  0  0  0  0  0  1/0  1 , i  i
0  0  0  0  0  0  1/0  1 , i  f
0  0  0  0  0  0  1/0  1 , f  i
0  0  0  0  0  0  1/0  1 , f  f
1  0  0  0  0  0  1/0  1 , i  f
1  0  0  0  0  0  1/0  1 , f  f
1  1  0  0  0  0  1/0  1 , i  f
1  1  0  0  0  0  1/0  1 , f  f
1  1  1  0  0  0  1/0  1 , i  f
1  1  1  0  0  0  1/0  1 , f  f
1  1  1  1  0  0  1/0  1 , i  f
1  1  1  1  0  0  1/0  1 , f  f
1  1  1  1  1  0  1/0  1 , f  i
1  1  1  1  1  0  1/0  1 , f  f
[o s v]
--------------------------------
#6
1  0  0  0  0  0  1  0 , i  i
1  0  0  0  0  0  1  0 , f  i
1  0  0  0  0  0  1  1/0 , i  i
1  0  0  0  0  0  1  1/0 , f  i
[v s,  o v s]
--------------------------------
#7
0  0  0  0  0  1  1  1 , i  i
0  0  0  0  0  1  1  1 , i  f
0  0  0  0  0  1  1  1 , f  i
0  0  0  0  0  1  1  1 , f  f
1  0  0  0  0  1  1  1 , i  i
1  0  0  0  0  1  1  1 , i  f
1  0  0  0  0  1  1  1 , f  i
1  0  0  0  0  1  1  1 , f  f
1  1  0  0  0  1  1  1 , i  f
1  1  0  0  0  1  1  1 , f  f
1  1  1  0  0  1  1  1 , i  f
1  1  1  0  0  1  1  1 , f  f
1  1  1  1  0  1  1  1 , i  f
1  1  1  1  0  1  1  1 , f  f
1  1  1  1  1  1  1  1 , f  i

1  1  1  1  1  1  1  1 , f  f
1  0  0  0  0  0  1/0  0 , i  f
1  0  0  0  0  0  1/0  0 , f  f
0  0  0  0  0  1/0  1  0 , i  i
0  0  0  0  0  1/0  1  0 , i  f
0  0  0  0  0  1/0  1  0 , f  i
0  0  0  0  0  1/0  1  0 , f  f
1  1  0  0  0  0  1/0  0 , i  f
1  1  0  0  0  0  1/0  0 , f  f
1  0  0  0  0  1/0  1  0 , i  f
1  0  0  0  0  1/0  1  0 , f  f
0  0  0  0  0  1  1  1/0 , i  i
0  0  0  0  0  1  1  1/0 , i  f
0  0  0  0  0  1  1  1/0 , f  i
0  0  0  0  0  1  1  1/0 , f  f
0  0  0  0  0  1/0  1  1 , i  i
0  0  0  0  0  1/0  1  1 , i  f
0  0  0  0  0  1/0  1  1 , f  i
0  0  0  0  0  1/0  1  1 , f  f
1  1  1  0  0  0  1/0  0 , i  f
1  1  1  0  0  0  1/0  0 , f  f
1  1  0  0  0  1/0  1  0 , i  f
1  1  0  0  0  1/0  1  0 , f  f
1  0  0  0  0  1  1  1/0 , i  i
1  0  0  0  0  1  1  1/0 , i  f
1  0  0  0  0  1  1  1/0 , f  i
1  0  0  0  0  1  1  1/0 , f  f
1  0  0  0  0  1  1/0  1 , i  f
1  0  0  0  0  1  1/0  1 , f  f
1  0  0  0  0  1/0  1  1 , i  f
1  0  0  0  0  1/0  1  1 , f  f
1  1  1  1  0  0  1/0  0 , i  f
1  1  1  1  0  0  1/0  0 , f  f
1  1  1  0  0  1/0  1  0 , i  f
1  1  1  0  0  1/0  1  0 , f  f
1  1  0  0  0  1  1  1/0 , i  f
1  1  0  0  0  1  1  1/0 , f  f
1  1  0  0  0  1  1/0  1 , i  f
1  1  0  0  0  1  1/0  1 , f  f
1  1  0  0  0  1/0  1  1 , i  f
1  1  0  0  0  1/0  1  1 , f  f
1  1  1  1  1  0  1/0  0 , f  i
1  1  1  1  1  0  1/0  0 , f  f
1  1  1  1  0  1/0  1  0 , i  f
1  1  1  1  0  1/0  1  0 , f  f
1  1  1  0  0  1  1  1/0 , i  f
1  1  1  0  0  1  1  1/0 , f  f
1  1  1  0  0  1  1/0  1 , i  f
1  1  1  0  0  1  1/0  1 , f  f
1  1  1  0  0  1/0  1  1 , i  f
1  1  1  0  0  1/0  1  1 , f  f
1  1  1  1  1  0  1/0  1 , f  i
1  1  1  1  1  0  1/0  1 , f  f
1  1  1  1  0  1  1  1/0 , i  f
1  1  1  1  0  1  1  1/0 , f  f
1  1  1  1  0  1  1/0  1 , i  f
1  1  1  1  0  1  1/0  1 , f  f
1  1  1  1  0  1/0  1  1 , i  f
1  1  1  1  0  1/0  1  1 , f  f
1  1  1  1  1  1  1  1/0 , f  i
1  1  1  1  1  1  1  1/0 , f  f
1  1  1  1  1  1  1/0  1 , f  i
```

```
1  1  1  1  1  1  1/0  1 , f  f
1  1  1  1  1  1/0  1  1 , f  i
1  1  1  1  1  1/0  1  1 , f  f
1  0  0  0  0  0  1/0  1/0 , i  f
1  0  0  0  0  0  1/0  1/0 , f  f
1  0  0  0  0  1/0  1/0  0 , i  f
1  0  0  0  0  1/0  1/0  0 , f  f
0  0  0  0  0  1/0  1  1/0 , i  i
0  0  0  0  0  1/0  1  1/0 , i  f
0  0  0  0  0  1/0  1  1/0 , f  i
0  0  0  0  0  1/0  1  1/0 , f  f
1  1  0  0  0  0  1/0  1/0 , i  f
1  1  0  0  0  0  1/0  1/0 , f  f
1  1  0  0  0  1/0  1/0  0 , i  f
1  1  0  0  0  1/0  1/0  0 , f  f
1  0  0  0  0  1  1/0  1/0 , i  f
1  0  0  0  0  1  1/0  1/0 , f  f
1  0  0  0  0  1/0  1  1/0 , i  f
1  0  0  0  0  1/0  1  1/0 , f  f
1  0  0  0  0  1/0  1/0  1 , i  f
1  0  0  0  0  1/0  1/0  1 , f  f
1  1  1  0  0  0  1/0  1/0 , i  f
1  1  1  0  0  0  1/0  1/0 , f  f
1  1  1  0  0  1/0  1/0  0 , i  f
1  1  1  0  0  1/0  1/0  0 , f  f
1  1  0  0  0  1  1/0  1/0 , i  f
1  1  0  0  0  1  1/0  1/0 , f  f
1  1  0  0  0  1/0  1  1/0 , i  f
1  1  0  0  0  1/0  1  1/0 , f  f
1  1  0  0  0  1/0  1/0  1 , i  f
1  1  0  0  0  1/0  1/0  1 , f  f
1  1  1  1  0  0  1/0  1/0 , i  f
1  1  1  1  0  0  1/0  1/0 , f  f
1  1  1  1  0  1/0  1/0  0 , i  f
1  1  1  1  0  1/0  1/0  0 , f  f
1  1  1  0  0  1  1/0  1/0 , i  f
1  1  1  0  0  1  1/0  1/0 , f  f
1  1  1  0  0  1/0  1  1/0 , i  f
1  1  1  0  0  1/0  1  1/0 , f  f
1  1  1  0  0  1/0  1/0  1 , i  f
1  1  1  0  0  1/0  1/0  1 , f  f
1  1  1  1  1  0  1/0  1/0 , f  i
1  1  1  1  1  0  1/0  1/0 , f  f
1  1  1  1  1  1/0  1/0  0 , f  i
1  1  1  1  1  1/0  1/0  0 , f  f
1  1  1  1  0  1  1/0  1/0 , i  f
1  1  1  1  0  1  1/0  1/0 , f  f
1  1  1  1  0  1/0  1  1/0 , i  f
1  1  1  1  0  1/0  1  1/0 , f  f
1  1  1  1  0  1/0  1/0  1 , i  f
1  1  1  1  0  1/0  1/0  1 , f  f
1  1  1  1  1  1  1/0  1/0 , f  i
1  1  1  1  1  1  1/0  1/0 , f  f
1  1  1  1  1  1/0  1  1/0 , f  i
1  1  1  1  1  1/0  1  1/0 , f  f
1  1  1  1  1  1/0  1/0  1 , f  i
1  1  1  1  1  1/0  1/0  1 , f  f
1  0  0  0  0  1/0  1/0  1/0 , i  f
1  0  0  0  0  1/0  1/0  1/0 , f  f
1  1  0  0  0  1/0  1/0  1/0 , i  f
1  1  0  0  0  1/0  1/0  1/0 , f  f
1  1  1  0  0  1/0  1/0  1/0 , i  f
```

```
1  1  1  0  0  1/0  1/0  1/0 , f  f
1  1  1  1  0  1/0  1/0  1/0 , i  f
1  1  1  1  0  1/0  1/0  1/0 , f  f
1  1  1  1  1  1/0  1/0  1/0 , f  i
1  1  1  1  1  1/0  1/0  1/0 , f  f
[s o v,  o s v,  s v]
---------------------------------
#8
1  0  0  0  0  0  1  1 , i  i
1  0  0  0  0  0  1  1 , f  i
1  1  0  0  0  0  1  1 , i  i
1  1  0  0  0  0  1  1 , f  i
1  1  1  0  0  0  1  1 , i  i
1  1  1  0  0  0  1  1 , f  i
1  1  1  1  0  0  1  1 , i  i
1  1  1  1  0  0  1  1 , f  i
1  1  1  1  1  0  1  1 , i  i
1  1  1  1  1  0  1  1 , i  f
1  0  0  0  0  0  1/0  1 , i  i
1  0  0  0  0  0  1/0  1 , f  i
1  1  0  0  0  0  1/0  1 , i  i
1  1  0  0  0  0  1/0  1 , f  i
1  1  1  0  0  0  1/0  1 , i  i
1  1  1  0  0  0  1/0  1 , f  i
1  1  1  1  0  0  1/0  1 , i  i
1  1  1  1  0  0  1/0  1 , f  i
1  1  1  1  1  0  1/0  1 , i  i
1  1  1  1  1  0  1/0  1 , i  f
[o v s]
---------------------------------
#9
1  1  0  0  0  0  1  0 , i  i
1  1  0  0  0  0  1  0 , f  i
1  1  1  0  0  0  1  0 , i  i
1  1  1  0  0  0  1  0 , f  i
1  1  1  1  0  0  1  0 , i  i
1  1  1  1  0  0  1  0 , f  i
1  1  1  1  1  0  1  0 , i  i
1  1  1  1  1  0  1  0 , i  f
[v s,  v o s]
---------------------------------
#10
1  1  0  0  0  1  1  1 , i  i
1  1  0  0  0  1  1  1 , f  i
1  1  1  0  0  1  1  1 , i  i
1  1  1  0  0  1  1  1 , f  i
1  1  1  1  0  1  1  1 , i  i
1  1  1  1  0  1  1  1 , f  i
0  0  0  0  0  0  1/0  0 , i  i
0  0  0  0  0  0  1/0  0 , i  f
0  0  0  0  0  0  1/0  0 , f  i
0  0  0  0  0  0  1/0  0 , f  f
1  1  0  0  0  1  1  1/0 , i  i
1  1  0  0  0  1  1  1/0 , f  i
1  1  0  0  0  1  1/0  1 , i  i
1  1  0  0  0  1  1/0  1 , f  i
1  1  1  0  0  1  1  1/0 , i  i
1  1  1  0  0  1  1  1/0 , f  i
1  1  1  0  0  1  1/0  1 , i  i
1  1  1  0  0  1  1/0  1 , f  i
1  1  1  1  0  1  1  1/0 , i  i
1  1  1  1  0  1  1  1/0 , f  i
```

```
1  1  1  1  0  1  1/0  1 , i  i
1  1  1  1  0  1  1/0  1 , f  i
0  0  0  0  0  0  1/0  1/0 , i  i
0  0  0  0  0  0  1/0  1/0 , i  f
0  0  0  0  0  0  1/0  1/0 , f  i
0  0  0  0  0  0  1/0  1/0 , f  f
1  1  0  0  0  1  1/0  1/0 , i  i
1  1  0  0  0  1  1/0  1/0 , f  i
1  1  1  0  0  1  1/0  1/0 , i  i
1  1  1  0  0  1  1/0  1/0 , f  i
1  1  1  1  0  1  1/0  1/0 , i  i
1  1  1  1  0  1  1/0  1/0 , f  i
[o s v,  s v,  s v o]
--------------------------------
#11
1  1  1  1  1  1  1  1 , i  i
1  1  1  1  1  1  1  1 , i  f
1  1  1  1  1  1  1/0  1 , i  i
1  1  1  1  1  1  1/0  1 , i  f
1  1  1  1  1  1/0  1  1 , i  i
1  1  1  1  1  1/0  1  1 , i  f
1  1  1  1  1  1/0  1/0  1 , i  i
1  1  1  1  1  1/0  1/0  1 , i  f
[s v o,  s v,  o v s]
--------------------------------
#12
1  0  0  0  0  1/0  0  0 , i  i
1  0  0  0  0  1/0  0  0 , f  i
1  1  0  0  0  1/0  0  0 , i  i
1  1  0  0  0  1/0  0  0 , f  i
1  1  1  0  0  1/0  0  0 , i  i
1  1  1  0  0  1/0  0  0 , f  i
1  1  1  1  0  1/0  0  0 , i  i
1  1  1  1  0  1/0  0  0 , f  i
1  1  1  1  1  1  0  1/0 , i  i
1  1  1  1  1  1  0  1/0 , i  f
1  0  0  0  0  1/0  0  1/0 , i  i
1  0  0  0  0  1/0  0  1/0 , f  i
1  1  0  0  0  1/0  0  1/0 , i  i
1  1  0  0  0  1/0  0  1/0 , f  i
1  1  1  0  0  1/0  0  1/0 , i  i
1  1  1  0  0  1/0  0  1/0 , f  i
1  1  1  1  0  1/0  0  1/0 , i  i
1  1  1  1  0  1/0  0  1/0 , f  i
1  1  1  1  1  1/0  0  1/0 , i  i
1  1  1  1  1  1/0  0  1/0 , i  f
[s v o,  s v,  v s,  v s o]
--------------------------------
13
0  0  0  0  0  1  1/0  0 , i  i
0  0  0  0  0  1  1/0  0 , i  f
0  0  0  0  0  1  1/0  0 , f  i
0  0  0  0  0  1  1/0  0 , f  f
1  0  0  0  0  1  1/0  0 , i  i
1  0  0  0  0  1  1/0  0 , f  i
[s o v,  s v,  s v o]
--------------------------------
#14
1  0  0  0  0  0  1/0  0 , i  i
1  0  0  0  0  0  1/0  0 , f  i
1  0  0  0  0  0  1/0  1/0 , i  i
1  0  0  0  0  0  1/0  1/0 , f  i
[o v s,  v s,  v s o]
--------------------------------
#15
1  1  0  0  0  0  1/0  0 , i  i
1  1  0  0  0  0  1/0  0 , f  i
1  1  1  0  0  0  1/0  0 , i  i
1  1  1  0  0  0  1/0  0 , f  i
1  1  1  1  0  0  1/0  0 , i  i
1  1  1  1  0  0  1/0  0 , f  i
1  1  1  1  1  0  1/0  0 , i  i
1  1  1  1  1  0  1/0  0 , i  f
1  1  1  1  1  1/0  1  0 , i  i
1  1  1  1  1  1/0  1  0 , i  f
1  1  1  1  1  1/0  1/0  0 , i  i
1  1  1  1  1  1/0  1/0  0 , i  f
[v o s,  v s,  v s o]
--------------------------------
#16
0  0  0  0  0  1  1/0  1 , i  i
0  0  0  0  0  1  1/0  1 , i  f
0  0  0  0  0  1  1/0  1 , f  i
0  0  0  0  0  1  1/0  1 , f  f
1  0  0  0  0  1  1/0  1 , i  i
1  0  0  0  0  1  1/0  1 , f  i
0  0  0  0  0  1/0  1/0  0 , i  i
0  0  0  0  0  1/0  1/0  0 , i  f
0  0  0  0  0  1/0  1/0  0 , f  i
0  0  0  0  0  1/0  1/0  0 , f  f
0  0  0  0  0  1  1/0  1/0 , i  i
0  0  0  0  0  1  1/0  1/0 , i  f
0  0  0  0  0  1  1/0  1/0 , f  i
0  0  0  0  0  1  1/0  1/0 , f  f
0  0  0  0  0  1/0  1/0  1 , i  i
0  0  0  0  0  1/0  1/0  1 , i  f
0  0  0  0  0  1/0  1/0  1 , f  i
0  0  0  0  0  1/0  1/0  1 , f  f
1  0  0  0  0  1  1/0  1/0 , i  i
1  0  0  0  0  1  1/0  1/0 , f  i
0  0  0  0  0  1/0  1/0  1/0 , i  i
0  0  0  0  0  1/0  1/0  1/0 , i  f
0  0  0  0  0  1/0  1/0  1/0 , f  i
0  0  0  0  0  1/0  1/0  1/0 , f  f
[o s v,  s o v,  s v,  s v o]
--------------------------------
#17
1  0  0  0  0  1/0  1  0 , i  i
1  0  0  0  0  1/0  1  0 , f  i
[s o v,  s v,  v s,  o v s]
--------------------------------
#18
1  1  0  0  0  0  1  1/0 , i  i
1  1  0  0  0  0  1  1/0 , f  i
1  1  1  0  0  0  1  1/0 , i  i
1  1  1  0  0  0  1  1/0 , f  i
1  1  1  1  0  0  1  1/0 , i  i
1  1  1  1  0  0  1  1/0 , f  i
1  1  1  1  1  0  1  1/0 , i  i
1  1  1  1  1  0  1  1/0 , i  f
[o v s,  v s,  v o s]
--------------------------------
#19
1  1  0  0  0  1/0  1  0 , i  i
```

