In Japanese Sentence Processing
Eds. Mazuka et al. Lawrence Erlbaum 1995

CHAPTER 8

Madama Butterfly Redux: Parsing English and Japanese with a Principles and Parameters Approach

Robert C. Berwick
Massachusetts Institute of Technology

Sandiway Fong
NEC Research Institute, Princeton, NJ

INTRODUCTION

Puccini's familiar opera *Madama Butterfly* tells of a metamorphosis from Japanese to English and then back again—what appears, at first glance, to be wildly implausible. In this chapter we attempt a little of the same: probe the validity of the principles and parameters approach by exhibiting a single parser, parsing algorithm, and parameterized grammar that works for *both* English and Japanese, while at the same time exploring some of the computational differences and difficulties that arise.

As is familiar, the past decade has seen many advances in answering the question of how knowledge of grammar is represented; see Chomsky (1981). There has been a shift in transformational generative grammar from homogeneous, language-particular systems of rules such as passive, raising, and so forth to a highly modular, nonhomogeneous, and parameterized deductive system of universal principles. We call such systems *principles and parameters* (P & P) models, developed from the theory of government and binding (GB). However, until recently there has been far less progress in constructing efficient parsers that can take as input orthographic representations of sentences and output the representations demanded by the P & P approach or GB theory, namely, S-structure, LF, and D-structure. There has been even less progress in constructing cross-linguistic systems that do the same: That is, following the P & P approach, we ought to be able to keep the parser and the grammar essentially fixed, varying just a few parameters and the lexicon, and yet parse Japanese, say, instead of English. This is often touted as the litmus test for the P & P approach.

The aim of this chapter is twofold. First, we show that one can now address all the issues of principles and parameters linguistic theory in a precise computational framework. We do so by exhibiting a full implementation of P & P theory, parametrically varying across multiple languages, including English and Japanese. The parser demonstrates, by construction, that the same set of 25 principles used to parse 286 example sentences in chapters 1–4 of Lasnik and Uriagereka (1988) can be easily parameterized—with just four binary switches—to handle the Japanese *Wh*-questions in Lasnik and Saito (1984), and much more Japanese syntax besides. As far as we know, this is the first time that a full-fledged principles and parameters linguistic theory has been implemented as a parser for a broad range of constructions across distinctive, multiple languages. The implementation provides insight into the computational structure of the P & P approach. In particular: (a) the differences between English and Japanese, (b) how these differences affect computation and linguistic theory, and (c) whether a single processing algorithm suffices for distinct languages. For example, it has sometimes been suggested (Mazuka, 1991; Mazuka & Lust, 1988) that the head-final character of Japanese (as well as other languages) should pose special problems for a left-to-right (top-down) parser. Does it? Similarly for the omission of NPs, scrambling, lack of relative pronouns, and so forth (see Mazuka, 1991; Mazuka & Lust, 1990; Frazier & Rayner, 1988). Superficially these would seem to demand more guesswork by a parser. Do they?

Although these questions have been posed before, especially from a psycholinguistic viewpoint, to our knowledge they have never been thoroughly treated from a strictly computational vantage point. For example, Mazuka (1991) argues that "we will first present linguistic data showing that detecting an [empty category] and computing its possible antecedents during on-line processing is difficult in Japanese" (p. 5).

These and other hypotheses can be *rigorously* investigated from a computational point of view only with a fully implemented P & P parser. But until now, there has been no working parser that implements a complete P & P theory.[1] Without such a device, informal testing and *gedanken* experiments can be an immense problem, because there are just too many possibilities to overlook—with a few embeddings, possibly many thousands of candidate structures, as we shall see. We seriously doubt that anyone has ever managed to explore the whole solution space for such examples, and indeed there are some surprises that arise as a result.

[1]To be sure, fragments of such theories have been implemented, but the largest such efforts we know of (e.g., those of Stabler, 1992) have not been true parsers at all, but proof checkers: One must supply the parsing "answer" to such systems first (the LF or s-structure forms), and the system checks that the answer is derivable.

The second aim of this chapter, then, is to show how one can readily do computational "surgery" to investigate the logical and computational consequences of changes in linguistic theory, arriving at a space of possibilities that might be further narrowed by additional psycholinguistic evidence. For instance, one can use such a machine to tease apart the computational effects of Head-final structure from that of scrambling by building a grammar, J*, that is head-final and otherwise behaves like Japanese, but does not include scrambling and free omission of NPs. These experiments are demonstrated later in this chapter.

The results are sometimes surprising, sometimes not. Not surprisingly, by and large Japanese taxes the parser more: The freedom to drop NPs, scrambling, and head-final structure generally leads to more computational work expended. More surprisingly perhaps, note that the parsing engine we use is a canonical LR(1) parser, the optimum deterministic bottom-up parser, suitably modified to handle ambiguity. This *single* algorithm is used for all natural languages—left-branching as well as right-branching. Thus we do not need to parameterize the parser across different languages.

At least as far as our analyses indicate, branching direction does not seem to be an issue with an LR parser, *if* that machine can be modified to factor out such effects such as scrambling and free *pro*-drop in Japanese. To the extent that we can localize these differences in a precise way, we can partially answer Hasegawa's (1990) plea that "algorithms and formalizations do matter if the issues they [Mazuka & Lust] raise are to be productively discussed" (p. 222). Putting the same point another way, for Japanese there are just more ways to guess, even for a bottom-up parser, and we can demonstrate this concretely.

Arriving at more speculative psychological matters then, if one can judge at all from a machine that operates not quite like a person (for one thing, it can draw on unlimited lookahead when it has to),[2] these results reinforce what has often been said about the role of context and heuristic strategies in limiting hypotheses in Japanese sentence processing. If a machine that is in some ways more powerful than we winds up doing more guessing than we do, then clearly there is a mismatch somewhere: Machine design and approach, theory, or both could be wrong. For the required strategies and heuristics, one must turn to real psycholinguistics as reported in the other chapters in this volume. For the logical possibilities, though, psycholinguists might well turn to a P & P parser like this one.

In this operatic sequel, we follow the outline just given. First, we briefly sketch how the principle-based parser works with the Lasnik and Uriagereka

[2]On the other hand, it is not immediately clear that this lookahead power alone is necessary. We have not yet investigated the computational consequences of a *bounded* version of the same machinery. In other papers, we have explored an initial attempt to show how a *different* principle ordering, hence in a sense a different parser, could be built *dynamically* given a different past history of sentences encountered. In this way, context could be given a solid role within the system we have designed. See Fong and Berwick (1991).

theory, using a parasitic gap sentence as an illustrative example and focusing on the recovery of S-structure. Second, we turn to Japanese and show how simple parametric variation suffices to cover the *Wh*-sentence types in Lasnik and Saito (1984), including LF movement, scrambling, and the like, including some surprising parses. The next section addresses extensions to that system to cover additional Japanese example sentences, such as questions and embedded relatives. It then continues by looking at the computational consequences of the resulting parser, why Japanese is harder even for an LR (bottom-up) parsing machine, and what can be done to improve matters via techniques drawn from programming language compilers.

EFFICIENT PARSING WITH PRINCIPLES: HOW IT CAN BE DONE

The focus of this chapter is not on the structure of the P & P parser per se. However, because understanding how it works is central to what follows, we briefly sketch the overall system here. There is neither time nor space here to justify all the design decisions that were made; complete details may be found in Fong and Berwick (1991, 1994).

Like all the parsers we know of that use some version of transformational grammar in the post-*Lectures on Government and Binding* era, ours carries out the analysis of an orthographic form in essentially two steps. First, one must recover some representation(s) of augmented X–bar phrase structure (either partial S-structures or a pseudo-S-structures), generating all possibilities. This is done by a full LR(1) parser that uses a 30 rule grammar to generate quasi-S-structures. We use the term *quasi-S-structure* because the phrase structure that is generated does not meet all the constraints on S-structure; in particular, empty categories are inserted in all possible locations without further checking and without their features (as traces, *pro*, etc.) being determined. For instance, the first stage LR parser assigns the same structure to *Bill seems to like ice-cream* and *Bill wants to like ice-cream*, inserting a generic empty category as the subject of the complement phrase CP (S).[3] Later principles must fix this as, for example, a trace in the first case, but not the second. In particular, chain formation has not yet taken place. It is this simplification that allows the first stage LR machine to be small and efficient; it does not try to check all principle conditions at once.[4]

This two-stage process seems appropriate simply because the predicates of P & P theories are structural, and it doesn't make sense to apply, for example,

[3]We assume the conventional functional category notation, for example, CP = Complementizer Phrase; IP = Inflection Phrase, etc., and omit indices that indicate head movement for verbs. Spec and head of CP are referred to as Comp (for Complementizer Phrase).

