2. PROLOG does not properly implement SLD-resolution. Standard PROLOG systems have two major defects. (1) The full unification algorithm (with the occurs check) is not usually implemented. The consequence is that soundness is not preserved. (2) PROLOG uses an unfair computation rule, namely depth-first search. The consequence is that completeness is not preserved.

3. PROLOG does not properly implement NAF (Negation as Failure). PROLOG allows Horn clauses to be extended with negated goals in the body of a clause, e.g. $L_1 :- \backslash+L_2, \ldots, L_m$ where '$\backslash+L_2$' represents a negated goal. It is well-known that SLDNF-resolution (SLD-resolution with NAF), Lloyd [36], is a sound and complete computation rule. However, standard PROLOG systems only implement an unsound version of NAF.

The reason why standard PROLOG implementations have so many compromises is that it is much more efficient *in practice* to omit the occurs check, to use depth-first search (rather than, say, breadth-first search), and to implement a non-safe negation rule.[4] However, this efficiency comes at the cost of the grammar writer having to worry about procedural details to avoid getting into trouble. For example, the grammar writer can avoid the incompleteness of depth-first search and the unsoundness of PROLOG negation in many cases by re-ordering goals. However, it is unreasonable to expect potential grammar writers to deal with such defects.

On the other hand, better PROLOG implementations, in the sense of having fewer compromises with respect to soundness and completeness, whilst not yet in widespread use, do exist. For example, there are systems, e.g. Naish [40], that automatically delay negated goals until all unbound variables have been instantiated (an example of a sound implementation of NAF). The current principle definitions can be interpreted directly by such systems without any changes. Hence, we conclude that many of the present deficiencies in PROLOG are not fundamental, but simple artifacts of current implementations that will probably be remedied in the near future.

However, the restriction of Horn clause logic to a single positive literal is potentially a more serious limitation. The natural question to ask is: Are there are any instances of principles where two (or more) positive literals are required? Equivalently, do we ever need to express something of the form, say, $p \vee q \leftarrow r$? Stabler [52] suggests one example of such a situation:

(1) moveA(Tree0,Tree) $\leftrightarrow$ (substitute(Tree0,Tree) §*6(7)*
$\vee$ adjoin(Tree0,Tree)
)

[4] We have ignored the so-called 'extra-logical' features such as the 'cut' that can also cause problems for completeness.

The $\rightarrow$ direction of rule (1) contains two positive literals. Note that the following two PROLOG clauses are *not* equivalent to the $\rightarrow$ form:

(2)
```
substitute(Tree0,Tree) :- moveA(Tree0,Tree).
adjoin(Tree0,Tree) :- moveA(Tree0,Tree).
```

The reason, of course, is that the two clauses in (2) constitute an *exclusive-or* disjunction, whereas the 1st order logic statement allows the possibility that both disjuncts might be simultaneously satisfied, i.e. an *inclusive-or*. Hence, at first glance, it would seem that the PROLOG representation is inadequate. However, it turns out that (1) is an incorrect formalization of the English definition in *Barriers* Chomsky [11]. From pg.4, we have:

(3) "Assume that there are two types of movement: substitution and adjunction."

In (3), the *intended* interpretation is that substitution and adjunction are *distinct* operations. Hence the PROLOG definition *does* capture the intent of Chomsky's definition. Moreover, an informal survey of the 108 1st order sentences in Stabler's axiomatization of *Barriers* suggests that the intended interpretation of all such disjunctive statements is the exclusive-or one. Finally, there also seems to be no need for inclusive-or disjunctions in the definitions employed in the current system. To conclude, there seems to be no need to go to a more powerful representation (and pay the associated performance penalty) than Horn clause logic extended with NAF.[5]

2.3 Free Overgeneration, Separation of Principles and Efficiency

We have chosen to use as straightforward and direct an implementation of principle definitions as possible. In particular, each principle has been encoded separately. We have deliberately chosen not to attempt to combine principles or allow one principle to, partially or otherwise, abrogate another purely for the sake of efficiency. Depending on the linguistic theory, this can result in a very inefficient parser. For example, in the type of theory assumed by Lasnik & Uriagereka, the basic model is

[5] Stabler also suggests other theoretical reasons for using FOL. He observes (§2.9) that there may be sentences *which are not in the completion of any Horn theory.* He also points out the possibility that selective SLDNF computation rule may abort *so much as to make the theorem proving system worthless.* Finally, he also notes that the NAF rule can fail to terminate. However, despite these theoretical limitations, it is not clear that a non-contrived, linguistically-motivated example that illustrates these deficiencies can be found.

one that allow certain operations such as movement (Move-α) to operate as freely as possible; for example, to allow any syntactic element to move anywhere. (See also section 1.2.) The essential assumption here is that principles will interact so as to eliminate all but the well-formed instances of movement. Even standard assumptions about movement such as "traces are obligatorily left by movement" and "a trace and an antecedent are obligatorily coindexed" are considered to be stipulations best reduced from being obligatory to being merely optional, with other principles forcing the stipulation to hold only when necessary. In this way, the theory becomes a stronger one with fewer redundancies because (optional) stipulations will have independent justification.

Let us briefly consider the *LF movement* operation, an *optional* process (motivated by semantic interpretation considerations) that moves quantifiers and *wh*-elements to positions of sentential scope at LF. For example, *everyone* and *nani* ('what') both undergo movement at LF in (4a) and (4c) (= (11) in chapter 1), respectively:

(4) a. John saw everyone
b. everyone$_i$ [John saw x_i] (LF)
c. watashi-wa Taro-ga nani-o katta ka shitte iru
(= I know what Taro bought)
d. watashi-wa nani$_i$ [Taro-ga x_i katta] ka shitte iru (LF)

In general, such elements, unless they appear in sentential scope positions, will be ruled out by principles that operate at LF, such as the *wh*-Comp requirement (which requires [+*wh*], i.e. questioned, clauses to have *wh*-elements head them) or perhaps operator–variable licensing (operators must Bind some variable). Hence, one temptation when encoding LF movement is to make movement obligatory, say, for all *wh*-elements and quantifiers not already in positions of sentential scope. This would avoid much unnecessary work from ruling out obviously ill-formed LFs. But we take the position that this is wrong, not only because it changes the theory and introduces redundancy, but it is also dangerous in the sense that the formulation of the *wh*-Comp requirement, for example, will not be properly tested. This can lead to insidious problems when the theory is updated. For example, if the *wh*-Comp requirement is incorrectly altered for some reason, the parser will continue to produce the correct parses (for the wrong reason). Admittedly, this is a relatively trivial example.[6] However, as we shall see below, reformulation has led to major (probably unintended) deviations from the theory for at least one existing parser.

[6] Incidentally, in the current system, the theory as implemented is underconstrained. The LF principles fail to force both *wh*-elements in *kimi-wa nani-o doko-de katta no* (example (60) from figure 1.3) to move to Comp. This discrepancy would not have been apparent if LF movement had been stipulated to be obligatory for *wh*-elements not in Comp.

It should be pointed out that we should have no objection to the alteration of principles for efficiency reasons provided that such operations can be carried out automatically and transparently, i.e. without missing well-formed parses, accepting ill-formed sentences, or altering the structures that the user sees in any way. Unfortunately, the current system (or any other existing parser as far as we are aware) cannot automatically deduce theorems such as: *wh*-elements must move to position of sentential scope (if they are not already at such a position). Because of the lack of such a compilation procedure, the *LF movement* operation in the current system faithfully mimics the *free overgeneration* that is a *property* of the linguistic theory described in Lasnik & Uriagereka. However, principle-based theories need not have this freely-generate-and-test characteristic. For example, Chomksy [14] has described a theory that explains important properties of verbal inflection (that we will not cover here), but relies on a 'guiding' principle to prune the space of possibilities for Move-α. The essential difference is that the guiding principle is a 'least effort' condition on derivations. This can considerably simplify the problem for parsing. That is, we no longer have an obligation to fully explore the space of possible derivations, but simply to produce all the derivations of shortest length.[7]

However, many existing parsers have been designed with no such prohibition against combining or reformulating principles. Such systems will avoid the performance penalty due to overgeneration that occurs in the system described here. We will now turn to discuss one example of such a system.

2.4 Problems with Derived Principles

Correa [15] (also [16]) describes an attribute grammar formulation that has several interesting features. One of the features that Correa's parser has in common with the one described in this thesis is that both parsers employ a theory of empty categories (ECs) based on *Functional Determination*. In this theory, empty noun phrases are initially devoid of syntactic features. However, their distribution are restricted depending on whether or not they are classified as anaphors or pronominals (or both or neither). For example, if an EC is classified as an anaphor, then just like an overt anaphor such *himself*, it must be subject to Binding condition A, as discussed in chapter 1. The Binding status of such elements is indicated by the values assigned to two features [±A] (anaphoric) and [±P] (pronominal) — discussed earlier in section 1.2. Let us examine in more detail how these features are assigned.

Normally, these values are determined by the function that the EC plays in syntactic structure. For example, if an EC is bound by a *wh*-element (an operator)

[7] Note that the actual theory is slightly more complicated because not all derivation steps have the same 'cost'.

Trace Theory (non-deterministic)	
↑	
Free Indexation (non-deterministic)	*Case Assignment* (deterministic[8])
↑	↑
Functional Determination	*Structural Determination*

Figure 2.1 Operational dependencies for functional and structural determination.

occupying a clause-initial position, as in *which report*$_1$ *did you file* e_1, then the EC, e_1 in this case, should be determined as a variable ([−A,−P]). The precise details of how the assignments are made will not be important for the purpose of this discussion. We will just note that functional determination is a complex procedure that depends on many factors, including: (1) whether the EC is bound or not; (2) the position that the binder occupies if the EC is bound, and (3) whether the bound EC is a trace formed originally by movement of the binder.

For parsing, this implies that a great deal of information must be recovered before functional determination can be applied. In particular, we should apply the principles that help to identify which ECs are bound to what elements. Consider again the dependency graph that was shown earlier in figure 1.6. Here, each Binding condition, *A*, *B* and *C*, is directly dependent on *Functional Determination*. In turn, *Functional Determination* is dependent on *Free Indexation* (an operation that can establish Binding relations by freely coindexing, i.e. assigning the same indices to, syntactic elements). *Functional Determination* is also dependent on *Trace theory* (an operation that can also establish Binding relations by recovering trace-antecedent relations, i.e. the occurrences of movement). (No other operation can establish Binding relations, so *Functional Determination* does not depend on anything else, apart from *Parse S-structure* to recover phrase structures.)

However, Correa's parser uses a reformulation of functional determination, called *Structural Determination*, that assigns [±A,±P] features to ECs without needing to identify the possible binders first. Instead, these features are assigned by inspecting properties of the EC itself. (The properties used are whether the EC is: (1) in an A-position or not, (2) governed or not, and (3) assigned Case or not.) This reformulation greatly simplifies the computation of Binding features for ECs since there is now no need to recover the trace-antecedent relations or do free indexation first. (The relevant dependencies for both operations are summarized in figure 2.1.)

[8] Not strictly true. See the discussion of optional Case assignment in section 3.3.2. However, Case assignment is relatively deterministic compared to trace theory and free indexation.

Moreover, this also reduces the amount of non-determinism in the system since, in general, there will be many assignments of trace-antecedent and coindexing relations compatible with any particular structure. However, we do not adopt the (more efficient) structural determination approach for the following reasons:

1. *Problems with theory maintenance.*

   Correa is able to bypass having to compute coindexing relations because he has pre-computed whether various assignments of [$\pm$A,$\pm$P] are allowed for certain positions. For example, one of the cases of structural determination is the following rule:

   (5) If an EC occurs in an A-position, and is assigned Case, and is governed; then it *must* have features [$-$A,$-$P].

   Any other assignment of [$\pm$A,$\pm$P] will be ruled out by some filter. To reach this conclusion, information from many other principles such as the Case filter and the the various Binding conditions must have been applied. In other words, (5) must be a logical consequence of the linguistic theory. Now, consider the problem of maintaining (5). What happens when the theory is updated? Because it relies on many assumptions about other principles, how can we be sure (5) still holds? Neither Correa's parser nor the system described in this thesis has the capability to automatically derive this rule. Hence, we would be forced into the unsatisfactory position of having to to re-derive the theorem by hand.

2. *Problems with representability and explanation.*

   Another problem with adopting structural determination is that the system described here has the goal of being able to, or at least have the option of, building structures that violate principles of Case and Binding theory. (Linguistics texts including Lasnik & Uriagereka often make use of such examples). It would be very hard to implement such an option if the consequences of a given principle were embedded (along with those of other several principles) in different operations. The problem of not being able to represent violations of particular principles also translates into a problem for explanation. If a structure is eliminated by structural determination; then how can we tell if it is a Case or Binding theory violation? There is no easy way to distinguish betweeen the two. This would be a serious problem for the system described in this thesis because one of the tests used to inspire confidence in the implementation is that it should report exactly the same violations as predicted by Lasnik & Uriagereka.