```
1  1  0  0  0  1/0  1  0 , f  i
1  1  1  0  0  1/0  1  0 , i  i
1  1  1  0  0  1/0  1  0 , f  i
1  1  1  1  0  1/0  1  0 , i  i
1  1  1  1  0  1/0  1  0 , f  i
[s v o,  s v,  v s,  v o s]
--------------------------------
#20
1  0  0  0  0  1/0  1  1 , i  i
1  0  0  0  0  1/0  1  1 , f  i
[s o v,  s v,  o s v,  o v s]
--------------------------------
#21
1  1  0  0  0  1/0  1  1 , i  i
1  1  0  0  0  1/0  1  1 , f  i
1  1  1  0  0  1/0  1  1 , i  i
1  1  1  0  0  1/0  1  1 , f  i
1  1  1  1  0  1/0  1  1 , i  i
1  1  1  1  0  1/0  1  1 , f  i
1  1  0  0  0  1/0  1/0  1 , i  i
1  1  0  0  0  1/0  1/0  1 , f  i
1  1  1  0  0  1/0  1/0  1 , i  i
1  1  1  0  0  1/0  1/0  1 , f  i
1  1  1  1  0  1/0  1/0  1 , i  i
1  1  1  1  0  1/0  1/0  1 , f  i
[s v o,  s v,  o s v,  o v s]
--------------------------------
#22
1  1  1  1  1  1  1  1/0 , i  i
1  1  1  1  1  1  1  1/0 , i  f
1  1  1  1  1  1  1/0  1/0 , i  i
1  1  1  1  1  1  1/0  1/0 , i  f
[s v o,  s v,  o v s,  v s,  v s o]
--------------------------------
#23
1  0  0  0  0  1/0  1/0  0 , i  i
1  0  0  0  0  1/0  1/0  0 , f  i
[o v s,  s o v,  s v,  s v o,  v s,
 v s o]
--------------------------------
#24
1  1  0  0  0  0  1/0  1/0 , i  i
1  1  0  0  0  0  1/0  1/0 , f  i
1  1  1  0  0  0  1/0  1/0 , i  i
1  1  1  0  0  0  1/0  1/0 , f  i
1  1  1  1  0  0  1/0  1/0 , i  i
1  1  1  1  0  0  1/0  1/0 , f  i
1  1  1  1  1  0  1/0  1/0 , i  i
1  1  1  1  1  0  1/0  1/0 , i  f
[v o s,  o v s,  v s,  v s o]
--------------------------------
#25
1  1  0  0  0  1/0  1/0  0 , i  i
1  1  0  0  0  1/0  1/0  0 , f  i
1  1  1  0  0  1/0  1/0  0 , i  i
1  1  1  0  0  1/0  1/0  0 , f  i
1  1  1  1  0  1/0  1/0  0 , i  i
1  1  1  1  0  1/0  1/0  0 , f  i
[v o s,  s v,  s v o,  v s,  v s o]
--------------------------------
#26
1  0  0  0  0  1/0  1  1/0 , i  i
1  0  0  0  0  1/0  1  1/0 , f  i
[o s v,  s v,  s o v,  v s,  o v s]
--------------------------------
#27
1  0  0  0  0  1/0  1/0  1 , i  i
1  0  0  0  0  1/0  1/0  1 , f  i
[s v o,  s v,  o s v,  s o v,  o v s]
--------------------------------
#28
1  1  0  0  0  1/0  1  1/0 , i  i
1  1  0  0  0  1/0  1  1/0 , f  i
1  1  1  0  0  1/0  1  1/0 , i  i
1  1  1  0  0  1/0  1  1/0 , f  i
1  1  1  1  0  1/0  1  1/0 , i  i
1  1  1  1  0  1/0  1  1/0 , f  i
[o v s,  o s v,  s v,  s v o,  v s,
 v o s]
--------------------------------
#29
1  1  1  1  1  1/0  1  1/0 , i  i
1  1  1  1  1  1/0  1  1/0 , i  f
1  1  1  1  1  1/0  1/0  1/0 , i  i
1  1  1  1  1  1/0  1/0  1/0 , i  f
[v o s,  s v,  s v o,  o v s,  v s,
 v s o]
--------------------------------
#30
1  0  0  0  0  1/0  1/0  1/0 , i  i
1  0  0  0  0  1/0  1/0  1/0 , f  i
[o v s,  s v o,  s v,  s o v,  o s v,
 v s,  v s o]
--------------------------------
#31
1  1  0  0  0  1/0  1/0  1/0 , i  i
1  1  0  0  0  1/0  1/0  1/0 , f  i
1  1  1  0  0  1/0  1/0  1/0 , i  i
1  1  1  0  0  1/0  1/0  1/0 , f  i
1  1  1  1  0  1/0  1/0  1/0 , i  i
1  1  1  1  0  1/0  1/0  1/0 , f  i
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
```

B.5

```
#1    0  0  0  0  0  0  0  0
[aux s v,  aux s v o]
--------------------------------
#2    0  0  0  0  0  1  0  0
[s aux v,  s aux v o]
--------------------------------
#3    1  0  0  0  0  0  0  0
[aux v s,  aux v s o]
--------------------------------
#4    0  0  0  0  0  0  1  0
[aux o s v,  aux s v]
--------------------------------
#5    1  0  0  0  0  0  1  0
[aux v s,  aux o v s]
--------------------------------
#6    0  0  0  0  0  1  1  0
```

```
[s aux v,  s aux o v]
--------------------------------
#7    0  0  0  0  0  0  1  1
[o aux s v]
--------------------------------
#8    1  1  1  0  0  0  0  0
[v s,  v s o]
--------------------------------
#9    1  0  0  0  0  0  1  1
[o aux v s]
--------------------------------
#10   1  1  0  0  0  0  1  0
[aux v s,  aux v o s]
--------------------------------
#11   0  0  0  0  0  1  1  1
[o s aux v,  s aux v,  s aux o v]
--------------------------------
#12   1  1  1  0  0  1  0  0
[s v,  s v o]
--------------------------------
#13   1  1  1  0  0  0  1  0
[v s,  v o s]
--------------------------------
#14   0  0  0  1  1  1  1  0
[aux s v,  aux s o v]
--------------------------------
#15   1  1  1  0  0  0  1  1
[o v s]
--------------------------------
#16   1  1  0  0  0  1  1  1
[o s aux v,  s aux v,  s aux v o]
--------------------------------
#17   0  0  0  1  1  1  1  1
[s aux o v,  s aux v,  o aux s v]
--------------------------------
#18   1  1  1  0  0  1  1  1
[o s v,  s v,  s v o]
--------------------------------
#19   1  1  0  1  1  1  1  1
[o aux s v,  s aux v,  s aux v o]
--------------------------------
#20   1  1  1  1  1  1  1  1  1
[s v o,  s v,  o v s]
--------------------------------
#21   0  0  0  0  0  0 1/0  0  0
[s aux v o,  s aux v,  aux s v,
 aux s v o]
--------------------------------
#22   0  0  0  0  0  0  0 1/0  0
[aux o s v,  aux s v,  aux s v o]
--------------------------------
#23   1  0  0  0  0  0 1/0  0  0
[s aux v o,  s aux v,  aux v s,
 aux v s o]
--------------------------------
#24   1  0  0  0  0  0  0 1/0  0
[aux o v s,  aux v s,  aux v s o]
--------------------------------
#25   0  0  0  0  0  0  1 1/0  0
[s aux o v,  s aux v,  s aux v o]
--------------------------------
#26   0  0  0  0  0  0  0  1 1/0
```

```
[o aux s v,  aux o s v,  aux s v]
--------------------------------
#27   0  0  0  0  0  0 1/0  1  0
[s aux o v,  s aux v,  aux o s v,
 aux s v]
--------------------------------
#28   1  1  0  0  0  0  0 1/0  0
[aux v o s,  aux v s,  aux v s o]
--------------------------------
#29   1  0  0  0  0  0  0  1 1/0
[o aux v s,  aux v s,  aux o v s]
--------------------------------
#30   0  0  0  0  0  0  1 1/0  1
[o s aux v,  s aux o v,  s aux v,
 s aux v o]
--------------------------------
#31   1  0  0  0  0  0 1/0  1  0
[s aux o v,  s aux v,  aux v s,
 aux o v s]
--------------------------------
#32   0  0  0  0  0  0 1/0  1  1
[s aux o v,  s aux v,  o s aux v,
 o aux s v]
--------------------------------
#33   1  1  1  0  0 1/0  0  0
[s v o,  s v,  v s,  v s o]
--------------------------------
#34   1  1  1  0  0  0 1/0  0
[v o s,  v s,  v s o]
--------------------------------
#35   1  1  0  0  0  0  0  1 1/0
[o aux v s,  aux v s,  aux v o s]
--------------------------------
#36   1  0  0  1  1 1/0  0  0
[aux s v o,  aux s v,  aux v s,
 aux v s o]
--------------------------------
#37   1  1  0  0  0 1/0  1  0
[s aux v o,  s aux v,  aux v s,
 aux v o s]
--------------------------------
#38   0  0  0  1  1  1 1/0  0
[aux s o v,  aux s v,  aux s v o]
--------------------------------
#39   1  0  0  0  0 1/0  1  1
[s aux o v,  s aux v,  o s aux v,
 o aux v s]
--------------------------------
#40   0  0  0  1  1 1/0  1  0
[aux s o v,  aux o s v,  aux s v]
--------------------------------
#41   1  1  1  0  0  0  1 1/0
[o v s,  v s,  v o s]
--------------------------------
#42   1  1  1  0  0 1/0  1  0
[s v o,  s v,  v s,  v o s]
--------------------------------
#43   1  1  0  0  0 1/0  1  1
[s aux v o,  s aux v,  o s aux v,
 o aux v s]
--------------------------------
#44   0  0  0  1  1  1  1 1/0
```

```
[s aux o v,  s aux v,  o aux s v,
 aux s v,  aux s o v]
--------------------------------
#45   0  0  0  1  1  1  1/0  1
[s aux v o,  s aux v,  s aux o v,
 o aux s v]
--------------------------------
#46   1  0  0  1  1  1/0  1  0
[aux s o v,  aux s v,  aux v s,
 aux o v s]
--------------------------------
#47   1  1  1  0  0  1/0  1  1
[s v o,  s v,  o s v,  o v s]
--------------------------------
#48   1  1  0  1  1  1/0  1  0
[aux s v o,  aux s v,  aux v s,
 aux v o s]
--------------------------------
#49   1  0  0  1  1  1/0  1  1
[s aux o v,  s aux v,  o aux s v,
 o aux v s]
--------------------------------
#50   1  1  0  1  1  1  1  1/0
[s aux v o,  s aux v,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#51   1  1  0  1  1  1/0  1  1
[s aux v o,  s aux v,  o aux s v,
 o aux v s]
--------------------------------
#52   1  1  1  1  1  1  1  1/0
[s v o,  s v,  o v s,  v s,  v s o]
--------------------------------
#53   0  0  0  0  0  0  1/0  1/0
[aux o s v,  o aux s v,  aux s v,
 aux s v o]
--------------------------------
#54   0  0  0  0  0  1/0  1/0  0
[aux o s v,  s aux o v,  s aux v,
 s aux v o,  aux s v,  aux s v o]
--------------------------------
#55   1  0  0  0  0  0  1/0  1/0
[aux o v s,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#56   1  0  0  0  0  1/0  1/0  0
[aux o v s,  s aux o v,  s aux v,
 s aux v o,  aux v s,  aux v s o]
--------------------------------
#57   0  0  0  0  0  1/0  1  1/0
[o aux s v,  o s aux v,  s aux v,
 s aux o v,  aux o s v,  aux s v]
--------------------------------
#58   0  0  0  0  0  1/0  1/0  1
[s aux v o,  s aux v,  o s aux v,
 s aux o v,  o aux s v]
--------------------------------
#59   1  1  0  0  0  0  1/0  1/0
[aux v o s,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#60   1  1  0  0  0  1/0  1/0  0
[aux v o s,  s aux v,  s aux v o,
 aux v s,  aux v s o]
--------------------------------
#61   1  0  0  0  0  1/0  1  1/0
[o aux v s,  o s aux v,  s aux v,
 s aux o v,  aux v s,  aux o v s]
--------------------------------
#62   1  0  0  0  0  1/0  1/0  1
[s aux v o,  s aux v,  o s aux v,
 s aux o v,  o aux v s]
--------------------------------
#63   0  0  0  1  1  1/0  1/0  0
[aux o s v,  aux s o v,  aux s v,
 aux s v o]
--------------------------------
#64   1  1  1  0  0  0  1/0  1/0
[v o s,  o v s,  v s,  v s o]
--------------------------------
#65   1  1  1  0  0  1/0  1/0  0
[v o s,  s v,  s v o,  v s,  v s o]
--------------------------------
#66   1  0  0  1  1  1/0  0  1/0
[aux s v o,  aux s v,  s aux v,
 s aux v o,  aux v s,  aux v s o]
--------------------------------
#67   1  1  0  0  0  1/0  1  1/0
[o aux v s,  o s aux v,  s aux v,
 s aux v o,  aux v s,  aux v o s]
--------------------------------
#68   0  0  0  1  1  1  1/0  1/0
[s aux v o,  s aux v,  s aux o v,
 o aux s v,  aux s o v,  aux s v,
 aux s v o]
--------------------------------
#69   1  0  0  1  1  1/0  1/0  0
[aux o v s,  aux s o v,  aux s v,
 aux s v o,  aux v s,  aux v s o]
--------------------------------
#70   0  0  0  1  1  1/0  1  1/0
[o aux s v,  s aux o v,  s aux v,
 aux s o v,  aux o s v,  aux s v]
--------------------------------
#71   1  1  1  0  0  1/0  1  1/0
[o v s,  o s v,  s v,  s v o,  v s,
 v o s]
--------------------------------
#72   1  1  0  1  1  1/0  1/0  0
[aux v o s,  aux s v,  aux s v o,
 aux v s,  aux v s o]
--------------------------------
#73   1  0  0  1  1  1/0  1  1/0
[o aux v s,  s aux o v,  s aux v,
 o aux s v,  aux s v,  aux s o v,
 aux v s,  aux o v s]
--------------------------------
#74   1  0  0  1  1  1/0  1/0  1
[s aux v o,  s aux v,  o aux s v,
 s aux o v,  o aux v s]
--------------------------------
#75   1  1  0  1  1  1/0  1  1/0
[o aux v s,  s aux v o,  s aux v,
 o aux s v,  aux s v,  aux s v o,
```

```
 aux v s,  aux v o s]
--------------------------------
#76   1  1  1  1  1  1/0  1  1/0
[v o s,  s v,  s v o,  o v s,  v s,
 v s o]
--------------------------------
#77   0  0  0  0  0  1/0  1/0  1/0
[aux o s v,  s aux v o,  s aux v,
 s aux o v,  o s aux v,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#78   1  0  0  0  0  1/0  1/0  1/0
[aux o v s,  s aux v o,  s aux v,
 s aux o v,  o s aux v,  o aux v s,
 aux v s,  aux v s o]
--------------------------------
#79   1  1  0  0  0  1/0  1/0  1/0
[aux v o s,  s aux v o,  s aux v,
 o s aux v,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#80   0  0  0  1  1  1/0  1/0  1/0
[aux o s v,  aux s o v,  s aux o v,
 s aux v,  s aux v o,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#81   1  1  1  0  0  1/0  1/0  1/0
[v o s,  s v o,  s v,  o s v,  o v s,
 v s,  v s o]
--------------------------------
#82   1  0  0  1  1  1/0  1/0  1/0
[aux o v s,  aux s v o,  aux s v,
 aux s o v,  o aux s v,  s aux o v,
 s aux v,  s aux v o,  o aux v s,
 aux v s,  aux v s o]
--------------------------------
#83   1  1  0  1  1  1/0  1/0  1/0
[aux v o s,  aux s v o,  aux s v,
 o aux s v,  s aux v,  s aux v o,
 o aux v s,  aux v s,  aux v s o]
--------------------------------
```

B.6