[4]For conceptual purposes, this description has been simplified from that of the actual parser, which in practice permits *interleaving* of some of the principles; see Fong and Berwick (1994) for details.

binding theory without the structures on which c-command is defined. Moreover, by dividing up the work in this way, we appear to gain in overall speed.[5]

The overall conception is based on the Lasnik and Uriagereka (1988) model of free and optional application of all principles (e.g., movement and indexing). We can picture the parser as a generate-and-test device that, searching through the space of all possible quasi-S-structures, produces a stream of candidate S-structures, with (underspecified) empty categories in place, and then funnels these through a cascade of 24 additional principle modules as each candidate emerges.[6] As soon as one candidate fails or makes it all the way through the gauntlet, the system goes back and retrieves another candidate S-structure. What emerges out the back end of this cascade is zero or more LFs (as defined in P & P theory, e.g., with operator-bound variables, coindexed NPs, quantifiers in operator position, etc.).

Although the actual process of running the S-structure parser is itself straightforward, in part because the grammar is small enough to be comparable or smaller than those used for programming languages, what is true is that the S-structure grammar *cannot* miss any candidate examples; this is its contractual obligation. Of course, the fewer candidates it generates, the better. In a later section we describe in detail how this may be done by taking X-bar schemas and systematically adding to them rules that generate gaps and adjoined phrases.

Similarly, the actual declarative statements that implement principles, for example structural case assignment, are completely straightforward. Let us examine in some detail this example, just to dispel any mystery that might remain.

---

[5]If we generate all possible valid X-bar structures in order to check well-formedness at d-structure first, we will waste much time because many of these will be plainly illicit given a particular input sentence (e.g., the complement position cannot be filled by an NP in *John thought*, nor can an empty category be subsumed by an NP). Clearly, constraining information on structure must be brought to bear as soon as possible. Computationally, the picture is more complex. The current parser steers a middle course. It is of course possible to *interleave* structure building and filtering, and there are a number of ways of doing so that we and others have implemented. Briefly, interleaving applies constraints simultaneously rather than sequentially. For instance, in the current system, one can select to interleave any or all principles, but it generally proves most efficient that all s-structures built must also obey the possible constraints of agreement between Subject and Inflection (via coindexing), the licensing of syntactic adjuncts (they all must be one-place predicates, including examples such as *the sad person, the person who I saw*), and S-bar deletion. Then *all* the quasi-s-structures that are generated are guaranteed to obey not only X-bar constraints plus movement, but also these additional three constraints. In many other situations, however, interleaving becomes computationally too expensive given the current design, because the interleaved principle involves too much (wasted) computation over s-structure trees that will eventually have to be thrown away. The exact cost depends in a complex way on the nondeterminancy of the particular s-structure space to be explored, the principle involved, and so on. Thus, although some (Crocker, 1992; Johnson, 1989) have advocated full interleaving as always more efficient, the matter is actually much more delicate. See Fong (1991) and Fong and Berwick (1991) for additional discussion.

[6]This approach is to be distinguished from traditional analysis by synthesis, in that we can have a "smart" generator and tester that need not wait for an entire sentence to be built up before dismissing it.

First, we must state that structural case assignment (sCaseAssign) has been satisfied. This holds whenever, in all tree configurations CF, if CF satisfies the properties of structural Case configuration (sCaseConfig), namely, a relation between CF, the assigner, the case to be assigned, and an NP, then assignment of case (assignCase) also holds between the assigner, case, and the NP:

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

An Example: A Parasitic Gap Sentence

Let us see how this works in more detail via a concrete example, the English parasitic gap sentence (example 17 of chapter 3 in Lasnik & Uriagereka, 1988), *Which report did you file without reading*, to which the parser (and Lasnik & Uriagereka) assigns a single (correct) LF. Following Lasnik and Uriagereka, for this sentence the parser builds an LF with a controlled PRO indexed to *you* as the subject of *reading*, although a pure variable (marked –A(naphoric), –P(ronominal) via the functional determination of empty categories, as adopted here) fills the position after *reading*; a *Wh* trace is the object of *file*, coindexed to the parasitic empty category and to *which report*. Note also that Head movement has taken place: *do* is raised to Infl, so as to receive tense, and then the V-I complex raised and adjoined to C(comp), yielding Subject-Aux inversion. All this detail is captured by the 29 principles shown in Fig. 8.1, shown along the right-hand side of the figure, along with the resulting LF tree output. It seems quite remarkable that among the many thousands of possible principle interactions, exactly one—the right analysis—survives, notably without any special stipulation at all.

Expanding on this overview, observe that a *single* S-structure is recovered by the special-purpose LR machine in Stage I, with generic empty categories already placed in the positions of the object of *file*, the subject of *reading*, and the object of *reading*, and with Head movement and inversion computed (but sans indices for NPs, the identity of the empty categories, etc.). Free (optional) movement blows this up to 49 candidates, which are whittled down to 5; free indexing expands these back to 36, and then the θ-criterion, control, and Condition B cut these back to just a single final LF.

(1) Which report did you file without reading?

(2) Stage I (s-structure with underdetermined empty categories):
$[_{C2}$ $[_{NP}$ $[_{Det}$ *which*$][_{N1}$ *report*$]]$ $[_{C1}$ $[_{C}$ [C] $[_{I}$ I(Agr) $[_{V}$ *did*$]]]$ $[_{I2}$ $[_{NP}$ *you*$]$ $[_{I1}$ $[_{I}$ trace-I $[_{VP}$ $[_{VP}$ $[_{V}$ trace-*do* $[_{VP}$ $[_{V}$ *file* $[_{NP}$ NP-ec±A ± P]]] $[_{PP}$ $[_{P}$ *without*$][_{I2}$ $[_{NP}$ NP-ec±A ± P]$[_{I1}$ $[_{I}$ trace-I $[_{VP}$ [V I *reading*]]$[_{NP}$ NP-ec±A ± P]]]]]]]]]]]]

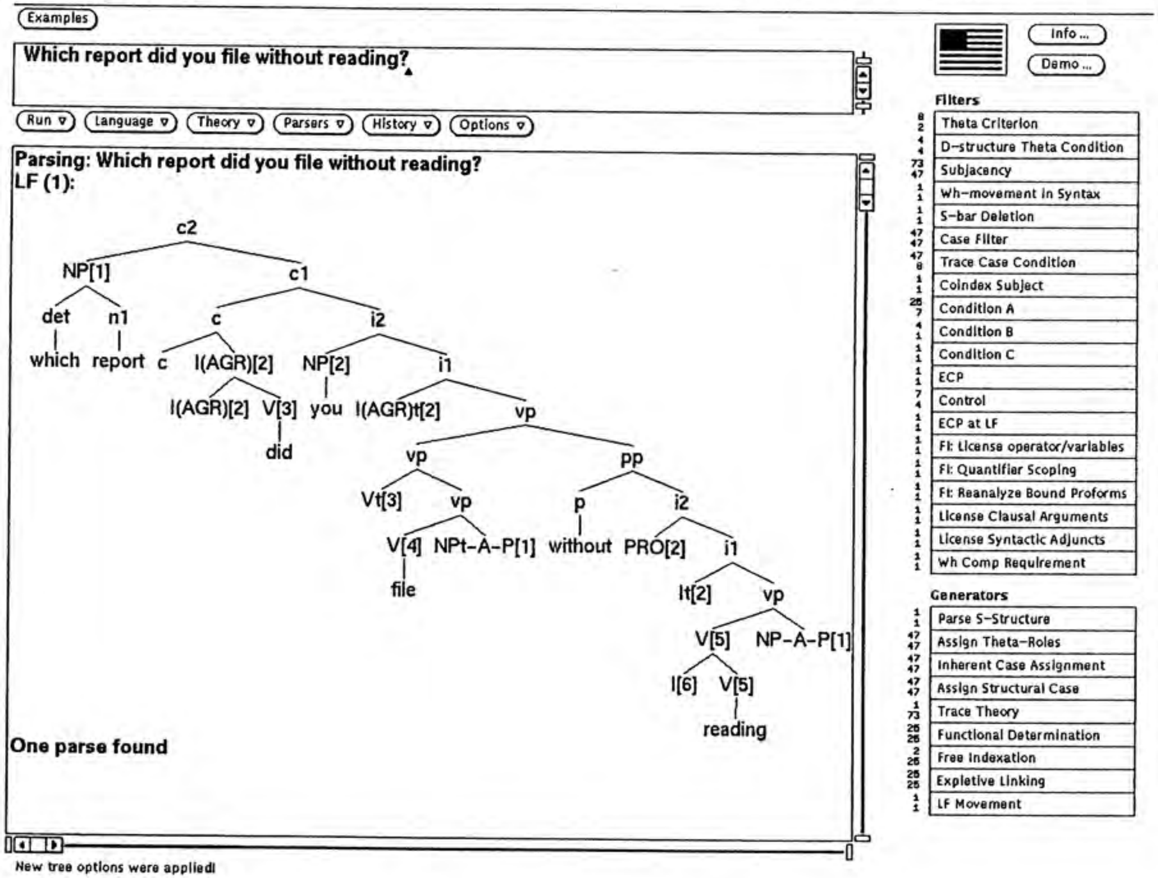

FIG. 8.1. A computer snapshot of the P & P parser in actual operation, processing the sentence, *Which report did you file without reading*. The right-hand side lists the principle filters and generators used, effectively the entire "rule system" at work.

