ROBERT C. BERWICK

PRINCIPLES OF PRINCIPLE-BASED PARSING

1. INTRODUCTION: PRINCIPLES AND PARSING

This book chronicles the first stirrings of a revolution in the study of natural language processing, language variation, and psycholinguistics—what some have called *principle-based parsing*.

To begin, perhaps it is simplest to say what principle-based parsing is *not*. A traditional view of grammar description, and so parsing, relies on many thousands of individual, language-particular, and construction-specific rules. This is true whether the rules are used by a context-free parser, an augmented transition network, a deterministic LR-type parser like the Marcus parser, or a logic grammar of almost any stripe.

Whatever the parsing method used, the key point is that rule-based systems attempt to spell out surface word order phrase patterns such as passive or dative, pattern by pattern and language by language. For example, a typical rule-based system will encode the format of a passive sentence such as *Mary was kissed by John* in a particular *if-then* format that includes the details of English-particular morphology (the *be* form followed by a verb with an *en* ending) plus the absence of a logical object in its expected position, along with a particular left-to-right ordering of phrases. Note that this is as true of the context-free rule that might be written as S→NP *be* V *ed+passive* as it is of the *if-then* grammar rules of Marcus' (1980) system, or the augmented transition network rules handling passive as described in Bates (1978) that use register assignments and arc ordering. Each encodes the same construction-based information in roughly the same way. Further, the same view pervades language acquisition systems grounded on rules, like that of Berwick (1985): acquiring a grammar amounts to the piecemeal acquisition of many construction-specific rules.

Principle-based language analysis replaces this standard paradigm with another world view: rules are covered by a much smaller set of *principles* that reconstitute the vocabulary of grammatical theory in such a way that constructions like passive *follow* from the deductive in-

R. C. Berwick et al. (eds.),
Principle-Based Parsing: Computation and Psycholinguistics, 1–37.
© 1991 *Kluwer Academic Publishers. Printed in the Netherlands.*

teractions of a relatively small set of primitives. On this view there is no 'passive rule'. Passive constructions result from the interactions of deeper morphological and syntactic operations that 'bubble to the surface' as the sentences we happen to describe as active or passive. The goal of the rest of this book is to show how this principle-based approach leads to exciting new models for parsing and language translation, different psycholinguistic avenues, and changes in our view of language acquisition.

Figure 1 illustrates the fundamental difference between rule- and principle-based approaches. The top half of the figure shows a conventional rule-based approach. Each sentence type is described by a different rule. The bottom half of the figure shows a principle-based approach. Intuitively, note that one can get the multiplicative effect of $n_1 \times n_2 \times \ldots$ rules by the interaction of $n_1 + n_2 + \ldots$ principles. A dozen principles, each with 2 or 3 degrees of freedom or *parameters* can thus encode many thousands of rules. This approach has therefore been dubbed *principles-and-parameters* theory.[1]

Let us see how principles can replace rules in our passive example. One general principle says that verb phrases in sentences must either *begin* with a verb in some languages, or *end* with a verb in others (those are the degrees of freedom or parameterization in this particular principle). This basic description of the tree shapes in a language, dubbed $\overline{\text{X}}$ *theory*, gives us part of the variation between languages like English and Spanish on the one hand, and languages like German and Japanese on the other. In English, the verb must come first, with the object after; in Japanese, the reverse. A second principle, called the *Case filter*, says that all pronounced or *lexical* noun phrases like *ice-cream* must receive Case, where Case, roughly speaking, is an abstract version of the Latinate system that gives objective Case to objects, oblique Case to objects of prepositions, nominative Case to sentence subjects, and so forth. Case is assigned either from an active verb like *ate* or an auxiliary verb like *was*; the adjectival verb *eaten* does not assign case. A third principle, called the *Theta-criterion* or θ-criterion, insists that every verb must discharge its *Thematic arguments* and every noun phrase must receive a thematic role, completing a description or *thematic structure* representation of 'who did what to whom'. So for example, *eat* can mean to eat *something* (the 'Affected Object' in earlier parlance) and discharges a thematic role of a noun phrase, while *persuade* could have either a noun phrase or a noun phrase and a proposition as its thematic

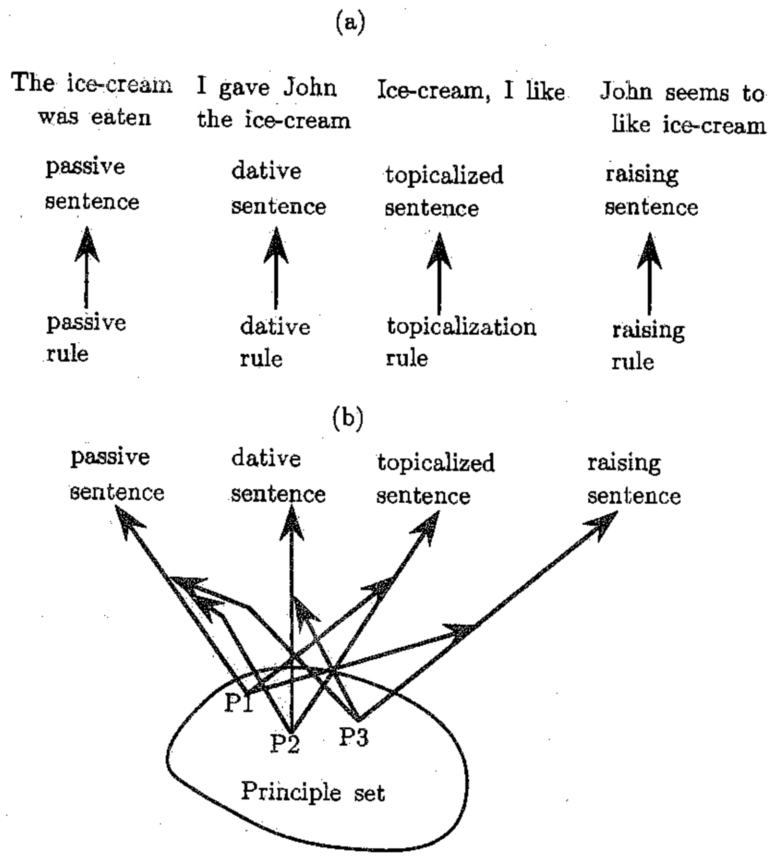

Figure 1: The difference between rule-based and principle-based systems is shown schematically in this figure. The top half (a) illustrates a rule-based system, with one rule per construction like passive or dative. The bottom half of the figure (b) illustrates a principle-based system. Sentence types are derived from a much smaller basis set of more fundamental principles that deductively interact to yield the effect of constructions like passive.

roles (*persuade John* or *persuade John that Bill is a fool.* A fourth principle, *Movement* (or *Move-α*), lets one move any phrase α to any available 'landing site'. A fifth general principle is *Trace theory*: any moved phrase leaves behind a phonologically empty category, a *trace*, coindexed with the original phrase, and bearing a certain configurational relationship with the moved phrase. A sixth constraint, *Binding theory*, determines when one noun phrase (a trace or not) can be coidentified with another, as in *John thinks that he likes ice-cream* where *John* and *he* can refer to the same person. And so on; we will see details of these principles in the chapters to come. A useful taxonomy for principles developed by Fong in this volume is to brand them either as *filters*, ruling out possible structures fed into them, or *generators*, admitting possible new structures. Thus the Case filter and Theta-criterion behave like filters because they are gates permitting only certain structures to pass, while $\overline{\text{X}}$ theory and movement act as generators, because they output at least as many structures as they receive.

Seeing how these principles operate in concert gives us a chance to understand how the passive conspiracy works and at the same time review the standard model of phrase structure assumed by all the authors of this book, as shown in figure 2. Conceptually (but not computationally!) the principles fit together on a four-fold scaffolding tied together in an inverted 'Y': a representation of a sentence's underlying thematic structure or *D-structure*; a sentence's surface structure or *S-structure*, roughly, an augmented parse tree; a phonetic form or *PF*; and a sentence's *Logical form*, or *LF*. Returning to our passive example sentence, we begin with a representation of its D-structure. Our goal is to show how this D-structure can surface as a 'passive' form *the ice-cream was eaten* without ever making reference to an explicit 'passive rule'.

$[_{S}\ [_{NP}\ \triangledown]\ [_{VP}\ \text{was eaten}\ [_{NP}\ \text{the ice-cream}]]]$

Here $\triangledown$ is an empty position, a legitimate landing site for a noun phrase (NP), and the basic tree shape, indicated by the usual bracketing, is set by $\overline{\text{X}}$ constraints. Note that Thematic structure makes explicit reference to the properties of lexical items, in this case that *eat* requires a thing that gets eaten.

As figure 2 shows, D-structure is related to S-structure by the Move-α relation plus some of the other constraints mentioned earlier. S-structure then serves as a springboard for two 'interface systems': on the left it maps via phonological rules to an interface with the outside world, namely, a spoken or phonetic form (PF)—this is the sentence that we actually see (more properly hear). On the right S-structure maps to an interface for other cognitive systems of an inside mental world, a representation of quantificational and noun phrase–noun phrase relationships, or Logical form; this relation is also mediated by a general relation of phrase movement, suitably constrained. (*Warning*: though this sketches a *logical* picture of the principle-based components, it does *not* say *how* these representations are to be computed, piece-by-piece or even if at all, or in what order. It is not even clear that we need all these representations, computationally speaking. *That* is a serious topic that all the authors must tackle.)