```
#1  0  0  0  0  0  0  0  0 , i  i
[aux s v,  aux s v o]
--------------------------------
#2  0  0  0  0  0  0  0  0 , i  f
[s v aux,  s v o aux]
--------------------------------
#3  1  0  0  0  0  0  0  0 , i  f
[s v aux,  s o v aux]
--------------------------------
#4  0  0  0  0  0  1  0  0 , i  i
[s aux v,  s aux v o]
--------------------------------
#5  0  0  0  0  0  0  1  0 , i  i
[aux o s v,  aux s v]
--------------------------------
#6  0  0  0  0  0  0  1  0 , i  f
[o s v aux,  s v aux]
--------------------------------
#7  1  0  0  0  0  0  0  0 , i  i
[aux v s,  aux v s o]
--------------------------------
#8  0  0  0  0  0  1  1  0 , i  i
[s aux v,  s aux o v]
--------------------------------
#9  0  0  0  0  0  0  1  1 , i  i
[o aux s v]
--------------------------------
#10 0  0  0  0  0  0  1  1 , i  f
[o s v aux]
--------------------------------
#11 1  0  0  0  0  0  1  0 , i  i
[aux v s,  aux o v s]
--------------------------------
#12 1  1  1  0  0  0  0  0 , i  i
[v s,  v s o]
--------------------------------
#13 1  1  1  0  0  0  0  0 , i  f
[s v,  s o v]
--------------------------------
#14 0  0  0  0  0  1  1  1 , i  f
[aux s v,  aux s o v]
--------------------------------
#16 1  0  0  0  0  0  1  1 , i  i
[o aux v s]
--------------------------------
#17 1  1  0  0  0  0  1  0 , i  i
[aux v s,  aux v o s]
--------------------------------
#18 1  0  0  1  1  0  0  0 , f  i
[v s aux,  v s o aux]
--------------------------------
#19 0  0  0  0  0  1  1  1 , i  i
[o s aux v,  s aux v,  s aux o v]
--------------------------------
#20 1  1  1  0  0  1  0  0 , i  i
[s v,  s v o]
--------------------------------
#21 1  1  1  0  0  0  1  0 , i  i
[v s,  v o s]
--------------------------------
#22 1  1  1  0  0  0  1  0 , i  f
[o s v,  s v]
--------------------------------
#23 1  0  0  1  1  0  1  0 , f  i
[v s aux,  o v s aux]
--------------------------------
#24 1  1  1  0  0  0  1  1 , i  i
[o v s]
--------------------------------
#25 1  1  1  0  0  0  1  1 , i  f
[o s v]
--------------------------------
#26 0  0  0  1  1  1  1  1 , i  i
[s aux o v,  s aux v,  o aux s v]
--------------------------------
#27 1  0  0  1  1  0  1  1 , f  i
[o v s aux]
--------------------------------
#28 1  1  0  1  1  0  1  0 , f  i
```

```
[v s aux,  v o s aux]
--------------------------------
#29 1  1  0  0  0  1  1  1 , i  i
[o s aux v,  s aux v,  s aux v o]
--------------------------------
#30 1  1  1  0  0  1  1  1 , i  f
[o s v,  s v,  s o v]
--------------------------------
#31 1  1  1  0  0  1  1  1 , i  i
[o s v,  s v,  s v o]
--------------------------------
#32 1  1  0  1  1  1  1  1 , i  i
[o aux s v,  s aux v,  s aux v o]
--------------------------------
#33 1  1  0  1  1  1  1  1 , f  i
[o s v aux,  s v aux,  s v o aux]
--------------------------------
#34 1  1  1  1  1  1  1  1 , i  i
[s v o,  s v,  o v s]
--------------------------------
#35 0  0  0  0  0  1/0  0  0 , i  i
[s aux v o,  s aux v,  aux s v,
 aux s v o]
--------------------------------
#36 0  0  0  0  0  0  1/0  0 , i  i
[aux o s v,  aux s v,  aux s v o]
--------------------------------
#37 0  0  0  0  0  0  1  1/0 , i  i
[o aux s v,  aux o s v,  aux s v]
--------------------------------
#38 1  0  0  0  0  1/0  0  0 , i  i
[s aux v o,  s aux v,  aux v s,
 aux v s o]
--------------------------------
#39 0  0  0  0  0  1  1/0  0 , i  f
[s o v aux,  s v aux,  s v o aux]
--------------------------------
#40 1  0  0  0  0  0  1/0  0 , i  i
[aux o v s,  aux v s,  aux v s o]
--------------------------------
#41 0  0  0  0  0  1  1/0  0 , i  i
[s aux o v,  s aux v,  s aux v o]
--------------------------------
#42 0  0  0  0  0  1/0  1  0 , i  i
[s aux o v,  s aux v,  aux o s v,
 aux s v]
--------------------------------
#43 1  1  0  0  0  0  1/0  0 , i  i
[aux v o s,  aux v s,  aux v s o]
--------------------------------
#44 0  0  0  0  0  1  1/0  1 , i  f
[o s v aux,  s o v aux,  s v aux,
 s v o aux]
--------------------------------
#45 1  0  0  0  0  0  1  1/0 , i  i
[o aux v s,  aux v s,  aux o v s]
--------------------------------
#46 0  0  0  0  0  1  1/0  1 , i  i
[o s aux v,  s aux o v,  s aux v,
 s aux v o]
--------------------------------
#47 1  0  0  0  0  1/0  1  0 , i  i
[s aux o v,  s aux v,  aux v s,
 aux o v s]
--------------------------------
#48 0  0  0  0  0  1/0  1  1 , i  i
[s aux o v,  s aux v,  o s aux v,
 o aux s v]
--------------------------------
#49 1  1  1  0  0  1/0  0  0 , i  i
[s v o,  s v,  v s,  v s o]
--------------------------------
#50 1  1  1  0  0  0  1/0  0 , i  i
[v o s,  v s,  v s o]
--------------------------------
#51 1  1  0  0  0  0  1  1/0 , i  i
[o aux v s,  aux v s,  aux v o s]
--------------------------------
#52 1  0  0  1  1  0  1/0  0 , i  f
[aux s o v,  aux o s v,  aux s v]
--------------------------------
#53 1  0  0  1  1  1/0  0  0 , i  i
[aux s v o,  aux s v,  aux v s,
 aux v s o]
--------------------------------
#54 1  0  0  1  1  1/0  0  0 , f  i
[s v o aux,  s v aux,  v s aux,
 v s o aux]
--------------------------------
#55 1  1  0  0  0  1/0  1  0 , i  i
[s aux v o,  s aux v,  aux v s,
 aux v o s]
--------------------------------
#56 0  0  0  1  1  1  1/0  0 , i  i
[aux s o v,  aux s v,  aux s v o]
--------------------------------
#57 1  0  0  1  1  0  1/0  0 , f  i
[o v s aux,  v s aux,  v s o aux]
--------------------------------
#58 1  0  0  0  0  1/0  1  1 , i  i
[s aux o v,  s aux v,  o s aux v,
 o aux v s]
--------------------------------
#59 1  1  1  0  0  0  1  1/0 , i  i
[o v s,  v s,  v o s]
--------------------------------
#60 1  1  1  0  0  1/0  1  0 , i  i
[s v o,  s v,  v s,  v o s]
--------------------------------
#61 0  0  0  1  1  1  1  1/0 , i  i
[s aux o v,  s aux v,  o aux s v,
 aux s v,  aux s o v]
--------------------------------
#62 1  0  0  1  1  1  0  1/0 , i  f
[s aux v,  s aux o v,  aux s v,
 aux s o v]
--------------------------------
#63 1  1  0  1  1  0  1/0  0 , f  i
[v o s aux,  v s aux,  v s o aux]
--------------------------------
#64 1  1  0  0  0  1/0  1  1 , i  i
[s aux v o,  s aux v,  o s aux v,
 o aux v s]
--------------------------------
```

```
#65 0  0  0  1  1  1  1/0  1 , i  i
[s aux v o,  s aux v,  s aux o v,
 o aux s v]
--------------------------------
#66 1  0  0  1  1  1/0  1  0 , i  i
[aux s o v,  aux s v,  aux v s,
 aux o v s]
--------------------------------
#67 1  0  0  1  1  1/0  1  0 , f  i
[s o v aux,  s v aux,  v s aux,
 o v s aux]
--------------------------------
#68 1  1  1  0  0  1/0  1  1 , i  i
[s v o,  s v,  o s v,  o v s]
--------------------------------
#69 1  1  0  1  1  0  1  1/0 , f  i
[o v s aux,  v s aux,  v o s aux]
--------------------------------
#70 1  1  0  1  1  1/0  1  0 , i  i
[aux s v o,  aux s v,  aux v s,
 aux v o s]
--------------------------------
#71 1  1  0  1  1  1/0  1  0 , f  i
[s v o aux,  s v aux,  v s aux,
 v o s aux]
--------------------------------
#72 1  0  0  1  1  1/0  1  1 , i  i
[s aux o v,  s aux v,  o aux s v,
 o aux v s]
--------------------------------
#73 1  0  0  1  1  1/0  1  1 , f  i
[s o v aux,  s v aux,  o s v aux,
 o v s aux]
--------------------------------
#74 1  1  0  1  1  1  1  1/0 , i  i
[s aux v o,  s aux v,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#75 1  1  0  1  1  1/0  1  1 , i  i
[s aux v o,  s aux v,  o aux s v,
 o aux v s]
--------------------------------
#76 1  1  0  1  1  1/0  1  1 , f  i
[s v o aux,  s v aux,  o s v aux,
 o v s aux]
--------------------------------
#77 1  1  1  1  1  1  1  1/0 , i  i
[s v o,  s v,  o v s,  v s,  v s o]
--------------------------------
#78 0  0  0  0  0  0  1/0  1/0 , i  i
[aux o s v,  o aux s v,  aux s v,
 aux s v o]
--------------------------------
#79 0  0  0  0  0  1/0  1/0  0 , i  i
[aux o s v,  s aux o v,  s aux v,
 s aux v o,  aux s v,  aux s v o]
--------------------------------
#80 1  0  0  0  0  0  1/0  1/0 , i  i
[aux o v s,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#81 1  0  0  0  0  1/0  1/0  0 , i  i
[aux o v s,  s aux o v,  s aux v,
 s aux v o,  aux v s,  aux v s o]
--------------------------------
#82 0  0  0  0  0  1/0  1  1/0 , i  i
[o aux s v,  o s aux v,  s aux v,
 s aux o v,  aux o s v,  aux s v]
--------------------------------
#83 0  0  0  0  0  1/0  1/0  1 , i  i
[s aux v o,  s aux v,  o s aux v,
 s aux o v,  o aux s v]
--------------------------------
#84 1  1  0  0  0  0  1/0  1/0 , i  i
[aux v o s,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#85 1  1  0  0  0  1/0  1/0  0 , i  i
[aux v o s,  s aux v,  s aux v o,
 aux v s,  aux v s o]
--------------------------------
#86 1  0  0  0  0  1/0  1  1/0 , i  i
[o aux v s,  o s aux v,  s aux v,
 s aux o v,  aux v s,  aux o v s]
--------------------------------
#87 1  0  0  0  0  1/0  1/0  1 , i  i
[s aux v o,  s aux v,  o s aux v,
 s aux o v,  o aux v s]
--------------------------------
#88 0  0  0  1  1  1/0  1/0  0 , i  i
[aux o s v,  aux s o v,  aux s v,
 aux s v o]
--------------------------------
#89 1  1  1  0  0  0  1/0  1/0 , i  i
[v o s,  o v s,  v s,  v s o]
--------------------------------
#90 1  1  1  0  0  1/0  1/0  0 , i  i
[v o s,  s v,  s v o,  v s,  v s o]
--------------------------------
#91 1  0  0  1  1  0  1/0  1/0 , i  f
[aux o s v,  o aux s v,  aux s v,
 aux s o v]
--------------------------------
#92 1  0  0  1  1  1/0  0  1/0 , i  i
[aux s v o,  aux s v,  s aux v,
 s aux v o,  aux v s,  aux v s o]
--------------------------------
#93 0  0  0  1  1  1/0  1  1/0 , i  i
[o aux s v,  s aux o v,  s aux v,
 aux s o v,  aux o s v,  aux s v]
--------------------------------
#94 1  1  0  0  0  1/0  1  1/0 , i  i
[o aux v s,  o s aux v,  s aux v,
 s aux v o,  aux v s,  aux v o s]
--------------------------------
#95 0  0  0  1  1  1  1/0  1/0 , i  i
[s aux v o,  s aux v,  s aux o v,
 o aux s v,  aux s o v,  aux s v,
 aux s v o]
--------------------------------
#96 1  0  0  1  1  1/0  1/0  0 , i  i
[aux o v s,  aux s o v,  aux s v,
 aux s v o,  aux v s,  aux v s o]
--------------------------------
```

```
#97 1  0  0  1  1  1/0  1/0  0 , f  i
[o v s aux,  s o v aux,  s v aux,
 s v o aux,  v s aux,  v s o aux]
--------------------------------
#98 1  1  1  0  0  1/0  1  1/0 , i  i
[o v s,  o s v,  s v,  s v o,  v s,
 v o s]
--------------------------------
#99 1  1  0  1  1  0  1/0  1/0 , f  i
[v o s aux,  o v s aux,  v s aux,
 v s o aux]
--------------------------------
#100 1  1  0  1  1  1/0  1/0  0 , i  i
[aux v o s,  aux s v,  aux s v o,
 aux v s,  aux v s o]
--------------------------------
#101 1  1  0  1  1  1/0  1/0  0 , f  i
[v o s aux,  s v aux,  s v o aux,
 v s aux,  v s o aux]
--------------------------------
#102 1  0  0  1  1  1/0  1  1/0 , i  i
[o aux v s,  s aux o v,  s aux v,
 o aux s v,  aux s v,  aux s o v,
 aux v s,  aux o v s]
--------------------------------
#103 1  0  0  1  1  1/0  1  1/0 , f  i
[o s v aux,  s v aux,  s o v aux,
 v s aux,  o v s aux]
--------------------------------
#104 1  0  0  1  1  1/0  1/0  1 , i  i
[aux v o,  s aux v,  o aux s v,
 s aux o v,  o aux v s]
--------------------------------
#105 1  0  0  1  1  1/0  1/0  1 , f  i
[s v o aux,  s v aux,  o s v aux,
 s o v aux,  o v s aux]
--------------------------------
#106 1  1  0  1  1  1/0  1  1/0 , i  i
[o aux v s,  s aux v o,  s aux v,
 o aux s v,  aux s v,  aux s v o,
 aux v s,  aux v o s]
--------------------------------
#107 1  1  0  1  1  1/0  1  1/0 , f  i
[o v s aux,  o s v aux,  s v aux,
 s v o aux,  v s aux,  v o s aux]
--------------------------------
#108 1  1  1  1  1  1/0  1  1/0 , i  i
[v o s,  s v,  s v o,  o v s,  v s,  v s o]
--------------------------------
#109 0  0  0  0  0  1/0  1/0  1/0 , i  i
[aux o s v,  s aux v o,  s aux v,
 s aux o v,  o s aux v,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#110 1  0  0  0  0  1/0  1/0  1/0 , i  i
[aux o v s,  s aux v o,  s aux v,
 s aux o v,  o s aux v,  o aux v s,
 aux v s,  aux v s o]
--------------------------------
#111 1  1  0  0  0  1/0  1/0  1/0 , i  i
[aux v o s,  s aux v o,  s aux v,
 o s aux v,  o aux v s,  aux v s,
 aux v s o]
--------------------------------
#112 0  0  0  1  1  1/0  1/0  1/0 , i  i
[aux o s v,  aux s o v,  s aux o v,
 s aux v,  s aux v o,  o aux s v,
 aux s v,  aux s v o]
--------------------------------
#113 1  1  1  0  0  1/0  1/0  1/0 , i  i
[v o s,  s v o,  s v,  o s v,
 o v s,  v s,  v s o]
--------------------------------
#114 1  0  0  1  1  1/0  1/0  1/0 , i  i
[aux o v s,  aux s v o,  aux s v,
 aux s o v,  o aux s v,  s aux o v,
 s aux v,  s aux v o,  o aux v s,
 aux v s,  aux v s o]
--------------------------------
#115 1  0  0  1  1  1/0  1/0  1/0 , f  i
[o v s aux,  s v o aux,  s v aux,
 s o v aux,  o s v aux,  v s aux,
 v s o aux]
--------------------------------
#116 1  1  0  1  1  1/0  1/0  1/0 , i  i
[aux v o s,  aux s v o,  aux s v,
 o aux s v,  s aux v,  s aux v o,
 o aux v s,  aux v s,  aux v s o]
--------------------------------
#117 1  1  0  1  1  1/0  1/0  1/0 , f  i
[v o s aux,  s v o aux,  s v aux,
 o s v aux,  o v s aux,  v s aux,
 v s o aux]
--------------------------------
```

B.7