3. *The correctness of the reformulation.*

Actually, a more serious question is: How do we know that the reformulation is correct in the first place? Correa does not mention any incompatibilities nor does he provide a formal proof that structural determination produces the same assignments of [±A,±P] as functional determination taken together with whatever other principles needed to derive a rule like (5). However, we can show by means of a simple example that structural determination is inconsistent with, and therefore cannot replace, functional determination. Consider the following *strong crossover* example (example (3:1) in Lasnik & Uriagereka):

(6) *Who$_1$ does he$_1$ think [Mary likes e_1]

(The impossible intepretation is "for which person x, x thinks Mary likes x.") According to Lasnik & Uriagereka, functional determination will assign [+A,+P] to e_1. Being an anaphor, e_1[+A] must be bound in the embedded clause. But since the nearest binder he_1 occurs outside this clause, this is an example of a Binding condition A violation. However, structural determination makes a different prediction. Since e_1 is in an object position, it satisfies all the preconditions of rule (5).[9] Hence, it will be assigned [−A,−P]. Since, e_1[−A] is not an anaphor, condition A will be trivially satisfied, and the sentence will be incorrectly analysed.[10]

There is a similar problem with another derived rule used by Correa. This is the *Chain Formation* algorithm which recovers the trace-antecedent relations produced by movement. We will not go into the details of this rule here except to mention that it subsumes Subjacency and integrates information from a variety of sources including parts of Case theory, Theta theory, and, surprisingly enough, the features assigned by structural determination.[11] However, Is chain formation correct? For example, does it recover the correct antecedent-trace relations as predicted by the principles that have been integrated? Correa mentions that, *inter alia*, it cannot handle parasitic gap sentences such as *which report did you file without reading*. However, the theory (in particular the functional determination analysis) as assumed by

[9] (1) Object positions are A-positions, i.e. positions to which θ-roles may be assigned (e.g. *John likes Mary*). (2) Objects are always governed (in this case, by the verb). (3) Case is assigned under government and *like* is a Case assigner in this context, so e_1 will get Case.

[10] For completeness: under a non-functional determination account, Condition C can be used instead to rule out the strong crossover example: "R-expressions (overt and non-overt [−A,−P] elements) must be A-free (in the domain of the head of its non-trivial chain if one exists)."

[11] The latter dependency is surprising because functional determination as formulated by Chomsky [10] is dependent on trace-antecedent relations being identified first. That is, the dependency in Correa's formulation is the opposite of what we might expect — see the dependency graph in figure 1.6.

> Correa does correctly permit such parasitic gap sentences. Actually, structural determination does make the right prediction in this case for the empty object of *reading* to be analysed as a variable. Hence, the fact that parasitic gap sentences are not handled, must be due to some peculiarity of the chain formation algorithm. We should add that this one case does not imply that structural determination correctly handles other parasitic-gap-type examples. For example, structural determination as well as chain formation must be fixed before the parser can handle an example like **The report was filed without reading.*[12]

To conclude, although it is often tempting to manually compile principles for efficiency, it should be avoided if possible because of the ease of introducing errors and the difficulty of ensuring correctness. Furthermore, we have argued that such reformulations also come with a loss of explanatory power. Perhaps it is not surprising that it is easy to get derived rules wrong; the free and complex interactions between principles means that it is easy to miss cases where incompatibilities can result. In the current system, principles are formulated independently of each other for these very reasons. However, it can be argued that principle interleaving causes a similar loss of explanatory power. Still, this is not fatal since the user always has the option of determining filter violations by re-configuring the parser to operate without interleaving. Note also that reformulation may lead to redundancies in the parser, which will not occur with interleaving. For example, the structural determination and chain formation algorithms must have much of the same force as the Case filter and θ-criterion, two of the principles that they integrate information from. That is, the latter two principles may not actually do much work because many such violations will have already been accounted for by the derived rules. However, it remains a mystery under what circumstances, if any, the original formulation of the two principles will remain independently useful. (Correa does not mention discarding these rules.) By comparison, when a principle is interleaved, it is wholly merged into the phrase structure recovery procedure. There is no possibility of introducing such unnecessary redundancy; that is, there is no need to keep the non-interleaved version around.

In the next section, we will turn our attention from the topic of principle representation to that of control organization.

2.5 Licensing Parsers

Abney [1] (also [2]) describes a parser that builds *licensing structures*. The basic idea here is that the parser builds structure only in accordance with certain licensing

[12] The object of *reading* will be (incorrectly) licensed as [−A,−P] by (5).

relations. Now, licensing relations form a distinguished subset of the set of general linguistic relations. For example, one way for a noun phrase to be properly licensed is to appear as an argument of some verb that will assign it a θ-role. The noun phrase is also free to participate in other (non-licensing) linguistic relations such as Case assignment or Binding; however, irrespective of what other relations are formed, only θ-role assignment will license the presence of the noun phrase in syntactic structure. The licensing of non-arguments by *predication* is another example of such a relation. For example, a relative clause such as *who arrested John* in the noun phrase *the man who arrested John* is usually considered to be an modifier rather than an argument of the noun *man*. Because modifiers are not arguments, they must be licensed in some way other than via θ-role assignment. In some theories (see van Riemsdijk & Williams [46] for example), relative clauses are properly licensed if they happen to be one-place predicates. For example, this can be used to explain the difference between the following two noun phrases:

(7) a. The man [who arrested John]
b. *The man [(that) Bill arrested John]

The two noun phrases will have the following forms at LF:

(8) a. (the man) ($\lambda x, x$ arrested John)
b. *(the man) (Bill arrested John)

That is, the relative clause in (7a) may be interpreted as a simple λ-expression, but the lack of an operator in (7b) prevents it from having the same analysis.[13]

Predication and θ-role-assignment are two of the four fundamental licensing relations that Abney describes.[14] (We will not describe the other two here.) There are two main reasons why we have not adopted a licensing approach:

1. *Representation of ill-formed sentences.*

   We are primarily interested in parsing both well-formed and ill-formed sentences such as those found in Lasnik & Uriagereka. The licensing approach does not allow structures that violate its restrictions to be represented. Examples like (7b) and (8b) do violate Abney's licensing restrictions. There is no reason why we should prevent these examples from being represented, especially since, unlike Abney, we make no claims with regard to psycholinguistic plausibility.

---

[13] Here, it is assumed that the operator *who* has moved to a clause-initial position to get scope, therefore leaving behind a trace in subject position to be interpreted as a bound variable.

[14] This view has some intuitive appeal in the sense that identification of the thematic and predication relations in a sentence, be it ill-formed or well-formed, seems to be a necessary pre-condition if semantic interpretation is to take place.

2. *Licensing is largely subsumed under the principle interleaving model.*

   In any case, irrespective of whether we need to represent licensing violations or not, the licensing model represents a case of unnecessary 'hardwiring' of control strategy. In the current system, the user can construct a 'licensing' parser with predication and θ-role assignment as privileged operations simply by specifying that the two operations, *FI: License Syntactic Adjuncts* (which requires that all adjuncts must satisfy predication) and *Assign Theta-Roles*, should be interleaved. Recall that if an operation is interleaved, it is applied (where relevant) to individual phrases as they are recovered. In particular, *FI: License Syntactic Adjuncts* will be applied as a precondition on any phrase that is about to be attached as a modifier. Thus, [*CP (that) Bill saw John*] can never be attached as a relative clause to a noun phrase such as [*NP the man*].[15]

2.6 Designing for Efficient Parsing

In this section, we will address the issue of selecting methods for efficient parsing. We will survey the standard methods that can be applied to parsing and point out those that have been utilized in the current implementation. Additionally, we will comment on any specific problems that the methods might pose for implementation.

Let us begin by outlining the preconditions for adopting any particular method:

- *Must be completely automatic.*

  For example, it is unreasonable to expect the grammar writer or user to have to guide or otherwise intervene to make the selected method terminate. Furthermore, the method must not expect the grammar writer to have to annotate, say, the description of the principles in any way.

- *Must preserve linguistic judgments.*

  That is, the correct parses or explanations of ill-formed sentences must be maintained.

To summarize, any method chosen must be completely transparent to the grammar writer. However, as in the case of the principle ordering and interleaving models, there may exist simple parameters of application that will be controllable by the user.

[15] Note that Abney also discusses the application of non-licensing principles such as chain formation and Subjacency after recovering a licensing structure. The same strategy can be used in this system by simply not interleaving these operations. However, there are many aspects of Abney's system pertaining to performance such as ambiguity resolution that we cannot mimic so easily.

In the following discussion, we will distinguish between *off-line* and *on-line* methods. For our purposes, off-line methods, i.e. methods that can be applied to improve the efficiency of the grammar independently of the input, will be preferable to those that are on-line, i.e. to be applied during parsing. The obvious reason is that off-line methods extract no penalty during parsing. On the other hand, on-line methods need to be carefully evaluated since any overhead caused by bookkeeping or extra evaluation can, especially unless tightly coded, swamp any benefits derived from the application of the method itself. Having said that, one possible disadvantage of an expensive off-line method is that re-parameterizing the grammar for different languages may take a considerable amount of time. This could also prove to be inefficient if the parser were to be embedded within a system that required rapid re-parameterization, such as in the case of a system that attempts to learn parameter values through iteration.

1. *Use existing grammar compilation techniques.* [Off-line]

   Although one might be tempted to abandon traditional parsing techniques at the same time as abandoning traditional theories of syntax, it would be unconscionable to ignore the large body of well-understood and efficient parsing algorithms for theories such as context-free grammars (CFGs). General CFGs can be parsed using a push down automata (PDA), or any number of tabular methods including the Cocke-Young-Kasami (CKY) and Earley's algorithms. Also, from the domain of programming languages where linear time parsing is obviously vital, tabular methods with different tradeoffs between generative capacity and program size have been developed for the SLR, LL, LALR, and LR subsets of context-free languages. (See Aho & Ullman [7].)

   An obvious candidate for such techniques is the core set of grammar rules based on $\overline{\text{X}}$-theory that was described in section 1.6. Of the basic methods outlined above, the extended LR-based parsing methods, on the basis of the on-line/off-line distinction as observed by Tomita [56], seem to be preferable to other general methods such as Earley's algorithm, as used by Dorr [20]. In the current system, such an extended-LR(1) scheme is used for the S-structure grammar.

2. *Dynamic programming.* [On-line]

   This is a general technique that is useful for problems that can be recursively subdivided into smaller problems of the same type. If this results in many instances of the same sub-problems having to be solved, then it is usually more efficient to solve the sub-problem just once and 'save' the answers for subsequent occasions. This is also sometimes known as (simple) lemma generation. The essential tradeoff is one of the cost of computing each solution and the cost of saving and subsequent matching and retrieval of the solutions.

Tomita's extended-LR(1) *graph-structured stack* and *shared packed-forest* representations are an example of where this could be exploited to good effect in LR-style parsers. However, experiments with straightforward implementations of optimizations of this sort so far have not been promising. We will not go into the details here, save to mention that the cost of maintaining large phrase structure representations seemed to drastically outweigh any benefits.

3. *Partial Evaluation.* [Off-line]

   Partial evaluation, or the off-line partial execution of programs, has been widely discussed in the logic programming literature. There have been some notable successes such as the more-or-less automatic derivation of RETE-like optimizations for production systems (described in Takeuchi & Fujita [54]), and the derivation of a version of the efficient KMP-type (Knuth-Morris-Pratt) string-matching algorithm from a 'naïve' string-matching algorithm (Fujita [26]). (The latter case is especially interesting since it also incorporates off-line constraint propagation techniques.) However, there remain a number of outstanding problems (discussed in Fujita [26]) that need to be addressed before such methods will be generally applicable:

   (a) Simple partial evaluation, or just simply 'unravelling' goals and using substitution propagation, will not generally lead to a more than linear improvement in run-time performance.

   (b) Shortcutting (or 'folding') can produce large, e.g order-of-magnitude, improvements, but seem to rely on fortuitous circumstances to work. For example, the KMP-type shortcuts only seem to hold for certain matching patterns. Moreover, the string matching process has the advantage of a pre-determined partial input (an option not available in parsing), which provides additional information for program specialization.

   (c) Detailed control of the partial evaluation process seems necessary to achieve good results. In particular, annotations are generally used to indicate to the evaluator when it can safely completely execute code, or avoid non-termination. This violates our working assumption that adopted methods be completely automatic and transparent.

   In the area of principle-based parsers, Johnson [31] has shown that partial evaluation (including shortcutting) can produce parsers with different control strategies. However, our own experiments with Johnson's parsers suggests that the observations listed above seem to hold. For example, a wide range of performance including both significant slow-downs and speed-ups over an unevaluated parser are possible. However, it seems difficult to predict exactly which combinations will produce significant speed-ups, or whether the same results will hold for more substantial parsers, as Johnson's system only implements a small fraction of a principle-based theory.

At the moment, as will be described in chapter 4, the current implementation employs only simple partial evaluation, chiefly applied to the LR(1) parser-generation process.