How then does the passive cabal get off the ground? *Ice-cream* can-

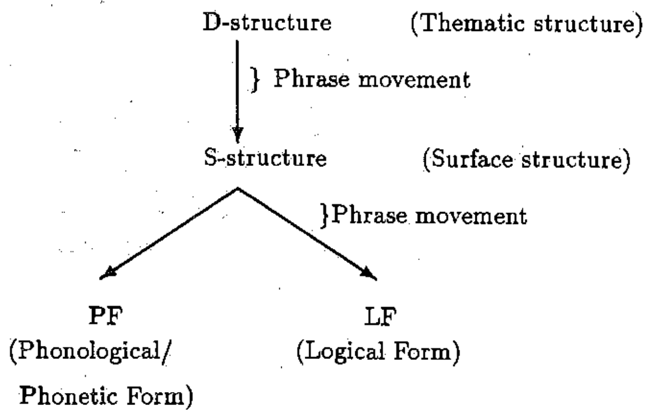

Figure 2: The conventional inverted 'Y' model of phrase structure used in the principle-and-parameters theories described in this book. It includes four levels of representation: (1) D-structure, or the level of *Thematic structure*, essentially 'who did what to whom'; (2) S-structure, or *Surface structure*, related to D-structure by the displacement of NPs from their D-structure positions; (3) PF, the interface to speech, related to S-structure by a phonological mapping; and (4) LF, *Logical form*, an interface to other cognitive systems such as inferential processes, representing quantifier and NP-NP relations and related to S-structure again by the displacement of certain phrases (so-called *LF movement*). The lexicon, not shown, is a source of thematic and phonological information.

not receive Case from an adjectival form like *eaten*. (We know that *eaten* is adjectival by the way that it mimics the analogous *John was sad*.) So *ice-cream* must move, and can move, to the available empty landing site at the front. Now *ice-cream* receives nominative Case from *was*, meeting the Case filter. Further, *eat* can now discharge its thematic role of Affected Object, and all visible noun phrases in the sentence receive a distinct thematic role, because the trace left behind by movement gets the discharged thematic role just as if it were the object of *eat*, and the trace's link with *ice-cream* ensures that *ice-cream* receives that thematic role as well. Taken together, there is no explicit passive rule. It is implicit in the inference from the principles. Importantly, the deductive chain from principles to surface form is very much longer than the link from the usual if-then rules to surface form. It is very much like reasoning from first principles, and it gives us the first hint of the computational thickets we must cut through to gain from a principle-based approach, as we shall see below.

If you concluded this was a lot of pain for little gain, then you were right. In just the same way, making *one* chemical compound out of a vast stock of molecules and bonding principles seems like overkill. Still, we can learn much from this small example, namely what the principle-based enterprise is all about.

The first lesson is that the *same* small set of principles can be recombined, over and over in different ways, yielding the entire array of surface sentences, and, by varying the parameters, different dialects and languages.

Second, note that the principles themselves are highly *abstract* and *heterogeneous*, often stated as a set of *declarative constraints*, unlike a more uniform representation like a set of context-free rules in a derivational framework. That is, we can imagine the set of well-formed sentences as those that dodge their way through a gauntlet of quite different representational constraints, rather than as the generative derivation from some S starting symbol. To this extent, the principle-and-parameters approach casts off the old garb of formal language theory with its emphasis on strings and languages. To take but one example, the notion of a *language* with its *grammatical* and *ungrammatical* strings becomes nearly meaningless: instead, there are simply structures that pass more or less of the way through the gauntlet. (To be sure, as note 1 points out, many recent linguistic theories have adopted a similar declarative framework. Even so, the declarative character of

the principles-and-parameters approach itself is hotly debated, as noted in the chapters by Correa and Johnson.) In addition, the principles themselves *apply at different places*. For instance, movement of phrases and constraints on movement apply at the D-structure and S-structure interfaces, while the Case filter applies at S-structure itself (on some accounts).

Third, the principles-and-parameters approach stresses the importance of the *lexicon* (for example, as the source of thematic role constraints and possibly all language-particular variation), leading quite naturally to a focus on cross-linguistic variation, universal constraints, and language acquisition. On this view, there is one basic 'chassis' for all human grammars, with parametric variations setting the range of possible variations. For instance, Japanese differs from English in part because Japanese $\overline{\text{X}}$ constraints place phrasal heads—verbs in verb phrases, prepositions in prepositional phrases—*last* rather than *first*. Learning Japanese rather than English, then, is partly a matter of setting this $\overline{\text{X}}$ parameter to *head final* rather than *head first* (see Kazman's chapter for more on this). Finally, the highly modular and abstract character of principles leads directly to a heterogeneous view of language use, and an emphasis on how linguistic principles actually enter into human cognition.

Each of the chapters focuses on one or more of these major issues. Abney, Correa, Epstein, Fong, Johnson, and Stabler stress the major architectural and computational design features of principle-based parsers, including the issues of computational efficiency and program control flow in parsing. Dorr, Kashket, and Kazman treat the topic of cross-linguistic variation and the lexicon as it bears on machine translation; the parsing of languages as diverse as English and its near opposite, the Australian aboriginal language Warlpiri; and language acquisition. Thankfully, these parsers are not just speculative fancies; each author has a computer-implemented parsing model to show and tell. Finally, completing the circle, Abney, Gorrell, Kurtzman, Crawford, Nychis-Florence, Pritchett, and, to some extent, Stabler, take up specific psycholinguistic issues that connect facts about human sentence processing to the architectural modularity of principle-based systems. (A fourth area, language change, deserves a place as well but regrettably there are few computational studies of diachronic syntax—particularly ironic given Chomsky's original inspiration—1955, 1975—from language change.)

2. THE PRINCIPAL PROBLEM WITH PRINCIPLES

So far, all this discussion has been mere advertisement. The principle-based approach asks as many questions as it answers. Can it really work at all? To see that principle-based parsing *can* work, we shall first sketch in broadbrush the chief problems with principle-based parsing. We then review how the authors have solved these problems, showing how their solution fit under two unifying umbrellas: the division of parsing algorithms into control plus data structures (section 3); and parsing as search (section 4).

To begin, note that the picture in figure 2 merely sets out *what* is to be computed, in the sense of Marr (1982), not *how* the computation should proceed. Yet the gap between simply stating constraints and solving them can be huge; as noted by Abelson and Sussman (1985), it's one thing to define `square-root(x)` as the set of all y's such that $y^2 = x$, and quite another matter to write a square root algorithm. So it is by no means clear that principle-based parsing is possible at all—perhaps it is just as difficult as with earlier transformational theories.

Second, two related computational difficulties lie at the heart of principle-based parsing: *overgeneration* and *slow parsing*. The first problem plagues such systems precisely because the constraints of a principle-based system are heterogeneous and parceled out to many different components that we run into a computational brick wall at the start. By design, any single principle will not constrain the ultimate sentence structures very much. Thus we already know before we begin that a key problem faced in principle-based analysis is overgeneration: too many illicit structures will be produced that never mate with the input sentence. For instance, even a *simple* $\overline{\text{X}}$ theory, without recursion, can generate thousands of possible structures—just imagine three layers of tree structure with left- and righthand sides, each filled with any one of, say, 10 possibly different lexical categories. Then, each of these several thousand structures branches out a half-dozen or more new possible S-structures each, via Move-α. This is not just idle speculation, but hard-won computational and psycholinguistic experience. For instance, Fong has confirmed that there are tens of thousands of $\overline{\text{X}}$ possibilities for even a simple sentence, a result seconded by Kolb and Thiersch (1988) for German. Coping with overgeneration is therefore an important theme that runs throughout the parsing analyses in this book.

Slow parsing is the natural heir of overgeneration. But there is more to poor performance than that. As the bottom half of figure 1.1 shows, the other reason is that the deductive chains from principles to surface forms are now much longer than before. Modularity, then, is both a blessing and a curse. Simply enumerating possibilities will be slow because there are too many of them and a long way between sentence and structure. If one has to figure out from first principles how to open the front door, then getting around can become very slow indeed.

This raises yet another immediate question. Why then use principles at all? Why not simply 'compile out' small *lemmas* that store the repeated deductive chains, like the piece of reasoning about passive that we carried out earlier? But wouldn't this lemma then be a rule? Would we lose what little we've gained? One can see then that every *architectural* question about the relation between linguistic theory and the data structures of an implemented theory looms behind the overgeneration and parsing time problems. We won't answer these vital questions completely here—that's what the rest of the book is about—but we can at least see right away the centrality of overgeneration and slow parsing to the whole enterprise.

The general outline of what to do seems clear enough: starting from an input sentence, an orthographic representation of PF, we must 'invert the Y diagram' and recover at least the S-structure and the information in D-structure and LF, if not those structures themselves. Since all the filters and generators are grounded in phrase structure definitions—the Case filter must look at a particular *configuration* of tree structure, such as a verb next to an NP—we must somehow bootstrap ourselves from the input sentence and start building phrases. This is by no means easy. One cannot just reverse the arrows in the figure because the mappings are not obviously one-to-one and the constraints one needs aren't always immediately at hand (but see below on this point).

For instance, given the orthographic PF form *the ice-cream was eaten*, in order to build an S-structure it makes sense to bring thematic constraints from the lexicon into play, namely that *eat* might demand something eaten, because after all S-structure is in part a product of thematic constraint. But this information is hidden from PF's direct view back in D-structure. At the same time we have the problem of guessing that there is a phonologically null *trace* after *was eaten* that does not even show up in the input sentence. Similarly, consider the $\overline{\text{X}}$ component. Typical English structures would include forms for *John ate*, *John* *ate the ice-cream*, *John thought ice-cream was wonderful*, and so on. But the bare $\overline{X}$ component says simply that $\overline{X} \rightarrow X$ {*Complements*}, where *Complements* can be nothing, or an NP; or a PP; or a CP (a propositional complement); or NP CP, and so on. However, some of these will be impossible depending on the verb. We can't have an NP as a complement of *think* as in *John thought ice-cream* (of course there is an elliptical construction that looks like this but is simply a truncation of a full propositional complement). If this is so, then expanding the skeletal $\overline{X}$ template out to a full set of complements will waste time and computer money because all expansions will be tested even though we know in advance that some are not possible. For instance, if the sentence were *John ate*, then an overzealous system might just invent an empty element after *ate*, to satisfy the NP complement possibility. But it is far better to consult the lexical entry for the allowable complements. We need to factor in information from the lexicon into the system at the PF–S-structure interface—even though this is not the logical arrangement of the 'Y' diagram where the lexicon hooks into the D-structure alone.