```
#1    0  0  0  0  0  1  0  0 , i  i , 0-1  1-0  1-0  0-1
[s-[c(1)] aux-[tns] v-[asp],  s-[c(1)] aux-[tns] v-[asp] o-[c(2)]]
-----------------------------------------------------
#2    0  0  0  0  0  1  0  0 , i  i , 0-1  1-0  1-0  0-0
[s-[c(1)] aux-[tns] v-[],  s-[c(1)] aux-[tns] v-[] o-[c(2)]]
-----------------------------------------------------
#3    0  0  0  0  0  1  0  0 , i  i , 0-0  1-0  1-0  0-1
[s-[] aux-[tns] v-[asp],  s-[] aux-[tns] v-[asp] o-[]]
-----------------------------------------------------
#4    0  0  0  0  0  1  0  0 , i  i , 0-0  1-0  1-0  0-0
[s-[] aux-[tns] v-[],  s-[] aux-[tns] v-[] o-[]]
-----------------------------------------------------
#5    0  0  0  0  0  1  0  0 , i  f , 0-1  1-0  1-0  0-1
[s-[c(1)] v-[asp] aux-[tns],  s-[c(1)] v-[asp] o-[c(2)] aux-[tns]]
-----------------------------------------------------
#6    0  0  0  0  0  1  0  0 , i  f , 0-1  1-0  1-0  0-0
[s-[c(1)] v-[] aux-[tns],  s-[c(1)] v-[] o-[c(2)] aux-[tns]]
-----------------------------------------------------
#7    0  0  0  0  0  1  0  0 , i  f , 0-0  1-0  1-0  0-1
[s-[] v-[asp] aux-[tns],  s-[] v-[asp] o-[] aux-[tns]]
-----------------------------------------------------
#8    0  0  0  0  0  1  0  0 , i  f , 0-0  1-0  1-0  0-0
[s-[] v-[] aux-[tns],  s-[] v-[] o-[] aux-[tns]]
-----------------------------------------------------
#9    0  0  0  1  0  1  0  0 , i  i , 0-1  1-0  1-0  0-1
[s-[c(1)] aux-[agr,tns] v-[asp],  s-[c(1)] aux-[agr,tns] v-[asp] o-[c(2)]]
-----------------------------------------------------
#10   0  0  0  1  0  1  0  0 , i  i , 0-1  1-0  1-0  0-0
[s-[c(1)] aux-[agr,tns] v-[],  s-[c(1)] aux-[agr,tns] v-[] o-[c(2)]]
-----------------------------------------------------
#11   0  0  0  1  0  1  0  0 , i  i , 0-0  1-0  1-0  0-1
[s-[] aux-[agr,tns] v-[asp],  s-[] aux-[agr,tns] v-[asp] o-[]]
-----------------------------------------------------
#12   0  0  0  1  0  1  0  0 , i  i , 0-0  1-0  1-0  0-0
[s-[] aux-[agr,tns] v-[],  s-[] aux-[agr,tns] v-[] o-[]]
-----------------------------------------------------
#13   0  0  0  1  0  1  0  0 , i  f , 0-1  1-0  1-0  0-1
[s-[c(1)] v-[asp] aux-[agr,tns],  s-[c(1)] v-[asp] o-[c(2)] aux-[agr,tns]]
-----------------------------------------------------
#14   0  0  0  1  0  1  0  0 , i  f , 0-1  1-0  1-0  0-0
[s-[c(1)] v-[] aux-[agr,tns],  s-[c(1)] v-[] o-[c(2)] aux-[agr,tns]]
-----------------------------------------------------
#15   0  0  0  1  0  1  0  0 , i  f , 0-0  1-0  1-0  0-1
[s-[] v-[asp] aux-[agr,tns],  s-[] v-[asp] o-[] aux-[agr,tns]]
-----------------------------------------------------
#16   0  0  0  1  0  1  0  0 , i  f , 0-0  1-0  1-0  0-0
[s-[] v-[] aux-[agr,tns],  s-[] v-[] o-[] aux-[agr,tns]]
-----------------------------------------------------
#17   1  1  1  0  0  1  0  0 , i  i , 0-1  1-0  0-1  0-1
[s-[c(1)] v-[tns,asp],  s-[c(1)] v-[tns,asp] o-[c(2)]]
-----------------------------------------------------
#18   1  1  1  0  0  1  0  0 , i  i , 0-1  1-0  0-1  0-0
[s-[c(1)] v-[tns],  s-[c(1)] v-[tns] o-[c(2)]]
-----------------------------------------------------
#19   1  1  1  0  0  1  0  0 , i  i , 0-1  1-0  0-0  0-1
[s-[c(1)] v-[asp],  s-[c(1)] v-[asp] o-[c(2)]]
-----------------------------------------------------
#20   1  1  1  0  0  1  0  0 , i  i , 0-1  1-0  0-0  0-0
[s-[c(1)] v-[],  s-[c(1)] v-[] o-[c(2)]]
-----------------------------------------------------
```

```
#21   1  1  1  0  0  1  0  0 , i  i , 0-1  0-1  0-1  0-1
[s-[c(1)] v-[agr,tns,asp],  s-[c(1)] v-[agr,tns,asp] o-[c(2)]]
-----------------------------------------------------------
#22   1  1  1  0  0  1  0  0 , i  i , 0-1  0-1  0-1  0-0
[s-[c(1)] v-[agr,tns],  s-[c(1)] v-[agr,tns] o-[c(2)]]
-----------------------------------------------------------
#23   1  1  1  0  0  1  0  0 , i  i , 0-1  0-1  0-0  0-1
[s-[c(1)] v-[agr,asp],  s-[c(1)] v-[agr,asp] o-[c(2)]]
-----------------------------------------------------------
#24   1  1  1  0  0  1  0  0 , i  i , 0-1  0-1  0-0  0-0
[s-[c(1)] v-[agr],  s-[c(1)] v-[agr] o-[c(2)]]
-----------------------------------------------------------
#25   1  1  1  0  0  1  0  0 , i  i , 0-0  1-0  0-1  0-1
[s-[] v-[tns,asp],  s-[] v-[tns,asp] o-[]]
-----------------------------------------------------------
#26   1  1  1  0  0  1  0  0 , i  i , 0-0  1-0  0-1  0-0
[s-[] v-[tns],  s-[] v-[tns] o-[]]
-----------------------------------------------------------
#27   1  1  1  0  0  1  0  0 , i  i , 0-0  1-0  0-0  0-1
[s-[] v-[asp],  s-[] v-[asp] o-[]]
-----------------------------------------------------------
#28   1  1  1  0  0  1  0  0 , i  i , 0-0  1-0  0-0  0-0
[s-[] v-[],  s-[] v-[] o-[]]
-----------------------------------------------------------
#29   1  1  1  0  0  1  0  0 , i  i , 0-0  0-1  0-1  0-1
[s-[] v-[agr,tns,asp],  s-[] v-[agr,tns,asp] o-[]]
-----------------------------------------------------------
#30   1  1  1  0  0  1  0  0 , i  i , 0-0  0-1  0-1  0-0
[s-[] v-[agr,tns],  s-[] v-[agr,tns] o-[]]
-----------------------------------------------------------
#31   1  1  1  0  0  1  0  0 , i  i , 0-0  0-1  0-0  0-1
[s-[] v-[agr,asp],  s-[] v-[agr,asp] o-[]]
-----------------------------------------------------------
#32   1  1  1  0  0  1  0  0 , i  i , 0-0  0-1  0-0  0-0
[s-[] v-[agr],  s-[] v-[agr] o-[]]
-----------------------------------------------------------
#33   1  1  1  0  0  1  0  0 , i  f , 0-1  1-0  0-1  0-1
[s-[c(1)] o-[c(2)] v-[tns,asp],  s-[c(1)] v-[tns,asp]]
-----------------------------------------------------------
#34   1  1  1  0  0  1  0  0 , i  f , 0-1  1-0  0-1  0-0
[s-[c(1)] o-[c(2)] v-[tns],  s-[c(1)] v-[tns]]
-----------------------------------------------------------
#35   1  1  1  0  0  1  0  0 , i  f , 0-1  1-0  0-0  0-1
[s-[c(1)] o-[c(2)] v-[asp],  s-[c(1)] v-[asp]]
-----------------------------------------------------------
#36   1  1  1  0  0  1  0  0 , i  f , 0-1  1-0  0-0  0-0
[s-[c(1)] o-[c(2)] v-[],  s-[c(1)] v-[]]
-----------------------------------------------------------
#37   1  1  1  0  0  1  0  0 , i  f , 0-1  0-1  0-1  0-1
[s-[c(1)] o-[c(2)] v-[agr,tns,asp],  s-[c(1)] v-[agr,tns,asp]]
-----------------------------------------------------------
#38   1  1  1  0  0  1  0  0 , i  f , 0-1  0-1  0-1  0-0
[s-[c(1)] o-[c(2)] v-[agr,tns],  s-[c(1)] v-[agr,tns]]
-----------------------------------------------------------
#39   1  1  1  0  0  1  0  0 , i  f , 0-1  0-1  0-0  0-1
[s-[c(1)] o-[c(2)] v-[agr,asp],  s-[c(1)] v-[agr,asp]]
-----------------------------------------------------------
#40   1  1  1  0  0  1  0  0 , i  f , 0-1  0-1  0-0  0-0
[s-[c(1)] o-[c(2)] v-[agr],  s-[c(1)] v-[agr]]
-----------------------------------------------------------
#41   1  1  1  0  0  1  0  0 , i  f , 0-0  1-0  0-1  0-1
[s-[] o-[] v-[tns,asp],  s-[] v-[tns,asp]]
```

```
-----------------------------------------------------
#42  1 1 1 0 0 1 0 0 , i f , 0-0 1-0 0-1 0-0
[s-[] o-[] v-[tns], s-[] v-[tns]]
-----------------------------------------------------
#43  1 1 1 0 0 1 0 0 , i f , 0-0 1-0 0-0 0-1
[s-[] o-[] v-[asp], s-[] v-[asp]]
-----------------------------------------------------
#44  1 1 1 0 0 1 0 0 , i f , 0-0 1-0 0-0 0-0
[s-[] o-[] v-[], s-[] v-[]]
-----------------------------------------------------
#45  1 1 1 0 0 1 0 0 , i f , 0-0 0-1 0-1 0-1
[s-[] o-[] v-[agr,tns,asp], s-[] v-[agr,tns,asp]]
-----------------------------------------------------
#46  1 1 1 0 0 1 0 0 , i f , 0-0 0-1 0-1 0-0
[s-[] o-[] v-[agr,tns], s-[] v-[agr,tns]]
-----------------------------------------------------
#47  1 1 1 0 0 1 0 0 , i f , 0-0 0-1 0-0 0-1
[s-[] o-[] v-[agr,asp], s-[] v-[agr,asp]]
-----------------------------------------------------
#48  1 1 1 0 0 1 0 0 , i f , 0-0 0-1 0-0 0-0
[s-[] o-[] v-[agr], s-[] v-[agr]]
-----------------------------------------------------
```

Appendix C

Partial Ordering of Parameter Settings

(1) 0.0.0

[0 0 0 0 0 0 0 0 0]

(2) 0.0.1

[0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 1 0 0 0]
[0 0 0 0 1 0 0 0 0]
[0 0 0 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0]
[0 0 0 0 0 1 1 0 0]

(3) 0.0.2

[0 0 0 0 0 1 0 1 0]
[0 0 0 0 0 1 1 0 0]
[0 0 0 0 1 0 0 1 0]
[0 0 0 0 1 0 1 0 0]
[0 0 0 0 1 1 0 0 0]
[0 0 0 1 0 0 0 1 0]
[0 0 0 1 0 0 1 0 0]
[0 0 0 1 0 1 0 0 0]
[0 0 0 1 1 0 0 0 0]
[0 0 1 0 0 0 0 1 0]
[0 0 1 0 0 0 1 0 0]
[0 0 1 0 0 1 0 0 0]
[0 0 1 0 1 0 0 0 0]
[0 0 1 1 0 0 0 0 0]
[0 1 0 0 0 0 0 1 0]
[0 1 0 0 0 0 1 0 0]
[0 1 0 0 0 1 0 0 0]
[0 1 0 0 1 0 0 0 0]
[0 1 0 1 0 0 0 0 0]
[0 1 1 0 0 0 0 0 0]
[1 0 0 0 0 0 0 1 0]
[1 0 0 0 0 0 1 0 0]
[1 0 0 0 0 1 0 0 0]
[1 0 0 0 1 0 0 0 0]
[1 0 1 0 0 0 0 0 0]
[1 1 0 0 0 0 0 0 0]

(4) 0.0.3

[0 0 0 0 1 1 1 0]
[0 0 0 1 0 1 1 0]
[0 0 0 1 0 1 1 0]
[0 0 0 1 1 0 1 0]
[0 0 0 1 1 1 0 0]
[0 0 1 0 0 1 1 0]
[0 0 1 0 1 0 1 0]
[0 0 1 0 1 1 0 0]
[0 0 1 1 0 0 1 0]
[0 0 1 1 0 1 0 0]
[0 0 1 1 1 0 0 0]
[0 1 0 0 0 1 1 0]
[0 1 0 0 1 0 1 0]
[0 1 0 0 1 1 0 0]
[0 1 0 1 0 0 1 0]
[0 1 0 1 0 1 0 0]
[0 1 0 1 1 0 0 0]
[0 1 1 0 0 0 1 0]
[0 1 1 0 0 1 0 0]
[0 1 1 0 1 0 0 0]
[0 1 1 1 0 0 0 0]
[1 0 0 0 0 1 1 0]
[1 0 0 0 1 0 1 0]
[1 0 0 0 1 1 0 0]
[1 0 0 1 0 0 1 0]
[1 0 0 1 0 1 0 0]
[1 0 0 1 1 0 0 0]
[1 0 1 0 0 0 1 0]
[1 0 1 0 0 1 0 0]
[1 0 1 0 1 0 0 0]
[1 0 1 1 0 0 0 0]
[1 1 0 0 0 0 1 0]
[1 1 0 0 0 1 0 0]
[1 1 0 0 1 0 0 0]
[1 1 0 1 0 0 0 0]
[1 1 1 0 0 0 0 0]

(5) 0.0.4

[0 0 0 1 1 1 1 0]
[0 0 1 0 1 1 1 0]
[0 0 1 1 0 1 1 0]
[0 0 1 1 1 0 1 0]
[0 0 1 1 1 1 0 0]
[0 1 0 0 1 1 1 0]

```
[ 0 1 0 1 0 1 1 0 ]
[ 0 1 0 1 1 0 1 0 ]
[ 0 1 0 1 1 1 0 0 ]
[ 0 1 1 0 0 1 1 0 ]
[ 0 1 1 0 1 0 1 0 ]
[ 0 1 1 0 1 1 0 0 ]
[ 0 1 1 1 0 0 1 0 ]
[ 0 1 1 1 0 1 0 0 ]
[ 0 1 1 1 1 0 0 0 ]
[ 1 0 0 0 1 1 1 0 ]
[ 1 0 0 1 0 1 1 0 ]
[ 1 0 0 1 1 0 1 0 ]
[ 1 0 0 1 1 1 0 0 ]
[ 1 0 1 0 0 1 1 0 ]
[ 1 0 1 0 1 0 1 0 ]
[ 1 0 1 0 1 1 0 0 ]
[ 1 0 1 1 0 0 1 0 ]
[ 1 0 1 1 0 1 0 0 ]
[ 1 0 1 1 1 0 0 0 ]
[ 1 1 0 0 0 1 1 0 ]
[ 1 1 0 0 1 0 1 0 ]
[ 1 1 0 0 1 1 0 0 ]
[ 1 1 0 1 0 0 1 0 ]
[ 1 1 0 1 0 1 0 0 ]
[ 1 1 0 1 1 0 0 0 ]
[ 1 1 1 0 0 0 1 0 ]
[ 1 1 1 0 0 1 0 0 ]
[ 1 1 1 0 1 0 0 0 ]
[ 1 1 1 1 0 0 0 0 ]
```

(6) 0.0.5

```
[ 0 0 1 1 1 1 1 0 ]
[ 0 1 0 1 1 1 1 0 ]
[ 0 1 1 0 1 1 1 0 ]
[ 0 1 1 1 0 1 1 0 ]
[ 0 1 1 1 1 0 1 0 ]
[ 0 1 1 1 1 1 0 0 ]
[ 1 0 0 1 1 1 1 0 ]
[ 1 0 1 0 1 1 1 0 ]
[ 1 0 1 1 0 1 1 0 ]
[ 1 0 1 1 1 0 1 0 ]
[ 1 0 1 1 1 1 0 0 ]
[ 1 1 0 0 1 1 1 0 ]
[ 1 1 0 1 0 1 1 0 ]
[ 1 1 0 1 1 0 1 0 ]
[ 1 1 0 1 1 1 0 0 ]
[ 1 1 1 0 0 1 1 0 ]
[ 1 1 1 0 1 0 1 0 ]
[ 1 1 1 0 1 1 0 0 ]
[ 1 1 1 1 0 0 1 0 ]
[ 1 1 1 1 1 0 0 0 ]
```

(7) 0.0.6

```
[ 0 1 1 1 1 1 1 0 ]
[ 1 0 1 1 1 1 1 0 ]
[ 1 1 0 1 1 1 1 0 ]
[ 1 1 1 0 1 1 1 0 ]
[ 1 1 1 1 0 1 1 0 ]
[ 1 1 1 1 1 0 1 0 ]
[ 1 1 1 1 1 1 0 0 ]
```

(8) 0.0.7

```
[ 1 1 1 1 1 1 1 0 ]
```

(9) 0.1.1

```
[ 0 0 0 0 0 0 0 1 ]
```

(10) 0.1.2

```
[ 0 0 0 0 0 0 1 1 ]
[ 0 0 0 0 0 1 0 1 ]
[ 0 0 0 0 1 0 0 1 ]
[ 0 0 0 1 0 0 0 1 ]
[ 0 0 1 0 0 0 0 1 ]
[ 0 1 0 0 0 0 0 1 ]
[ 1 0 0 0 0 0 0 1 ]
```

(11) 0.1.3