4. *Constraint satisfaction methods.* [On-line]

   Because of the relatively free interaction between principles and the common use of finite domains, e.g. [±A,±P] features, indices (bounded with respect to a structure), constraint satisfaction methods should find wide application in principle-based systems. Standard techniques of propagating constraints offer the possibility of pruning the search space ahead of time to reduce the number of possibilities that have to be examined by standard backtracking schemes such as that employed in our system. The tradeoff with respect to efficiency is the added time that has to be spent periodically checking constraints. However, there are several other drawbacks. Perhaps the most important is that this usually changes the 'natural', or simplest formulation of programs (as observed by van Hentenryck [28]). It is unreasonable to expect a grammar writer to explicitly program in, say, forward checking, into principle definitions. Assuming that the system could be configured to recognize and transform such problems into one that uses constraints (a capability that the present system does not have), the lack of general (low-level) programming language support for even simple forms of constraints such as simple inequality in standard PROLOG makes explicit checking expensive. For instance, a built-in facility for inequality constraints would be useful for implementing the *i-within-i* condition used in the definition of chain formation (described in section 3.4.3). Currently, a non-transparently programmed constraint checker is used.

5. *Theorem proving.* [Off-line]

   We have already seen examples of hand-derived theorems, e.g. Correa's Structural Determination (and Chain Formation) definitions which integrate knowledge from various principles to reduce non-determinism. Another example where theorem proving can be useful employed is the theorem that *PRO* must be ungoverned. In Lasnik & Uriagereka, this is shown to be a consequence of the fact that *PRO* has Binding features [+A,+P] (and therefore must satisfy both Binding condition A and B). The present system does not make direct use of the *PRO* theorem, but effectively re-derives instances of the theorem each time it applies the two Binding conditions to *PRO*. Since the current system is oriented towards parsing rather than doing general theorem proving, it does not have the capability to generate such theorems.[16]

2.7 Other Issues: Directness and Faithfulness

The definitions of commonsense notions such as directness and faithfulness are somewhat unclear in the context of parser implementation. We will briefly comment on what they might mean in the framework of the implementation described in this thesis.

As discussed earlier, we have adopted a practical definition of correctness. That is, a parser must return the same parses for well-formed sentences and produce the same explanations in terms of the underlying principles as the reference, in our case Lasnik & Uriagereka. This is the minimum requirement that we can expect any implementation to satisfy. Let us now consider the other issues:

- *Directness.*

  As we have seen in the previous sections, the level of representation 'actually used to parse' is largely an illusionary one due to the possibility of having many layers of abstraction with automatic and transparent procedures to compile or interpret each succeeding level. As long as correctness with respect to the original specification, i.e. the high-level definition of grammar, is preserved at each stage, the grammar writer need only be concerned with the topmost level. That is, it only makes sense to 'ground' terms such as directness or faithfulness with respect to the level of representation seen by the grammar writer. Thus, perhaps one reasonable measure of the directness of an implementation is the amount of translation required to encode a given principle. For example, a 1st order logic definition would be more 'direct' than an set of attribute grammar rules (as used by Correa), or LISP or PASCAL code. Ideally, the framework we have adopted, i.e. a strict separation of principles and a representation that minimizes the amount of translation effort required of the grammar writer by using similar combinative and primitive elements, should be considered to be maximally direct. However, the current implementation does fall short of this goal in several respects. As we have mentioned earlier, in some cases, e.g. free indexation, linguistic definitions are not directly constructive. Also, the grammar writer has to 'program around' the deficiencies of standard PROLOG with respect to completeness and soundness.

- *Faithfulness.*

[16] Note that, as mentioned earlier, Stabler does have a 1st logic theorem prover that, presumably, has the capability to prove the *PRO* theorem; although, designing a system that could 'discover' such useful facts is probably another matter entirely.

> The relevant question here is: Given the many layers of abstraction possible, what stronger practical requirements on parsing can be made other than simple correctness? Let us consider one such requirement. In the generate-and-test framework of the current system, we can distinguish between *linguistically-motivated choice points* and all others. The idea being that the parsing problem can be viewed abstractly as a general search problem. In general, at each point in the search space, there many be many choices of particular actions to take. Certainly, if we wish to take ambiguity into account by generating all possible parses, then parsing must be a non-deterministic process. Other theory-internal examples will be discussed in later chapters; let us briefly consider one of these. For example, in Lasnik & Uriagereka Case assignment is assumed to be an optional process in order to be able to account for examples such as:
>
> (9) a. I want John to win
>   b. I want *PRO* to win
>
> (The subject *John* must get Case in (9a), but *PRO* must not get Case in (9b).)
>
> The optionality of Case assignment is represented using two PROLOG clauses, one for each situation, (described in section 3.3.2). Thus during parsing, a (linguistically-motivated) choice point will be created for Case assignment. In general, implementations of a theory must implement all linguistic choice points or else risk getting the wrong analysis for some omitted case. However, all other choice points, call them *computational choice points*, may be eliminated without affecting the linguistic judgements rendered by the parser. Generally, computational choice points may be generated from 'supporting' definitions such as bookkeeping operations and maintenance of internal data structures. To summarize, we might then say that an implementation with only linguistic choice points is more faithful to the theory than one that has additional computational choice points. More specifically, the relative performance, e.g. in terms of parsing time, of the former implementation with respect to different inputs will more faithfully reflect the non-determinism of the theory than the latter implementation.[17]

Note that in the case of a parser that includes principles of performance — provided we regard performance constraints as being part of the original specification of grammar and correctness is maintained — then the same basic observations should hold.[18]

[17] We assume here that the branching factor in a given search space is the dominant factor that affects the time taken to explore that search space.

[18] One interesting question does arise. For example, if the specification requires the use of a

stack representation, then does this mean that a stack must be used at every level of abstraction? Or would it suffice to use a stack, say, only at the lowest level?

Chapter 3

The Representation of Linguistic Principles

One of the most important tasks in implementing a linguistic theory is how to turn the natural language, or semi-formal, definitions that are commonly found in the literature into the precise, formal definitions required for computation. The problem is to choose an appropriate level of representation between the abstract definitions used in the linguistics literature and the low-level, but efficient machinery used in computation. We will present a high-level notation for encoding principles that is abstract from certain control strategy choices, such as whether to interleave principle application with phrase structure construction or not, yet may be efficiently targeted to a variety of different machine architectures. As preliminary groundwork, first, let us review the important issues for representation design and how they are addressed in this system.

3.1 Design Issues

In this section, we discuss the reasons for adopting the following three design goals for our system:

1. A perspicuous representation that is fairly 'close' to linguistic definitions.
2. Efficiently executable definitions.
3. A representation that is abstract from control strategy choices.

An important goal of representation design is to select a representation that minimizes the amount of translation effort required of the grammar writer. There are many reasons why a 'direct' encoding should be adopted. Perhaps the most important is the correctness problem: that is, the problem of ensuring that all encoded forms are faithful to the original definitions. This is made all the more difficult by the lack of a standardized formal notation in the framework. Minimizing the encoding effort is also important from the point of view of theory maintenance. The rapid evolution of linguistic theories implies that frequent revisions will be necessary if parsing systems are not to become obsolete. The large number of extant versions of 'standard' theory is yet another motivating factor. Ideally, it should be possible to re-target a system for comparison purposes with minimal effort. All these factors suggest that the level of representation should be pitched as 'close' to that of the original definitions as possible.

The most obvious route to achieving a direct encoding is to design the representation to have the same vocabulary, i.e. the same primitive and combinative elements, as that used by linguists. For example, informal definitions often make use of pseudo-logic notation. Consider the following θ-marking uniformity condition on D-structure from Chomsky [12]:

(1) *"If a position X is T-governed by α, then X is occupied by an argument if and only if X is θ-marked by α."* *(from (3:86))*

For the purpose of this discussion, we will ignore the precise definition of linguistic terms such as T-government, θ-marking and argument. Instead, let us focus on the natural language connectives in the condition. They resemble their logical counterparts. For example, *"if... then..."* and *"if and only if"* are basically the logical implication and equivalence connectives, respectively. Hence, a natural step towards formalizing such a definition would be to substitute the formal logic connectives in place of the natural languages ones. To complete the formalization, we will also have to make explicit the quantifiers that range over the variables X and α, determine their relative scopes, and introduce a logical variable variable (plus an accompanying quantifier) to represent the element that occupies X. Linguistic terms like *"argument"* and *"D-structure position"* may be represented as one-place predicates. Similarly, binary linguistic relations such as *"T-governed by"*, *"occupied by"* and *"θ-marked by"* may be represented as two-place predicates. Hence, a formalized version of (1) might look something like:

$$\begin{aligned}(2)\quad & \forall X\, \exists\alpha [(\text{ D-structure-position}(X) \wedge \text{T-governed-by}(X,\alpha)\,) \rightarrow \\ & \quad (\exists Z[\text{ occupied-by}(X,Z) \wedge \text{argument}(Z)\,] \leftrightarrow \theta\text{-marked-by}(X,\alpha)\,)]\end{aligned}$$

The full 1st-order logic (FOL) notation employed by Stabler [52] is an example of this approach. However, there are well-known tractability problems for computing directly with FOL. Also, it is by no means clear that such a powerful notation is required to encode principle-based theories. (See the discussion of PROLOG in chapter 2 for details.)

At the other end of the spectrum, one could use a imperative programming language such as PASCAL (Wehrli [58]) or LISP (Kashket [32] and Dorr [20]) to provide a lower-level, but immediately operational specification. For example, a corresponding formulation in LISP might look something like:

(3)
```
(defun satisfies-condition (node)
   (if (T-governed-p node)
       (let ((alpha (T-governor-of node)))
          (if (argument-p (element-occupying node))
              (theta-marked-by-p node alpha)
              (not (theta-marked-by-p node alpha))))
       't))

(defun DS-filter (node)
   (if (satisfies-condition node)
       (if (has-subconstituents-p node)
           (every #'DS-filter (subconstituents node))
           't)))
```

Although the judgement is ultimately a subjective one, intuitively speaking, this definition is significantly less direct and perspicuous than the FOL formulation. In general, it will be harder to convince oneself both that the translation is correct and that changes will be propagated correctly should the original definition have to be updated. Also, unlike the FOL formulation, the LISP version (unnecessarily) commits the parser to a top-down traversal of phrase structure. This 'hardwiring' of control strategy is not flexible enough for our purposes: recall that one of the major motivating factors in building the system was to explore different control strategies. In the current implementation, principles may be interleaved with phrase structure construction, or applied only after skeletal S-structures are built, or some combination in between. As chapters 5 and 6 will describe, these two strategies operate using quite different methods. To insulate the grammar writer from such concerns, principles should be encoded 'declaratively' such that the semantics of the encoded forms can be understood independently without reference to the actual operational realization.

3.1.1 The Approach

In the remainder of the chapter, we will describe a representation that attempts to retain as much of the perspicuity of the FOL approach as possible without incurring the associated performance penalty. The gap between an abstract representation and the underlying operational specification will be bridged (albeit, somewhat

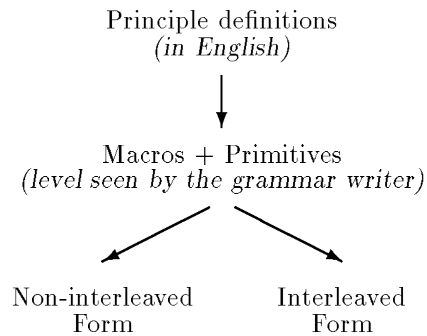


Figure 3.1 Multiple-targeting of linguistic principles.

incompletely) by an automatic translation procedure. This translation step, shown in figure 3.1, will allows us to 'multiply-target' different control strategy options in a manner that is transparent to the grammar writer. Basically, the representation will consist of two components:

1. *Two top-level principle-defining macros.*

   The simpler and more widely used macro is **`in_all_configurations`** which is used to encode universally quantified conditions over tree structures. (See figure 3.2 for a list of principles in the current system encoded using this macro.) The condition has the basic form $\forall X((X \in T \wedge p(X)) \rightarrow q(X))$, where X ranges over the constituents of a tree T, and p and q represent predicates that involve at least X. The second macro **`compositional_cases_on`** is used to encode compositionally defined operations on n-ary trees. For example, let Z be a constituent with two immediate sub-constituents X and Y. The macro may be used to define some function f over constituents such that $f(Z)$ is defined (recursively) in terms of the corresponding values for its children X and Y, i.e. $f(Z) = g(f(X), f(Y))$ for some function g.

2. *Linguistically-motivated primitives.*

   A set of linguistically-motivated primitives covering the access and manipulation of constituent features, category labels and sub-constituents.