(3) Final LF output:
[$_{\mathrm{CP}}$ [NP [$_{\mathrm{Det}}$ *which*] [$_{\mathrm{N1}}$ *report*]$_1$] [$_{\mathrm{C1}}$ [$_{\mathrm{C}}$ [$_{\mathrm{C}}$] [$_{\mathrm{I}}$ I(Agr) *did*]]] [$_{\mathrm{IP}}$ [$_{\mathrm{NP}}$ *you*]$_2$ [$_{\mathrm{I1}}$ [$_{\mathrm{I}}$ trace I-*do* [$_{\mathrm{VP}}$ [$_{\mathrm{VP}}$ [$_{\mathrm{V}}$ trace-*do* [$_{\mathrm{VP}}$ [$_{\mathrm{V}}$ *file*]]] [$_{\mathrm{NP}}$ NP-t[–A–P]]$_1$]]$_1$ [$_{\mathrm{PP}}$ [$_{\mathrm{P}}$ *without*] [$_{\mathrm{I2}}$ [$_{\mathrm{NP}}$ PRO$_2$ [$_{\mathrm{I1}}$ [$_{\mathrm{I}}$ trace-I [$_{\mathrm{VP}}$ [$_{\mathrm{V}}$ I [$_{\mathrm{V}}$ *reading*]] [$_{\mathrm{NP}}$ NP-A – P]]]]]]]]]]

Between Stage I's S-structure output in (2) and the single, final LF output in (3), there is much work done in Stage II. The reader can follow along by noting the numbers at the top and bottom of each principle box in Fig. 8.1; numbers on top denote number of structures input to a principle module, and numbers on bottom are those that make it through (either generated or filtered).

Briefly, the cascade of the remaining 24 principles runs like this:[7] full interpretation (FI) of syntactic adjuncts at s-structure and S-bar deletion don't weed out any structures.

Next, Move-α applies freely, compositionally computing all possible chains and assigning indices, in this example yielding 49 candidates. The actual algorithm is sophisticated, but the central concept is not. The mechanism used here is essentially to build all possible chains by computing the set cross-product of possible links between existing empty categories and partial chains as the parser walks a tree structure, extending partial chains or not as the parser compositionally traverses the tree structure it has already built, starting with some selected empty category, and keeping track of partial chains built so far as well as remaining free empty NP candidates or a final nontrace NP head for the chain.

This nondeterminism reflects the complete optionality of the Lasnik and Uriagereka system. All movement is optional. Hence, for each empty category, the parser can either do nothing, cause it to participate in an already existing partial chain, make it a 1-element trivial chain, or start a new chain, all nondeterministically. Naturally, this process must meet some constraints; for example, no chain can cross more than one bounding node, which would violate subjacency. In addition, an overt NP optionally heads a chain; an element cannot participate in more than one chain; and all chains must be complete, that is, headed by a nontrace element (hence, an empty Operator can head a chain, for relative clauses).[8]

As a simple example, consider the generic empty NP after *reading*. The parser may freely decide to have this empty NP start a chain, be somewhere in the

---

[7]The principles are ordered as shown in the figure, but not the computer snapshot. Note that the principles may be statically or dynamically reordered, often with significant computational effects. For optimal performance, the operative principle is to delay candidate hypothesis expansion by generating principles like movement as long as possible, and apply filtering principles like the case filter or Condition B as soon as possible, subject to the logical dependencies of the theory. The connection between principle ordering and "guiding principles" such as earliness or least effort in recent linguistic discussion has not escaped our attention.

[8]For reasons of space we must omit here how the parser makes sure it is not building redundant chains as well as checking the *i*–within–*i* condition of circularly referential chains. We have also taken some liberties with the full description of the composition process.

middle of a chain, be a trivial 1-element chain, or decide to do nothing at all with this empty NP. Note that some (movement) indices are now in place. Of course, we must ensure that this Chain Formation algorithm is complete, so that among all these possibilities we generate at least all the licit chains. One result of chain formation is given in (4).

(4) $[_{C2}\ [_{NP}\ [_{Det}$ *which*$]\ [_{N1}$ *report*$]]_1\ [_{C1}\ [_{C}\ [_{C}\ ]\ [_{I}$ (Agr) $[_{V}$ *did*$]]]\ [_{I2}\ [_{NP}$ *you*$]_2\ [_{I1}\ [_{I1}\ [_{VP}\ [_{VP}\ [_{V1}\ [_{VP}\ [_{V}$ *file*$]\ [_{NP}$ NP-t[–A–P]$]_1]]\ [_{PP}\ [_{P}$ *without*$]\ [_{I2}\ [_{NP}$ NP-t[–A–P]$]_1\ [_{I1}\ [_{I1}\ [_{VP}\ [_{V}$ I $[_{V}$ *reading*$]]\ [_{NP}$ NP-–A – P]]]]]]]]]]]]

Next, these chain-augmented structures are assigned structural and inherent case, and run through the case filter (in this example, with no effect on eligible candidates). Of these 49 different chain outputs with case now assigned, all but 5 will pass through both the trace case condition (TCC: NP traces cannot have case; *Wh* traces must have case) and subjacency, with θ-Roles being assigned in between these two constraints, and *Wh*-movement in syntax checked (as appropriate for the language).[9] Specifically, the fourth chain output from these two constraints will ultimately prove to be the winning structure (though the system cannot know that yet, of course). This has a single chain linking the object of *file* to *which report*, and leaves the subject and object of *reading* as unspecified empty categories.

Proceeding, free indexing greatly expands these 5 possibilities to 36, by computing compositionally the number of ways of dividing n elements (the NPs) into m distinct sets (the indices and unindexed NPs).[10] Thus, the NP empty category in subject position may be linked to *which report*, or to *you*, or be arbitrary, etc. The traces and empty categories are then instantiated with anaphoric and pronominal feature values via Functional Determination, which uses local context to determine the features ±A(naphoric) ±P(ronominal), again following Lasnik and Uriagereka. Here, for the correct LF, it determines that the subject of *reading* is a PRO, the trace is –A–P, and the object of *reading* is simply an empty category marked –A–P, hence a pure variable.[11]

Finally, control theory cuts these 36 structures down to 10 (ruling out uncontrolled PRO); passing through the θ-criterion, binding conditions A, B, and C, the ECP, LF movement, and full interpretation at LF, the parser eliminates all but 1 of the viable candidates, striking out (via Condition B for the most part) cases where

[9]The reader may note that in this example then it might have been fruitful to reorder Subjacency ahead of case or theta-role assignment after TCC, saving some work as suggested earlier because fewer candidates will have survived.

[10]See Fong (1989, 1991) for mathematical details of this procedure.

[11]This algorithm, as given in Lasnik and Uriagereka, just happens to be deterministic for English, which does not have *pro*-drop, but it is nondeterministic for Japanese. Of course, our selection of functional determination is a theoretical choice that can in fact be altered; the parser would then simply be using a different grammatical theory.

PRO is bound in its minimal governing category. Out of many hundreds of possible interactions, the single right result emerges, as displayed earlier.

Having surveyed how a basic sentence parse works, we now return to examine how some of the parser modules work in detail, focusing on the recovery of S-structure.

Building Phrase Structure

How does this system actually get its job done? As mentioned earlier, the central problem for a P & P model is building some scaffolding on which to hang the predicates of the theory. The key idea in solving this structural bootstrapping problem for the first stage of P & P parsing, which will prove crucial for our later discussion of parameterization and Japanese parsing, is the formulation of an intermediate *covering grammar* that builds an underspecified s-structure (including underspecified empty categories), but overgenerates, constructing at least all the legal output phrase structures.

This method is strongly reminiscent of solutions proposed in the 1960s in the MITRE (Zwicky, Friedman, Hall, & Walker, 1965) and Petrick (1965) parsers to solve a similar problem with *Aspects*-style TGs. Briefly, the notion is to build a (small) phrase structure grammar for s-structure by writing phrase structure rules for d-structure and then adding rules to account for adjunction and movement, both at d-structure and s-structure. This results in 31 rule schemas in English, which are further augmented by additional context tests into 74 augmented context-free rules. Fig. 8.2 displays an overview of this process, as summarized in Fong (1991).