```
[ 0 0 0 0 0 1 1 1 ]
[ 0 0 0 0 1 0 1 1 ]
[ 0 0 0 0 1 1 0 1 ]
[ 0 0 0 1 0 0 1 1 ]
[ 0 0 0 1 0 1 0 1 ]
[ 0 0 0 1 1 0 0 1 ]
[ 0 0 1 0 0 0 1 1 ]
[ 0 0 1 0 0 1 0 1 ]
[ 0 0 1 0 1 0 0 1 ]
[ 0 0 1 1 0 0 0 1 ]
[ 0 1 0 0 0 0 1 1 ]
[ 0 1 0 0 0 1 0 1 ]
[ 0 1 0 0 1 0 0 1 ]
[ 0 1 0 1 0 0 0 1 ]
[ 0 1 1 0 0 0 0 1 ]
[ 1 0 0 0 0 0 1 1 ]
[ 1 0 0 0 0 1 0 1 ]
[ 1 0 0 0 1 0 0 1 ]
[ 1 0 0 1 0 0 0 1 ]
[ 1 0 1 0 0 0 0 1 ]
[ 1 1 0 0 0 0 0 1 ]
```

(12) 0.1.4

```
[ 0 0 0 0 1 1 1 1 ]
[ 0 0 0 1 0 1 1 1 ]
[ 0 0 0 1 1 0 1 1 ]
[ 0 0 0 1 1 1 0 1 ]
[ 0 0 1 0 0 1 1 1 ]
[ 0 0 1 0 1 0 1 1 ]
[ 0 0 1 0 1 1 0 1 ]
[ 0 0 1 1 0 0 1 1 ]
[ 0 0 1 1 0 1 0 1 ]
[ 0 0 1 1 1 0 0 1 ]
[ 0 1 0 0 0 1 1 1 ]
[ 0 1 0 0 1 0 1 1 ]
[ 0 1 0 0 1 1 0 1 ]
[ 0 1 0 1 0 0 1 1 ]
[ 0 1 0 1 0 1 0 1 ]
[ 0 1 0 1 1 0 0 1 ]
```

```
[ 0 1 1 0 0 0 1 1 ]
[ 0 1 1 0 0 1 0 1 ]
[ 0 1 1 0 1 0 0 1 ]
[ 0 1 1 1 0 0 0 1 ]
[ 1 0 0 0 0 1 1 1 ]
[ 1 0 0 0 1 0 1 1 ]
[ 1 0 0 0 1 1 0 1 ]
[ 1 0 0 1 0 0 1 1 ]
[ 1 0 0 1 0 1 0 1 ]
[ 1 0 0 1 1 0 0 1 ]
[ 1 0 1 0 0 0 1 1 ]
[ 1 0 1 0 0 1 0 1 ]
[ 1 0 1 0 1 0 0 1 ]
[ 1 0 1 1 0 0 0 1 ]
[ 1 1 0 0 0 0 1 1 ]
[ 1 1 0 0 0 1 0 1 ]
[ 1 1 0 0 1 0 0 1 ]
[ 1 1 0 1 0 0 0 1 ]
[ 1 1 1 0 0 0 0 1 ]
```

(13) 0.1.5

```
[ 0 0 0 1 1 1 1 1 ]
[ 0 0 1 0 1 1 1 1 ]
[ 0 0 1 1 0 1 1 1 ]
[ 0 0 1 1 1 0 1 1 ]
[ 0 0 1 1 1 1 0 1 ]
[ 0 1 0 0 1 1 1 1 ]
[ 0 1 0 1 0 1 1 1 ]
[ 0 1 0 1 1 0 1 1 ]
[ 0 1 0 1 1 1 0 1 ]
[ 0 1 1 0 0 1 1 1 ]
[ 0 1 1 0 1 0 1 1 ]
[ 0 1 1 0 1 1 0 1 ]
[ 0 1 1 1 0 0 1 1 ]
[ 0 1 1 1 0 1 0 1 ]
[ 0 1 1 1 1 0 0 1 ]
[ 1 0 0 0 1 1 1 1 ]
[ 1 0 0 1 0 1 1 1 ]
[ 1 0 0 1 1 0 1 1 ]
[ 1 0 0 1 1 1 0 1 ]
[ 1 0 1 0 0 1 1 1 ]
[ 1 0 1 0 1 0 1 1 ]
[ 1 0 1 0 1 1 0 1 ]
[ 1 0 1 1 0 0 1 1 ]
[ 1 0 1 1 0 1 0 1 ]
[ 1 0 1 1 1 0 0 1 ]
[ 1 1 0 0 0 1 1 1 ]
[ 1 1 0 0 1 0 1 1 ]
[ 1 1 0 0 1 1 0 1 ]
[ 1 1 0 1 0 0 1 1 ]
[ 1 1 0 1 0 1 0 1 ]
[ 1 1 0 1 1 0 0 1 ]
[ 1 1 1 0 0 0 1 1 ]
[ 1 1 1 0 0 1 0 1 ]
[ 1 1 1 0 1 0 0 1 ]
[ 1 1 1 1 0 0 0 1 ]
```

(14) 0.1.6

```
[ 0 0 1 1 1 1 1 1 ]
[ 0 1 0 1 1 1 1 1 ]
[ 0 1 1 0 1 1 1 1 ]
[ 0 1 1 1 0 1 1 1 ]
[ 0 1 1 1 1 0 1 1 ]
[ 0 1 1 1 1 1 0 1 ]
[ 1 0 1 1 1 1 1 1 ]
[ 1 0 1 1 1 1 1 1 ]
[ 1 0 1 1 0 1 1 1 ]
[ 1 0 1 1 1 0 1 1 ]
[ 1 0 1 1 1 1 0 1 ]
[ 1 1 0 0 1 1 1 1 ]
[ 1 1 0 1 0 1 1 1 ]
[ 1 1 0 1 1 0 1 1 ]
[ 1 1 0 1 1 1 0 1 ]
[ 1 1 1 0 0 1 1 1 ]
[ 1 1 1 0 1 0 1 1 ]
[ 1 1 1 0 1 1 0 1 ]
[ 1 1 1 1 0 0 1 1 ]
[ 1 1 1 1 0 1 0 1 ]
[ 1 1 1 1 1 0 0 1 ]
```

(15) 0.1.7

```
[ 0 1 1 1 1 1 1 1 ]
[ 1 0 1 1 1 1 1 1 ]
[ 1 1 0 1 1 1 1 1 ]
[ 1 1 1 0 1 1 1 1 ]
[ 1 1 1 1 0 1 1 1 ]
[ 1 1 1 1 1 0 1 1 ]
[ 1 1 1 1 1 1 0 1 ]
```

(16) 0.1.8

```
[ 1 1 1 1 1 1 1 1 ]
```

(17) 1.0.1

```
[ 0 0 0 0 0 1/0 0 0 ]
[ 0 0 0 0 0 0 1/0 0 ]
[ 0 0 0 0 0 1/0 1 0 ]
```

(18) 1.0.2

```
[ 0 0 0 0 0 1 1/0 0 ]
[ 0 0 0 0 0 1 1/0 0 0 ]
[ 0 0 0 0 0 1 0 1/0 0 ]
[ 0 0 0 0 1 0 1/0 0 0 ]
[ 0 0 0 0 1 0 0 1/0 0 ]
[ 0 0 0 1 0 0 1/0 0 0 ]
[ 0 0 0 1 0 0 0 1/0 0 ]
[ 0 1 0 0 0 1/0 0 0 ]
[ 0 1 0 0 0 0 1/0 0 ]
[ 1 0 0 0 0 1/0 0 0 ]
[ 1 0 0 0 0 0 1/0 0 ]
```

(19) 1.0.3

```
[ 0 0 0 0 1 1/0 1 0 ]
[ 0 0 0 0 1 1 1/0 0 ]
[ 0 0 0 0 1 0 1/0 1 0 ]
[ 0 0 0 0 1 0 1 1/0 0 ]
[ 0 0 0 0 1 1 1/0 0 0 ]
[ 0 0 0 0 1 1 0 1/0 0 ]
```

```
[ 0 0 1 0 0 1/0 1 0 ]
[ 0 0 1 0 0 1 1/0 0 ]
[ 0 0 1 0 1 1/0 0 0 ]
[ 0 0 1 0 1 0 1/0 0 ]
[ 0 0 1 1 0 1/0 0 0 ]
[ 0 0 1 1 0 0 1/0 0 ]
[ 0 1 0 0 0 1/0 1 0 ]
[ 0 1 0 0 0 1 1/0 0 ]
[ 0 1 0 0 1 1/0 0 0 ]
[ 0 1 0 0 1 0 1/0 0 ]
[ 0 1 0 1 0 1/0 0 0 ]
[ 0 1 0 1 0 0 1/0 0 ]
[ 0 1 1 0 0 1/0 0 0 ]
[ 0 1 1 0 0 0 1/0 0 ]
[ 1 0 0 0 0 1/0 1 0 ]
[ 1 0 0 0 0 1 1/0 0 ]
[ 1 0 0 0 1 1/0 0 0 ]
[ 1 0 0 0 1 0 1/0 0 ]
[ 1 0 0 1 0 1/0 0 0 ]
[ 1 0 0 1 0 0 1/0 0 ]
[ 1 0 1 0 0 1/0 0 0 ]
[ 1 0 1 0 0 0 1/0 0 ]
[ 1 1 0 0 0 1/0 0 0 ]
[ 1 1 0 0 0 0 1/0 0 ]
```

(20) 1.0.4

```
[ 0 0 0 1 1 1/0 1 0 ]
[ 0 0 0 1 1 1 1/0 0 ]
[ 0 0 1 0 1 1/0 1 0 ]
[ 0 0 1 0 1 1 1/0 0 ]
[ 0 0 1 1 0 1/0 1 0 ]
[ 0 0 1 1 0 1 1/0 0 ]
[ 0 0 1 1 1 1/0 0 0 ]
[ 0 0 1 1 1 0 1/0 0 ]
[ 0 1 0 0 1 1/0 1 0 ]
[ 0 1 0 0 1 1 1/0 0 ]
[ 0 1 0 1 0 1/0 1 0 ]
[ 0 1 0 1 0 1 1/0 0 ]
[ 0 1 0 1 1 1/0 0 0 ]
[ 0 1 0 1 1 0 1/0 0 ]
[ 0 1 1 0 0 1/0 1 0 ]
[ 0 1 1 0 0 1 1/0 0 ]
[ 0 1 1 0 1 1/0 0 0 ]
[ 0 1 1 0 1 0 1/0 0 ]
[ 0 1 1 1 0 1/0 0 0 ]
[ 0 1 1 1 0 0 1/0 0 ]
[ 1 0 0 0 1 1/0 1 0 ]
[ 1 0 0 0 1 1 1/0 0 ]
[ 1 0 0 1 0 1/0 1 0 ]
[ 1 0 0 1 0 0 1/0 0 ]
[ 1 0 0 1 1 1/0 0 0 ]
[ 1 0 0 1 1 0 1/0 0 ]
[ 1 0 1 0 0 1/0 1 0 ]
[ 1 0 1 0 0 1 1/0 0 ]
[ 1 0 1 0 1 1/0 0 0 ]
[ 1 0 1 0 1 0 1/0 0 ]
[ 1 0 1 1 0 1/0 0 0 ]
[ 1 0 1 1 0 0 1/0 0 ]
[ 1 1 0 0 0 1/0 1 0 ]
[ 1 1 0 0 0 1 1/0 0 ]
[ 1 1 0 0 1 1/0 0 0 ]
```

```
[ 1 1 0 0 1 0 1/0 0 ]
[ 1 1 0 1 0 1/0 0 0 ]
[ 1 1 0 1 0 0 1/0 0 ]
[ 1 1 1 0 0 1/0 0 0 ]
[ 1 1 1 0 0 0 1/0 0 ]
```

(21) 1.0.5

```
[ 0 0 1 1 1 1/0 1 0 ]
[ 0 0 1 1 1 1 1/0 0 ]
[ 0 1 0 1 1 1/0 1 0 ]
[ 0 1 0 1 1 1 1/0 0 ]
[ 0 1 1 0 1 1/0 1 0 ]
[ 0 1 1 0 1 1 1/0 0 ]
[ 0 1 1 1 0 1/0 1 0 ]
[ 0 1 1 1 0 1 1/0 0 ]
[ 0 1 1 1 1 1/0 0 0 ]
[ 0 1 1 1 1 0 1/0 0 ]
[ 1 0 0 1 1 1/0 1 0 ]
[ 1 0 0 1 1 1 1/0 0 ]
[ 1 0 1 0 1 1/0 1 0 ]
[ 1 0 1 0 1 1 1/0 0 ]
[ 1 0 1 1 0 1/0 1 0 ]
[ 1 0 1 1 0 1 1/0 0 ]
[ 1 0 1 1 1 1/0 0 0 ]
[ 1 0 1 1 1 0 1/0 0 ]
[ 1 1 0 0 1 1/0 1 0 ]
[ 1 1 0 0 1 1 1/0 0 ]
[ 1 1 0 1 0 1/0 1 0 ]
[ 1 1 0 1 0 1 1/0 0 ]
[ 1 1 0 1 1 1/0 0 0 ]
[ 1 1 0 1 1 0 1/0 0 ]
[ 1 1 1 0 0 1/0 1 0 ]
[ 1 1 1 0 0 1 1/0 0 ]
[ 1 1 1 0 1 1/0 0 0 ]
[ 1 1 1 0 1 0 1/0 0 ]
[ 1 1 1 1 0 1/0 0 0 ]
[ 1 1 1 1 0 0 1/0 0 ]
```

(22) 1.0.6

```
[ 0 1 1 1 1 1/0 1 0 ]
[ 0 1 1 1 1 1 1/0 0 ]
[ 1 0 1 1 1 1/0 1 0 ]
[ 1 0 1 1 1 1 1/0 0 ]
[ 1 1 0 1 1 1/0 1 0 ]
[ 1 1 0 1 1 1 1/0 0 ]
[ 1 1 1 0 1 1/0 1 0 ]
[ 1 1 1 0 1 1 1/0 0 ]
[ 1 1 1 1 0 1/0 1 0 ]
[ 1 1 1 1 0 1 1/0 0 ]
[ 1 1 1 1 1 1/0 0 0 ]
[ 1 1 1 1 1 0 1/0 0 ]
```

(23) 1.0.7