Both components are written on 'top' of the underlying implementation language, PROLOG. Although the grammar writer has access to the full generality of the programming language, the multiple-targeting procedure is only defined for the two macro forms. This effectively constrains linguistic principles to be written using either one of these two forms. Note however that we deviate from this scheme for the principles of $\overline{\mathrm{X}}$-theory that define phrase structure. Such principles are treated

Operation	Encoded Using
θ-role assignment	`in_all_configurations`
θ-criterion	`in_all_configurations`
D-structure θ-condition	`in_all_configurations`
Subjacency	`compositional_cases_on`
Inherent Case assignment	`in_all_configurations`
Structural Case assignment	`in_all_configurations`
$\overline{\text{S}}$-deletion	`in_all_configurations`
Case filter	`in_all_configurations`
Case condition on traces	`in_all_configurations`
Trace theory (chain formation at S-structure)	`compositional_cases_on`
Wh-movement in syntax	`in_all_configurations`
Coindex Subject and INFL	`in_all_configurations`
Free Indexation	`compositional_cases_on`
Functional Determination	`compositional_cases_on`
Condition A	`in_all_configurations` (transformed)
Condition B	`in_all_configurations` (transformed)
Condition C	`in_all_configurations` (transformed)
Empty Category Principle (ECP)	`in_all_configurations` (transformed)
Control	`compositional_cases_on`
LF movement	Hand-coded
ECP at LF	`in_all_configurations` (transformed)
License operator-variables	`in_all_configurations`
License syntactic adjuncts	`in_all_configurations`
Wh-Comp requirement at LF	`in_all_configurations`

Figure 3.2 Parser Operations.

separately so that they may be compiled into an efficient machine for building phrase structure (to be described in chapter 4).

3.2 Overview of the Chapter

In the remaining sections of the chapter, we will illustrate the capabilities and limitations of the adopted representation by providing step-by-step examples of how principles in the current system are encoded. Let us briefly review the contents of the sections:

- *Using the* `in_all_configurations` *Form.*

  Section 3.3 contains a case study of how to directly encode English definitions of principles from Case theory using the universally quantified form `in_all_configurations`. We will address how *optionality* in Case assignment and access to constituents at D-structure (a level not directly computed by the phrase structure recovery procedure) are handled.

We will also take the opportunity to point out a (non-transparent) limitation of the underlying representation of phrase structures. In the current implementation, accessing constituents dominated by a larger constituent can be accomplished in a straightforward manner using various primitives supplied. However, due to the simplicity of the representation, there is no built-in method for accessing dominating constituents from a smaller constituent. The burden of ensuring that enough information gets passed to a parser operation for the operation to be carried out falls to the grammar writer. This is reflected in the definitions of many structural relations as an extra argument that holds 'context' information. For example, roughly speaking, government (see [Defn. 11] in appendix A) is normally considered to be a binary structural relation that obtains between two constituents that do not dominate one another. However, there is also the condition that there should be no intervening maximal projection, i.e. any phrasal node, between them. For example, the verb *sees* governs *Mary* but not *John* in [*John* [*VP* [*V sees*] *Mary*]]. To apply government, we must not only pass the two constituents but also enough surrounding context to determine if there is an intervening phrasal node. Hence, government is actually implemented as a ternary predicate `governs(X,Y,CF)` that holds if `X` governs `Y` *in* configuration `CF`.

- *Compositional Definitions. Part I*

  Certain parser operations (but not all as we shall see later) that require the establishment of *non-local*, i.e. potentially unbounded, structural relations pose difficult problems for two reasons:

  1. *Non-constructive definitions.*

     For example, the principle of free indexation in Lasnik & Uriagereka has the form:

     (4) "Freely assign indices" *(3:47b)*

     The relevant problem for computation is: How to turn an abstract definition of the form in (4) into a procedure that *sensibly* computes all possible indexings, i.e. all distinct assignments of indices, without emitting duplicate indexings? As it turns out, as we shall show later in the chapter, computing (4) basically amounts to enumerating the possible assignments of the *n-set partitioning problem*, a well-known problem in computational combinatorics. In the current system, an algorithm that enumerates the possible assignments is used.

     Note that this raises an important formalization issue. As will we see, the description of the algorithm cannot be claimed to be close to the notation used by linguists, i.e. (4). Also, it is not clear how one might attempt to raise the level of the representation to close the abstraction gap. For example, it is not clear how a compilation procedure could be constructed that automatically recognizes instances of well-known combinatorial problems from such abstract definitions.

2. *Interleaving.*
   There is also a conflict between the non-local nature of free indexation and the incremental processing employed by the interleaving model. Free indexation is obviously non-local because it allows any element to be coindexed with another element, or elements, anywhere in a sentence. Hence, if free indexation is to be interleaved with phrase structure construction, it must either: (1) simply wait until all elements to be indexed are available, i.e. have been recovered, before partitioning those elements into all possible combinations of distinct sets. Unfortunately, waiting to collect all possible elements would be tantamount to not interleaving at all. (2) An alternative is to compute the possible partitionings independently for each phrase as it is built. Then, as phrases are combined to become part of larger phrases, the partitionings of smaller phrases must be updated in accordance with the possible partitionings of the larger phrases. (Of course, care must be taken to ensure that all possible and no duplicate partitionings are produced.) This approach is the one adopted in the implemented system. General support for inductive definitions over tree structures — we will adopt the term *compositional definition* in this chapter — is provided by the `compositional_cases_on` form to be described in section 3.4.

Section 3.4 will illustrate the use of the `compositional_cases_on` form for two parser operations, free indexation and chain formation (recovery of the history of movement). Because neither operation is defined using compositional definitions in the linguistics literature, ensuring that the implemented definitions correctly cover all and only the possible cases is an especially important concern. Hence, to partially rectify the situation, we will prove that the definitions of both operations are *complete* in the sense that they are guaranteed to produce all possible assignments (of indices and chains), and *non-redundant* in the sense that they cannot produce duplicate assignments. Furthermore, as a side effect of the proof, we will also obtain a combinatorial analysis of the general problem of free indexation.

However, this does not constitute a completely satisfactory answer to the problem of representing non-constructively defined principles (or at least, definitions that do not immediately lead to constructive forms). A more satisfactory approach should include some procedure that recognizes non-constructive statements that are instances of well-known combinatorial problems, and then produce efficiently executable definitions automatically. Unfortunately, the current system does not have this capability. However, in section 3.5 (to be reviewed immediately below), we will discuss other cases involving non-local relations that the system does automatically transform into compositional definitions.

- *Compositional Definitions. Part II*

  Finally, section 3.5 will investigate the representation of principles that operate on non-local structural relations, but involve *minimality* in the sense of defining some smallest *domain* where those relations may or may not hold. For example, pronoun Binding is a non-local relation as illustrated in (5). However, there exist locality restrictions on Binding, namely the Binding conditions as discussed in chapter 1.

  (5) [John$_i$ knew [that [Mary [told her [she [thought [she [saw him$_i$]]]]]]]

  We will illustrate how such principles can be succinctly expressed using the `in_all_configurations` form. The system will automatically transform such definitions into equivalent compositional definitions. (See figure 3.2 for the list of parser operations that can take advantage of this transformation.)

  Apart from allowing certain principles to be more naturally expressed using the universal quantification form, an important advantage of the transformation procedure is that the transformed definitions also avoid certain kinds of repeated computation. Because a compositional definition computes structural relations in an incremental fashion, we can avoid any unnecessary repetition involved in finding minimal but *non-local* structural relations. For example, each time a larger phrase is built, a routine that looks for a Binder should not have to repeat its search in the smaller phrases it has already examined. The transformation procedure automatically transforms non-local relations such as Binding into local versions that can be incrementally applied in an efficient manner.

3.3 Using the `in_all_configurations` Form

A Case Study: Case Theory

Consider the following two sentences:

(6) a. *it it is likely John to be here
    b. it is likely that John is here

```
:- caseFilter in_all_configurations CF where
       lexicalNP(CF) then assignedCase(CF).

lexicalNP(NP) :- cat(NP,np), \+ ec(NP).
assignedCase(X) :- X has_feature case(Case), assigned(Case).
```

Figure 3.3 The definition of the Case filter.[2]

The usual explanation for the contrast between these two similar-looking examples is that *John* receives abstract Case in the latter example, but not in the former.[1] The key difference being that the non-matrix (or embedded) clause in (6b) is tensed, whereas the corresponding clause in (6a) is not. To exploit this distinction, we can say that a tense element in (6b) (which is missing or inactive for untensed clauses) is available to assign Case to, or Case-mark, *John*. Now, the Case filter, a principle of Case theory, requires that all lexical NPs such as *John* must receive Case. Hence, (6a) will violate the Case filter.

3.3.1 Case Filter

How can we implement this filter? The definition of the filter given in Lasnik & Uriagereka has the following semi-formal form:

(7) At S-structure, every lexical NP needs Case. *(1:20)*

Here, the basic elements being named are constituents (NPs) and features (lexicality, i.e. the property of having a morphological realization, and Case slot features). The combinatory terms *needs* and *every* imply that the representation, as a minimum, must include the ability to state feature membership conditions over constituents, and to universally quantify over particular classes of S-structure constituents.

[1]The term 'abstract' here refers to the fact that Case may not always be morphologically realized. For example, in *John saw him/*he* and *he/*him saw John*, the form of the pronoun *he* changes depending on whether it is in a subject or an object position, whereas the form of *John* remains invariant.

[2]For convenience, we will repeat the description of PROLOG notation introduced in chapter 1:

1. Logical variables are distinguished from constants by the use of an uppercase letter as the initial character. The scope of a variable is a clause. Also, '_' is used to indicate an anonymous or 'don't care' variable. The scope of '_' is itself.
2. Elements enclosed in square brackets denote lists. In particular, `[]` denotes the empty list. `[Y]` denotes a list with one element `Y`, `[Y1,Y2]` a list of two elements, and so on. `[X|L]` denotes a list with first element `X` and with `L` representing the remainder of the list.
3. Outside of lists, (`,`) is used to denote conjunction. (`:-`) — to be read as 'if' — is used to denote implication and (`\+`) is used to denote negation as failure to prove.

Compare (7) with the formulation shown in figure 3.3. The first statement represents a universally quantified formula over constituent structures. `caseFilter` names the following condition: "in every configuration `CF` where `lexicalNP(CF)` holds, then `assignedCase(CF)` must also hold." The next two clauses define what we mean by the terms *lexical NP* and (to be) *assigned Case*:

1. The first clause states that: "`NP` is a lexical NP if: (1) it has the category label `np`, and (2) it is not an empty category (`ec/1`)." Both `cat/2` and `ec/1` are primitives in the system.[3]

2. Two other two primitives are used in the second second, namely `has_feature/2` (which tests whether a constituent has the syntactic feature in question) and `assigned/1` (which tests if some feature slot has been bound). Hence, `assignCase/1` is meant to be read as: "`X` is assigned Case if: (1) `X` has a Case feature with slot `Case` and (2) that slot has been filled."

There are two important observations that we should make at this point:

- PROLOG uses a simple depth first (left-to-right) search strategy. Also, goals are matched with each clause in the order that clauses are written. The order of execution is unimportant for `lexicalNP/1`. However, `assignedCase/1` relies on the Case slot being identified before it is tested.

- Note also that the universally quantified form may be understood in a control-independent manner. In particular, the grammar writer need not know what PROLOG form or forms it expands into. For example, in the case when the principle is not being interleaved, the macro expands into a 'tree-walker' that recursively descends a given phrase structure representation testing each subtree using the specified conditional form. However, no tree-walking is done in the case when the principle is interleaved. Since the macro may been understood at a level abstract from its realization, there is no need to commit to any particular search strategy, as in, say, a direct operational encoding in an imperative programming language.

3.3.2 Structural Case Assignment

In the current implementation, the Case filter, i.e. testing for the presence of Case, can only operate after Case has been assigned. Now, according to the grammatical theory, Case may only be assigned to NPs at S-structure by certain constituents, called Case assigners, or markers. For simplicity, we will restrict the discussion of Case assigners to the following classes:

---

[3]Note: `p/n` is standard notation for a predicate `p` of arity `n`.

1. *AGR (Agreement).*

   Consider example (6) again, repeated below as (8):

   (8) a. *it it is likely John to be here
   b. it is likely that John is here

   Here, the (non-vacuous) agreement element of inflection (Infl) will assign nominative Case to *John* in the latter sentence. However, only finite, but not infinitival clauses have a non-vacuous agreement element (AGR). This implies that *John* will not get Case in the former case. As it turns out, there are no other possible Case assigners in the sentence. Hence, *John* will violate the Case filter.

2. *Verbs.*

   (a) Verbs usually assign accusative Case to their complements (if they exist) as in *John hit him* (cf. **John hit he*).[4]

   (b) Certain verbs, known as Exceptional Case Markers (ECM), may also assign Case to subjects of clausal complements as in *i believe John to be here* (cf. the ungrammatical example (8a)). However, this does not hold when the verb is passivized: for example, as in **it was believed John to be here*.

   (c) However, there are also certain 'technically intransitive' verbs such as *seem* or *ask* that do not assign Case to their complements. (See section 3.3.3.2 in Chomsky [12] for further discussion.) In the current implementation, such verbs (as well as the passivized ones) are marked with the lexical feature `noCasemark`.[5]

3. *Complementizers.*

   A complementizer such as *for* in English may assign oblique Case to the subject of a clause (another case of Exceptional Case Marking) . For example, *John* is Case-marked by *for* in *i am eager for John to be here*, thereby avoiding a violation of the Case filter — cf. **i am eager John to be here*.