Interestingly, the Zwicky et al. approach also built an initial covering grammar of 30–40 rules. The difference, however, is that the older procedure was meant to build the structural descriptions required for old-style transformations, which required elaborate contexts and proper ordering for their application. What has changed is that such rules can now be disposed of; only declarative constraints

PS Rules at s-structure=

PS Rules at d-structure

$\overline{\mathrm{X}}$-rules: e.g. $\overline{\overline{\mathbf{X}}} \rightarrow Spec, \overline{\mathbf{X}}$
$\mathbf{X} = N, Spec = Det$
$[_{NP}\ [_{Det}\ the]\ [_{\overline{N}}\ destruction\ of\ the\ city]]$

Empty category rules: e.g. NP $\rightarrow \lambda$
$[_{IP}\ [_{NP}\ NP\text{-}ec \pm \mathrm{A} \pm \mathrm{P}\ [_{VP}\ was\ followed\ John]]$

Adjunction rules: e.g. VP $\rightarrow$ VP, Adv
$[_{VP}\ [_{VP}\ was\ followed\ John]\ [_{Adv}\ why]]$

plus
Rules to account for movement

e.g. CP $\rightarrow$ Adv, $\overline{\mathrm{C}}$; Adv $\rightarrow \lambda$
$[_{CP}\ [_{Adv}\ why][_{\overline{C}}\ was\ John\ followed]]$

FIG. 8.2. Components of the phrase structure grammar for s-structure.

need be applied after the initial structures are recovered. For example, as is well known, we have no separate rules for passive, raising, and the like, but just a single movement operation. In addition, we employ more powerful LR parsing and compilation techniques in the first place. Both of these innovations let us build a system with considerably broader cross-linguistic coverage.

More specifically, D-structure phrases are comprised of two parts: the instantiated X–bar rules for a given language, as given by the values of the X–bar parameters in that language; plus empty category rules and adjunction rules, introducing empty NPs (NP-ec±A ± P *was followed John*) and, say, empty elements for Adverbial adjunction ($[_{VP}$ *was followed John*$][_{Adv}$ *why*$]$). Let us cover the basic X–bar schemas, and then turn to adjunction and empty category rules. Then, having completed the description of d-structure, we consider next s-structure augmentations.

Basic X–bar Schemas. The X–bar schemas are just what one would expect. They use unordered right-hand sides and binary branching. We assume for this parser that subjects are in Spec of IP. Parameters are incorporated by adding constraints on schemas that are automatically expanded. For example, the rule that reads something like, "XP derives X1 followed by the specifiers of X1 if the parameter specFinal holds such that XP is a maximal projection and X1 immediately projects to XP" can be written as in (5), where we leave undefined the obvious auxiliary predicates. Note how the language particular parameter specFinal enters in.

(5) rule XP $\rightarrow$ [X1|specifiers(X)] ordered specFinal st max(XP), proj(X1, XP).

A schema compiler turns this form into an actual context-free rule essentially by instantiating the theta-grid for a lexical item, letting *X* range over the requisite lexical categories plus the parameter settings, valid specifier and complement structures for particular lexical categories and items based on thematic roles. For instance, lexical V for V = *persuade* in English forces the addition of the instantiated schema in (6).

(6) VP $\rightarrow$ NP CP

Further dummy rules are added to factor in subcategorization and other "top-down" Head selectional properties. We return to the question of implementing these sorts of constraints in the next subsection, which will prove important in Japanese.

Addition of Empty Categories and Adjunction at D-Structure. To this set of basic d-structures we add any schemas that yield empty elements at d-structure (e.g., rules such as NP $\rightarrow \lambda$) and schemas that yield adjoined structures. Where can these occur? Empty NPs at d-structure can be subjects (as in *I want* e.c. *to like ice-cream*) or as parasitic gaps. C (Comp) and I (Inflection) can also be empty, as is familiar, due to I-movement and the like. Turning to adjunction, adopting van Riemsdijk and Williams' (1986) analysis, nonmovement adjunction is required for relative clauses and PPs (as in *the guy who likes ice-cream* or *book on the table*). Following Lasnik and Uriagereka, *Wh*-adverbs such as *why* are adjoined to VP at d-structure and then fronted to clause initial position (e.g., [*John* [$_{VP}$ [$_{Adv}$ *why* [$_{VP}$ *leave*]]]]). This completes the description of the d-structure covering grammar; next we must augment this with s-structure conditions, such as movement.

Addition of Movement. In general, Move-α allows any phrase to move anywhere. This is too underconstrained, as has been noted in the literature. In this respect the linguistic theory is incomplete. For the system described here, in order to maintain computational efficiency only the core examples of movement have been implemented: Head movement, adjunct fronting, and general movement of NPs.

Nonlocal Movement. The Lasnik and Uriagereka principles and parameters theory, like many others, divides the possible types of movement into various landing and launching sites: It considers argument (A-positions, such as subject or object) and nonargument (A–bar) positions as the main subcases for movement. *Wh*-movement is the classic example of A-to-A–bar movement: The NP moves from an argument position (such as the object of a verb) to a Comp position, as in *What*$_i$ *did you eat* e$_i$. Passive and raising are the classic examples of A–to–A movement, as in *John seems to be happy*. Adverbial movement is A–bar–to–A–bar: *why* can front in *You like ice-cream why* to *Why do you like ice-cream*. These possibilities are already covered—that is, nondeterministically generated—by existing d-structure rules because all A-positions (subject and object positions) and Specifier of Comp admit empty NPs at D-structure.

Scrambling. Let us turn next to scrambling. Although this matter is currently an open topic in linguistic theory,[12] for concreteness we adopt the approach of Hoji (1985). This approach uses adjunction of NPs at VP and IP. Hoji's four examples, augmented with traces, for the Japanese equivalent of *John gave Mary a book* show this kind of adjunction. The s-structure grammar covers these simply by adding adjunction rules like (7).

(7) a. VP $\rightarrow$ NP VP
    b. IP $\rightarrow$ NP IP

These adjunction structures cover Hoji's example sentences such as those in (8).

[12]For example, some theorists hypothesize that there is both short and long distance scrambling.

(8) a. [$_{IP}$ John-ga [$_{VP}$ Mary-ni hon-o ageta]]
b. [$_{IP}$ hon$_i$-o [$_{IP}$ John-ga [$_{VP}$ Mary-ni t$_i$ ageta]]]
c. [$_{IP}$ Mary$_i$-ni [$_{IP}$ John-ga [$_{VP}$ t$_i$ hon-o ageta]]]
d. [$_{IP}$ John-ga [$_{VP}$ hon$_i$-o [$_{VP}$ Mary-ni t$_i$ ageta]]]

Head Movement: Inflection or I and Verb Movement. Here we basically follow the treatment in Chomsky (1986), and assume that verbs may raise and adjoin to an inflectional element (Infl) or I can lower to the verb; for reasons of space we omit details of how this is implemented.

Adjunct and Adverbial Movement. Finally, schemas must also be added to handle adjunct movement at s-structure (e.g., the fronting of *Wh*-adverbs such as *why*, as in *why did John move*). The parser uses the position of Comp at s-structure for adverbs. A second, Head position is used as the landing site for zero-level categories.

This completes the outline of how the s-structure covering grammar is built. The actual phrase structure grammar constructed for s-structure is language dependent. For instance, the X–bar schema expansions will be different according to the values of parameters such as [±specInitial,±headInitial], and the lack of *Wh*-movement in syntax will bar certain movements in Japanese. Table 8.1 summarizes the system for the two grammars.

Precisely because the resulting grammar is small—one of the properties of P & P theories is to partition constraining work among different modules—we have only 31 covering grammar rule schemas for English and 25 for Japanese. We should emphasize this key property of a modular system: The small grammar size allows us to consider more powerful computational engines than are ordinarily deployed. In particular, we employ an algorithm that has provably optimal early error detection—what is called in the computer science literature a *canonical* LR(1) (bottom-up, 1 token of lookahead) parser. Early error detection is a must for a system that has to dispose of bad candidate structures as quickly as possible. In the next section we show how the system automatically compiles these rules into the form used by the LR parser.

Modifying the LR Grammar and Parsing S–structure

Let us now turn to how the system parses with this grammar, and what computational problems arise. The current system uses an augmented canonical LR parser. While LR(k) parsers are the the largest class of *deterministic* shift-reduce (bottom-up) parsers, we need to augment the basic LR machinery to handle ambiguity, factor in top-down information, and use arbitrary lookahead. Because these modifications play a role in the rest of the chapter, we review the relevant changes here.