```
[ 1 1 1 1 1 1/0 1 0 ]
[ 1 1 1 1 1 1 1/0 0 ]
```

(24) 1.1.1

```
[ 0 0 0 0 0 0 0 1/0 ]
```

(25) 1.1.2

[0 0 0 0 0 1/0 0 1]
[0 0 0 0 0 0 1/0 1]
[0 0 0 0 0 0 1 1/0]
[0 0 0 0 0 1 0 1/0]
[0 0 0 0 1 0 0 1/0]
[0 0 0 1 0 0 0 1/0]
[0 0 1 0 0 0 0 1/0]
[0 1 0 0 0 0 0 1/0]
[1 0 0 0 0 0 0 1/0]

(26) 1.1.3

[0 0 0 0 0 1/0 1 1]
[0 0 0 0 0 1 1/0 1]
[0 0 0 0 0 1 1 1/0]
[0 0 0 0 1 1/0 0 1]
[0 0 0 0 1 0 1/0 1]
[0 0 0 0 1 0 1 1/0]
[0 0 0 0 1 1 0 1/0]
[0 0 0 1 0 1/0 0 1]
[0 0 0 1 0 0 1/0 1]
[0 0 0 1 0 0 1 1/0]
[0 0 0 1 0 1 0 1/0]
[0 0 0 1 1 0 0 1/0]
[0 0 1 0 0 1/0 0 1]
[0 0 1 0 0 0 1/0 1]
[0 0 1 0 0 0 1 1/0]
[0 0 1 0 0 1 0 1/0]
[0 0 1 0 1 0 0 1/0]
[0 0 1 1 0 0 0 1/0]
[0 1 0 0 0 1/0 0 1]
[0 1 0 0 0 0 1/0 1]
[0 1 0 0 0 0 1 1/0]
[0 1 0 0 0 1 0 1/0]
[0 1 0 0 1 0 0 1/0]
[0 1 0 1 0 0 0 1/0]
[0 1 1 0 0 0 0 1/0]
[1 0 0 0 0 1/0 0 1]
[1 0 0 0 0 0 1/0 1]
[1 0 0 0 0 0 1 1/0]
[1 0 0 0 0 1 0 1/0]
[1 0 0 0 1 0 0 1/0]
[1 0 0 1 0 0 0 1/0]
[1 0 1 0 0 0 0 1/0]
[1 1 0 0 0 0 0 1/0]

(27) 1.1.4

[0 0 0 0 1 1/0 1 1]
[0 0 0 0 1 1 1/0 1]
[0 0 0 0 1 1 1 1/0]
[0 0 0 1 0 1/0 1 1]
[0 0 0 1 0 1 1/0 1]
[0 0 0 1 0 1 1 1/0]
[0 0 0 1 1 1/0 0 1]
[0 0 0 1 1 0 1/0 1]
[0 0 0 1 1 0 1 1/0]
[0 0 0 1 1 1 0 1/0]
[0 0 1 0 0 1/0 1 1]
[0 0 1 0 0 1 1/0 1]
[0 0 1 0 0 1 1 1/0]
[0 0 1 0 1 1/0 0 1]
[0 0 1 0 1 0 1/0 1]
[0 0 1 0 1 0 1 1/0]
[0 0 1 0 1 1 0 1/0]
[0 0 1 1 0 1/0 0 1]
[0 0 1 1 0 0 1/0 1]
[0 0 1 1 0 0 1 1/0]
[0 0 1 1 0 1 0 1/0]
[0 0 1 1 1 0 0 1/0]
[0 1 0 0 0 1/0 1 1]
[0 1 0 0 0 1 1/0 1]
[0 1 0 0 0 1 1 1/0]
[0 1 0 0 1 1/0 0 1]
[0 1 0 0 1 0 1/0 1]
[0 1 0 0 1 0 1 1/0]
[0 1 0 0 1 1 0 1/0]
[0 1 0 1 0 1/0 0 1]
[0 1 0 1 0 0 1/0 1]
[0 1 0 1 0 0 1 1/0]
[0 1 0 1 0 1 0 1/0]
[0 1 0 1 1 0 0 1/0]
[0 1 1 0 0 1/0 0 1]
[0 1 1 0 0 0 1/0 1]
[0 1 1 0 0 0 1 1/0]
[0 1 1 0 0 1 0 1/0]
[0 1 1 0 1 0 0 1/0]
[0 1 1 1 0 0 0 1/0]
[1 0 0 0 0 1/0 1 1]
[1 0 0 0 0 1 1/0 1]
[1 0 0 0 0 1 1 1/0]
[1 0 0 0 1 1/0 0 1]
[1 0 0 0 1 0 1/0 1]
[1 0 0 0 1 0 1 1/0]
[1 0 0 0 1 1 0 1/0]
[1 0 0 1 0 1/0 0 1]
[1 0 0 1 0 0 1/0 1]
[1 0 0 1 0 0 1 1/0]
[1 0 0 1 0 1 0 1/0]
[1 0 0 1 1 0 0 1/0]
[1 0 1 0 0 1/0 0 1]
[1 0 1 0 0 0 1/0 1]
[1 0 1 0 0 0 1 1/0]
[1 0 1 0 0 1 0 1/0]
[1 0 1 0 1 0 0 1/0]
[1 0 1 1 0 0 0 1/0]
[1 1 0 0 0 1/0 0 1]
[1 1 0 0 0 0 1/0 1]
[1 1 0 0 0 0 1 1/0]
[1 1 0 0 0 1 0 1/0]
[1 1 0 0 1 0 0 1/0]
[1 1 0 1 0 0 0 1/0]
[1 1 1 0 0 0 0 1/0]

(28) 1.1.5.

[0 0 0 1 1 1/0 1 1]
[0 0 0 1 1 1 1/0 1]
[0 0 0 1 1 1 1 1/0]
[0 0 1 0 1 1/0 1 1]
[0 0 1 0 1 1 1/0 1]

[0 0 1 0 1 1 1 1/0]
[0 0 1 1 0 1/0 1 1]
[0 0 1 1 0 1 1/0 1]
[0 0 1 1 0 1 1 1/0]
[0 0 1 1 1 1/0 0 1]
[0 0 1 1 1 0 1/0 1]
[0 0 1 1 1 0 1 1/0]
[0 0 1 1 1 1 0 1/0]
[0 1 0 0 1 1/0 1 1]
[0 1 0 0 1 1 1/0 1]
[0 1 0 0 1 1 1 1/0]
[0 1 0 1 0 1/0 1 1]
[0 1 0 1 0 1 1/0 1]
[0 1 0 1 0 1 1 1/0]
[0 1 0 1 1 1/0 0 1]
[0 1 0 1 1 0 1/0 1]
[0 1 0 1 1 0 1 1/0]
[0 1 0 1 1 1 0 1/0]
[0 1 1 0 0 1/0 1 1]
[0 1 1 0 0 1 1/0 1]
[0 1 1 0 0 1 1 1/0]
[0 1 1 0 1 1/0 0 1]
[0 1 1 0 1 0 1/0 1]
[0 1 1 0 1 0 1 1/0]
[0 1 1 0 1 1 0 1/0]
[0 1 1 1 0 1/0 0 1]
[0 1 1 1 0 0 1/0 1]
[0 1 1 1 0 0 1 1/0]
[0 1 1 1 0 1 0 1/0]
[0 1 1 1 1 0 0 1/0]
[1 0 0 0 1 1/0 1 1]
[1 0 0 0 1 1 1/0 1]
[1 0 0 0 1 1 1 1/0]
[1 0 0 1 0 1/0 1 1]
[1 0 0 1 0 1 1/0 1]
[1 0 0 1 0 1 1 1/0]
[1 0 0 1 1 1/0 0 1]
[1 0 0 1 1 0 1/0 1]
[1 0 0 1 1 0 1 1/0]
[1 0 0 1 1 1 0 1/0]
[1 0 1 0 0 1/0 1 1]
[1 0 1 0 0 1 1/0 1]
[1 0 1 0 0 1 1 1/0]
[1 0 1 0 1 1/0 0 1]
[1 0 1 0 1 0 1/0 1]
[1 0 1 0 1 0 1 1/0]
[1 0 1 0 1 1 0 1/0]
[1 0 1 1 0 1/0 0 1]
[1 0 1 1 0 0 1/0 1]
[1 0 1 1 0 0 1 1/0]
[1 0 1 1 0 1 0 1/0]
[1 0 1 1 1 0 0 1/0]
[1 1 0 0 0 1/0 1 1]
[1 1 0 0 0 1 1/0 1]
[1 1 0 0 0 1 1 1/0]
[1 1 0 0 1 1/0 0 1]
[1 1 0 0 1 0 1/0 1]
[1 1 0 0 1 0 1 1/0]
[1 1 0 1 0 1/0 1 1]
[1 1 0 1 0 0 1/0 1]
[1 1 0 1 0 0 1 1/0]
[1 1 0 1 1 0 1/0 1]
[1 1 0 1 1 0 0 1/0]
[1 1 1 0 0 1/0 1 1]
[1 1 1 0 0 1 1/0 1]
[1 1 1 0 0 1 1 1/0]
[1 1 1 0 1 1/0 0 1]
[1 1 1 0 1 0 1/0 1]
[1 1 1 0 1 0 1 1/0]
[1 1 1 0 1 1 0 1/0]
[1 1 1 1 0 1/0 0 1]
[1 1 1 1 0 0 1/0 1]
[1 1 1 1 0 0 0 1/0]

(29) 1.1.6.

[0 0 1 1 1 1/0 1 1]
[0 0 1 1 1 1 1/0 1]
[0 0 1 1 1 1 1 1/0]
[0 1 0 1 1 1/0 1 1]
[0 1 0 1 1 1 1/0 1]
[0 1 0 1 1 1 1 1/0]
[0 1 1 0 1 1/0 1 1]
[0 1 1 0 1 1 1/0 1]
[0 1 1 0 1 1 1 1/0]
[0 1 1 1 0 1/0 1 1]
[0 1 1 1 0 1 1/0 1]
[0 1 1 1 0 1 1 1/0]
[0 1 1 1 1 1/0 0 1]
[0 1 1 1 1 0 1/0 1]
[0 1 1 1 1 0 1 1/0]
[0 1 1 1 1 1 0 1/0]
[1 0 0 1 1 1/0 1 1]
[1 0 0 1 1 1 1/0 1]
[1 0 0 1 1 1 1 1/0]
[1 0 1 0 1 1/0 1 1]
[1 0 1 0 1 1 1/0 1]
[1 0 1 0 1 1 1 1/0]
[1 0 1 1 0 1/0 1 1]
[1 0 1 1 0 1 1/0 1]
[1 0 1 1 0 1 1 1/0]
[1 0 1 1 1 1/0 0 1]
[1 0 1 1 1 0 1/0 1]
[1 0 1 1 1 0 1 1/0]
[1 0 1 1 1 1 0 1/0]
[1 1 0 0 1 1/0 1 1]
[1 1 0 0 1 1 1/0 1]
[1 1 0 0 1 1 1 1/0]
[1 1 0 1 0 1/0 1 1]
[1 1 0 1 0 1 1/0 1]
[1 1 0 1 0 1 1 1/0]
[1 1 0 1 1 1/0 0 1]
[1 1 0 1 1 0 1/0 1]
[1 1 0 1 1 0 1 1/0]
[1 1 0 1 1 1 0 1/0]
[1 1 1 0 0 1/0 1 1]
[1 1 1 0 0 1 1/0 1]
[1 1 1 0 0 1 1 1/0]
[1 1 1 0 1 1/0 0 1]
[1 1 1 0 1 0 1/0 1]
[1 1 1 0 1 0 1 1/0]
[1 1 1 0 1 1 0 1/0]
[1 1 1 1 0 1/0 0 1]
[1 1 1 1 0 0 1/0 1]
[1 1 1 1 0 0 0 1/0]

(30) 1.1.7

```
[  0  1  1  1  1  1/0  1  1  ]
[  0  1  1  1  1  1  1/0  1  ]
[  0  1  1  1  1  1  1  1/0  ]
[  1  0  1  1  1  1/0  1  1  ]
[  1  0  1  1  1  1  1/0  1  ]
[  1  0  1  1  1  1  1  1/0  ]
[  1  1  0  1  1  1/0  1  1  ]
[  1  1  0  1  1  1  1/0  1  ]
[  1  1  0  1  1  1  1  1/0  ]
[  1  1  1  0  1  1/0  1  1  ]
[  1  1  1  0  1  1  1/0  1  ]
[  1  1  1  0  1  1  1  1/0  ]
[  1  1  1  1  0  1/0  1  1  ]
[  1  1  1  1  0  1  1/0  1  ]
[  1  1  1  1  0  1  1  1/0  ]
[  1  1  1  1  1  1/0  0  1  ]
[  1  1  1  1  1  0  1/0  1  ]
[  1  1  1  1  1  0  1  1/0  ]
[  1  1  1  1  1  1  0  1/0  ]
```

(31) 1.1.8

```
[  1  1  1  1  1  1/0  1  1  ]
[  1  1  1  1  1  1  1/0  1  ]
[  1  1  1  1  1  1  1  1/0  ]
```

(32) 2.0.2

```
[  0  0  0  0  0  1/0  1/0  0  ]
```

(33) 2.0.3

```
[  0  0  0  0  1  1/0  1/0  0  ]
[  0  0  0  1  0  1/0  1/0  0  ]
[  0  0  1  0  0  1/0  1/0  0  ]
[  0  1  0  0  0  1/0  1/0  0  ]
[  1  0  0  0  0  1/0  1/0  0  ]
```

(34) 2.0.4

```
[  0  0  0  1  1  1/0  1/0  0  ]
[  0  0  1  0  1  1/0  1/0  0  ]
[  0  0  1  1  0  1/0  1/0  0  ]
[  0  1  0  0  1  1/0  1/0  0  ]
[  0  1  0  1  0  1/0  1/0  0  ]
[  0  1  1  0  0  1/0  1/0  0  ]
[  1  0  0  0  1  1/0  1/0  0  ]
[  1  0  0  1  0  1/0  1/0  0  ]
[  1  0  1  0  0  1/0  1/0  0  ]
[  1  1  0  0  0  1/0  1/0  0  ]
```

(35) 2.0.5

```
[  0  0  1  1  1  1/0  1/0  0  ]
[  0  1  0  1  1  1/0  1/0  0  ]
[  0  1  1  0  1  1/0  1/0  0  ]
[  0  1  1  1  0  1/0  1/0  0  ]
[  1  0  0  1  1  1/0  1/0  0  ]
[  1  0  1  0  1  1/0  1/0  0  ]
[  1  0  1  1  0  1/0  1/0  0  ]
[  1  1  0  0  1  1/0  1/0  0  ]
[  1  1  0  1  0  1/0  1/0  0  ]
[  1  1  1  0  0  1/0  1/0  0  ]
```

(36) 2.0.6

```
[  0  1  1  1  1  1/0  1/0  0  ]
[  1  0  1  1  1  1/0  1/0  0  ]
[  1  1  0  1  1  1/0  1/0  0  ]
[  1  1  1  0  1  1/0  1/0  0  ]
[  1  1  1  1  0  1/0  1/0  0  ]
```

(37) 2.0.7

```
[  1  1  1  1  1  1/0  1/0  0  ]
```

(38) 2.1.2

```
[  0  0  0  0  0  1/0  0  1/0  ]
[  0  0  0  0  0  0  1/0  1/0  ]
```

(39) 2.1.3

```
[  0  0  0  0  0  1/0  1/0  1  ]
[  0  0  0  0  0  1/0  1  1/0  ]
[  0  0  0  0  0  1  1/0  1/0  ]
[  0  0  0  0  1  1/0  0  1/0  ]
[  0  0  0  0  1  0  1/0  1/0  ]
[  0  0  0  1  0  1/0  0  1/0  ]
[  0  0  0  1  0  0  1/0  1/0  ]
[  0  0  1  0  0  1/0  0  1/0  ]
[  0  0  1  0  0  0  1/0  1/0  ]
[  0  1  0  0  0  1/0  0  1/0  ]
[  0  1  0  0  0  0  1/0  1/0  ]
[  1  0  0  0  0  1/0  0  1/0  ]
[  1  0  0  0  0  0  1/0  1/0  ]
```

(40) 2.1.4

```
[  0  0  0  0  1  1/0  1/0  1  ]
[  0  0  0  0  1  1/0  1  1/0  ]
[  0  0  0  0  1  1  1/0  1/0  ]
[  0  0  0  1  0  1/0  1/0  1  ]
[  0  0  0  1  0  1/0  1  1/0  ]
[  0  0  0  1  0  1  1/0  1/0  ]
[  0  0  0  1  1  1/0  0  1/0  ]
[  0  0  0  1  1  0  1/0  1/0  ]
[  0  0  1  0  0  1/0  1/0  1  ]
[  0  0  1  0  0  1/0  1  1/0  ]
[  0  0  1  0  0  1  1/0  1/0  ]
[  0  0  1  0  1  1/0  0  1/0  ]
[  0  0  1  0  1  0  1/0  1/0  ]
[  0  0  1  1  0  1/0  0  1/0  ]
[  0  0  1  1  0  0  1/0  1/0  ]
[  0  1  0  0  0  1/0  1/0  1  ]
[  0  1  0  0  0  1/0  1  1/0  ]
[  0  1  0  0  0  1  1/0  1/0  ]
[  0  1  0  0  1  1/0  0  1/0  ]
[  0  1  0  0  1  0  1/0  1/0  ]
[  0  1  0  1  0  1/0  0  1/0  ]
[  0  1  0  1  0  0  1/0  1/0  ]
```

```
[ 0 1 1 0 0 1/0 0 1/0 ]
[ 0 1 1 0 0 0 1/0 1/0 ]
[ 1 0 0 0 0 1/0 1/0 1 ]
[ 1 0 0 0 0 1/0 1 1/0 ]
[ 1 0 0 0 0 1 1/0 1/0 ]
[ 1 0 0 0 1 1/0 0 1/0 ]
[ 1 0 0 0 1 0 1/0 1/0 ]
[ 1 0 0 1 0 1/0 0 1/0 ]
[ 1 0 0 1 0 0 1/0 1/0 ]
[ 1 0 1 0 0 1/0 0 1/0 ]
[ 1 0 1 0 0 0 1/0 1/0 ]
[ 1 1 0 0 0 1/0 0 1/0 ]
[ 1 1 0 0 0 0 1/0 1/0 ]
```

(41) 2.1.5

```
[ 0 0 0 1 1 1/0 1/0 1 ]
[ 0 0 0 1 1 1/0 1 1/0 ]
[ 0 0 0 1 1 1 1/0 1/0 ]
[ 0 0 1 0 1 1/0 1/0 1 ]
[ 0 0 1 0 1 1/0 1 1/0 ]
[ 0 0 1 0 1 1 1/0 1/0 ]
[ 0 0 1 1 0 1/0 1/0 1 ]
[ 0 0 1 1 0 1/0 1 1/0 ]
[ 0 0 1 1 0 1 1/0 1/0 ]
[ 0 0 1 1 1 1/0 0 1/0 ]
[ 0 0 1 1 1 0 1/0 1/0 ]
[ 0 1 0 0 1 1/0 1/0 1 ]
[ 0 1 0 0 1 1/0 1 1/0 ]
[ 0 1 0 0 1 1 1/0 1/0 ]
[ 0 1 0 1 0 1/0 1/0 1 ]
[ 0 1 0 1 0 1/0 1 1/0 ]
[ 0 1 0 1 0 1 1/0 1/0 ]
[ 0 1 0 1 1 1/0 0 1/0 ]
[ 0 1 0 1 1 0 1/0 1/0 ]
[ 0 1 1 0 0 1/0 1/0 1 ]
[ 0 1 1 0 0 1/0 1 1/0 ]
[ 0 1 1 0 0 1 1/0 1/0 ]
[ 0 1 1 0 1 1/0 0 1/0 ]
[ 0 1 1 0 1 0 1/0 1/0 ]
[ 0 1 1 1 0 1/0 0 1/0 ]
[ 0 1 1 1 0 0 1/0 1/0 ]
[ 1 0 0 0 1 1/0 1/0 1 ]
[ 1 0 0 0 1 1/0 1 1/0 ]
[ 1 0 0 0 1 1 1/0 1/0 ]
[ 1 0 0 1 0 1/0 1/0 1 ]
[ 1 0 0 1 0 1/0 1 1/0 ]
[ 1 0 0 1 0 1 1/0 1/0 ]
[ 1 0 0 1 1 1/0 0 1/0 ]
[ 1 0 0 1 1 0 1/0 1/0 ]
[ 1 0 1 0 0 1/0 1/0 1 ]
[ 1 0 1 0 0 1/0 1 1/0 ]
[ 1 0 1 0 0 1 1/0 1/0 ]
[ 1 0 1 0 1 1/0 0 1/0 ]
[ 1 0 1 0 1 0 1/0 1/0 ]
[ 1 0 1 1 0 1/0 0 1/0 ]
[ 1 0 1 1 0 0 1/0 1/0 ]
[ 1 1 0 0 0 1/0 1/0 1 ]
[ 1 1 0 0 0 1/0 1 1/0 ]
[ 1 1 0 0 0 1 1/0 1/0 ]
[ 1 1 0 0 1 1/0 0 1/0 ]
[ 1 1 0 0 1 0 1/0 1/0 ]
```