Thus as it stands the inverted 'Y' model does not seem well suited for parsing. Historians of transformational grammar may recall that this is precisely the same paradox that faced a much earlier generation of transformational grammar parsing pioneers (early systems at MIT and IBM, described in Petrick, 1965; and those at MITRE, reviewed by Zwicky *et al.*, 1965): transformations only apply to some structure, so we must first conjure up some structure from the sentence to start with, perhaps by employing a distinct *covering grammar*—a modified S-structure that can be used to start the jigsaw puzzle—and by carefully building inverse transformations that can be coupled with their forward generative counterparts. As we shall see, some of their solutions can be used in modern dress, but in addition we can do better now because we know more about control structures and computation, in particular trade-offs in search techniques that can model the covering grammar idea and much more besides. What is more, the theory has changed: there are no ordered, language-particular rules; traces and moved NPs can appear only in certain positions, not anywhere, so deletion is restricted. All this helps.

Let us then turn to our two umbrella views of the authors' solutions to overgeneration and slow parsing, beginning with algorithms viewed as control plus data structures.

3. MAKING PRINCIPLE-BASED PARSING WORK: CONTROL AND DATA STRUCTURES

Figure 3 summarizes where all the authors stand on these matters. In what follows we shall refer to that figure and table I for a bird's eye view of the entire book. We shall now explore each branch of this taxonomy.

For those who have implemented parsers—Abney, Correa, Dorr, Epstein, Fong, Johnson, Kashket, Kazman, and, at the borderline with psycholinguistics, Stabler—there are two chief clamps to place on overgeneration, following the familiar division of computer algorithms into *control structures* (*how* something is computed) plus *data structures* (*what* is computed). Half of these parsers cut down on overgeneration by adopting both flexible control structures and data structures, while the other half do their work with flexible data structures alone, leaving the control structure fixed. Those who work at principle-based psycholinguistics—Gorrell, Kurtzman *et al.*, and Pritchett—carve up their hypotheses in the same way, only their game is complementary to the computer investigations. They aim at confirming or disconfirming 'the facts of the matter' about the use of principles. Do human sentence processors *actually* interweave principles and constraints—Do people actually use the Case filter or access lexical or thematic information when parsing? If so, when?

3.1. *Flexible Control Structures*

Of those using flexible control structures—Epstein, Fong, Johnson, and Stabler—Epstein alone commits to Lisp-based, hand-built exploration of six different control regimes, based on well-known heuristics that attempt to do as much filtering as early as possible, or as cheaply as possible, or by building as little structure as possible. (We shall take a closer look at these control regimes in section 4 on search below.) His domain is that of quantifier scope interpretation, and the tradeoff between the modularity of four simple principles for fixing scope and computational efficiency, as in *Every professor expects several students to read many books*, which has 70 possible candidate LFs, but just a few viable ones. By carefully attending to principle ordering, and banking on the local and compositional nature of scope, Epstein achieves remarkably good parsing time, often reducing many thousands of possible LFs to just a handful.

The remaining flexible control structure devotees—Fong, Johnson,

Table I

A summary of what each author has proposed computationally for their parser, in terms of flexible and fixed control structures.

	Flexible Control Structure
Epstein	Heuristic ordering strategies & others for computing logical form
Fong	Manual & automatic ordering of principle modules for optimal speed Covering grammar for S-structure
Johnson	5 different control structures, incl. coroutining, ordering, & combining principles
Stabler	Clause selection ordering for online semantic interpretation
	Fixed Control Structure
Correa	Attribute grammar to enforce local and long-distance constraints; Covering S-structure grammar that includes moved phrases
Dorr	Coroutined Earley parser and principle-based constraints Covering S-structure grammar
Kashket	Bottom-up multi-pass parser projecting words into $\overline{X}$ structures separating linear order & hierarchical structure for free word order languages
Abney	Bottom-up multi-pass parser separating prosodic structure; small phrase 'chunks' (projected bottom-up from $\overline{X}$ structures); and full S-structure
Kazman	Coroutined bottom-up $\overline{X}$ parser plus principle-based constraints; parameterized for language acquisition
	Psycholinguistics: Fixed Control Structure
Gorrell	Verb subcategory constraints applied as soon as possible
Pritchett	Thematic (theta) theory applied as soon as possible
Kurtzman	Verb constraints applied as soon as possible

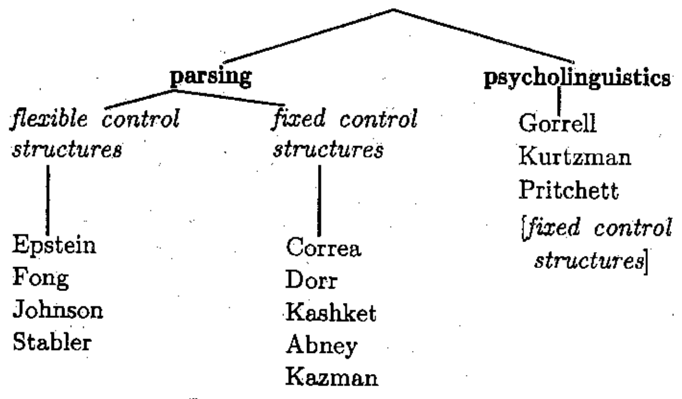

Figure 3: A tree diagram showing the computational territory the authors cover. The basic breakdown is into parsing and psycholinguistics. The implemented parsers fall into two camps: those with fixed control structures, and those separating control structure from data structure. The psycholinguistic studies all implicitly adopt fixed control structures.

and Stabler—have explicitly chosen an implementation language that allows one to make a clean separation of the parsing algorithm into control plus data structures—Prolog or some variant of first-order logic. This is a big advantage, because it allows one to separately abstract away from control structure issues. For example, Johnson's chapter amounts to a skillful set of finger exercises that explore an entire range of possible control structures, embodied in a series of five parsers, including some that order principles automatically, some that coroutine structure building and principle application, and some that avoid explicit construction of D-structure or S-structure entirely, building only Logical form. The beauty of using logic here is that the new parsers are *provably* equivalent to Johnson's original. It is very difficult to do this by hand, as Correa's attempt shows. We sketch these parsers further in section 4 below on search.

Fong and Stabler drive home the flexibility point, demonstrating how data and control structure partitioning yields tremendous power: Fong shows that one can reorder principles to construct many *hundreds* of different parsers within a single theory, even parsers that dynamically adapt their structure (the order in which principles are applied) according to what sentence types have been parsed before. This is an intriguing result for psycholinguistics, since it allows for individual variation in parsers from person to person, within a certain parametric envelope. Stabler establishes that ordering principles lets semantic interpretation proceed online, because one can start interpretation before an entire S-structure is built, thanks to a control regime that differentially queues at the front those statements that deal with semantic interpretation. In *the ice-cream was eaten* we can get an interpretation for *the ice-cream* before we arrive at *eaten*.

Logic also admits relatively transparent renderings of the English that linguists use in theories. There is a problem here—the usual one of getting from English to logic—but that's a problem for everybody. Logic still has all the virtues of directness: readability; easy modification in the face of theories in flux; verifiability of faithfulness to the original text. It is even the *historical* wellspring of generative grammar: after all, Chomsky's original *Morphophonemics of Modern Hebrew* (1953) that started the modern generative enterprise in motion consciously borrowed the notion of an axiom system because that was the only machinery then *au courant* in which to run a recursive syntax. As Johnson's paper argues, deductive inference is still perhaps the clearest way to to think about how to 'use' knowledge of language. In a certain sense, it even seems straightforward. The terms in the definitions like the one above have a suggestive logical ring to them, and even include informal quantifiers like *every*; terms like *lexical NP* can be predicates, and so forth. In this way one is led to first-order logic or Horn clause logic implementation (Prolog) as a natural first choice for implementation, and there have been several such principle-based parsers written besides those described by the authors of this volume who have built Prolog implementations of principle-based theories. (see Sharp, 1985; Kolb and Thiersch, 1988). Parsing amounts to using a theorem prover to search through the space of possible satisfying representations to find a parse, a metaphor that we'll explore in detail just below.

How then can a theory written in English be mapped into logic? Here for example, following Fong (1991 forthcoming) is a typical statement of the Case filter: "At S-structure, every lexical NP needs Case" (Lasnik and Uriagereka, 1988, p. 20). How are we to implement this statement as part of a parsing program? Those who live by the Horn clause must also die by it: We must actually translate the English connectives such

as *needs* or *every* and the required feature checking (the property of being *lexical* on NPs), into logical formulas. But in fact this can readily be done. Fong's system (surveyed in his chapter but fully detailed in Fong, 1991 forthcoming) states principles purely declaratively, as in his version of the Case filter:

```
:- 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).
```

This says, close in spirit to the original English, that the Case filter is met if, in all configurations (CF) where there is a lexical NP, then make sure that Case has been assigned to that NP configuration. The Prolog has been augmented a bit with more abstract macros such as `in_all_configurations` that render the code closer to the English. (Following Fong we read the second clause `lexicalNP(NP)` as stating that a lexical NP is one that has the category feature *np* and is *not* an empty category. As is conventional, \+ means negation-as-failure—if we can't prove something to be an *ec* or empty category then it is not one. Finally, the comma is a conjunction.)

The point is that the person who writes the Case filter statement need not be concerned whether this statement is implemented by a tree-walking procedure that climbs over all the nodes one by one, or by some other method that combines nodes together first (for more on this, see section 4).

It is interesting to compare Fong's encoding of the Case filter with Johnson's and Correa's. They are very much alike. Johnson, for instance, says, "The Case filter as formulated in PAD [his parser] applies recursively throughout the S-structure, associating each node with one of the three atomic values *assigner*, *receiver*, or *null*" (p. 49). He also observes (p. 62) that "this association of constituents with 'properties' described here is reminiscent of the way a tree automaton associates tree nodes with automaton 'states' ". Further, it's the locality of the constraint that makes this possible—it applies only to a restricted configuration of a verb and verb phrase adjacent to a noun phrase, for example.