TABLE 8.1

Language	X–bar-Prototype Rules	Empty Category Rules	Adjunction Rules	Others
English	12	5	8	6
Japanese	12	4	7	2

One problem with left–to–right, bottom-up parsers is that they do not make efficient use of top-down constraints or information "on the right" that could eliminate dead-end computations. This, after all, is what the head-first/head-final problem is all about. What can be done about this here? (Later we shall see how well these techniques work empirically in English and Japanese.[13])

Consider first the question of verb subcategorization and selection. Given the various expansions of VP, it would be inefficient to hypothesize all possibilities for every verb, because it might posit empty categories fruitlessly. The resulting machine would do extra work, inventing empty categories to try to match possibilities such as $V' \rightarrow V, NP$; $V' \rightarrow V$; $V' \rightarrow V, NP, CP$, etc. To avoid this, the system adds an extra condition to the X–bar instantiations, based on theta grids available from the lexicon. For example, the system adds to the rule $V' \rightarrow V$ for $V =$ *sleep*, the condition that no objects can follow.

Similarly, consider an example such as *John visited Mary*. When the parser was first designed, it was found that the system attempted to build relative clauses with *John* as the head, and then added either nothing (as in *the guy*[CP]), or *visited* (as in *the guy* [*who*] *visited*), or *visited Mary* (as in *the guy* [*who*] *visited Mary*). These partial analyses will all fail because there would be either an empty CP or no V in the matrix CP, but they cause much wasted effort.

In short, some way was needed so that the parser could "lift" constraints on the right *up* the tree being constructed to be tested immediately, rather than

[13]We first should review a common misconception about LR parsers: that they somehow have difficulty with left-branching as opposed to right-branching phrase structures. This has even been advanced as an argument for parameterizing parsing algorithms in different ways for left-branching and right-branching languages, with top-down parsing being more appropriate for right-branching, etc. Although this distinction may be true of pure bottom-up parsers, for LR machines it is, strictly speaking, incorrect: The class of languages parsable by LR grammars properly subsumes that of the deterministic top-down (LL) parsers. Further, the way that the LR parser is built does incorporate information about what is at the end of a phrase structure rule (e.g., the LR machine will contain a state $VP \rightarrow \bullet NP_1$ that is distinct from the state $VP \rightarrow \bullet NP_1NP_2$, etc.). Note that these states are predictive: One branch claims that an NP will follow; the other, that two will. The distinct states are built into a finite control table. In brief, LR machines do not have more difficulty with either right-branching or left-branching phrase structures. Of course, it may be that the stack depth is different in the parser using one grammar than another, and that other, finer complexity distinctions emerge, as we shall see. However, it is easy to construct simple grammars demonstrating that left-branching and right-branching languages are equally easily parsed by LR, or even simple LR (SLR) machines. We have not investigated this in full detail, but the LR machines for both English and Japanese must both, at times, stack 10 to 20 items, even though Japanese appears to be somewhat worse in this regard.

waiting until these right-hand elements were actually traversed in the input. Again, note that this is precisely the matter of how to handle the left-to-right character of a language, a problem that arises in Japanese.

The current implementation solves this problem by using a standard trick from programming language compilers to add extra conditions on rules. The system introduces a rule with an added pseudo-nonterminal condition (not a real phrase name), here, checkInput.

(9) V′ $\rightarrow$ V takesCPObject(top-of-stack(SS)) NP

(10) CP $\rightarrow$ checkInput(Input) Spec C′

In fact, what we are really doing is "flipping" the phrase structure around so that the information will now be at the lefthand edge, rather than at the right. Thus, this is a pseudogrammatical transformation (which does not really alter the grammar). The new predicate will do the required work (check if the top of the s-structure stack takes a CP Object and if the input contains the appropriate elements *anywhere* to the left or right). It in effect acknowledges the left-to-right bias in the order of tokens in the input stream.

Some care is needed in implementing this. Attaching the condition *directly* to a bottom-up parser will not work, because we want to check the CP or VP *before* we have done any work on the rest of the right-hand side of the rule. Here, too, we use a standard programming language technique of inserting a *dummy production* that does not build any input but has the side effect of testing for the condition in a top-down way (we do not go into the necessary implementation details about the stack machine to get this to work).

(11) a. CP $\rightarrow$ Dummy Spec C′

b. Dummy $\rightarrow \lambda$ if checkinput(Input)

When the dummy node is completely built (immediately, without consuming any input, because it builds nothing), it will also invoke the action clause checkInput(Input), testing whatever we want. In this case, the action will be to scan in the input to the right for a verb. Note that the dummy node is at the left-hand edge of the rule. This will have the desired effect even in a head-final language, for even if the head is last, the information requested has now been passed to the front of the expansion. Other restrictions on the right can be imposed in a similar way. Any movement to the right will require some kind of licensing of this kind (e.g., in Verb Inflection adjunction structures). The rule that introduces an empty I, leaving a trace, must look ahead in the input for licensing. In a head-final language like Japanese, this *must* happen when a verb raises to Infl. The necessity of this approach in Japanese for other constructions will become apparent in our experiments in a later section.

PARSING JAPANESE: LASNIK AND SAITO

We begin with a very simple parameterization of Japanese that will nonetheless be able to cover the Lasnik and Saito *Wh*-questions, scrambling, and so forth.[14]

We consider first the *Wh*-movement sentences found in Lasnik and Saito (1984). These sentences are listed in Fig. 8.3 and display many familiar of the typological Japanese–English differences. Let us review some of these:

• *SOV Language.* As is familiar, Japanese is often classified as a verb final or SOV (Subject–Object–Verb) language. Heads such as verbs and adjectives are preceded by their objects and modifiers. However, subjects do normally appear before verbs and objects, as in English. This distinction can be encoded by two binary parameters that specify head/complement and specifier/head order. The X–bar system compiles out schemas with C, I, N, V, etc. last rather than first. We also assume, without further discussion, the existence of a VP node.

• *Scrambling.* Japanese phrase order is more or less free, apart from the Verb final constraint. Direct and indirect objects may be interchanged, and appear before the subject in an initial position (which is evidently not a process of topicalization). We saw earlier the examples of *John gave Mary a book* from Hoji (1985). Movement can account for such examples. Suppose the canonical order is subject followed by indirect object followed by direct object. The direct object, *hon-o* in this case, is free to move (by VP-adjunction) to a position in front of the indirect object, *Mary-ni*, as in the fourth example in (8), or to a sentence initial position (by S-adjunction), as in the second example. Similarly, the indirect object may move to a sentence initial position as in the third example. We take the elements *ga*, *o*, *ni*, etc. to be essentially case-marking, clitic-like particles that do not project to phrases. (We shall see that in more complex examples, the scrambled element can itself be further moved at LF.) In addition, we alter structural case assignment slightly to transmit case from an A to an A–bar position, since a scrambled NP will be adjoined to VP, and would otherwise be unable to receive case. (This is a temporary move that we have used pending a better account.)

• *Empty subjects.* As is also familiar, Subjects and other NPs can be omitted in a super *pro*-drop language like Japanese. (In general the conditions that determine which elements can or cannot be omitted are largely dependent on discourse considerations, which are not considered here. However, as pointed out earlier, the system can be modified to take context into account in a general way, if a theory of context becomes available.) As an example, consider (12b), taken from Makino and Tsutsui (1986).

---

[14]We scramble only from direct object positions here, even though it is straightforward to scramble from indirect object positions. Informally, we have noted that scrambling from the IO position greatly increases computation time. A tighter set of constraints on scrambling seems to be called for.

(2)	Watashi-wa Taro-ga nani-o katta ka shitte iru *(I know what John bought)*	*Basic* wh-*questions*
(6)	Kimi-wa dare-ni Taro-ga naze kubi-ni natta tte itta no *(To whom did you say that John was fired why)*	*Good in Japanese but not in English*
(32)	*Meari-wa Taro-ga nani-o katta ka do ka shiranai *(Mary does not know whether or not John bought what)*	*Semantic parallelism: non-absorption of* ka do ka
(37a)	Taro-wa naze kubi-ni natta no *(Why was John fired)*	
(37b)	Biru-wa Taro-ga naze kubi-ni natta tte itta no *(Why did Bill say that John was fired)*	*Comp-to-Comp movement at LF*
(39a)	Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no *(What are you so angry about the fact that Taro obtained)*	*Complement-noncomplement asymmetries*
(39b)	*Taro-ga naze sore-o te-ni ireta koto-o sonnani okotteru no *(Why are you so angry about the fact that Taro obtained it)*	
(41a)	Hanoko-ga Taro-ga nani-o te-ni ireta tte itta koto-o sonnani okotteru no *(What are you so angry about the fact that Hanoko said that Taro obtained)*	
(41b)	*Hanoko-ga Taro-ga naze sore-o te-ni ireta tte itta koto-o sonnani okotteru no *(Why are you so angry about the fact that Hanoko said that Taro obtained it)*	
(60)	Kimi-wa nani-o doko-de katta no *(Where did you buy what)*	*Multiple-whs in Comp*
(63)	Kimi-wa nani-o sagashiteru no *(Why are you looking for what)*	

FIG. 8.3. *Wh*-movement examples in Japanese from Lasnik and Saito (1984).