---

[4] The actual implementation is slightly more complicated. Verbs sometimes may also assign dative Case to direct objects — see the Japanese examples shown previously in figure 1.3.

[5] Passivized verbs in English are implemented as follows: the passive auxiliary *be* subcategorizes for a verb phrase to be headed by a verb with the condition that the verb must take the past participle form, as in [*VP be* [*VP* [*V*`[+en,noCaseMark]` *arrest*]...]]. It also marks the subcategorized verb with the feature `noCasemark`. For technically intransitive verbs, `noCasemark` will appear directly in their lexical entries.

[6] In the current implementation, the two components of Infl, i.e. agreement and tense, are not represented as separate constituents, but simply as features of Infl.

```
caseAssigner(Infl,nom) :- % Class 1: AGR assigns nominative Case.
        cat(Infl,i),
        Infl has_feature agr(_).6
caseAssigner(Verb,acc) :- % Class 2: V assigns accusative Case
        cat(Verb,v),      % unless marked otherwise.
        \+ Verb has_feature noCasemark.
caseAssigner(ECM,obq) :-  % Class 3: ecm complementizer ('for') assigns
        cat(ECM,c),       % oblique Case.
        ECM has_feature ecm.
```

Figure 3.4 The definition of a Case assigner.

These cases are summarized in the definition of the predicate `caseAssigner` shown in figure 3.4. Here, `caseAssigner(X,C)` holds only if constituent `X` is a Case assigner that assigns Case of type `C`. Each case that we have mentioned above is encoded separately by a different clause that may be regarded as a single disjunct of an aggregate formula for the predicate. That is, if any clause holds, then the predicate also holds. (As discussed in chapter 2, the disjunction represented by the separate clauses encodes an exclusive-or rather than an inclusive-or.)

Having identified the possible Case assigners, Lasnik & Uriagereka define the conditions under which an element β can receive Case from an assigner α as follows:

(9) α assigns Case to β if *(1:26)*
   a. *α is a Case assigner;*
   b. *α governs β;*
   c. *α is adjacent to β.*

α assigns Case to β only in configurations of *government* where there cannot be "too much intervening" structure. Configurations that contain no intervening maximal projection, i.e. a phrasal boundary, between the assigner and assignee, as in [$\bar{V}$ *V NP*], i.e. a head-complement configuration, or [*IP NP* [$\bar{I}$ [*I(AGR)*]...]], i.e. a specifier-head configuration, will satisfy government. Obviously, there must be certain exceptions to this rule such as the case where the subject of a clause may be Case-marked by an element outside the first maximal projection, as in *i believe* [*John to be here*]; the bracketing here is used to indicate the relevant phrasal boundary. (For brevity, we omit the formal definition of government. See [Defn. 11] in appendix A.) Finally, in English there is the additional constraint that the elements involved must be adjacent. For example, Lasnik & Uriagereka cites **i believe sincerely John to be here* as evidence where intervening material, in this case the adverb *sincerely*, may block Case assignment.

```
sCaseConfig(CF,Assigner,Case,NP) :-
    governs(Assigner,NP,CF),          % Assigner governs NP in CF, and
    cat(NP,np),                       % NP is a noun phrase, and
    caseAssigner(Assigner,Case),      % Assigner is a Case-marker assigning
                                      % Case of type Case, and
    adjacent(Assigner,NP,CF)          % Assigner and NP must be adjacent
            if caseAdjacency.         % sub-constituents in CF if parameter
                                      % caseAdjacency holds.
```

Figure 3.5 Configurations of structural Case assignment.

```
:- sCaseAssign in_all_configurations CF
        where sCaseConfig(CF,Assigner,Case,NP)
          then assignCase(Assigner,Case,NP).
```

Figure 3.6 The top-level definition of structural Case assignment.

These conditions may be encoded in a parallel fashion to (9), as shown in figure 3.5. `sCaseConfig/4` identifies a a structural Case assignment configuration `CF` as one that contains the following three components:

1. `Assigner`: the Case-marking element.
2. `Case`: the type of Case, e.g. nominative, accusative, etc., to be assigned.
3. `NP`: the noun phrase to receive Case.

We have assumed the existence of a three-place predicate `governs` such that a goal `governs(X,Y,Z)` holds only if `X` and `Y` are sub-constituents (note: not necessarily immediate sub-constituents) of `Z`, such that `X` governs `Y`. Similarly, `adjacent` is a three-place predicate such that `adjacent(X,Y,Z)` holds only if `X` and `Y` are adjacent sub-constituents of `Z`. Finally, `caseAdjacency` is a language parameter with the following interpretation: `caseAdjacency` holds if and only if the language requires Case adjacency.

We can now use `sCaseConfig/4` in the definition of structural Case assignment as shown in figure 3.6. As with the Case filter, the top-level definition makes use of the universal quantification macro. This form states that in all configurations satisfying the pre-conditions of Case marking, the operation of assigning Case must be applied. The act of assigning Case, encoded using the three-place predicate `assignCase`, is implemented by unifying the Case slot of the NP with `Case`.

Optionality in Case Assignment

One complication for the definition of `assignCase/3` is that Case marking may sometimes be optional. For example, consider the difference between *believe* and *want* when the non-matrix subject is non-overt (*PRO* being an empty pronoun):

(10) a.* I believe *PRO* to be here
b. I want *PRO* to be here
c. I believe John to be here
d. I want John to be here

One explanation for this contrast, discussed in Lasnik & Uriagereka, is that *want* optionally assigns Case whereas *believe* is an obligatory Case marker. In (10d), *want* must Case-mark *John*; otherwise the Case filter will be violated. Now, one property of *PRO* is that it is a special element that must not be Case-marked.[7] Since *want* is an optional Case marker, we can avoid violating this condition by simply not assigning Case in (10b). On the other hand, *believe*, being an obligatory Case marker, must assign Case to the forbidden element; hence the ungrammaticality of (10a). This distinction is encoded in the lexical entries of *want* and *believe*. Although both verbs are classified as ECM verbs, i.e. they both have the feature `ecm(Type)`, `Type` will be instantiated to `opt` (for optional) for *want*. Now, consider the two clauses that define `assignCase/3` as shown in figure 3.7. The first clause handles the situation when no Case is actually assigned, and the other clause handles the situation when Case is assigned (by unifying the Case to be assigned with the Case slot feature of the assignee). Note however, that these two clauses have 'overlapping scope'; that is, a Case assigner with the feature `ecm(opt)` can satisfy *either* clause, whereas an element without this feature, e.g. `ecm(oblig)` (for obligatory) for *believe*, can only satisfy the second clause. The parser accommodates both possibilities by trying each clause in the order in which they appear.

Configurations and Phrase Structure Representation

Consider again the configurations of structural Case assignment shown in figure 3.5. Note that *binary* structural relations such as 'governs' and 'adjacent' have been defined using three-place predicates. This situation arises because the phrase structure representation adopted effectively allows only 'one-way' access between constituents. More precisely, the representation of a phrase will contain all of its sub-constituents,

[7] In the account adopted in this thesis, this condition is not explicitly stated anywhere in the set of principles used by the parser. Instead, it falls out from the condition that *PRO* cannot be governed, since Case-marking presupposes government. This latter condition, which is known as the *PRO* theorem, is not explicitly stated anywhere either. In fact, it is a consequence of the need for *PRO* to satisfy two fundamental principles that are explicitly stated, namely, conditions A and B of the Binding theory.

```
% assignCase: Assigner × Case × Assignee
assignCase(ECM,_,_) :-                    % Case 1: Optional ECM
        ECM has_feature ecm(opt).         % do not assign Case.
assignCase(_,Case,NP) :-                  % Case 2: assign Case.
        NP has_feature case(Case),
        morphCaseAgreement(NP,Case).      % 'he/him', 'they/their'
```

Figure 3.7 The optionality of Case assignment.

but there will be no (direct) way to access any other part of a structure that does not form some sub-constituent of the phrase. For example, given a structure [*IP* [*NP I*] [*VP pointed out* [*NP the man who I saw*]]], the object of the matrix clause is represented not as a single 'node' NP, but as the entire 'sub-tree' rooted at the object position, i.e. [*NP the man who I saw*]. From this sub-tree, it is a simple and direct procedure to (recursively) access any node that it dominates (i.e. any descendant). However, there is insufficient information to access any dominating node, e.g. the matrix VP node; or indeed, any node that does not dominate it nor is dominated by it, e.g. the subject *I*. Hence, it is important that predicates that encode structural relations be provided not only with the elements on which some relation is supposed to hold, but also with enough of the 'surrounding' phrase structure, or context, so that the structural connection between the elements can be established. For example, *the man* is adjacent to *who* in [*NP* [*NP the man*] [*CP* [*NP who*] [$\bar{C}$ *I saw*]]]. But the predicate `adjacent/3` requires a third argument, i.e. the context in which two elements are to be adjacent. This argument must be filled by a node that dominates both arguments; in this case, (at least) the NP node that represents the entire relative clause. In general, it will be up to the grammar writer to supply a configuration that dominates all arguments of a structural relation. Returning to figure 3.5, we have that `CF`, a configuration of structural Case assignment with three components (namely, a Case assigner, the type of Case to be assigned, and a NP to receives Case), must be a node that includes both the assigner and the assignee. For example, [*CP for* [*IP John to be here*]] is a Case assignment configuration, but not [*IP John to be here*].

Conceptually speaking, it would be relatively straightforward to augment the current representation with the necessary 'back-pointers' to parent nodes, and thus avoid having to 'manually' pass the extra parameter.[8] However, to avoid the com-

[8] Mark Johnson *(p.c.)* has brought to the attention of the author an interesting phrase structure representation (in PROLOG) that does not require explicit pointers to accomplish the doubly-linked scheme. Basically, each node with category `C` is represented as a triple `n(Ith,C/L,P)` where `L`

plexity involved in devising an efficient scheme for mainpulating and updating back-pointers in PROLOG, the present system has remained with the simple and non-transparent representation.[9] Actually, this unfortunate situation is remedied to some extent because the system also records certain kinds of positional information in the feature sets of constituents that would otherwise require back-pointers to verify. For example, constituents that occupy A-positions, short for *argument positions*([Defn. 2]) and/or complement positions have the features `apos` and/or `compl` automatically recorded in their feature sets, as a side-effect of the phrase structure recovery routine. Hence, instead of having to be able to access dominating nodes to check whether a certain element X occupies an A-position, we can simply test whether X `has_feature apos` holds.

This completes our discussion of structural Case assignment. Up to now, the principles outlined operate on phrase structure at the level of S-structure only. As chapter 4 will describe, for efficiency reasons, complete phrase structure representations are only recovered for the levels of S-structure and LF. In particular, D-structure constituents are recovered from S-structure only on a demand basis (to be made clear below). To illustrate how principles that operate on D-structure constituents are encoded, we now turn to the formalization of another instance of Case marking, that of inherent Case assignment.

Accessing D-structure constituents

Consider the nominal form [*NP* [*NP their*] [$\bar{N}$ [*N destruction*] [*NP of the city*]]]. Following Chomksy [12], we assume that both *the city* and *they* (*their* being the corresponding genitive form) are overt NPs that must be assigned Case at D-structure in order not to violate the Case filter. As with the corresponding non-nominal form *they destroyed the city*, the arguments *the city* and *they* are assumed to play the part of the 'target of destruction' and 'agent of destruction', respectively. Hence, they will receive a 'patient' and an 'agent' θ-role (respectively) from the noun *destruction* that heads the construction. In general, we have:

(11) *"Inherent Case is assigned by α to NP if and only if α θ-marks NP."*
from (pg. 193, Chomsky [12])

is a list of the nodes that it immediately dominates, `P` being the node that immediately dominates the current node, and `Ith` being the (unique) index of the current node among its siblings. Then the parent relation and i-th child relations can be simply defined as `P` *parent_of* `n(_,_,P)` and `n(Ith,T,Node)` *is* `Ith` *child_of* `Node`, respectively, where `Node=n(_,_/L,_)` and `T` is the `Ith` member of `L`.

[9] To properly implement a transparent representation will also require 'delays' in the case of principle interleaving. For example, the parent of a node X may not be known at the same time the node X is first made available by the phrase structure recovery routine. Hence, a goal such as Y `immediately_dominates` X would have to be delayed until Y has been generated.

```
:- inCaseAssign in_all_configurations CF where
       inCaseConfig(CF,Case,Items) then assignInCase(Case,Items).

% CF is a configuration of inherent Case assignment with two components:
% (1) Case, and (2) a list of constituents Items to receive Case.
inCaseConfig(CF,Case,Items) :-
    thetaConfig(CF,_,Items),         % CF is a θ-config. with arguments
    Head head_of CF,                 % Items, and Head heads CF, and
    cat(Head,C),                     % Head is of category C, and
    inherentCaseAssigner(C,Case).    % C is an inherent Case marker that
                                     % assigns Case of type Case.

% Category × Case
inherentCaseAssigner(n,gen).
inherentCaseAssigner(a,gen).
inherentCaseAssigner(p,obq).
```

Figure 3.8 Inherent Case assignment.