Correa has done precisely the same thing in other garb: he uses *attribute grammars* where the grammar has built into it an augmented S-structure that includes the effects of the lexicon plus local movement. An attribute grammar is an ordinary context-free grammar plus attribute information, which we can think of as separate memory registers attached to the different parse tree nodes. The attributes are 'typed'—that is, each has a certain *domain* defined in advance that can hold only certain values (*e.g.*, the *Case* type might hold possible Case assignment values like nominative or Accusative) We assign values to the attributes by means of rules attached to the context-free expansions; this information can be passed either from mother and sisters to daughter (an *inherited* attribute) or from daughter to mother (a *synthesized* attribute). Thus attributes can pass information down and from tree substructures below so that local constraints like the Case filter can be applied, as Correa notes: "the attribute *Case* is associated with NP, the categories of potential Case assigners, such as V[erb] and P[reposition]... The domain of the attribute is [nominative, accusative, dative, ... nil], which includes the special value *nil*" (p. 96). Here, the Case *assigner* and *assignee* are determined by the attribute rules themselves, where the attribute values are in brackets:

S→ NP VP

attribution:

NP[*Case*] ← if VP[*tensed*] then *Nominative*, else VP[*Case*]

Case filter:

if ¬NP[*empty*] (an empty category) then NP[*Case*]≠ *nil*

Not only is this almost exactly Fong's definition, but Correa even implicitly hints at Johnson's suspicions: as is known, *unrestricted* attribute grammars are more powerful than tree automata, but if one restricts the attribute grammar so that it can be evaluated in one depth-first left-to-right pass over the derivation tree, as seems to hold in Correa's system, then such a grammar's language is equal to the yield (fringe) of the languages recognizable by deterministic tree automata (Engelfriet and Filè, 1979). Thus there seems to be a connection between tree automata and restricted attribute grammars of the kind needed for natural grammars, unifying all three accounts. All structurally *local* constraints, including $\overline{\text{X}}$ theory and Thematic theory, can be formulated in this way, it appears. It would be a useful probe this connection in more detail to *prove* it to be so.

Besides these local conditions, there remain 'long-distance' principles: movement of NPs and *Wh*-phrases in sentences like *What did you*

think John ate, where there is a *chain* of traces or empty categories linking *what* to a position before *John* and then to a position after *ate*. The same things happens with LF where quantifier phrases like *several students* are moved about so that they can take wide scope in a sentence like Epstein's above. How can these be modeled? Here again, the authors are in basic agreement: instead of explicitly computing these chains via a derivation from D-structure to S-structure or S-structure to LF, they have all opted to formulate chains as a set of constraints *on* S-structure itself. Astonishingly enough, this *works*. For instance, Fong's parser can actually parse basically all of the several hundred example sentences in Lasnik and Uriagereka's textbook (1988), just the way the theory intends. *Astonishingly* is the right word because it hardly seems believable at first that one could actually translate all of several linguists' often incompatible written ideas into a logical formulation that zooms in on *just* the correct analysis of many hundreds of sentence types. It is by no means clear that this translation can even be done into Prolog (as Stabler, 1991 forthcoming, argues), and that powerful first-order theorem provers will be required. But it has been done; it works; and it works quickly (in just a few seconds on an older model Lisp machine).

For whatever reason, this declarative formulation seems much more computationally tractable. Again a full proof waits in the wings, but historically at least, under close scrutiny generative conditions have slowly given way to declarative admissibility constraints (for example, compare the discovery by Peters and Ritchie, 1973, that context-sensitive rewrite rules have been used for local tree *analysis*, and under this interpretation they admit only context-free languages. See section 4 for more discussion). This is also a viable linguistic option, advanced by Koster (1978) and others. Thus it is no surprise that Correa, Dorr, Epstein, Fong, and Johnson all use constraints on S-structure representations to encode long-distance phenomena. This choice also has a linguistic effect: D-structures are not actually computed by any of these parsers, as Johnson demonstrates in detail, even though D-structure information is factored into the parsers (again see section 4 for how this is done). The psycholinguists paint the same picture: D-structure information is tapped, but nowhere need one compute D-structures explicitly.

3.2. *Fixed Control Structures, Flexible Data Structures*

Turning now from those who have advocated the separation of control and data structures, those authors who have chosen Lisp or a close cousin for their implementation language—Abney, Correa, Dorr, Kashket, and Kazman (and, more casually, psycholinguists like Gorrell and Pritchett)—have bound their fates to a fixed parsing control structure.

Of those embracing fixed control structures, most, like Abney, Correa, Dorr, Kashket, and Kazman, have opted for a basically bottom-up implementation using the two-decade old idea of a *covering grammar*—a phrase structure grammar that augments S-structure, tapping into the lexicon, movement, and other constraints to come up with a different grammar than S-structure, yet one that incorporates all these D-structure effects at once. D-structure is no longer directly computed; the modularity of the theory is partially destroyed in order to salvage computational efficiency. The psycholinguists Gorrell, Kurtzman *et al.*, and Pritchett also find this nonexplicit D-structure approach attractive.

The key point to attend to, though, is *why* covering grammars can be successfully revived at all, after more than two decades. The answer is that a theory built on principles rather than rules has two advantages: first, a natural and simple covering grammar—as mentioned, the one formed by augmenting $\bar{X}$ structure with movement and some thematic and Case constraints; and second, a much simpler notion of transformation. In the older approach, each transformational rule had to be spelled out by means of a complex structural description (an *if* condition) that dictated when it applied, followed by a structural change (the *then* part), as in this subject formation example from Petrick (1973, p. 33) that attaches a *Wh*-NP to the Auxiliary position, to convert *e.g.*, *will who eat ice-cream* into *who will eat ice-cream*, checking that the sentence has not already been turned into a question. Each subtree component is marked with a number (1 through 6, 6= the Sentence; 1= boundary marker #, 2= the Auxiliary tree; 3= NP; 4= any subtree X; and 5= a boundary marker #). The transformation adjoins component 3, the *Wh*-NP, to subtree 2, leaving behind nothing:

```
RULE SUBJFRMA (subject formation)
structural
description:  S1   #   AUX    NP   X   #
                   1    2     3    4   5
constraints:  ∨ not S1 marked +Ques
                NP is marked +Wh
structural
change:            1  (3 2)   0    4   5
```

Note that on top of these detailed surface-patterned conditions, rules were marked as obligatory or optional, cyclic or postcyclic, and so forth. *None* of this is required in the principle-based framework. The movement of the *Wh*-NP is allowed because there is a 'landing site' (the Complementizer position at the front of a clause, not known in the earlier formulation), and because the resulting move passes all the other modular conditions. Nothing more has to be stated beyond what is already needed for other general properties of the system; there isn't any specific rule of subject formation. Because each principle-based constraint contributes just a bit to the overall constraint that fixes possible surface sentence forms, we have a fighting chance. For example, one can show that an S-structure covering grammar can be small enough to succumb to LR-parsing techniques, with a parsing table of only a few hundred entries (smaller than what's needed for a typical computer programming language; see Fong, 1991 forthcoming). This is possible only because each component does not try to do too much at once; the allowable $\bar{X}$ structures are really quite simple.

We have already briefly outlined Correa's technique in this computational war: using attribute grammars, he can handle the local constraints like Thematic theory that are obvious candidates for local tree attribute assignments. What about long-distance constraints? Again, a chain is composed from strictly local tree attribute assignments: in the *What did you think John ate* example, we can build the chain link by link in both directions. From the top down, we can propagate a chain value ever downwards, step by step; from the bottom up, we can propagate the proper value upwards: first a link is made between the empty category NP position after *ate* and the complement position before *John*, by 'passing up' the attribute via attributed assignment from the NP position to the complement position; then this attribute value is passed up once more to the front of the sentence to link up with the value passed down from *what*. No D-structure is used; instead, Correa assumes, following Koster (1978), that empty categories can be *base generated* in S-structure itself.

Because there are two types of chains in the theory—movement to so-called argument or A positions, like the Subject of a sentence in passive constructions, and movement to nonargument or $\overline{A}$ positions like the complement position at the head of a sentence in *Wh*-questions, Correa uses two attributes, A and AB ('A-bar' or $\overline{A}$). However, as he notes, this chain composition algorithm has its own gaps. There can be only one A or $\overline{A}$ chain formed per phrase. Since only two attributes are used, one can only move out one NP to an argument position. This is a problem, since Scandanavian languages evidently admit many more than one *Wh* word at the head of a sentence, and even problematic in English, where so-called *parasitic gap* sentences such as *Which book did I buy without reading*, where there are *two* empty categories, one after *buy* and one after *reading*, that are both linked to *which book*. One can show that these empty categories are both $\overline{A}$—roughly, they are both variables, linked to the logical operator *which book* (see Lasnik and Uriagereka, 1988, p. 78 for discussion). But this cannot work with only one AB attribute. This problem underscores the difficulty of hand-crafting an algorithm that combines many different linguistic principles: one can never be quite sure that it is logically correct. The reader is invited to check whether Correa's account of 'structural' determination of empty categories is in fact correct, and then judge whether a logic programming approach would have been better.

Dorr adopts a *coroutined*, hand-built covering grammar to get parsers for Spanish, German, and English translation. To do this, Dorr interleaves a conventional Earley parser with principles by applying constraints when phrases are started or completed. The system also assumes an augmented covering grammar that incorporates phrase movement and thematic constraints, thus leaving aside D-structure at run-time. To cover several languages, like Fong and Johnson, Dorr implements a specification language as a buffer between the parametric vocabulary of linguistic theory and the algorithmic guts beneath: at the top level, different languages apply the Case filter or not, and have their phrasal Heads first or final; this information is then compiled behind the scenes into a covering grammar that includes some details about possible phrase movement and Thematic constraints. Note that many constraints remain untested, so this grammar will overgenerate. It is this covering grammar that the Earley parser actually works with. The

system starts work by projecting from an input word the maximal covering $\overline{\text{X}}$ augmented structure possible, *e.g.*, the Spanish *vio* (*ver, to see*) is projected to a VP. When a phrase is completed, then, much like Correa's system, one can stop structure building and propagate certain features and carry out certain tests like Case assignment and Thematic role assignment; these are done procedurally rather than via the formal machinery of attribute grammar. For example, Dorr shows how the projection of the V to a VP then institutes a search for an NP clitic (*la, he*) as a thematic role of *ver* to the left of the verb, given information in the lexicon. Despite these procedural differences, the bottom-up algorithm and its subtree by subtree construction remain close to Correa's model.