(12) a. Taro-wa sono mise-de nani-o kaimashita ka
'What did Taro buy at the store?'

b. Pen-o kaimashita.
'He bought a pen.'

In the standard theory, the omitted subject is actually represented in the syntax by an empty pronoun, *pro*.[15] Again following conventional practice, we represent the binary option that determines whether *pro* is available or not as the *pro*-drop parameter.

• *No visible Wh-movement.* Following Lasnik and Saito and other recent work, we assume that Japanese LF looks like English LF: There is no *Wh*-movement in the syntax, but there is movement at LF. Thus, the option of whether to allow *Wh*-movement between d- and s-structure is a parameter. As we review later, it is this distinction that enables Lasnik and Saito to explain a variety of facts, including why the counterpart of a sentence such as (6) in Fig. 8.3 (which is well formed in Japanese) is ill formed in English.

• *Wh-Movement at LF.* We follow standard principle-and-parameter theory arguments in moving a *Wh* in situ at s-structure to a presentential scopal position at LF. In a sentence such as *watashi-wa Taro-ga nani-o katta ka shitte iru* 'I know what John bought', as shown in the computer output in Fig. 8.4, the question word *nani* is moved at LF to a position that has scope over the embedded

[15]We follow Takezawa (1987) in making this empty category a small *pro*. This option is evidently not available in English.

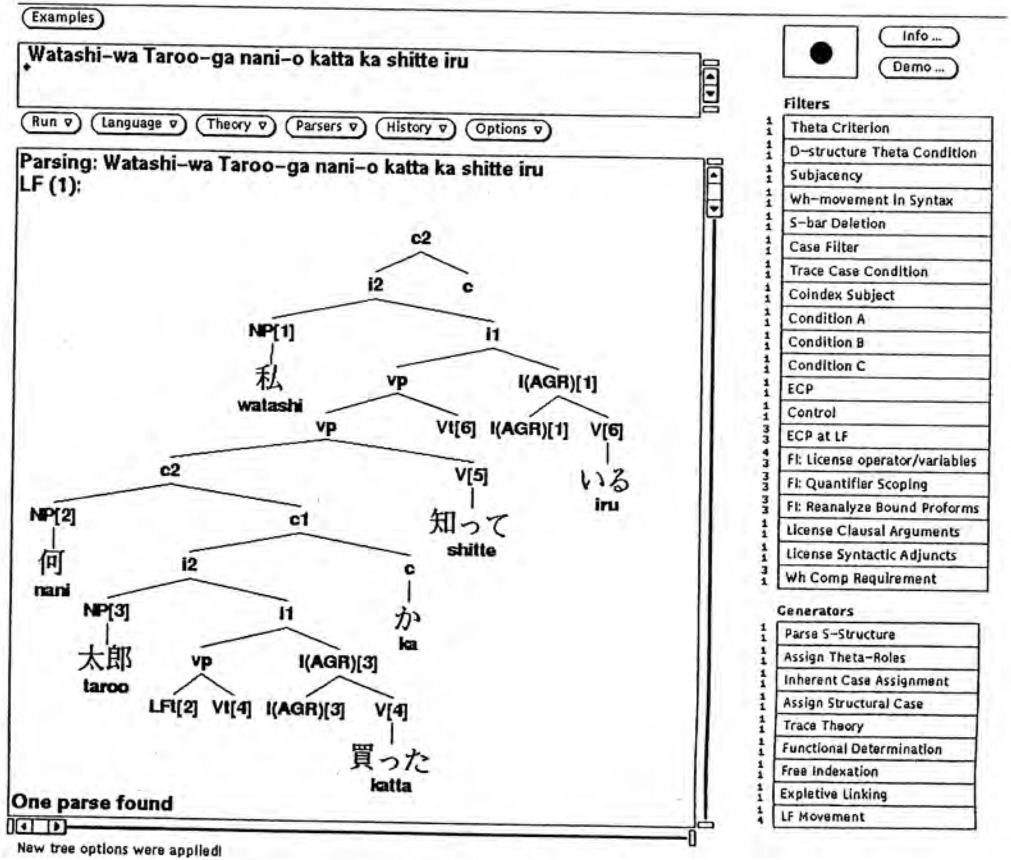

FIG. 8.4. Computer snapshot the parse of the Japanese sentence *watashi-wa Taro-ga nani-o katta ka shitte iru* 'I know what John bought'.

sentence (as indicated by the bracketing) leaving behind an (LF) trace LF-*t* to be interpreted as a variable in its original position. Because we want our Japanese and English grammars to be as uniform as possible and follow a full CP/IP system, we deviate from Lasnik and Saito's approach that puts the *Wh* element in a single (head of) Comp position. Instead we move it to a Spec of CP position. Additional constituents moved at LF are adjoined to this Spec position. In this example the question particle *ka* fills the Head of the embedded CP (=C2), and *nani* fills Spec of this CP the left, immediately after the C2 bracket.

To be sure, this is not in any way meant to be a complete characterization of the differences between these two languages. We defer for now all the intriguing questions of case marking, passives, causatives, and so forth. Rather, it is designed to be sufficient to demonstrate what we set out to show: to cover the examples shown in Fig. 8.3 with just a handful of parameter switches, literally as shown in Fig. 8.5, and provide the groundwork for the computational experiments in the next section.[16]

Even so, it is intriguing that the same set of principles for English recombine in different ways to handle the Japanese examples. The important point here again is that the system gets (by design) precisely the parses required in Lasnik and Saito, and blocks ungrammatical sentences by the same means as well.

PARSING JAPANESE: THE COMPUTATIONAL EFFECTS OF SCRAMBLING, *PRO*–DROP, AND PHRASE STRUCTURE

With the sketch of English–Japanese parameterization behind us, in this section we turn to the investigation of the *computational* differences between the two languages that we have explored: How do English and Japanese differ with respect to their difficulty for parsing? As a simple source of examples, we took sentences from Hosokawa (1990). For the most part, with the exception of our discussion of center-embedded/left-branching constructions in the next section, we follow the simple examples as they appeared in that paper, describing various problems that arose.

In the discussion that follows, we shall need to draw on comparisons between the complexity of different parses. While this is a delicate matter, there are two obvious metrics to use in comparing this parser's complexity. The first is the total number of principle operations used to analyze a sentence—the number of s-structures, chain formations, indexings, various constraint applications, etc. We can treat these individually and as a whole to give an account of the entire "search space" the parser moves through to discover analyses. However, this is often not a good measure of the total time spent in analysis, because some operations take

[16]Some of these parametric variations lead to implicational universals. For example, Lasnik and Saito (1987) states that if a language has syntactic *Wh*-movement then it obeys the *Wh*-Comp requirement at s-structure. See discussion later in text.

	English and Japanese parameter settings	
	English	Japanese
Spec order	specInitial.	specInitial.
	specFinal :- \+ specInitial.	specFinal :- \+ specInitial.
*Head order	headInitial.	headFinal.
	headFinal :- \+ headInitial.	headInitial :- \+ headFinal.
Agreement	agr(weak).	agr(weak).
Bounding	boundingNode(i2).	boundingNode(i2).
	boundingNode(np).	boundingNode(np).
*Case Adjacency	caseAdjacency.	:- no caseAdjacency.
**Wh* in Syntax	whInSyntax.	:-no whInSyntax.
*Pro-Drop	:- no proDrop.	proDrop.

FIG. 8.5. The differences between English and Japanese are captured by just a few parameter switches, shown here as actually written in the Prolog program. Distinct parameter settings for the two languages are marked by the four asterisks.

more time than others. The second measure we use is more particular and precisely tailored to the specific backtracking-LR design we have built to recover structural descriptions: We can count the total number of LR finite-state control steps taken in recovering the s-structure(s) for a given sentence. We shall more often rely on this measure.

A third, obvious, and perhaps even more accurate alternative is to time the overall parse and individual principle modules. While this is a legitimate approach, we have decided to avoid machine-dependent timing for the moment. In addition, we have found informally that such timing, where it is stable, is highly correlated with our other measures.

Turning now to the sentences themselves, to demonstrate how we can extend the P & P system beyond the Lasnik and Saito examples, we illustrate some of the (sometimes slightly modified) sentences covered in Hosokawa and their parses. (Again we stress that this is not meant in any sense to be complete, but rather a demonstration of how to build a P & P system that covers multiple languages. Enough questions arise even with this small additional sample to raise many interesting issues.) Fig. 8.6 lists the sentences discussed in this section. In the next section we turn to a more general computational optimization analysis.