This uniformity condition ensures that the two arguments will not violate the Case filter. The correspondence between the nominalized and non-nominalized forms extends to passive constructions. For example, the nominalized form of *the city was destroyed* $[_{NP}$-t] will be $[_{NP}$ $[_{NP}$ *the city's*] $[_{\bar{N}}$ $[_{N}$ *destruction*] $[_{NP}$-t]]], in which Move-α has (optionally) caused the *the city* to move from the complement position at D-structure. Of course, the other form $[_{NP}$ $[_{\bar{N}}$ $[_{N}$ *destruction*] $[_{NP}$ *of the city*]]] is also available in the situation where there is no movement.

In general, adjectives such as *proud*, as in $[_{AP}$ $[_{A}$ *proud*] $[_{NP}$ *of John*]], as well as nouns assign genitive Case. Genitive Case is morphologically realized by affixing a marker that depends on the position of the NP at S-structure. For instance, in English it is realized as the semantically empty marker *of* for NPs in complement position (*of*-insertion), as in *destruction of the city* and *proud of John* (cf. **proud John*). However, in subject position, genitive Case is realized as the possessive marker *'s*, as in the passive form *the city's destruction*. For *their destruction of the city*, we assume that *'s* will combine with the subject *they* to form *their*.[10] Finally, prepositions are also assumed to Case-mark the NPs that they assign a θ-role to, as in *the city was destroyed* $[_{PP}$ $[_{P}$ *by*] $[_{NP}$ *John*]]. However, in English, oblique Case (unlike genitive Case) is not morphologically realized.

[10] For simplicity, in the present system, no derivation *they* + *'s* $\rightarrow$ *their* is actually done. The genitive form *their* has its own lexical entry with a special feature indicating that it 'already has' genitive Case.

```
assignInCase(Case,SSItem) :-
        recoverDsConstituent(DSItem,SSItem),
        DSItem has_feature case(Case) if cat(DSItem,np).
```

Figure 3.9 Case assignment at D-structure.

To summarize, inherent Case assignment can be encoded in a parallel fashion to structural Case assignment, as shown in figure 3.8. Here, we assume a predicate `thetaConfig/3` that defines the configurations of θ-marking, `CF`, where `Items` holds a list of the elements in the configuration to be θ-marked. (We will not go into the details of `thetaConfig` here; the definition of θ-role assignment will be described in detail in chapter 6.) For example, `Items` would be a list containing the two elements *their* and *the city* for the θ-configuration *their destruction of the city*. Then, the configurations of inherent Case assignment are simply those θ-marking configurations that are headed by an inherent Case assigner. But because the phrase structure recovery routine produces structures at the level of S-structure, not D-structure, the θ-configuration items returned are the elements that occupy θ-positions at S-structure. For example, the complement position in the phrase [*NP the city's* [$\bar{N}$ *destruction* [*NP-t*]]] is occupied by the trace of *the city* at S-structure.[11] These S-structure constituents are passed to the predicate `assignInCase` that will unify `Case` with the Case slots of the relevant constituents at D-structure. To retrieve the corresponding constituent that occupies a given θ-position at D-structure, we make use of a primitive operation `recoverDsConstituent` (to be described in section 4.3.2), as shown in figure 3.9. (For simplicity, we have only shown the definition of `assignInCase` for the instance where only a single item is to be assigned Case.[12])

In the current implementation, it is left to the grammar writer to specify when to access D-structure constituents. An alternative (but equivalent procedure) is to first re-express a principle that operates on D-structure as a (derived) principle

[11] Chomsky [12] distinguishes between positions and the elements that occupy those positions. Strictly speaking, a θ-role is assigned to a position, whereas Case is assigned to the element that occupies a given position. Of course, the effects of such a distinction will only be apparent under movement. In the current representation of phrase structure, both kinds of syntactic features such as Case and θ-slots are held in the feature set associated with an element. This can probably be regarded as a design flaw in the current implementation. Hence, when elements are moved by Move-α, we should be careful to copy certain features to the new destination, e.g. Case, and leave others, e.g. θ-slots, behind in the feature set of the trace that occupies the original position.

[12] Note that we have retreated considerably from the strict uniformity condition (11). Not all configurations of θ-marking are configurations of inherent Case assignment, e.g. verbs θ-mark their complements, but verbs assign structural, not inherent, Case. Also, clausal arguments must be θ-marked, e.g. *the belief* [*that Mary left*], but it is not clear whether clauses should also be Case-marked (see note 9 of chapter 1 in Lasnik & Uriagereka).

that operates at S-structure. This is often done in the literature. For example, Chomsky [12] re-expresses the uniformity condition shown in (11) as:

(12) *"If α is an inherent Case-marker, then α Case-marks NP if and only if [it] θ-marks the chain headed by NP."* *(from ch.3, no.272)*

The reformulated definition which operates at S-structure makes explicit the use of the chains constructed through movement that the call to the D-structure constituent recovery primitive encoded.

This concludes our discussion on the use of the `in_all_configurations` form. This universal quantification macro is used to encode many of the principles in the current implementation. In fact, as figure 3.2 shows, a majority of the current parser operations are encoded directly in this form. In the next two sections, we will turn to illustrate the second principle-defining form `compositional_cases_on` which accommodates definitions that involve structural induction over tree structures.

3.4 Compositional Definitions
Part I: Enumeration of Combinatorial Operations

In this section, we will describe how inherently combinatorial operations such as free indexation and chain formation are encoded in the current system. These operations share the general characteristic of having many logically possible assignments, of indices and chains, respectively, for any given phrase structure. In general, there are two major issues to be considered in enumerating such assignments:

- How can we guarantee that the implementation will recover each logically possible assignment? Unless this can be proved, there will be the risk of failing to produce an assignment that corresponds to a well-formed structure for some particular sentence, or an assignment that is ill-formed for a particular sentence will be ruled out for the wrong reason.

- How can we construct a procedure that works irrespective of whether the operation is to be interleaved or not? As will be explained below, the obvious procedures for enumerating such assignments do not accommodate the interleaved model. In fact, it will be necessary to define such procedures compositionally over tree structures in order for assignments to be produced incrementally as partial structures are recovered.

In this section, we will describe compositional definitions for free indexation and chain formation that provably enumerate all and only those assignments, i.e. without duplicates, that are logically possible.

Section Roadmap

In section 3.4.1, we will use linguistic examples to introduce the operations of free indexation and chain formation. We will also take the opportunity to expand on the two problems for enumeration that were briefly mentioned above. Next, sections 3.4.2 and 3.4.3 will each present a compositional definition together with a proof of correctness for free indexation and chain formation, respectively.

3.4.1 The Problems of Chain Formation and Free Indexation

Completing Phrase Structure Representations

As chapter 4 will describe, the phrase structure recovery procedure initially recovers representations at the level of S-structure for reasons of computational efficiency. These representations are incomplete in many respects. For example, the phrase structure recovery procedure does not recover the chains formed by syntactic movement between D- and S-structure. Also, indices that indicate coreference possibilities for noun phrase constituents such as pronouns and anaphors are missing. Empty noun phrases are particularly underspecified. Although they are inserted (where possible) to occupy positions normally occupied by overt noun phrases, e.g. subject and object positions, during the initial phrase structure recovery stage, they lack many syntactic features. For instance, their status as being empty elements generated through Move-α, or not, as may be the case, remains undetermined. For example, the phrase structure recovery procedure will assign essentially the same structure to (13a) and (13b):

(13) a. John seems [[*NP*-e] to be happy]
     b. John wants [[*NP*-e] to be happy]

In particular, the empty element [*NP*-e] will occupy the subject position of the clausal complement in both cases. However, in order to satisfy θ-theory, [*NP*-e] must be a trace in the former case (in fact, it must be the trace of *John*, thus forming a chain $\langle$*John*, [*NP*-e]$\rangle$), but not in the latter case where it is usually assumed that *John* has not been subjected to Move-α. For example, compare (13) with (14):

(14) a. *John seems that he is happy
     b. John wants Mary to be happy

To capture both possibilities shown in (14), the [*NP*-e] constituent must be allowed to *optionally* participate in movement. The task of recovering the possible movement chains falls under the province of the chain formation algorithm. Principles of grammar from other modules such as θ-theory will restrict the possibilities to those that are well-formed.

As mentioned above, noun phrases also lack the indices that are necessary to indicate various coreference possibilities. For example, the sentence *John believes Bill will identify him* is ambiguous. Here, the pronoun *him* can be interpreted as being coreferential with *John* or with some person not named in the sentence, but not with *Bill*. We can represent these various cases by assigning indices to all noun phrases in a sentence together with the interpretation that two noun phrases are coreferential if and only if they are coindexed, that is, if they have the same index. Hence the following indexings represent the three coreference options for pronominal *him*:

(15) a. John_1 believes Bill_2 will identify him_1
b. John_1 believes Bill_2 will identify him_3
c. *John_1 believes Bill_2 will identify him_2

Once indices have been assigned, Binding conditions that state constraints on the locality of reference of pronominals and R-expressions, i.e. *referential* elements such as *John* and *Bill*, will conspire to rule out the impossible interpretation (15c). At the same time, the constraints will allow the other two (valid) interpretations. *Free indexation* is the process of assigning indices to noun phrases. (16) contains an updated version of the definition shown previously in (4): form:

(16) Assign indices freely to all noun phrases.[13]

Hence, free indexation accounts for the fact that we have coreferential ambiguities in language. Other principles of grammar will interact so as to limit the number of indexings generated by free indexation to those that are semantically well-formed.

Finally, it is worth mentioning at this point that the phrase structure recovery procedure also leaves the Binding theory status of empty noun phrases unspecified. As explained in chapter 1, some empty noun phrases exhibit the same locality restrictions as overt anaphors or pronouns (or both or neither of them). As explained in chapter 2, a theory of *functional determination* can be used to determine the appropriate status. By contrast, the status of overt nouns as anaphors or pronominals are specified in the lexicon.

The Problems of Formalization and Interleaving

One common aspect of both free indexation and chain formation is that, in general, there are many possible assignments of indexings and chains to a given phrase structure at S-structure. In general, free indexation must be capable of

[13] The exact form of (16) varies according to different theories in the principles-and-parameters framework. For example, in Chomsky [10] (pg.59), free indexation is restricted to apply to A-positions at the level of S-structure, and to $\bar{\text{A}}$-positions at the level of logical form.

5 indexings for sentences with 3 noun phrases:

(111)	John$_1$ wanted PRO_1 to forgive himself$_1$
(112)	John$_1$ wanted PRO_1 to forgive him$_2$
(121)	John$_1$ wanted Mary$_2$ to forgive him$_1$
(122)	John$_1$ wanted Mary$_2$ to forgive herself$_2$
(123)	John$_1$ wanted Mary$_2$ to forgive him$_3$

15 indexings for sentences with 4 noun phrases:

(1111)	John$_1$ persuaded himself$_1$ that he$_1$ should give himself$_1$ up
(1222)	John$_1$ persuaded Mary$_2$ PRO_2 to forgive herself$_2$
(1112)	John$_1$ persuaded himself$_1$ PRO_1 to forgive her$_2$
(1221)	John$_1$ persuaded Mary$_2$ PRO_2 to forgive him$_1$
(1223)	John$_1$ persuaded Mary$_2$ PRO_2 to forgive him$_3$
(1233)	John$_1$ wanted Bill$_2$ to ask Mary$_3$ PRO_3 to leave
(1122)	John$_1$ wanted PRO_1 to tell Mary$_2$ about herself$_2$
(1211)	John$_1$ wanted Mary$_2$ to tell him$_1$ about himself$_1$
(1121)	John$_1$ wanted PRO_1 to tell Mary$_2$ about himself$_1$
(1232)	John$_1$ wanted Bill$_2$ to tell Mary$_3$ about himself$_2$
(1123)	John$_1$ wanted PRO_1 to tell Mary$_2$ about Tom$_3$
(1213)	John$_1$ wanted Mary$_2$ to tell him$_1$ about Tom$_3$
(1231)	John$_1$ wanted Mary$_2$ to tell Tom$_3$ about him$_1$
(1234)	John$_1$ wanted Mary$_2$ to tell Tom$_3$ about Bill$_4$

Figure 3.10 Example of various indexings

generating each possible indexing or else risk missing a valid interpretation for some sentence. For example, there are five and fifteen possible indexings for a sentence containing three and four noun phrases, respectively; all of which correspond to some valid interpretation as shown in figure 3.10. As discussed at the beginning of the chapter, the relevant problem for computation is how to turn an abstract definition of the form shown in (16) into a procedure that provably generates all possible indexings? Moreover, in the interest of computational efficiency, we would also like the procedure to (provably) generate *only* the possible indexings without redundancy. Finally, the system should derive such a procedure *automatically*, i.e. without the intervention of the grammar writer, for reasons of perspicuity and abstraction.