```
[ 1 1 0 1 0 1/0 0 1/0 ]
[ 1 1 0 1 0 0 1/0 1/0 ]
[ 1 1 1 0 0 1/0 0 1/0 ]
[ 1 1 1 0 0 0 1/0 1/0 ]
```

(42) 2.1.6

```
[ 0 0 1 1 1 1/0 1/0 1 ]
[ 0 0 1 1 1 1/0 1 1/0 ]
[ 0 0 1 1 1 1 1/0 1/0 ]
[ 0 1 0 1 1 1/0 1/0 1 ]
[ 0 1 0 1 1 1/0 1 1/0 ]
[ 0 1 0 1 1 1 1/0 1/0 ]
[ 0 1 1 0 1 1/0 1/0 1 ]
[ 0 1 1 0 1 1/0 1 1/0 ]
[ 0 1 1 0 1 1 1/0 1/0 ]
[ 0 1 1 1 0 1/0 1/0 1 ]
[ 0 1 1 1 0 1/0 1 1/0 ]
[ 0 1 1 1 0 1 1/0 1/0 ]
[ 0 1 1 1 1 1/0 0 1/0 ]
[ 0 1 1 1 1 0 1/0 1/0 ]
[ 1 0 0 1 1 1/0 1/0 1 ]
[ 1 0 0 1 1 1/0 1 1/0 ]
[ 1 0 0 1 1 1 1/0 1/0 ]
[ 1 0 1 0 1 1/0 1/0 1 ]
[ 1 0 1 0 1 1/0 1 1/0 ]
[ 1 0 1 0 1 1 1/0 1/0 ]
[ 1 0 1 1 0 1/0 1/0 1 ]
[ 1 0 1 1 0 1/0 1 1/0 ]
[ 1 0 1 1 0 1 1/0 1/0 ]
[ 1 0 1 1 1 1/0 0 1/0 ]
[ 1 0 1 1 1 0 1/0 1/0 ]
[ 1 1 0 0 1 1/0 1/0 1 ]
[ 1 1 0 0 1 1/0 1 1/0 ]
[ 1 1 0 0 1 1 1/0 1/0 ]
[ 1 1 0 1 0 1/0 1/0 1 ]
[ 1 1 0 1 0 1/0 1 1/0 ]
[ 1 1 0 1 0 1 1/0 1/0 ]
[ 1 1 0 1 1 1/0 0 1/0 ]
[ 1 1 0 1 1 0 1/0 1/0 ]
[ 1 1 1 0 0 1/0 1/0 1 ]
[ 1 1 1 0 0 1/0 1 1/0 ]
[ 1 1 1 0 0 1 1/0 1/0 ]
[ 1 1 1 0 1 1/0 0 1/0 ]
[ 1 1 1 0 1 0 1/0 1/0 ]
[ 1 1 1 1 0 1/0 0 1/0 ]
[ 1 1 1 1 0 0 1/0 1/0 ]
```

(43) 2.1.7

```
[ 0 1 1 1 1 1/0 1/0 1 ]
[ 0 1 1 1 1 1/0 1 1/0 ]
[ 0 1 1 1 1 1 1/0 1/0 ]
[ 1 0 1 1 1 1/0 1/0 1 ]
[ 1 0 1 1 1 1/0 1 1/0 ]
[ 1 0 1 1 1 1 1/0 1/0 ]
[ 1 1 0 1 1 1/0 1/0 1 ]
[ 1 1 0 1 1 1/0 1 1/0 ]
[ 1 1 0 1 1 1 1/0 1/0 ]
[ 1 1 1 0 1 1/0 1/0 1 ]
[ 1 1 1 0 1 1/0 1 1/0 ]
[ 1 1 1 0 1 1 1/0 1/0 ]
```

```
[  1   1   1   1   0  1/0  1/0   1  ]
[  1   1   1   1   0  1/0   1   1/0 ]
[  1   1   1   1   0   1   1/0  1/0 ]
[  1   1   1   1   1  1/0   0   1/0 ]
[  1   1   1   1   1   0   1/0  1/0 ]
```

(44) 2.1.8

```
[  1   1   1   1   1  1/0  1/0   1  ]
[  1   1   1   1   1  1/0   1   1/0 ]
[  1   1   1   1   1   1   1/0  1/0 ]
```

(45) 3.1.3

```
[  0   0   0   0   0  1/0  1/0  1/0 ]
```

(46) 3.1.4

```
[  0   0   0   0   1  1/0  1/0  1/0 ]
[  0   0   0   1   0  1/0  1/0  1/0 ]
[  0   0   1   0   0  1/0  1/0  1/0 ]
[  0   1   0   0   0  1/0  1/0  1/0 ]
[  1   0   0   0   0  1/0  1/0  1/0 ]
```

(47) 3.1.5

```
[  0   0   0   1   1  1/0  1/0  1/0 ]
[  0   0   1   0   1  1/0  1/0  1/0 ]
[  0   0   1   1   0  1/0  1/0  1/0 ]
[  0   1   0   0   1  1/0  1/0  1/0 ]
[  0   1   0   1   0  1/0  1/0  1/0 ]
[  0   1   1   0   0  1/0  1/0  1/0 ]
[  1   0   0   0   1  1/0  1/0  1/0 ]
[  1   0   0   1   0  1/0  1/0  1/0 ]
[  1   0   1   0   0  1/0  1/0  1/0 ]
[  1   1   0   0   0  1/0  1/0  1/0 ]
```

(48) 3.1.6

```
[  0   0   1   1   1  1/0  1/0  1/0 ]
[  0   1   0   1   1  1/0  1/0  1/0 ]
[  0   1   1   0   1  1/0  1/0  1/0 ]
[  0   1   1   1   0  1/0  1/0  1/0 ]
[  1   0   0   1   1  1/0  1/0  1/0 ]
[  1   0   1   0   1  1/0  1/0  1/0 ]
[  1   0   1   1   0  1/0  1/0  1/0 ]
[  1   1   0   0   1  1/0  1/0  1/0 ]
[  1   1   0   1   0  1/0  1/0  1/0 ]
[  1   1   1   0   0  1/0  1/0  1/0 ]
```

(49) 3.1.7

```
[  0   1   1   1   1  1/0  1/0  1/0 ]
[  1   0   1   1   1  1/0  1/0  1/0 ]
[  1   1   0   1   1  1/0  1/0  1/0 ]
[  1   1   1   0   1  1/0  1/0  1/0 ]
[  1   1   1   1   0  1/0  1/0  1/0 ]
```

(50) 3.1.8

```
[  1   1   1   1   1  1/0  1/0  1/0 ]
```

Appendix D

Learning Sessions

D.1

```
| ?- sp.
The initial setting is [0  0  0  0  0  0  0  0 ]          %a
Next? [s,iv].                                        %1
Current setting remains unchanged.
Next? [s,tv,o].                                      %2
Current setting remains unchanged.
Next? [s,o,tv].                                      %3
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  0 ]          %b
Next? generate.
Language generated with current setting:
[[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                      %4
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  0 ]          %c
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                      %5
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1  0 ]          %d
Next? generate.
Language generated with current setting:
[[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                      %6
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1  0  0 ]          %e
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [o,s,tv].                                      %7
Unable to parse [o,s,tv]
Resetting the parameters ...
```

```
Parameters reset to: [0  0  0  0  0  0  1  1  ]                %f
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,tv]]
Next? [s,tv,o].                                          %8
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  0  1  ]                %g
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                          %9
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  1  ]                %h
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                          %10
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  1  ]                %i
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [o,s,tv].                                          %11
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1  1  ]                %j
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                          %12
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1  0  1  ]                %k
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                          %13
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  0  ]              %l
Next? generate.
Language generated with current setting:
[[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                          %14
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  0  ]              %m
Next? generate.
Language generated with current setting:
[[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? [o,s,tv].                                          %15
```

```
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1/0  1  ]              %n
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,tv]]
Next? [s,tv,o].                                               %16
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  0  1/0  ]              %o
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                               %17
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  1  ]              %p
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                               %18
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  1  ]              %q
Next? [s,iv].                                                 %19
Current setting remains unchanged.
Next? [s,often,iv].                                           %20
Current setting remains unchanged.
Next? [s,tv,o].                                               %21
Current setting remains unchanged.
Next? [o,s,tv].                                               %22
Current setting remains unchanged.
Next? [s,o,tv].                                               %23
Current setting remains unchanged.
Next? [s,often,tv,o].                                         %24
Current setting remains unchanged.
Next? [o,s,often,tv].                                         %25
Current setting remains unchanged.
Next? [s,often,o,tv].                                         %26
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? bye.

yes
| ?-
```

D.2

```
| ?- sp.
The initial setting is [0  0  0  0  0  0  0  0  ]          %a
Next? [s,o,tv].                                            %1
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  0  ]              %b
```

```
Next? [o,s,tv].                                          %2
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1  1  ]                  %c
Next? [s,o,tv].                                          %3
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  1  ]                  %d
Next? [s,tv,o].                                          %4
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  1  ]                  %e
Next? [s,o,tv].                                          %5
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1  1  ]                  %f
Next? [s,tv,o].                                          %6
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1  0  1  ]                  %g
Next? [o,s,tv].                                          %7
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1  1  1  ]                  %h
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                          %8
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  0  ]                %i
Next? generate.
Language generated with current setting:
[[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                          %9
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  0  ]                %j
Next? [o,s,tv].                                          %10
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1/0  1  ]                %k
Next? [s,o,tv].                                          %11
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  1  ]                %l
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
Next? [s,tv,o].                                          %12
Unable to parse [s,tv,o]
Resetting the parameters ...
```

```
Parameters reset to: [0  0  0  0  0  1  1/0  1  ]                    %m
Next? [s,iv].                                                        %13
Current setting remains unchanged.
Next? [s,often,iv].                                                  %14
Current setting remains unchanged.
Next? [s,tv,o].                                                      %15
Current setting remains unchanged.
Next? [o,s,tv].                                                      %16
Current setting remains unchanged.
Next? [s,o,tv].                                                      %17
Current setting remains unchanged.
Next? [s,often,tv,o].                                                %18
Current setting remains unchanged.
Next? [o,s,often,tv].                                                %19
Current setting remains unchanged.
Next? [s,often,o,tv].                                                %20
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? bye.

yes
| ?-
```

D.3

```
| ?- learn_all_langs.
Trying to learn [[s,iv],[s,tv,o]] ...
Final setting: 0  0  0  0  0  0  0  0
Language generated: [[s,iv],[s,tv,o]]
The language [[s,iv],[s,tv,o]] is learnable.

Trying to learn [[s,iv],[s,o,tv],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1  1/0  0
Language generated: [[s,iv],[s,o,tv],[s,tv,o]]
The language [[s,iv],[s,o,tv],[s,tv,o]] is learnable.

Trying to learn [[s,iv],[s,o,tv]] ...
Final setting: 0  0  0  0  0  1  1  0
Language generated: [[s,iv],[s,o,tv]]
The language [[s,iv],[s,o,tv]] is learnable.

Trying to learn [[o,tv,s],[s,iv],[s,tv,o]] ...
Final setting: 1  1  1  1  1  1  1  1
Language generated: [[o,tv,s],[s,iv],[s,tv,o]]
The language [[o,tv,s],[s,iv],[s,tv,o]] is learnable.

Trying to learn [[o,tv,s]] ...
Final setting: 1  0  0  0  0  0  1  0
Language generated: [[iv,s],[o,tv,s]]
 which is a superset of [[o,tv,s]]
The language [[o,tv,s]] is NOT learnable.


Trying to learn [[o,s,tv],[s,iv],[s,tv,o]] ...
Final setting: 1  1  0  0  0  1  1  1
Language generated: [[o,s,tv],[s,iv],[s,tv,o]]
The language [[o,s,tv],[s,iv],[s,tv,o]] is learnable.
```

```
Trying to learn [[o,s,tv],[s,iv],[s,o,tv],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1  1/0  1
Language generated: [[o,s,tv],[s,iv],[s,o,tv],[s,tv,o]]
The language [[o,s,tv],[s,iv],[s,o,tv],[s,tv,o]] is learnable.

Trying to learn [[o,s,tv],[s,iv],[s,o,tv]] ...
Final setting: 0  0  0  0  0  1  1  1
Language generated: [[o,s,tv],[s,iv],[s,o,tv]]
The language [[o,s,tv],[s,iv],[s,o,tv]] is learnable.

Trying to learn [[o,s,tv],[s,iv]] ...
Final setting: 0  0  0  0  0  0  1  0
Language generated: [[o,s,tv],[s,iv]]
The language [[o,s,tv],[s,iv]] is learnable.

Trying to learn [[o,s,tv],[o,tv,s],[s,iv],[s,tv,o]] ...
Final setting: 1  1  0  0  0  1/0  1  1
Language generated: [[o,s,tv],[o,tv,s],[s,iv],[s,tv,o]]
The language [[o,s,tv],[o,tv,s],[s,iv],[s,tv,o]] is learnable.

Trying to learn [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  1
Language generated: [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o]]
The language [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o]] is learnable.

Trying to learn [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  1
Language generated: [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]]
The language [[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]] is learnable.

Trying to learn [[o,s,tv]] ...
Final setting: 0  0  0  0  0  0  1  0
Language generated: [[o,s,tv],[s,iv]]
 which is a superset of [[o,s,tv]]
The language [[o,s,tv]] is NOT learnable.


Trying to learn [[iv,s],[tv,s,o]] ...
Final setting: 1  0  0  0  0  0  0  0
Language generated: [[iv,s],[tv,s,o]]
The language [[iv,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  0  1/0  0
Language generated: [[iv,s],[tv,o,s],[tv,s,o]]
The language [[iv,s],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[tv,o,s]] ...
Final setting: 1  1  0  0  0  0  1  0
Language generated: [[iv,s],[tv,o,s]]
The language [[iv,s],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[s,iv],[s,tv,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  0  0
Language generated: [[iv,s],[s,iv],[s,tv,o],[tv,s,o]]
The language [[iv,s],[s,iv],[s,tv,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  1/0  1/0  0
Language generated: [[iv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] is learnable.
```

```
Trying to learn [[iv,s],[s,iv],[s,tv,o],[tv,o,s]] ...
Final setting: 1  1  0  0  0  1/0  1  0
Language generated: [[iv,s],[s,iv],[s,tv,o],[tv,o,s]]
The language [[iv,s],[s,iv],[s,tv,o],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[tv,s,o]] ...
Final setting: 1  0  0  0  0  0  1/0  0
Language generated: [[iv,s],[o,tv,s],[tv,s,o]]
The language [[iv,s],[o,tv,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  0  1/0  1/0
Language generated: [[iv,s],[o,tv,s],[tv,o,s],[tv,s,o]]
The language [[iv,s],[o,tv,s],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[tv,o,s]] ...
Final setting: 1  1  0  0  0  0  1  1/0
Language generated: [[iv,s],[o,tv,s],[tv,o,s]]
The language [[iv,s],[o,tv,s],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,s,o]] ...
Final setting: 1  1  1  1  1  1  1  1/0
Language generated: [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,s,o]]
The language [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  1  1  1  1/0  1  1/0
Language generated: [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  0
Language generated: [[iv,s],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]]
The language [[iv,s],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[s,iv],[s,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  0
Language generated: [[iv,s],[o,tv,s],[s,iv],[s,o,tv]]
The language [[iv,s],[o,tv,s],[s,iv],[s,o,tv]] is learnable.

Trying to learn [[iv,s],[o,tv,s]] ...
Final setting: 1  0  0  0  0  0  1  0
Language generated: [[iv,s],[o,tv,s]]
The language [[iv,s],[o,tv,s]] is learnable.

Trying to learn [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  1/0  1/0  1/0
Language generated: [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s]] ...
Final setting: 1  1  0  0  0  1/0  1  1/0
Language generated: [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s]]
The language [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,tv,o],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  1/0
Language generated: [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]]
The language [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,tv,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  1/0
```

```
Language generated: [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]]
The language [[iv,s],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv]] is learnable.

yes
|- ?
```

D.4

```
| ?- learn_all_langs.
Trying to learn [[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1  0  0
Language generated: [[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
The language [[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1  1/0  0
Language generated: [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
The language [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 0  0  0  0  0  1  1  0
Language generated: [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] is learnable.

Trying to learn [[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] ...
Final setting: 1  1  1  1  0  1  0  0
Language generated: [[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]]
The language [[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] is learnable.

Trying to learn [[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]] ...
Final setting: 0  0  0  0  0  0  0  0
Language generated: [[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]]
The language [[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]] is learnable.

Trying to learn [[often,s,iv],[often,s,tv,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1/0  0  0
Language generated: [[often,s,iv],[often,s,tv,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
The language [[often,s,iv],[often,s,tv,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] ...
Final setting: 1  1  1  1  1  1  1  1
Language generated: [[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]]
The language [[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] is learnable.

Trying to learn [[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] ...
Final setting: 1  1  1  1  1  1/0  1  1
Language generated: [[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]]
The language [[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]]
is learnable.

Trying to learn [[o,tv,often,s],[o,tv,s]] ...
Final setting: 1  1  1  1  0  0  1  1
Language generated: [[o,tv,often,s],[o,tv,s]]
The language [[o,tv,often,s],[o,tv,s]] is learnable.