The first two example sentences illustrate marking by the particle *no*. (For a computer snapshot of the parser analyzing Japanese, see Fig. 8.7 in the next section.) In the first example, *no* marks *linguistics* with genitive case (effectively nominalizing it, hence the parser displays it as an NP), while *student* is marked nominative, and so on; branching is to the right, and I lowers to V.[17]

[17]In fact, two LFs are output because free indexing operates purely syntactically. The second LF, with distinct indices on all NPs is the correct one. However, the first LF shown, with the two NPs, *linguistics* and *cheese*, both assigned the same index, 2, is not blocked, because *linguistics* is A-free in its minimal nontrivial chain. Thus, without any syntactic principles to block it, the indices *i* and *j* here may be equal, which is of course anomalous. Similarly, in the English *the rat the cat killed ate cheese*, *cat* and *cheese* yield two parses, one where *cat* and *cheese* receive the same index. As we discuss later, this "problem" is one repaired by a (tacit) assumption of the Lasnik and Saito

(1b)′	Gengogaku-no	gakusei-ga	tiizu-o	tabeta			
	linguistics-Gen	student-Nom	cheese-Acc	eat-Past			
	'A student of linguistics ate cheese'						
(2b)′	Nagai	kami-no	gakusei-ga	tiizu-o	tabeta		
	long	hair-Gen	student-Nom	cheese-acc	eat-Past		
	'A long haired student ate cheese'						
(3b)	Taro-ga	hon-o	katta				
	Taro-Nom	book-Acc	buy-Past				
	'John bought a book'						
(4b)	Taro-ga	Hanoko-ni	hon-o	ageta			
	Taro-Nom	Hanoko-Dat	book-Acc	give-Past			
	'Taro bought Hanoko a book'						
(5b)	Taro-ga	hon-o	table-no	ue-ni	oita		
	Taro-Nom	book-Acc	table-Gen	top-Dat	put-Past		
	'Taro put the book on top of the table'						
(6b)	Taro-wa	gakkoo-ni	itta				
	Taro-Top	school-Dat	go-Past				
	'Taro went to school'						
(15b)	Watashi-wa	Taro-ga	nani-o	katta ka	shiranai		
	I-Top	Taro-Nom	what-Acc	bought-Ques	know-not		
	'I don't know what John bought'						
(17b)	Taro-wa	Chomsky-no	*Barriers*-o	yomimashita ka			
	Taro-Top	Chomsky-Gen	*Barriers*-Acc	read-Past-Ques			
	'Did Taro read Chomsky's *Barriers*'						
(18b)	Hanoko-wa	Biru-ga	Chomsky-no	*Barriers*-o	yonda ka-do-ka	shiranai	
	Hanoko-Top	Bill-Nom	Chomsky-Gen	*Barriers*-Acc	read-Ques	know-not	
	'Mary does not know whether or not Bill read Chomsky's *Barriers*'						

FIG. 8.6. A list of the sentences from Hosokawa (1990). Numbering corresponds to the original article, with primes denoting slight modifications or extensions of fragments to full sentences; we drop the primes in subsequent discussion.

As for the second sentence, *long haired student ate cheese*, note that *no* marks the entire nominal clause for genitive case, as desired. We also get two parses as before.[18]

The third and fourth sentences pose no apparent difficulties, each producing a single (correct) analysis. (The system could also parse a sentence with the direct object position scrambled, given our limited implementation.)

---

account of relative clauses as open sentences with a bound variable corresponding to the head of the clause (that is, $\lambda(x)P(x\ \mathit{atethecheese}).\mathit{the\ rat}$). Ordinary locality considerations of such logical forms would rule out the odd bindings, and we implement this approach later.

[18]A direct counterpart of the English possibility "student with long hair" ("long hair with student") evidently doesn't arise in Japanese (Miyagawa, personal communication). We may suppose that this latter possibility is excluded because *no* is not acting as a true postposition, that is, roughly as it is in English, but rather is clitic-like in nature, like the other particles. The reason that *no* must appear on each element in the nominal is left as a mystery, of course.

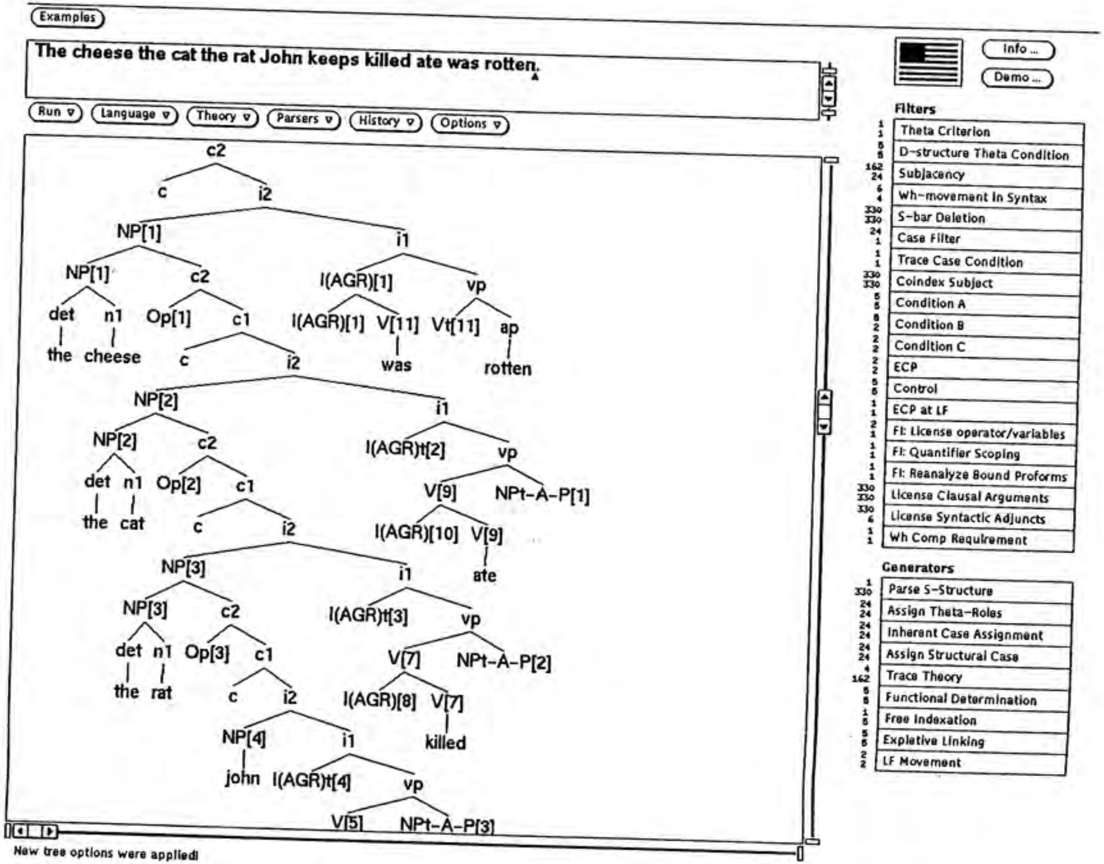

FIG. 8.7. A snapshot of the parse of the triply center-embedded English sentence, *The cheese the rat the cat John keeps killed ate was rotten.*

The fifth sentence, (5b), illustrates both scrambling (with the NP trace of *hon* 'book', moved from its canonical direct object position) and an extra parse arising from the "accidental" coindexing of *hon* and *table*.[19]

Sentences (6b) and (15b) pose no new problems, but example sentences (17b) and (18b), both yes–no questions, do. When the system initially tried to parse these sentences, it failed on all of them. The Wh-Comp Requirement blocked the correct parses. Sentence (17b) is typical. The problem is that there is a *ka* at the end of the sentence, a Q(uestion) marker, but no *Wh* element to pull out to Spec of the matrix CP. Thus the Wh-Comp requirement, which requires both spec and head to be marked, is not satisfied. We must artificially turn off the Wh-Comp requirement in such examples. The same tactic was used for other yes–no questions, which will all otherwise fail because there is no +*Wh* element to move to Spec of CP, again violating the Wh-Comp requirement. A fix is plainly in order here. In English, with no explicit Q marker, we simply marked the Comp +*Wh* if the Spec was also, because a *Wh*-phrase will be there in a matrix *wh*-question at s-structure. In Japanese, we need something like an abstract Q operator with a *Wh* feature to satisfy the Wh-Comp requirement, but so far this has not been implemented.

PROCESSING COMPLEXITY: A CASE STUDY

Given this initial set of analyses, let us now examine the complexity of Japanese sentence processing as compared to English. To do this, we initially examined sentences that we thought would highlight the ease of Japanese relative to English, namely, the "classic" English center-embedded versus the Japanese left-branching constructions from Kuno (1973).

(13) a. The cheese the rat the cat John keeps killed ate was rotten

b. Taro-ga katte-iru neko-ga korosita nezumi-ga tabeta
John-subj keeps cat-subj killed rat-subj ate
tiizu-wa kusatte ita
cheese-topic rotten was

On the conventional Chomsky–Miller account, the English construction is very difficult to parse, while the left-branching Japanese form is completely understandable. Does the same hold for our parser? The answer, initially, is No.