An additional problem is that the chain formation procedure must accommodate the possibility of interleaving chain formation. Let us briefly review the interleaving model. Basically, the idea here is to apply the well-formedness conditions that represent linguistic principles to partial structures 'as they are assembled' by the

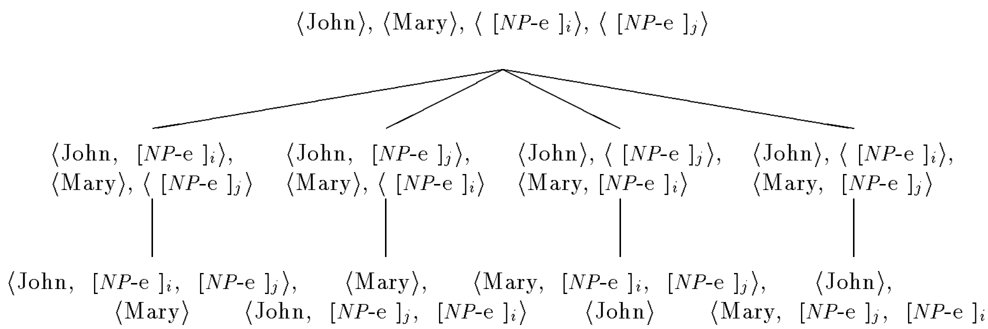


Figure 3.11 Computation tree for chain formation.

phrase structure recovery procedure. The motivating factor for this approach (as opposed to a scheme in which well-formedness principles are only applied after the structure for a complete sentence has been recovered), is that it will be possible to avoid building a complete, but ill-formed structure if some principle can be applied to rule out some portion of that structure. Let us now briefly consider the problems faced by chain formation in the light of this model.

Perhaps the simplest method for assigning chains to a structure is to first collect all the constituents that could possibly be members of a chain into a list. (We will consider only the movement of NPs in this discussion.) There is only one possible assignment in which chains may contain at most one element. That is, by having each element of the list head its own (trivial) chain. Then, the possible chain combinations with at most two elements can be found by optionally adding any [NP-e] element (only empty NPs can be traces) to any chain of length one. The possible combinations for chains with at most three elements can be incrementally derived from each combination of length up to two by optionally adding any [NP-e] not already part of a non-trivial chain, i.e. a chain with more than one member, to any chain with two members. In general, we only need to repeat this last step up a maximum of $n + 1$ times where n is the number of elements in the original list. Figure 3.11 illustrates this procedure for the list *John*, *Mary*, [NP-e]$_i$ and [NP-e]$_j$. Here, each of the nine nodes in the computation tree represents one of the possible chain assignments. This simple procedure produces each possible chain assignment without any duplicates and works well in the case where chain formation applies after structure building has been completed. Unfortunately, this procedure is not flexible enough to be used in the interleaved model because all noun phrases in a sentence must be identified before the computation tree can be constructed. But, to delay chain formation until all noun phrases in the sentence have been recovered by the phrase structure recovery procedure would negate much of the advantage of the interleaved model.

A similar procedure also works for free indexation. All and only the possible indexings can be easily found by, first, collecting all noun phrases into a list. Then, simply take each element in turn and optionally coindex it with each element following it in the list. However, this method also suffers from the same problem with delayed evaluation as the corresponding method for chain formation.

However, a procedure that is defined *compositionally* on phrase structures can be effectively interleaved. That is, chain formation, for example, should be defined so that the chains present in some phrase are some function of the chains present in that phrase's sub-constituents. Then, chains can be computed incrementally for all individual phrases as they are built. Such a method would have the advantage of being *unbiased* in the sense that it will be applicable irrespective of whether the operation is to be interleaved or not. In the remainder of this section we present compositional definitions for both chain formation and free indexation. However, despite the additional flexibility, these definitions also have the property of being very different in flavour from the 'original' specifications. As mentioned at the start of the chapter, this difference cannot be made transparent to the grammar writer because the current system does not have the capability to automatically synthesize compositional definitions from abstract specifications. This obviously compromises our goal of separating concerns about control options from principle definitions. Moreover, the correctness of representation becomes an important concern because of the extra 'derivation step' that has to be performed manually by the grammar writer.

In section 3.4.2, we will prove the 'correctness' of a compositional definition for free indexation, in the sense established at the beginning of the chapter. Furthermore, as a side-effect of the proof procedure, we will derive a tight bound for its complexity in terms of the number of referentially distinct index assignments as a function of the number of noun phrases in a sentence. Next, section 3.4.3 will repeat the process for chain formation.

3.4.2 Compositional Free Indexation

Before going on to discuss specific details of the free indexation operation, first let us review the model of indexation adopted by the present system (following (50) in Lasnik & Uriagereka(pg.82)):

(17) a. Coindex via movement.
b. Freely assign indices to NPs in A-positions only at S-structure.
c. Freely assign indices at LF.

We should also add the condition:

(17d) Indices can only be assigned through (17)a–c.[14]

Here, we have assumed that syntactic elements do not inherently possess indices. That is, the only way (apart from free indexation) that a constituent can acquire an index is via movement (condition (17a)). This condition ensures, *inter alia*, that resumptive pronoun constructions such as (19) cannot license parasitic gaps under the theory of functional determination adopted in the current system:[15]

(19) *The article which$_i$ I filed it$_i$ yesterday without reading e_i is over there
Lasnik & Uriagereka (3:46)

Here, the parasitic gap e_i will be (incorrectly) licensed by being functionally determined as a variable, since it is locally bound by the relative operator *which* in an $\overline{\text{A}}$-position. According to Lasnik & Uriagereka, condition (17b) prevents e_i from being licensed as a parasitic gap, since functional determination is assumed to apply at S-structure only. Finally condition (17c) permits elements that are base-generated in $\overline{\text{A}}$-positions to receive the indices that are necessary for interpretation in cases of resumptive pronoun constructions not involving parasitic gaps, such as *the man who I don't believe the claim that anyone saw him* (example [3.2.2:12(i)] in Chomsky [9]). For the remainder of this section, we will be concerned with the implementation of free indexation as defined in condition (17b). In particular, we will address the following three issues:

1. *The combinatorics of free indexation.*

   First, we will investigate the combinatorics of free indexation. By relating the problem to the well-known n-set partitioning problem, we will show that free indexation must produce an exponential number of referentially distinct phrase structures given a structure with n (independent) noun phrases.

---

[14] Of course, the subject of a clause and Infl will also need to be coindexed for agreement. Note that this constraint does not affect how indices are initially assigned because the subject position is an A-position and head movement will ensure that Infl will always have an index. The same reasoning will also apply to other processes such as the linking of non-arguments (not implemented in the current system), as in *there$_i$ is [a man]$_i$ in the room.*

[15] Condition (17b) also prevents overgeneration in cases such as (18):

(18) a. who left?
b. [$_{CP}$ who$_i$ [$_{IP}$ t_i left]]
c. [$_{CP}$ who$_i$ [$_{IP}$ e_i left]]

In the standard account for (18a), *who* is assumed to have been moved from the subject position to COMP, as shown in (18b). However, there is one other possibility that is consistent with a simple model of indexation, as shown in (18c). Here, *who* (base-generated in COMP) binds an empty NP in subject position. However, e_i is not a trace. Furthermore, at S-structure, e_i (as in the case of t_i) will be functionally determined as a variable. However, note that (18c) will not be generated under (17). (*who* does not receive an index and hence cannot bind anything. As a result, e_i will be functionally determined as *PRO*. Hence, 18c will be ruled by condition A of the Binding Theory.)

2. *The enumeration of indexings.*

   Secondly, we will describe a non-deterministic procedure for enumerating the logically possible indexings. This procedure will be compositionally defined over general tree structures. We will use the results of the first part to show that the procedure is correct, both in terms of completeness and non-redundancy, in the senses discussed earlier.

3. *The representation of the enumeration procedure.*

   Finally, we will introduce a general notation for encoding compositional definitions. We will illustrate its use by showing how compositional free indexation may be represented in a perspicuous fashion.

The Combinatorics of Free Indexation

Consider the general problem of partitioning a set of n elements into m *non-empty* (disjoint) subsets. For example, a set of four elements $\{w, x, y, z\}$ can be partitioned into two subsets in the following seven ways:

$$\begin{array}{ll} \{w,x,y\}\{z\} & \{w,x\}\{y,z\} \\ \{w,x,z\}\{y\} & \{w,y\}\{x,z\} \\ \{w,y,z\}\{x\} & \{w,z\}\{x,y\} \\ \{x,y,z\}\{w\} & \end{array}$$

The number of partitions obtained thus is usually represented using the notation $\left\{ {n \atop m} \right\}$ (Knuth [33]). In general, the number of ways of partitioning n elements into m sets is given by the following formula. (See Purdom & Brown [44] for the derivation of (20).)

(20)

$$\left\{ {n+1 \atop m+1} \right\} = \left\{ {n \atop m} \right\} + (m+1)\left\{ {n \atop m+1} \right\}$$

$$\text{for } n, m \geq 0$$

The number of ways of partitioning n elements into zero sets, $\left\{ {n \atop 0} \right\}$, is defined to be zero for $n > 0$ and one when $n = 0$. Similarly, $\left\{ {0 \atop m} \right\}$, the number of ways of partitioning zero elements into m sets is zero for $m > 0$ and one when $m = 0$.

We observe that the problem of free indexation may be expressed as the problem of assigning $1, 2, \ldots, n$ distinct indices to n noun phrases where n is the number of noun phrases in a sentence. Now, the general problem of assigning m distinct indices to n noun phrases is isomorphic to the problem of partitioning n elements into m non-empty disjoint subsets. The correspondence here is that each partitioned subset represents a set of noun phrases with the same index. Hence, the number of indexings for a sentence with n noun phrases is:

(21)
$$\sum_{m=1}^{n} \left\{ {n \atop m} \right\}$$

(The quantity in (21) is commonly known as Bell's Exponential Number B_n; see Berge [5].) The recurrence relation in (20) has the following solution (Abramowitz [3]):

(22)
$$\left\{ {n \atop m} \right\} = \frac{1}{m!} \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} k^n$$

Using (22), we can obtain a finite summation form for the number of indexings:

(23)
$$B_n = \sum_{m=1}^{n} \sum_{k=0}^{m} \frac{(-1)^{m-k}}{(m-k)!\,k!} k^n$$

It can also be shown (Graham [27]) that B_n is asymptotically equal to (24):

(24)
$$\frac{{m_n}^n e^{m_n - n - \frac{1}{2}}}{\sqrt{\ln n}}$$

where the quantity m_n is given by:

(25)
$$m_n \ln m_n = n - \frac{1}{2}$$

That is, (24) is both an upper and lower bound on the number of indexings. To provide some idea of how fast the number of possible indexings increases with the number of noun phrases in a phrase structure, figure 3.12 exhibits the values of B_n for $n = 1, 2, \ldots, 10$.

A Procedure for Enumerating Indexings

For each node X in a tree structure, we will associate a set I_X of indexed noun phrases dominated by node X. Each index set I_X will represent the range of indices assigned within the phrase X. For example, figure 3.13 illustrates the set of indices associated with each constituent in *who*$_i$ *did John*$_j$ *see* t_i.[16]. Basically, the procedure

[16] Ignoring head movement. See section 4.4.1.

NPs	Indexings	NPs	Indexings
1	1	7	877
2	2	8	4140
3	5	9	21147
4	15	10	115975
5	52	11	678570
6	203	12	4123597

Figure 3.12 Table of the first dozen values of B_n.

$$\underset{\{i,j\}}{[_{CP}}\ \underset{\{i\}}{[_{NP}}\ \text{who}_i]\ \underset{\{i,j\}}{[_{\overline{C}}}\ \underset{\{\}}{[_C}\ \text{did}]\ \underset{\{i,j\}}{[_{IP}}\ \underset{\{j\}}{[_{NP}}\ \text{John}_j]\ \underset{\{i\}}{[_{VP}}\ \text{see}\ \underset{\{i\}}{[_{NP}}\ t_i]]]]]$$

Figure 3.13 Index sets for *who did John see?*

operates recursively by calculating the possible index assignments, or *indexings*, for a phrase in terms of the possible indexings for its immediate sub-constituents. There are two cases to consider:

1. Base Case: X is a leaf node.

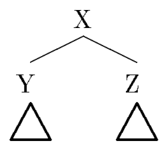
   (a) X is a noun phrase in an A-position. By condition (17b), X should be assigned a (new) index *unless* it already has one (through movement) by condition (17a). In either case, let i be the index. Then, $I_X = \{X_i\}$. For the remainder of the section, it will be convenient to represent a set of indexed NPs using just the indices present. Hence, $I_X = \{i\}$.

   (b) X is not a noun phrase in an A-position. In this case, X should not receive an index, i.e. $I_X = \emptyset$.

2. Compositional Case: X is an interior node.

   (a) Unary branching: Suppose X has a single immediate constituent Y. Since no additional elements are introduced by X, the range of indices in X will be the same as that for Y. In other words, X will inherit the index set of Y: that is, $I_X = I_Y$.