Kashket's parser for the free-word order language Warlpiri also uses a bottom-up parser that *starts* with the lexicon to project phrases, which are then tied to each other by other, thematic principles. This makes a great deal of sense given the word order freedom of Warlpiri. All the possible permutations of phrases are permitted, and noun phrases can even be interwoven with other noun phrases. There are 24 possible ways of saying the following sentence, where we have listed just one other possible permutation:

Karli	*ka-rna-rla*	*punta-rni*	*ngajulu-rlu*	*kurdu-ku*
boomerang	*imperf*-1s-3d	take-*nonpast*	I-*erg*	child-*dat*

'It is the boomerang I am taking from the child'

Kurdu-ku	*ka-rna-rla*	*ngajulu-rlu*	*karli*	*punta-rni*
child-*dat*	*imperf*-1s-3d	I-*erg*	boomerang	take-*nonpast*

'From the child I am taking the boomerang'

To make order out of this apparent word salad, Kashket again turns to the bottom-up and lexical projection style parsing used by Correa and Dorr. Each lexical item is first projected bottom-up into its $\overline{\text{X}}$ counterpart. For instance, *kurdu-ku* (child-dative) is projected to form an NP subtree, bottom-up. A second pass then applies the principles of Thematic theory and Case to sweep subtrees into larger structures, according to the Case markers, like the dative *ku*, attached to the ends of words. It is by using the Case markers and an S-structure that contains no left-to-right precedence information at all that absolutely free order can be handled. Importantly, Kashket shows that Case marking is also operative in languages like English, but here, since marking is carried out by the inflection in second position and the verb marking to its right, English winds up with a basically fixed Subject-Verb-Object order. Thus, on this view, there are no 'fixed' or 'free' word order languages. As with passive constructions, this is an artifact of a superficial analysis. Languages can vary according to a parameterization of Case marking, from free to fixed order, even within the same language (adjunct prepositional phrases are relatively free in English because their Case is inherently fixed, like the Warlpiri NPs, rather than assigned).

Abney's work is closely allied to Kashket's. Abney too uses a bottom-up parser (but a parallel bottom-up parser) for phrase *chunks*. His twist is to add a three-stage bridge between (1) prosody (intonational patterns); (2) phrase chunks as a mediating between prosodic phrases and syntactic phrases; and (3) full syntactic phrase construction via thematic role determination. The prosodic level aims to capture some of the constraint in spoken input. 'Chunks' are extracted from a sentence including prosodic salience peaks, like *the bald man was sitting on his suitcase*—namely, [the bald man], [was sitting], and [on his suitcase]. These are quite close to the projections that Kashket defines. Given the prominence of prosodic cues in Warlpiri, perhaps this is not so surprising. The grammar of chunks is again a covering grammar for $\overline{\mathrm{X}}$ structure plus some movement factored in. Next, a second-stage pass, like Kashket's, pastes some of these phrases together, following the constraints of Case theory and Thematic theory—for example, it attaches *on his suitcase* to *was sitting*.

Kazman is the only author to explicitly address the problem of language acquisition in a principle-based system. His parsing design mirrors Kashket's and Abney's: for each word in the sentence, the parser first constructs NPs, VPs, etc., according to $\overline{\mathrm{X}}$ theory. A second stage then attempts to attach the phrases to one another, subject to Case theory, Thematic theory, and so forth. Next, by parametrically varying the parameters—*e.g.*, by turning off the Case filter—Kazman strives to reproduce the patterns of child language, tested by using the altered parser to process actual examples of child speech. Using four sets of 50 sentences each from two children, he shows that this parameterization works: the lobotomized parser adequately describes child speech from about age 2, and, by changing the parameters, arrives at an 'adult' parsing competence.

The psycholinguistic studies also all (implicitly) adopt fixed control structures—bottom-up parsers, some (like Gorrell's) parallel and close in design to Kazman's or Abney's, some not. Regardless of the parsing design, the basic point that each one of them drives home is that principle application and interpretation takes place as *soon* as possible. In particular, Gorrell, Kurtzman *et al.*, and Pritchett show that information from subcategorization frames or thematic theory—roughly, what phrase subtypes/thematic roles can be associated with each verb, as whether a verb is transitive or not—seems to be used early on in parsing, and, more generally, principles and constraints are applied as soon as possible. This supports Stabler's point; not so surprising perhaps, but the authors do more. They go on to show that this early-as-possible constraint accounts for the human sentence processor's abilities in processing locally ambiguous sentences (*after the child had visited the doctor prescribed a course of injections/after the child has sneezed the doctor prescribed a course of injections*); garden path sentences, as in *after Steve ate the soup proved to be poisoned*; and the difficulty of finding the 'gaps' associated with *Wh*-phrases, as in *what did John escape from*, where *what* is associated with the position after *from*.

4. SEARCH AND PRINCIPLE-BASED PARSING

As we have seen, overgeneration and how to avoid it is *the* key theme of principle-based parsing and psycholinguistics. To better understand the overgeneration issue as the tie that binds all the chapters together, we now turn to our second unifying umbrella, the metaphor of *parsing as search*. Our parsing task is a classic search problem. We must enumerate a space of possibilities, shown conventionally as a branching tree (*not* a phrase structure tree!) of possible paths, and apply tests to rule out some, perhaps all of these paths; those surviving are the valid parses. Overgeneration is one symptom that our searching algorithm fares poorly. For each author the key to reducing overgeneration is some way to exploit the modularity of a principle-based system, recasting the principle modules to avoid exhaustive search. Let us see how.

Figure 4 sketches the main possibilities as a kind of graphical taxonomy of what the authors do. Each figure 4(a) to 4(d) shows a portion of a search tree for discovering the possible parses. Each circle standing for some *enumerator* or *next path generator* plus a (partial) structure already built; the result will be either a single new branch (if no constraints are applied); a single new branch or perhaps a dead end (if the enumerator is applying a constraint like the Case filter); or many new branches (if the enumerator is applying a generator like Move-α). Each possibility leads to further enumerations, possibly casting some structures out as we go, until we arrive at the fringe of the tree where entire parses are found, and either ruled OK or out. Note that in order not to miss any parses, the *entire* space of logical possibilities must be exhaustively explored, but we can limit the exhaustion by cutting down the number of nodes and branches generated (we want the *branching factor* or number of new paths generated at any step to be as small as possible) and reducing the cost of enumeration at each node. The graphical sketches lay out the basic methods for doing this. We shall now consider them, one by one.

Parts (a) and (b) of the figure illustrate the most direct and costly ways of coping with overgeneration: ignore it. In (a), given a sentence, we build *all* possible structures, without applying any constraints, not even looking at the input sentence, and *then* apply all constraints to each candidate structure in turn. This simple and well-known strategy, dubbed *analysis by synthesis*, was proposed in the the earliest discussions of parsing with transformational grammars (see Petrick, 1965 for additional historical discussion).[2] Alternatively, as in (b), we could generate each candidate structure in turn, and as each is completed, *test* it against the constraints, then generate the next structure, and so on. This is the familiar *generate-and-test* paradigm of artificial intelligence.

Left unconstrained, methods (a) and (b) suffer from serious, even fatal, overgeneration. This was was well known from earlier work in search and is demonstrated again Johnson's chapter via his first parsing model (PAD1) that adopts generate-and-test search method (b)—it might not even terminate, as Johnson indicates, because one could generate an infinite number of D-structures first, as would be true of any reasonable recursive syntax, and one might not ever get to the next enumeration stage). To take another example, Correa's chapter notes that Barss' (1983) and Chomsky's (1981) specifications of the conditions on chain formation are just that: *specifications* of algorithms, not algorithms themselves. Strictly interpreted as algorithms, they generate *all possible* movements in S-structure first, and then test each one in turn. Correa shows that this blind generate-and-test approach takes exponential time in the length of the input string, but that his attribution grammar that builds up chains by evaluating substructures piece by piece is evidently linear time—a vast improvement, if correct.

Somehow then we must mix the constraints from the lexicon and $\overline{X}$ theory together with the constraints like the Case filter that apply at other levels, and do so piecemeal *without* losing the advantages of