Trying to learn [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 0  0  0  0  0  1/0  1  0
Language generated: [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
is learnable.
```

```
Trying to learn [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]] ...
Final setting: 0  0  0  0  0  0  1  0
Language generated: [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]]
The language [[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]] is learnable.

Trying to learn [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]] ...
Final setting: 0  0  0  0  0  0  1/0  0
Language generated: [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]]
The language [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]] is learnable.

Trying to learn [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1/0  1/0  0
Language generated: [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
The language [[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[o,s,tv],[o,s,tv,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] ...
Final setting: 1  1  1  1  0  1  1  1
Language generated: [[o,s,tv],[o,s,tv,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]]
The language [[o,s,tv],[o,s,tv,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o]] is learnable.

Trying to learn [[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],[s,iv,often],[s,tv,o],
[s,tv,often,o]] ...
Final setting: 1  1  1  1  0  1/0  1  1
Language generated: [[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],[s,iv,often],[s,tv,o],
[s,tv,often,o]]
The language [[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],[s,iv,often],[s,tv,o],
[s,tv,often,o]] is learnable.

Trying to learn [[o,s,often,tv],[o,s,tv],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 1  1  0  0  0  1  1  1
Language generated: [[o,s,often,tv],[o,s,tv],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
The language [[o,s,often,tv],[o,s,tv],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],
[s,tv,o]] ...
Final setting: 0  0  0  0  0  1  1/0  1
Language generated: [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]]
The language [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],
[s,tv,o]] is learnable.

Trying to learn [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 0  0  0  0  0  1  1  1
Language generated: [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] is learnable.

Trying to learn [[o,often,tv,s],[o,tv,s]] ...
Final setting: 1  0  0  0  0  0  1  1
Language generated: [[o,often,tv,s],[o,tv,s]]
The language [[o,often,tv,s],[o,tv,s]] is learnable.

Trying to learn [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o]] ...
Final setting: 1  1  0  0  0  1/0  1  1
Language generated: [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,often,iv],
[s,often,tv,o],[s,tv,o]]
The language [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o]] is learnable.
```

```
Trying to learn [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  1
Language generated: [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
The language [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  1
Language generated: [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv]]
The language [[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv]] is learnable.

Trying to learn [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]] ...
Final setting: 0  0  0  0  0  0  1  1/0
Language generated: [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]]
The language [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv]] is learnable.

Trying to learn [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],
[s,tv,o]] ...
Final setting: 0  0  0  0  0  0  1/0  1/0
Language generated: [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],
[s,tv,o]]
The language [[o,often,s,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,tv,o]]
is learnable.

Trying to learn [[o,often,s,tv],[o,s,tv]] ...
Final setting: 0  0  0  0  0  0  1  1
Language generated: [[o,often,s,tv],[o,s,tv]]
The language [[o,often,s,tv],[o,s,tv]] is learnable.

Trying to learn [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1/0  1/0  1
Language generated: [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]]
The language [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 0  0  0  0  0  1/0  1  1
Language generated: [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
is learnable.

Trying to learn [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],
[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 0  0  0  0  0  1/0  1  1/0
Language generated: [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],
[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[s,iv],[s,o,tv],
[s,often,iv],[s,often,o,tv]] is learnable.

Trying to learn [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]] ...
Final setting: 0  0  0  0  0  1/0  1/0  1/0
Language generated: [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
```

```
The language [[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,s,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  0  0  0
Language generated: [[iv,s],[often,iv,s],[often,tv,s,o],[tv,s,o]]
The language [[iv,s],[often,iv,s],[often,tv,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],
[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  0  0
Language generated: [[iv,s],[often,iv,s],[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o],[tv,s,o]]
The language [[iv,s],[often,iv,s],[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],
[tv,s,o]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]] ...
Final setting: 1  1  0  0  0  0  1  0
Language generated: [[iv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]]
The language [[iv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,o,s],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],
[tv,o,s]] ...
Final setting: 1  1  0  0  0  1/0  1  0
Language generated: [[iv,s],[often,iv,s],[often,tv,o,s],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o],[tv,o,s]]
The language [[iv,s],[often,iv,s],[often,tv,o,s],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],
[tv,o,s]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  0  1/0  0
Language generated: [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  1/0  1/0  0
Language generated: [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[s,iv],[s,often,iv],
[s,often,tv,o],[s,tv,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],
[s,tv,o],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  0
Language generated: [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[s,iv],[s,o,tv],[s,often,iv],
[s,often,o,tv]]
The language [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
is learnable.

Trying to learn [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  0  1/0  0
Language generated: [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]]
The language [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[s,iv],[s,o,tv],
[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  0
Language generated: [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[s,iv],[s,o,tv],
[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]]
The language [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[s,iv],[s,o,tv],
[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]] is learnable.
```

```
Trying to learn [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]] ...
Final setting: 1  0  0  0  Q  0  1  0
Language generated: [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]]
The language [[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]] ...
Final setting: 1  1  0  0  0  0  1  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]]
The language [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],
[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  0  1/0  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],
[tv,o,s],[tv,s,o]]
The language [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[often,tv,s,o],[tv,o,s],
[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  0  1/0  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]]
The language [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[tv,s,o]]
is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]] ...
Final setting: 1  0  0  0  0  0  1  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]]
The language [[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],
[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s]] ...
Final setting: 1  1  0  0  0  1/0  1  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],
[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s]]
The language [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],[s,iv],
[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],
[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s],[tv,s,o]] ...
Final setting: 1  1  0  0  0  1/0  1/0  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],
[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s],[tv,s,o]]
The language [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,tv,o,s],
[often,tv,s,o],[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o],[tv,o,s],[tv,s,o]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] ...
Final setting: 1  0  0  0  0  1/0  1  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]]
The language [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv]] is learnable.

Trying to learn [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[often,tv,s,o],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]] ...
Final setting: 1  0  0  0  0  1/0  1/0  1/0
Language generated: [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[often,tv,s,o],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]]
The language [[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],
[often,tv,s,o],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]] is learnable.
```

```
Trying to learn [[iv,s],[iv,s,often],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1  0  0
Language generated: [[iv,s],[iv,s,often],[tv,s,o],[tv,s,often,o]]
The language [[iv,s],[iv,s,often],[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,s,o],
[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1  0  1/0
Language generated: [[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,s,o],
[tv,s,often,o]]
The language [[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,s,o],[tv,s,often,o]]
is learnable.

Trying to learn [[iv,s],[iv,s,often],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],
[s,tv,often,o],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1  1  1/0
Language generated: [[iv,s],[iv,s,often],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],
[s,tv,often,o],[tv,s,o],[tv,s,often,o]]
The language [[iv,s],[iv,s,often],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],
[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[tv,often,s,o],[tv,s,o]] ...
Final setting: 1  1  1  1  0  0  0  0
Language generated: [[iv,often,s],[iv,s],[tv,often,s,o],[tv,s,o]]
The language [[iv,often,s],[iv,s],[tv,often,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]] ...
Final setting: 1  1  1  1  0  0  1/0  0
Language generated: [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]]
The language [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s]] ...
Final setting: 1  1  1  1  0  0  1  0
Language generated: [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s]]
The language [[iv,often,s],[iv,s],[tv,o,s],[tv,often,o,s]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,often,s,o],
[tv,s,o]] ...
Final setting: 1  1  1  1  0  1/0  0  0
Language generated: [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,often,s,o],
[tv,s,o]]
The language [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,often,s,o],
[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],
[tv,often,o,s],[tv,often,s,o],[tv,s,o]] ...
Final setting: 1  1  1  1  0  1/0  1/0  0
Language generated: [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],
[tv,often,o,s],[tv,often,s,o],[tv,s,o]]
The language [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],
[tv,often,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],
[tv,often,o,s]] ...
Final setting: 1  1  1  1  0  1/0  1  0
Language generated: [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],
[tv,often,o,s]]
The language [[iv,often,s],[iv,s],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s]]
is learnable.

Trying to learn [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s],[tv,often,s,o],
[tv,s,o]] ...
```

```
Final setting: 1  1  1  1  0  0  1/0  1/0
Language generated: [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s],
[tv,often,s,o],[tv,s,o]]
The language [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s],[tv,often,s,o],
[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s]] ...
Final setting: 1  1  1  1  0  0  1  1/0
Language generated: [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s]]
The language [[iv,often,s],[iv,s],[o,tv,often,s],[o,tv,s],[tv,o,s],[tv,often,o,s]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]] ...
Final setting: 1  1  1  1  0  1/0  1/0  1/0
Language generated: [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]]
The language [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],[s,iv,often],
[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s]] ...
Final setting: 1  1  1  1  0  1/0  1  1/0
Language generated: [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s]]
The language [[iv,often,s],[iv,s],[o,s,tv],[o,s,tv,often],[o,tv,often,s],[o,tv,s],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[tv,often,s,o],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  0  0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[tv,often,s,o],[tv,s,o],[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[tv,often,s,o],[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  1  0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],
[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  1/0  0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],
[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],
[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],
[tv,often,s,o],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  0  1/0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],
[tv,often,s,o],[tv,s,o],[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],
[tv,often,s,o],[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  1  1/0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],
[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,s,o],[tv,s,often,o]] is learnable.

Trying to learn [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],
```

```
[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],[tv,s,often,o]] ...
Final setting: 1  1  1  1  1  1/0  1/0  1/0
Language generated: [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],
[s,iv],[s,iv,often],[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],[tv,s,often,o]]
The language [[iv,often,s],[iv,s],[iv,s,often],[o,tv,often,s],[o,tv,s],[o,tv,s,often],[s,iv],[s,iv,often],
[s,tv,o],[s,tv,often,o],[tv,o,s],[tv,often,o,s],[tv,often,s,o],[tv,s,o],[tv,s,often,o]] is learnable.

yes
|- ?
```

D.5

```
| ?- sp.
The initial setting is [0  0  0  0  0  0  0  0  ]              %a
Next? [s,o,tv].                                        %1
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  0  ]              %b
Next? [iv,s].                                          %2
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  0  1  0  ]              %c
Next? [s,iv].                                          %3
Unable to parse [s,iv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  0  ]              %d
Next? [o,tv,s].                                        %4
Unable to parse [o,tv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  0  1  1  ]              %e
Next? [o,s,tv].                                        %5
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1  1  ]              %f
Next? [tv,s,o].                                        %6
Unable to parse [tv,s,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  0  0  ]            %g
Next? [s,tv,o].                                        %7
Current setting remains unchanged.
Next? [s,o,tv].                                        %8
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  1  0  ]            %h
Next? [s,tv,o].                                        %9
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1/0  0  ]            %i
Next? [o,s,tv].                                        %10
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1/0  1  ]            %j
```

```
Next? [s,o,tv].                                          %11
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  1  ]                %k
Next? [s,tv,o].                                          %12
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  1  ]                %l
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                            %13
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  0  1  1/0  ]                %m
Next? generate.
Language generated with current setting:
[[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,o,tv,s]]
Next? [s,tv,o].                                          %14
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  1/0  ]                %n
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                          %15
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  1  1  ]                %o
Next? [o,s,tv].                                          %16
Current setting remains unchanged.
Next? [s,tv,o].                                          %17
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1/0  1  ]                %p
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                            %18
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  0  1  1/0  ]                %q
Next? generate.
Language generated with current setting:
[[iv,s],[o,often,tv,s],[o,tv,s],[often,iv,s],[often,tv,o,s],[tv,o,s]]
Next? [s,tv,o].                                          %19
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1  0  1/0  ]                %r
Next? generate.
Language generated with current setting:
[[s,iv],[s,often,iv],[s,often,tv,o],[s,tv,o]]
Next? [s,o,tv].                                          %20
Unable to parse [s,o,tv]
```

```
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1/0  0  ]            %s
Next? generate.
Language generated with current setting:
[[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                          %21
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  1/0  0  ]            %t
Next? generate.
Language generated with current setting:
[[iv,s],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o],[tv,s,o]]
Next? [s,tv,o].                                        %22
Current setting remains unchanged.
Next? [s,o,tv].                                        %23
Current setting remains unchanged.
Next? [o,s,tv].                                        %24
Unable to parse [o,s,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1/0  1/0  ]            %u
Next? [s,o,tv].                                        %25
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1/0  1  ]            %v
Next? [s,tv,o].                                        %26
Current setting remains unchanged.
Next? [o,s,tv].                                        %27
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],
[s,tv,o]]
Next? [iv,s].                                          %28
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  0  1/0  ]            %w
Next? [s,tv,o].                                        %29
Current setting remains unchanged.
Next? [s,o,tv].                                        %30
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  1/0  1  ]            %x
Next? [s,tv,o].                                        %31
Current setting remains unchanged.
Next? [o,s,tv].                                        %32
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],
[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                          %33
Unable to parse [iv,s]
Resetting the parameters ...
```

```
Parameters reset to: [1  0  0  0  0  1/0  1  1/0  ]          %y
Next? [s,tv,o].                                        %34
Unable to parse [s,tv,o]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  1/0  1/0  ]          %z
Next? [s,o,tv].                                        %35
Current setting remains unchanged.
Next? [o,s,tv].                                        %36
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,s,often,tv],[o,s,tv],[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                          %37
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1/0  0  1/0  ]          %a1
Next? [s,tv,o].                                        %38
Current setting remains unchanged.
Next? [s,o,tv].                                        %39
Unable to parse [s,o,tv]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1/0  1/0  ]          %b1
Next? [s,tv,o].                                        %40
Current setting remains unchanged.
Next? [o,s,tv].                                        %41
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,often,s,tv],[o,s,often,tv],[o,s,tv],[often,o,s,tv],[often,s,iv],[often,s,tv,o],[s,iv],[s,o,tv],
[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o]]
Next? [iv,s].                                          %42
Unable to parse [iv,s]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1/0  1/0  1/0  ]          %c1
Next? [s,tv,o].                                        %43
Current setting remains unchanged.
Next? [s,o,tv].                                        %44
Current setting remains unchanged.
Next? [o,s,tv].                                        %45
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[iv,s],[o,often,tv,s],[o,s,often,tv],[o,s,tv],[o,tv,s],[often,iv,s],[often,o,tv,s],[often,tv,s,o],
[s,iv],[s,o,tv],[s,often,iv],[s,often,o,tv],[s,often,tv,o],[s,tv,o],[tv,s,o]]
Next? [o,tv,s].                                        %46
Current setting remains unchanged.
Next? [tv,o,s].
Unable to parse [tv,o,s]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  1/0  1/0  1/0  ]          %d1
Next? [s,tv,o].                                        %47
Current setting remains unchanged.
Next? [s,o,tv].                                        %48
Unable to parse [s,o,tv]
Resetting the parameters ...
```

```
no
| ?-
```

D.6

```
| ?- sp.
The initial setting is [0  0  0  0  0  0  0  0  i  i  1-0 ]
Next? [s,iv,aux].
Unable to parse [s,iv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  0  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Current setting remains unchanged.
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  0  0  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  0  0  0  i  f  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  0  0  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  0  0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  1  0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  1  1  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1  0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1  0  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
```

```
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  0  0  f  i  1-0 ]
Next? [s,tv,o,aux].
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[s,iv,aux],[s,tv,o,aux]]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  0  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  0  1  i  f  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  1  0  1  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  1  0  1  i  f  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  1  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  1  0  1  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  1  1  1  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1  0  1  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...
```

```
Parameters reset to: [1  0  0  1  1  1  0  1  f  f  1-0  ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  0  1  f  i  1-0  ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  0  1  f  f  1-0  ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1  1  f  i  1-0  ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1  1  f  f  1-0  ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  1/0  0  i  f  1-0  ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  0  0  0  1/0  0  i  f  1-0  ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  0  1/0  0  i  f  1-0  ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  0  i  f  1-0  ]
Next? [s,tv,o,aux].
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[s,iv,aux],[s,o,tv,aux],[s,tv,o,aux]]
Next? [o,s,tv,aux].
Unable to parse [o,s,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1/0  1  0  i  f  1-0  ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  0  1/0  0  f  i  1-0  ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  1  1/0  0  i  f  1-0  ]
Next? [s,tv,o,aux].
```

```
Current setting remains unchanged.
Next? [o,s,tv,aux].
Unable to parse [o,s,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  1/0  1  0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1/0  0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1/0  0  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  1  1/0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Current setting remains unchanged.
Next? [o,s,tv,aux].
Unable to parse [o,s,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  1/0  1  0  f  i  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1/0  0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  i  0  1  1  1/0  0  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1  1/0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Current setting remains unchanged.
Next? [o,s,tv,aux].
Unable to parse [o,s,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  0  0  1  1  1/0  1  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1/0  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1  1/0  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
```

```
Resetting the parameters ...

Parameters reset to: [1  1  0  1  1  1/0  1  0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [i  1  0  1  1  1/0  1  0  f  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  0  0  1/0  i  f  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [i  0  0  0  0  0  0  1/0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  0  0  0  1/0  i  f  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [1  1  0  0  0  0  0  1/0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  1  1  0  0  1/0  f  i  1-0 ]
Next? [s,o,tv,aux].
Unable to parse [s,o,tv,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1  1/0  i  f  1-0 ]
Next? [s,tv,o,aux].
Unable to parse [s,tv,o,aux]
Resetting the parameters ...

Parameters reset to: [0  0  0  0  0  1  1/0  1  i  f  1-0 ]
Next? [s,o,tv,aux].
Current setting remains unchanged.
Next? [o,s,tv,aux].
Current setting remains unchanged.
Next? generate.
Language generated with current setting:
[[o,s,tv,aux],[s,iv,aux],[s,o,tv,aux],[s,tv,o,aux]]
Next? bye.

yes
| ?-
```