   (b) Binary branching: Suppose X has two immediate constituents Y and Z:



$I_Y = \{i_1, \ldots, i_m\}, m \geq 0$ $\qquad$ $I_Z = \{i_{m+1}, \ldots, i_n\}, n \geq m$

In order to compute the possible index sets of X, given particular index sets for Y and Z, we must allow for the possibility that elements in one phrase, say Y, could be coindexed with elements from the other phrase, Z. This will be accomplished by allowing indices from one set to be (optionally) merged with distinct indices from the other set. Let us call this process *cross-indexing*. For example, the phrases $[_{NP}$ *John* $]_i$ and $[_{VP}$ *likes* $[_{NP}$ *him* $]_j$ $]$ have index sets $\{i\}$ and $\{j\}$, respectively. Free indexation must allow for the possibilities that *John* and *him* could be coindexed or maintain distinct indices. Cross-indexing accounts for this by optionally merging indices i and j. Hence, we obtain:

(26) a. John$_i$ likes him$_i$, *i merged with j*
b. John$_i$ likes him$_j$, *i not merged with j*

More formally, let us assume for the moment that phrase structures are binary branching. Then:

(27) Operation: `crossIndex`: $I_Y \times I_Z$
Let $\bar{I}_Y$ represent those elements of I_Y which are not also members of I_Z, that is, $(I_Y - I_Z)$. Similarly, let $\bar{I}_Z$ be $(I_Z - I_Z)$.
i. If either $\bar{I}_Y$ or $\bar{I}_Z$ are empty sets, then stop.
ii. Let y and z be members of $\bar{I}_Y$ and $\bar{I}_Z$, respectively.
iii. *Either* merge indices y and z *or* do nothing.
iv. Repeat from step (i) with $\bar{I}_Y - \{y\}$ in place of $\bar{I}_Y$.
Replace $\bar{I}_Z$ with $\bar{I}_Z - \{z\}$ if y and z have been merged.

Next, we must find the index set of the larger phrase X. Hence:

(28) Index Set Propagation: $I_X = I_Y \cup I_Z$.

That is, the index set of an aggregate phrase is just the set union of the index sets of its sub-phrases *after* cross-indexation. Returning to our earlier example, *John likes him*, here, (26a) and (26b) will have index sets $\{i\}$ and $\{i, j\}$, respectively.

Note that the above procedure must have an additional step in cases where the larger phrase itself may be indexed; for instance, as in $[_{NP_i}$

$[NP_j$ *John's* $]$ *mother* $]$. In such cases, the extra step will be simply to merge the singleton set consisting of the index of the larger phrase with the result of cross-indexing in the first step. In other words, a structure of the form $[NP_i\ [NP_j \ldots \alpha \ldots\ ] \ldots \beta \ldots\ ]$ can always be handled as $[P_1\ [NP_i\ ]\ [P_2\ [NP_j \ldots \alpha \ldots\ ] \ldots \beta \ldots\ ]]$.

(c) n-ary branching: The algorithm generalizes to n-ary branching, for $n \geq 3$, using iteration. For example, a ternary branching structure such as $[_P\ X\ Y\ Z]$ can be handled in the same way as $[P\ X\ [P'\ Y\ Z\ ]]$.

The nondeterminism in step (iii) of cross-indexing will generate all and only all the possible indexings. We will show this in two parts. First, we will argue that the above procedure cannot generate duplicate indexings. As shown in the previous section, the combinatorics of free-indexing indicates that there are only B_n possible indexings. Secondly, we will demonstrate that the algorithm generates exactly that number of indexings. If the algorithm satisfies both of these conditions, then we have shown that it generates all the possible indexings exactly once:

1. Consider the definition of cross-indexing. Here, $\bar{I}_X$ represents those indices in X that do not appear in Y. (Similarly for $\bar{I}_Y$.) Note also that whenever two indices are merged in step (ii), they are 'removed' from $\bar{I}_X$ and $\bar{I}_Y$ before the next iteration. Thus, in each iteration, x and y must be 'new' indices that have not been merged with each other in a previous iteration. By induction on tree structures, it is easy to see that two distinct indices cannot be merged with each other more than once. Hence, the algorithm cannot generate duplicate indexings.

2. We now demonstrate why the algorithm generates exactly the correct number of indexings by means of a simple example. Without loss of generality, consider the right-branching phrase scheme shown in figure 3.14.

   Now consider the decision tree shown in figure 3.15 for computing the possible indexings of the right-branching tree in a bottom-up fashion.

   Each node in the tree represents the index set of the combined phrase depending on whether the noun phrase at the same level is cross-indexed or not. For example, $\{i\}$ and $\{i, j\}$ on the level corresponding to NP_j are the two possible index sets for the phrase P_{ij}. The path from the root to an index set contains arcs indicating what choices (either to coindex or to leave free) must have been made in order to build that index set. Next, let us just consider the cardinality of the index sets in the decision tree, and expand the tree one more level (for NP_l) as shown in figure 3.16.

   Informally speaking, observe that each decision tree node of cardinality i 'generates' i child nodes of cardinality i plus one child node of cardinality

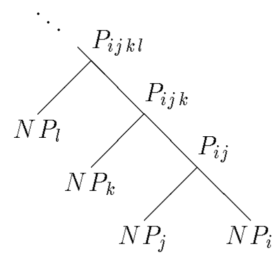


Figure 3.14 Right-branching tree.

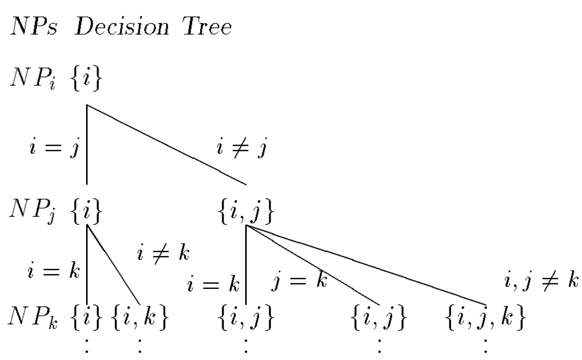


Figure 3.15 Decision tree.

$i+1$. Thus, at any given level, if the number of nodes of cardinality m is c_m, and the number of nodes of cardinality $m-1$ is c_{m-1}, then at the next level down, there will be $mc_m + c_{m-1}$ nodes of cardinality m. Let $c(n,m)$ denote the number of nodes at level n with cardinality m. Let the top level of the decision tree be level 1. Then:

(29)
$$c(n+1, m+1) = c(n, m) + (m+1)c(n, m+1)$$

Observe that this recurrence relation has the same form as equation (20). Hence the algorithm generates exactly the same number of indexings as de-

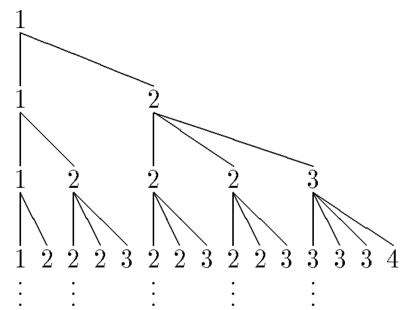


Figure 3.16 Condensed decision tree.

manded by combinatorial analysis.

The Representation of Free Indexation

Figure 3.17 contains the definition of free indexation. The first form is a direct instantiation of the compositional definition. The first line states that `freeIndexation` maintains a list of indexed noun phrases for each node of a phrase structure tree. The rest of the definition is divided into two parts:

1. The first part encodes the case for binary branching. Here, `CF` names a constituent with two immediate sub-constituents `X` and `Y`. (The next three lines just names the indexed NPs associated with each of these three constituents.) It specifies that the indexed NPs for `CF` are defined by cross-indexing the corresponding sets for `X` and `Y` and then taking the union of the two sets, as required by the compositional definition. The system will automatically generalize the procedure to unary and ternary branching (and above) by iteration (using the method described on page 96). In any case, no matter how many branches there are, the additional step to take care of the case when `CF` itself may be indexed is handled by the final call to `freelyIndexIfNP/3`.

2. The second part encodes the base case, i.e. when there is a noun phrase in an A-position.

The remaining predicates are a straightforward encoding of the `crossIndex` and Index Set Propagation operations defined in (27) and (28).

[16] To make the boundary cases match, just define $c(0,0)$ to be 1, and let $c(0,m) = 0$ and $c(n,0) = 0$ for $m > 0$ and $n > 0$, respectively.

```
:- freeIndexation produces [INPs]
        compositional_cases_on CF where   % Compositional case:
          CF with_constituents Y and Z st
                Y produces [IY],
                Z produces [IZ],
                CF produces [ICF]
        then crossIndex(IY,IZ),
             unionWrtI(IY,IZ,ICF)
        finally freelyIndexIfNP(CF,ICF,INPs)
      else
        npInApos(CF) then index(CF,_), INPs=[CF]. % Base case.

npInApos(CF) :- cat(CF,np), CF has_feature apos.

crossIndex(IX,IY) :-
        diffWrtI(IX,IY,IbX),              % $\bar{I}_X = I_X - I_Y$
        diffWrtI(IY,IX,IbY),              % $\bar{I}_Y = I_Y - I_X$
        freelyCoindex(IbX,IbY).

% freelyCoindex: $I_X \times I_Y$
freelyCoindex([],_).                      % Do nothing,  $I_1 = \emptyset$
freelyCoindex([NP|IX],IY) :-
        optCoindex(NP,IY,IYp),
        freelyCoindex(IX,IYp).

% optCoindex: $NP \times I \mapsto I'$
optCoindex(_,I,I).                        % Do nothing, or ...
optCoindex(NP1,I,Ip) :-                   % Select an NP in I to coindex with.
        select(NP2,I,Ip),                 % $NP_2 \in I, I' = I - \{NP_2\}$
        agree(NP1,NP2),
        coindex(NP1,NP2).

coindex(X,Y) :- index(X,I), index(Y,I).

index(X,I) :-
        X has_feature index(I)            % Is X already indexed?
        -> true
        ;  addFeature(index(I),X).        % If not, give it one.

freelyIndexIfNP(NP,I,Ip) :-
        npInApos(NP)
        -> index(NP,_),                   % Give the NP an index feature.
           crossIndex([NP],I),
           unionWrtI([NP],I,Ip)
        ;  Ip = I.                        % Do nothing if not an NP
```

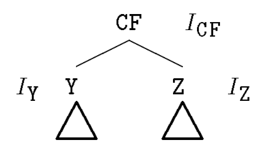


Figure 3.17 Representation of the free indexation algorithm.

This completes our formalization of free indexation. In the next section, we will turn to describe how chain formation, a similar but more complex procedure, can also be encoded using structural induction.

3.4.3 Compositional Chain Formation

In the following discussion, we will restrict the domain of application of Move-α to instances of syntactic movement, i.e. between D- and S-structure, of NPs only. Furthermore, we will assume that syntactic movement of NPs involves substitution only.[17] Movement of *wh*-adverbs such as *why* and *how* are handled by the same mechanism; but for simplicity of exposition, this will not be shown here. However, LF-movement of quantifiers, *wh*-elements, and operators will be covered by a separate parser operation. As chapter 4 will describe, cases of head movement are also handled by a different mechanism. Finally, we will also assume that movement must leave traces.[18]

First, let us formally define the notion of a movement chain:

(30) A (non-trivial) *movement chain* is a sequence of elements $\langle E_1, \ldots, E_n \rangle$, $n \geq 2$, such that E_n occupies the position occupied by E_1 before movement, and that each subsequence $\langle E_i, E_{i+1} \rangle$, for $i \geq 0$, represents a single instance of movement of E_1 from the position occupied by E_{i+1} to that occupied by E_i. Elements E_2 to E_n are *traces*. Furthermore, all elements of a chain will be coindexed. A *trivial chain* involving no movement will simply be a singleton sequence $\langle E_1 \rangle$.

(Unless otherwise stated, hereafter, we will use the term 'chain' to denote a 'non-trivial movement chain'.) The chain formation algorithm is based on the following conditions:

(31) a. [*NP*-e] optionally participates in a chain.
b. An overt NP optionally heads a chain.

[17] In the current system, a mechanism for adjunction of NPs was found to be only necessary for cases of multiple-*wh*-movement, e.g. as in *who will read what?* Hence, the current chain formation operation for syntactic movement lacks this capability. However, since such cases do occur at LF, the operation that implements LF-movement does indeed incorporate a mechanism for adjunction. (See Lasnik & Saito [34] for the proposal that adjunction to Comp is universally prohibited in the syntax, and only possible in LF.)

[18] However, a trace may be marked as being invisible to the ECP. For example, consider the (ill-formed) S-structure *who*$_i$ *do you think* [*CP* t'_i [$\bar{C}$ *that* [*IP* t_i *saw Bill*]]]. Here, we have assumed that *who* has been moved from the embedded subject position occupied by t_i in two steps (in order not to violate Subjacency). According to Lasnik & Uriagereka, this structure violates the ECP because t_i is not properly governed. For this to happen, it must be the case that the intermediate trace t'_i should not function as a proper governor. One way to guarantee this is to say that movement will optionally leave a trace. Since we have assumed that movement must leave traces, the same effect can be obtained by marking t'_i as being invisible to the ECP.