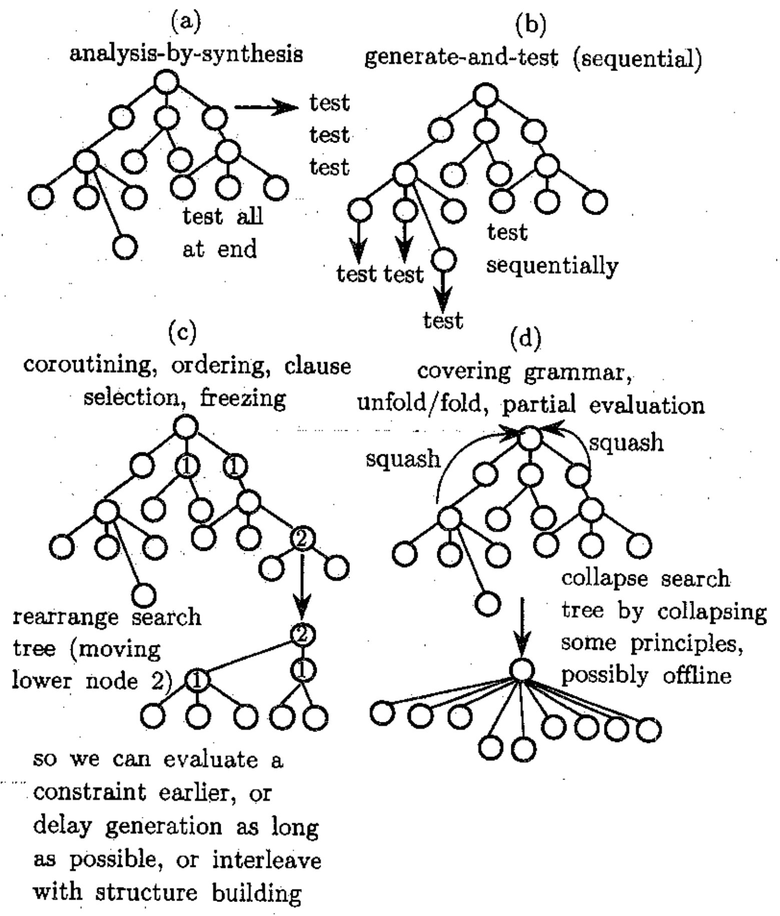
(a)
analysis-by-synthesis
test
test
test
test all
at end
(b)
generate-and-test (sequential)
test test
test
test
sequentially
(c)
coroutining, ordering, clause selection, freezing
1
1
2
2
1
1
rearrange search tree (moving lower node 2)
so we can evaluate a constraint earlier, or delay generation as long as possible, or interleave with structure building
(d)
covering grammar, unfold/fold, partial evaluation
squash
squash
collapse search tree by collapsing some principles, possibly offline

Figure 4: All the different strategies for speeding up principle-based parsing can be viewed as techniques for better enumerating or searching a space of possibilities. Parts (a) and (b) of the figure illustrate exhaustive enumeration methods that are too computationally intensive. The authors use the methods sketched in parts (c) and (d) of the figure to reduce the search space itself.

principles that we sought in the first place.

In fact, in one way or another all the authors have adopted ways to weave the constraints from the lexicon and D-structure back into S-structure, so as to get one foot into the phrase structure door as required to apply the remaining principles. Ultimately, all these methods are based on the observation that most principles and constraints apply locally to parse substructures. Therefore, if some structural subpart fails to make it past a principle, such as *I am proud John* (which violates the Case filter since there is no *of* between *proud* and *John* to Case mark *John*) then a larger structure built out of this, such as *I think that I am proud John* will inherit this failure. Therefore, there is no need to apply *all* constraints on the *entire* parse structure. It is enough to apply the right constraints to pieces of the parse tree as it is built.

Techniques (c) and (d) in the figure do exactly this, and *all* the authors have tried to cure the overgeneration problem by adopting one or more of these methods. Their approaches follow directly from the logical possibilities for reducing the enumerated space—either cutting down the number of nodes and paths generated, or reducing the enumeration cost. Let's run through each in turn.

The method pictured in figure 4(c) shows how the related techniqnes dubbed *freezing* (Johnson's second parser, PAD2), *principle reordering* (used by Fong), *clause selection* (Stabler), or *parse table coroutining* (used by Dorr) can help reduce search. These are control structure strategies. As the parse proceeds, different principles may be called on, now drawn from the lexicon, now from D-structure, checking the well-formedness of the currently hypothesized structures as we go. Such *coroutining* is quite common in compilers for programming languages as a way to interleave phrase building with other kinds of constraint checking, and that's obviously just the ticket we need here. If the experiences of the authors are any guide, these methods can quite effective (for example, Fong's parser just takes a few second on even relatively complex sentences such as *This is the book that I filed without reading*, which is difficult to accommodate at all in an ATN approach).

Though each of these authors use distinct tools, they amount graphically to the same thing as shown in figure 4 part (c) and in another form in figure 5(c). On the left, we have some search tree with the generators and constraints arranged in a particular order, resulting in a bushy tree with many enumeration circles. On the right, we have rearranged the search tree by moving one of the enumeration nodes, marked *2* higher

up in the tree. How can this help? If the newly promoted node winnows out many structures, then subsequent generators like Move-α will have fewer structures to expand on. The new search tree on the right illustrates.

How are each of these part (c) methods implemented? Johnson and Fong examine automatic means of determining the dataflow dependencies among principles and ordering the parsing system offline so that it does not compute unwanted constraints. For example, as we saw earlier, it makes little sense to apply the Case filter constraint if there is no assignment of Case (which in turn depends on establishing certain structural relationships between verbs, prepositions, and nouns).

Johnson concentrates on semi-automatic logic programming techniques for handling D-structure, S-structure, PF, and LF dependencies, including the so-called *freeze* technique drawn from Prolog research that suspends the computation of one conjunct, say, Move-α, while the parser works on another that the conjunct depends on, like S-structure. Thus we say that Move-α is *delayed* on S-structure.[3] Because no S-structure is initially available, this immediately delays, or freezes, any computations involving Move-α, until the first node of S-structure is built. See figure 5(b). Then Move-α is applied while the further recovery of S-structure is frozen; and so on. (Note also that this parser recovers D-structure by using a declarative description of Move-α and then running the system backwards, as can be done in Prolog.) In this way principles can be *automatically* and systematically coroutined with structure building.

Fong also focuses on principle ordering effects but *without* coroutining, and uses a finer grain size than Johnson, arranging a proposed twenty-odd principle bundles or modules like a deck of cards as in figure 5(a). By rearranging the deck, subject to the logical dependencies of the theory (as we have seen, some things *must* be computed before others: we can't apply the Case filter until Case is assigned) we can explore all the logically possible conjunct orderings, assess their effect on parsing efficiency—evidently there can be an order of magnitude difference in parsing times for the same sentence depending on the arrangement of the deck—and even add a means to learn the best way to juggle the deck for a particular type of input sentence.

Fong's method does assume a modified generate-and-test approach: an entire S-structure is built before any other constraints are applied. From a different angle, (see part (b) of the figure) but akin to John-

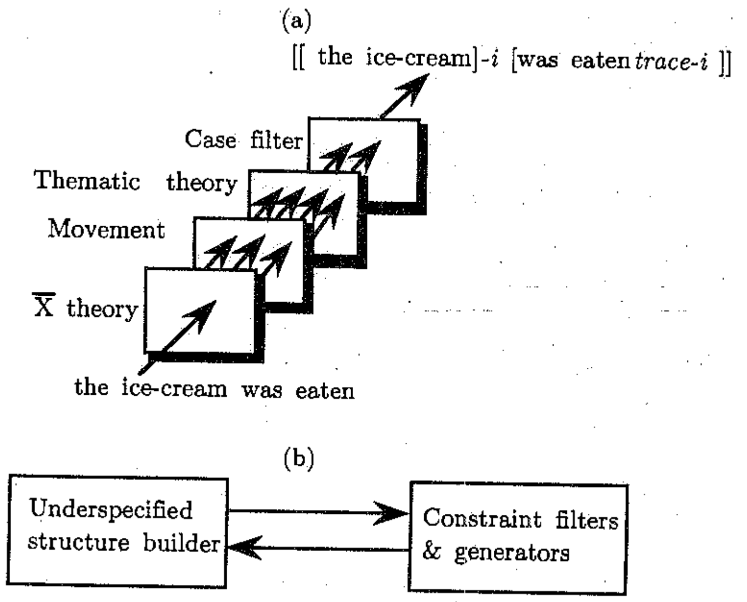
(a)
[[the ice-cream]-*i* [was eaten *trace-i*]]
Case filter
Thematic theory
Movement
$\overline{\text{X}}$ theory
the ice-cream was eaten
(b)
Underspecified structure builder
Constraint filters & generators

Figure 5: The principles-and-parameters theory can be organized for processing in two distinct ways, as shown in parts (a) and (b) of this figure. First, it can be pictured as set of 20 or so modules or groups of principles, arranged like a deck of cards as in the top half of the figure, (a). An orthographic representation of a sentence like *the ice-cream was eaten* is input at one end, and, if the sentence is completely well-formed, one or more valid parse structures emerges unscathed at the other end. The arrows emerging out of each box show that some modules act like filters that can cut down the number of structures that pass through them, while other principles act like generators that expand the space of possibly valid structures. The bottom half of the figure, (b), illustrates a coroutined processing model where structure building and constraint application are interleaved.

son, Stabler shows that the generate-and-test method is not forced upon us. The parsing algorithm need not produce an *entire* S-structure before beginning to incrementally compute an associated Logical form or calculating its truth-conditional interpretation, even assuming conventional branching for English phrase structure (contrary to what some researchers have assumed).[4]

Dorr develops a coroutining strategy that interleaves the phrase construction carried out by a standard context-free parser (swinging into action when phrase like an NP is started or completed) with additional constraint checking (does the NP meet the Case filter). As Fong, Stabler, and Johnson demonstrate, this can be done automatically in a logic-based approach, based on the widely explored notion of *selection rules* or *ordering strategies*. Citing Johnson and following Fong and Stabler, suppose we regard a well-formed (sentence, LF structure) pair (what Johnson dubs the *parse relation*) as the first-order logic *conjunction* of the various PF, S-structure, D-structure, and LF constraints:

$$\forall d\text{-}struc, s\text{-}struc, lf, pf,\ \overline{\text{X}}(d\text{-}struc) \wedge \text{CaseFilter}(s\text{-}struc) \wedge \text{LF}$$
$$PF(pf, s\text{-}struc) \wedge LF(lf) \Rightarrow Parse(pf, lf)$$

(Johnson, this volume, p. 49)

The details don't matter here. What does matter is that we can decide to work on *any* one of the conjuncts at any time—and then, as we dive down into those conjuncts themselves, we can stop and build a part of D-structure, or apply the Case filter, or whatever. Naturally, it makes the most sense to work on a conjunct whose 'internals' will be partly grounded on what we already know, and this in turn depends on the input sentence and the 'dataflow dependencies' of the theory itself—what principles are wired to what other principles. For instance, as we have seen, it's unwise to start out enumerating all possible D-structures before checking what's in the input sentence. It makes more sense to first build a piece of the S-structure by projecting it from the input sentence. For example, in our *ice-cream* sentence, we can use conventional techniques and $\overline{\text{X}}$ theory to deduce that *the ice-cream* is a noun phrase, and *was eaten* a verb phrase. Now we have a skeleton S-structure on which to hang the other inferences about Case and Trace theory. Better still, as Johnson and Stabler's chapters observe, by interleaving the construction of structures and the applications of constraints, we need not trap ourselves in the hopeless task of first enumerating an endless sequence of D-structures.

Plainly then the order in which one works on conjuncts can make a vast difference in parsing efficiency, as is well known from work by Smith and Genesereth (1985) and others on *search tree rearrangement*. To see this in more detail, consider the following simple example definition from their work: to check if x has an aunt, $aunt(x)$, see if x has a mother y and that person y has a sister z:

$$aunt(x) = mother(x, y) \wedge sister(y, z)$$

Now to compute, say, $aunt(elissa)$ it matters a great deal whether we start in on the first conjunct or on the second. If we start on the first, we will compute $mother(elissa, y)$, bind y to some value, say, $marilyn$, and then check $sister(marilyn, z)$, which will then succeed or fail accordingly. But if we started on the second conjunct, we would initially compute $sister(y, z)$, for all possible values of y and z—that is, we would first find *all* the sister pairs in the universe, and *then* check each one to see if one of them was the mother of Elissa. This would clearly be much more expensive! Smith and Genesereth show, sensibly enough, that the optimal conjunct ordering essentially depends on applying the best-winnowing constraints as early as possible delaying hypothesis-expanding modules as long as possible. The connection to principle-based parsing ought to be clear: computing all the sisters first before looking at Elissa is very much like enumerating all the D-structures before looking at the constraining input sentence.

The search speedup methods just described in part (c) of figure 4 are all geared to different control strategies that coroutine. Most of the remaining search improvements work as in figure 4 part (d): by squashing the enumeration tree, accordion-wise. As the figure shows, this attempts to collapse two (or more) search nodes into one larger one that meets *both* sets of constraints or generators covered by the individual nodes. The general idea is to apply constraints sooner, rather than later, and so avoid exploring structures that cannot possibly participate in a solution. As we shall see, there are many ways to collapse the search tree, but for now it is enough to see that collapsing reduces the size of the search space. It simply squashes together the constraints/enumerators in such a way that one larger predicate is applied rather than the sequential conjunction of several.

This helps, because the number of branches generated is cut down immensely. To take a concrete example, the number of bare $\overline{X}$ structures possible might number in the many thousands (roughly 80,000 according to Fong's experiments, 1991 forthcoming); but the number of $\overline{X}$ structures plus following NPs that meet the Case filter and thematic constraints are only about 100. Thus if we tested this system sequentially we would first generate 80,000 possibilities, then run each of these by the Case filter, reducing the number to about 2000, and then through the Thematic constraints to yield 100 outputs—82,100 tests. In contrast, if we carry out all the tests jointly, as one giant enumerator, we generate only about 100 branches.

Note that collapsing search tree nodes is not *necessarily* cheaper. Squashing the search tree can go too far and make the enumerator's job harder. If the enumerator is computationally expensive, then the savings from fewer branches explored can be swept under by the extra computation. Again consider one concrete example, *Mary likes her mother*, where there are two Logical forms, one where *her* is *Mary* and one where *her* is some other person, not named in the sentence; this is conventionally indicated by indexing, or linking of *Mary* and *her*. If we squash all constraints together, then we must in effect build *two* separate structures, checking all the constraints (in concert) twice. But we could squash everything *except* the computation of the indexing. Then we just have to apply our giant combined predicate, *sans* LF, yielding *one* structure, and from there use indexing to generate the two output possibilities. It turns out that this last approach takes less time because the enumerator does not have to go through the construction of another entire tree structure that is easily derived from a single common form. More generally, if computing the proper structural configurations required to apply a constraint or principle is expensive, then the benefits of collapsing search nodes can outweigh the costs. This is where a smart offline computation can help, by figuring out in advance in what configurations certain principles and constraints will never apply.

In general though there are big savings to be had in squashing the search space, so big that almost all the authors—Abney, Correa, Dorr, Epstein, Fong, Gorrell, Johnson, Kashket, Kazman, Kurtzman *et al.*, Pritchett, and Stabler—adopt it in one form or another. How then is this accordion move played out?

The first compression technique, covering grammars, simply collapses several constraints together—for example, $\overline{X}$ tree structures, plus Move-α plus thematic constraints—to create a new phrase structure grammar that has more detail than just $\overline{X}$ theory alone. The collapsed search space is smaller because it has additional constraints built in, but still covers the same sentence territory as before. We illustrated above how much this can reduce the search space: we look just at possible $\overline{X}$ plus movement configurations, instead of first generating all possible $\overline{X}$ configurations.

Most of the authors who have built parsers—Abney, Correa, Dorr, Fong, Epstein, Kashket, and Johnson—take this tack. In effect, they propose to recover not simply S-structure, but rather a closely related representation described by a covering grammar. As described earlier, this is simply a new grammar that has lexical (thematic) constraints and Move-α factored into it. The new representation is called a covering grammar simply because it does not include *all* the constraints of the original principle-based system—if it did, we would just have rules again—so it must of necessity *cover more* possible structures than the original principles. Once again this technique should be familiar to historians of the field: early transformational parsers used covering grammars in just this way to bootstrap an initial, but overgenerating, tree structure on which to decamp the structural descriptions of (inverse) transformations.

Such an approach is possible because of a number of changes in the theory of grammar advanced by the authors, of which the most important is perhaps the view promoted by Koster (1978), namely, the so-called *base-generation hypothesis*: if as standardly assumed the $\overline{X}$ structures are describable by a context-free grammar, and if S-structure is derived from D-structure by moving phrases into positions where they *could* have been generated originally by the $\overline{X}$ grammar, then the S-structure can plainly be generated by a context-free grammar straight out, without appealing to D-strncture at all.[5]

To be concrete, recall our passive example again: *the ice-cream was eaten*. To say that the noun phrase Subject 'landing site' in this sentence is base generated is to say simply that there is an $\overline{X}$ template in the form S→NP VP, which of course there must be for ordinary sentences anyway. To take another example, we could write a covering grammar to accommodate *Wh*-questions like *what did John eat* by changing the the $\overline{X}$ module to include the templates $\overline{S}\rightarrow$ *what* S and VP→V NP_e, where NP_e is a trace or lexically null noun phrase. The new grammar is *not* pure $\overline{X}$ theory—it actually looks more like a conventional context-free rule-based system, or even something akin to the original conception of Generalized Phrase Structure Grammar with its system of displaced 'fillers' like *what* and gaps like NP_e. Despite appearances though, the

new system is *not* really rule-based, since the principles like the Case filter or Binding theory still apply as before.

We might think of the new covering grammar as a run-time version of a principle-based system where certain of the principles have been 'compiled' together. The covering grammar computation is done off-line, much as a compiler would work. If such a compiler is smart enough, it can detect principle combinations that never arise, and so eliminate predicate tests that would be wasteful. For example, consider $\overline{X}$ theory, phrase movement, and Thematic theory again, in particular the assignment of thematic roles. Assignment occurs only under certain structural configurations. While we could apply the Thematic theory constraints to *all* configurations (as in fact is done by Johnson), in fact this may be computationally inefficient, since assignment patently does not occur in all configurations. We can compute *offline* that thematic role assignment occurs only if the structures involved include the phrasal categories NP, S, and the intermediate phrasal $\overline{X}$ categories $\overline{N}$, $\overline{V}$, $\overline{P}$, and $\overline{A}$(intermediate adjective phrases). We do *not* have to apply the Thematic constraints if the structural configuration happens to include other phrases, like PP, AP, etc. Thus, a clever compiler can save us from much useless work.

Of course, compared to programming languages, we don't know as much yet about natural language grammar 'compiling'. The authors in this book have taken two approaches to this compile-time/run-time trade-off: some, like Dorr, Epstein, Fong, and Kashket, have done it by hand; others, like Johnson in his PAD parsers 3–5, have taken a more formal, automatic route and used the notions of *partial interpretation* and the related notion of *unfold/fold* transformations possible in logic programs (Tamaki and Sato, 1984). The unfold/fold idea, which is a close cousin of the notion of *partial evaluation*, is to run deductive interaction of selected principles for a bounded number of steps, offline and without an input sentence—the principles are 'folded' together to derive new predicates that are a combination of the old. This is a provably valid reaxiomatization of the original system.

To take a simple example in another domain, if we had the combination $\sin(x)\cdot\sin(x)+\cos(x)\cdot\cos(x)$, then we can evaluate this expression offline to $\sin^2(x)+\cos^2(c)$ and then with a second step to 1. Here we have replaced a runtime computation with a constant—some computational work for zero runtime work. Note that this indeed exactly what a programming langnage compiler tries to do. It is also what the covering grammar method attempts by hand. The great advantage of the fold/unfold method is that the reaxiomatized system is provably equivalent to the original: we are guaranteed that the same parse relation can be deduced in the modified system as before. Ensuring this by hand can be a much more tedious task. Even so, the automatic partial evaluation methods are not completely understood. (For one thing, figuring out where and how deeply to partially evaluate can be difficult, and the resulting system *still* applies the predicates it has folded together to some structures where it cannot possibly succeed. Refer to Correa's efforts at reformulating chain algorithms.) Thus hand-coding covering grammars still has a place in the sun, until we have a better understanding of the trade-offs in offline compilation vs. runtime constraint application.

More importantly for computation, using such a covering grammar gets at the root of one of the problems in recovering the inverse mapping from S- to D-structure—by eliminating D-structure. We can ditch D-structure and incorporate the Theta-criterion and Move-α constraints as well-formedness conditions on representations S-structures—as filters that either let certain S-structures pass or not, rather than as conditions on the derivation of one representation from another. As mentioned earlier, this shift from generative to declarative constraints on representations is another hallmark of the principle-based approach. We have seen that this *seems* to make computation easier, and have given some reasons why, though this has never been formally proved. Correa, Johnson, and Fong all take the line, with Johnson establishing that D-structure surgical removal is logically correct (the same parsing relation can be 'inferred' as before). Historically this line of reasoning extends back to McCawley's 1968 paper viewing context-sensitive rewrite rules as declarative constraints on trees, essentially template filters, *e.g.*, the rules S→NP VP; VP→ Verb+*animate*; NP→noun__V+*animate* is interpreted as admitting a set of trees dominated by an S(entence) node with an NP dominating a noun on the left *if* there is a verb marked +*animate* to the immediate right of the noun. The work of Peters and Ritchie (1973) proved that this declarative formulation was computationally simpler than the generative one, since it admitted only context-free languages despite tapping context-sensitive admissibility conditions like the template above. Modern principle-based theory awaits something like this demonstration as a precise comparison of generative vs. declarative representations of grammars.

5. CONCLUSION

Still, the central achievement of principle-based parsing stands: two decades past the 1960s we now *can* build efficient principle-based parsers. The reason? Our computational armamentarium is stronger. We know more about principle interleaving, freezing, and clause selection. The theory is more constrained: there is no extrinsic ordering or complicated rule conditionals; landing sites are limited by structure preservation, and empty categories are restricted in their distribution.

The moral should be clear. There are more things on a principle-based heaven and earth than dreamed of in a simple philosophy of parsing. Thinking about principles liberates us from the homogeneous rule-based format to open before us much wider logical possibilities for parsing control structures, hence a much wider range of design strategies and psycholinguistic outcomes than before. The working systems and results described in this volume form the beachhead of a much broader wave of principle-based research to come.

ACKNOWLEDGEMENTS

This chapter has benefited from discussion with Steve Abney, Bonnie Dorr, Ed Stabler, and, especially, Sandiway Fong. This work has been supported by NSF Grant DCR-85552543 under a Presidential Young Investigator Award, by the Kapor Family Foundation, and by a John Simon Guggenheim Memorial Fellowship. All remaining errors are of course the responsibility of the author.

NOTES

[1] This shift is certainly not unique to principles-and-parameters theory. Many other current linguistic theories, among them Generalized Phrase Structure Grammar (GPSG) and Head-driven Phrase Structure Grammar), have gradually shifted to declarative constraints that are not construction-specific. For instance, modern GPSG theory is full of principles that fix the well-formedness of surface strings without spelling out explicitly their detailed phrase structure: the Foot Feature Principle, the Head Feature Convention, and the Control Agreement Principle replace explicit phrase structure rules with constraint sets. These theories too no longer contain large numbers of particular rules, but declarative schemas constrained according to syntactic and morphological principles.

[2] As defined here, generate-and-test subsumes the more particular strategy of analysis by synthesis, because in classic analysis-by-synthesis we would enumerate all possible D-structure, S-structure, PFs (and LF) quadruples first, and only then test them against the input sentence PF; generate-and-test subsumes this particular enumeration strategy by admitting any possible sequence of interleaved quadruple generation

and testing against the input.

[3] 'Freezing' is not the sole province of Prolog; it amounts roughly to the delayed binding technique of many other computer programming languages, as described for example for Scheme in Abelson and Sussman (1985).

[4] Indeed, in later work, Fong (1991 forthcoming) adopts just such an approach, *interleaving* structure-building and principle filtering or generating. Again, instead of waiting for an entire S-structure to be built before applying any constraints, and then 'tree walking' to jog through the entire structure, checking each constraint, one can combine two or more constraints—say, the Case filter and restrictions on Move-α—into one larger predicate, and use that super-predicate to guarantee that any tree nodes produced incrementally satisfy the constraints to begin with. This method saves the parser's feet from tree walking, and is generally faster; it is a close cousin of partial evaluation or the reaxiomatization 'unfold-fold' technique used by Johnson, described later on.

[5] Emonds' (1976) *Structure Preserving Hypothesis*—that landing sites for moved phrases are also base generated—is of course central to the base generation approach. On the base generation view, it is impossible for there to be a derivational history that records the mapping from D-structure to S-structure and so any approach that relies on intermediate steps between the two in a crucial way—such as first moving a phrase to an intermediate position and then deleting a trace—is not replicable in the base-generation model. The empirical examples that distinguish between the two approaches are quite subtle, however, as has been noted in the literature (see, *e.g.*, Chomsky, 1981, for discussion and the references cited in Correa's chapter). The sheer empirical fact that a context-free base plus movement can be replaced by a structurally equivalent 'covering grammar' that is exactly or close to context-free underscores the connection between the base-generation approach and efforts like GPSG that also strive to eliminate the D-structure to S-structure mapping.

REFERENCES

Abelson, H. and G. Sussman: 1985, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Massachusetts.

Bates, L.: 1978, 'The Theory and Practice of Augmented Transition Network Grammars', in L. Bolc (ed.), *Natural Language Communication with Computers*, Lecture Notes in Computer Science no. 63, Springer-Verlag, New York, pp. 191–260.

Berwick, R.: 1985, *The Acquisition of Syntactic Knowledge*, MIT Press, Cambridge, Massachusetts.

Chomsky, N.: 1953, *Morphophonemics of Modern Hebrew*, Garland Press, New York (republication of M.S. dissertation, University of Pennsylvania).

Chomsky, N.: 1955, 1975, *The Logical Structure of Linguistic Theory*, University of Chicago Press, Chicago, Illinois.

Chomsky, N.: 1981, *Lectures on Government and Binding: The Pisa Lectures*, Foris, Dordrecht, Holland.

Engelfriet, J. and Filè, G., 1979: 'The Formal Power of One-visit Attribute Grammars', Memorandum 217, Twente University of Technology, Twente, Netherlands.

Emonds, J.: 1976, *A Transformational Approach to Syntax*, Academic Press, New York.

Fong, S.: 1991 forthcoming, *Computational Properties of Principle-Based Grammatical Theories*, Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, Massachusetts.

Kolb, H. and C. Thiersch: 1988, 'Levels and Empty Categories in a Principles and Parameters Approach to Parsing', unpublished manuscript, Tilburg University, Tilburg, The Netherlands.

Koster, J.: 1978, 'Conditions, Empty Nodes, and Markedness', *Linguistic Inquiry* **9**, 551–593.

Lasnik, H. and J. Uriagereka: 1988, *A Course in GB Syntax: Lectures on Binding and Empty Categories*, MIT Press, Cambridge, Mass.

Marcus, M.: 1980, *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, Massachusetts.

Marr, D.: 1982, *Vision*, W.H. Freeman, San Francisco, California.

McCawley, J.: 1968, 'Concerning the Base Component of Transformational Grammar', *Foundations of Language* **4**, 55–81.

Peters, S. and R. Ritchie: 1973, 'Context-Sensitive Immediate Constituent Analysis: Context-Free Languages Revisited', *Mathematical Systems Theory* **6**, 324–333.

Petrick, S.: 1965, *A Recognition Procedure for Transformational Grammars*, Ph.D. dissertation, Department of Linguistics, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Petrick, S.: 1973, 'Transformational Analysis', in R. Rustin (ed.), *Natural Language Processing*, Algorithmics Press, New York, pp. 27–41.

Sharp, R.: 1985, *A Model of Grammar Based on Principles of Government and Binding*, M.S. dissertation, Department of Computer Science, University of British Columbia, Vancouver, British Columbia.

Smith, D. and M. Genesereth: 1985, 'Ordering Conjunctive Queries', *Artificial Intelligence* **26**, 171–215.

Stabler, E.: 1991 forthcoming, *The Logical Approach to Syntax: Foundations, Specifications and Implementations of Theories of Government and Binding*, MIT Press, Cambridge, Massachusetts.

Tamaki, H. and T. Sato: 1984, 'Unfold/Fold Transformation of Logic Programs', *Proceedings of The Second International Logic Programming Conference*, Uppsala University, Uppsala, Sweden, pp. 127–138.

Zwicky, A., J. Friedman, B. Hall, and D. Walker: 1965, 'The MITRE Syntactic Analysis Procedure for Transformational Grammars', *Proceedings of the 1965 Fall Joint Computer Conference* **27**, Spartan Books, Washington, D.C., pp. 317–326.

MIT Artificial Intelligence Laboratory, Rm. 838,
545 Technology Square,
Cambridge, MA 02139, U.S.A.

PRINCIPLE-BASED PARSING:

Computation and Psycholinguistics

Edited by

ROBERT C. BERWICK

Artificial Intelligence Laboratory, MIT

STEVEN P. ABNEY

Bell Communications Research

and

CAROL TENNY

Department of Linguistics, University of Pittsburgh

KLUWER ACADEMIC PUBLISHERS
DORDRECHT / BOSTON / LONDON

Library of Congress Cataloging-in-Publication Data

```
Principle-based Parsing : computation and psycholinguistics / edited
  by Robert C. Berwick, Steven P. Abney, Carol Tenny.
       p.   cm. -- (Studies in linguistics and philosophy ; v. 44)
    Includes indexes.
    ISBN 0-7923-1173-6 (HB : acid-free paper)
    1. Parsing (Computer grammar)  2. Computational linguistics.
  3. Psycholinguistics.   I. Berwick, Robert C.   II. Abney, Steven P.
  III. Tenny, Carol.  IV. Series.
  P98.5.P36P75  1991
  006.3'5--dc20
                                                             91-2587
```

ISBN 0-7923-1173-6

Published by Kluwer Academic Publishers,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

Kluwer Academic Publishers incorporates
the publishing programmes of
D. Reidel, Martinus Nijhoff, Dr W. Junk and MTP Press.

Sold and distributed in the U.S.A. and Canada
by Kluwer Academic Publishers,
101 Philip Drive, Norwell, MA 02061, U.S.A.

In all other countries, sold and distributed
by Kluwer Academic Publishers Group,
P.O. Box 322, 3300 AH Dordrecht, The Netherlands.

Printed on acid-free paper

TABLE OF CONTENTS