Why should this be? On a modern analysis, and the one adopted here, recall that restrictive relative clauses such as *the rat the cat killed* are open sentences, and so contain an Operator-variable structure indexed to *the rat*, roughly as in (14).

(14) $[_{NP}$ $[_{NP}$ the rat$]_1$ $[_{CP}$ Operator$_1$ . . . the cat killed NP-t[−A−P]$_1$]]

[19]Our LF Binding Condition C requires that an R-expression be A-free in the domain of the head of its nontrivial chain. In the example, *hon* is in a VP-adjoined, an A–bar position, so *table* is in fact A–bar-bound, not A-bound. Thus it is A-free, as required.

We assume that empty operators are base-generated in A–position and fronted by Move-α (Chomsky, 1986).

To the best of our knowledge, no P & P based parser had ever attempted to analyze sentences this complex, with a triple center-embedding. In this case, 904 possible indexings are tried before the single correct parse is discovered.

What of Japanese? Our initial parse is shown in Fig. 8.8 (see also the statistics in Fig. 8.9). We expected this sentence to produce a single parse, and it does. The P & P model still worked. (Remember that we changed just a few parameters to get this radically different structure to come out.) However, when we examined the number of computational operations (measured by LR transitions), the result was quite surprising: about 207,000 LR operations, compared to about 18,000 for the English version. Thus, according to this metric, the English center-embedded sentence is much *simpler* than the Japanese—an unexpected result. (Note that the simple Japanese sentence and one center-embedding are simpler in Japanese than in English; the effect appears only after two embeddings.)

However, a quick glance at the computer snapshot shows that the Japanese structures *are* center-embedded after all—the parser places a potentially arbitrary string of empty Operators at the front of the sentence. This problem is essentially that noted by Mazuka and Lust (1990), and others. The P & P parser certainly confirms their suspicions that there may be many more logical possibilities for analyzing a Japanese sentence, as compared to its English counterpart. Perhaps, then, the *formal* accounts of why this sentence should be easy to parse are incorrect; it is formally difficult. Or perhaps it is scrambling, or pro-drop, or the Head-final character of the language that makes such sentences difficult. What is the source of the complexity problem? In the remainder of this chapter we investigate this question.

To do so, we embarked on a series of optimization efforts that focused on the Spec position of CP and the head-final character of the language, with the goal of making the Japanese as easy, or easier than, the corresponding English sentences or determining why we could not make it easier. In all, we conducted three empirical tests: (a) using dummy nonterminals to lift information from the verb to the VP node, to test the head-first/final hypothesis; (b) placing Spec of CP on the left rather than the right, to test the center-embedding hypothesis; and (c) building a "restricted" pseudo-Japanese that eliminated scrambling and free *pro*-drop, but did *not* lift information up and to the left, leaving the Head-final nature of the language intact. We cover the first and third computer experiments in detail here, leaving aside discussion of the second, due to space limitations.[20]

[20] In brief, if it is center-embedding that causes parsing complexity, then an obvious strategy is to get rid of the center-embedding itself. Here, there is a grammatical move we can make. Evidently, in Japanese, the only elements that appear in Spec of CP are put there by LF movement. Thus, these elements can never be visible in this position on the surface. If this is so, then there is really nothing to prevent us from placing just the Spec of CP on the right, rather than the left, at least as a test. (Another advantage of the implementation is that this change takes exactly two lines of code.) Of course, one can argue against this move: If we maintain complement–head order, and Subjects in VP internal Spec position, unlike in this parser, then perhaps Spec should be on the left, uniformly.

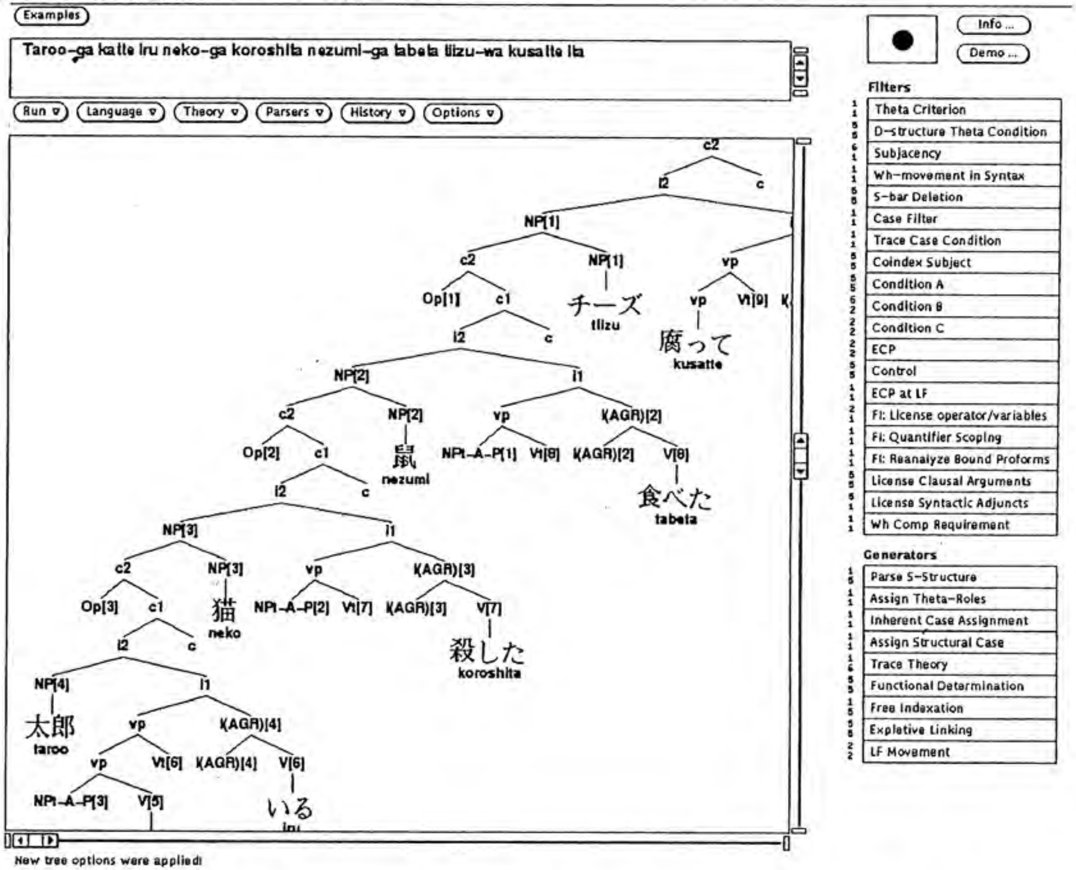

FIG. 8.8. The parse of the Japanese counterpart of the English center-embedded question. Tracing out the left-hand fringe of the tree, note the string of empty operators, as well as, on the right-hand column, the large number of parser operations required to build this single correct LF, as compared to English.

ce1. The cheese was rotten.

ce2. The cheese the rat ate was rotten.

ce3. The cheese the rat the cat killed ate was rotten.

ce4. The cheese the rat the cat John keeps killed ate was rotten.

Total number of LR state transitions

Sentence	Jp, Unoptimized	Jp, Optimized	English
ce1	57	62	745
ce2	689	563	2,431
ce3	8,627	6,041	4,979
ce4	207,909	155,125	21,074

FIG. 8.9. A comparison of Japanese unoptimized and optimized total LR state transitions to parse sentences (ce1–ce4), along with a comparison to the parsing effort for their English counterparts.

Optimization 1: Head-Final Information

Our first optimization centers on the head-final phrase structure of Japanese. As has often been noted, with heads at the end, valuable restriction information (subcategorization, selection) may be unavailable at the time the parser has to make a particular decision. However, for our LR machine, there is a well-known programming language optimization, discussed in a previous section, that we used for English. Thus, the comparison with Japanese is not entirely on level ground as yet, because the full optimization applied to English had not yet been applied to Japanese. Specifically, if verb information occurs on the right, we can, offline, lift that information up to the VP node, where it can then influence the LR state transitions that are made when examining material to the left of the head. This is precisely the mechanism we used to determine whether to insert an empty category or not in a Head-first language. For instance, in Japanese relative clauses, this is of importance because the parser may get valuable information from the verb to determine whether a preceding NP belongs to that relative clause or not. We emphasize here that this optimization is, in the limit, psychologically implausible, because it admits the possibility of unbounded lookahead. On the other hand, this method of transforming the grammar when doing parsing is roughly that used by other "noncanonical" parsing schemes (like that of Marcus): It simply says we need not build every subtree completely in a strict

On the other hand, the existence of a stack of verb-final particles in Japanese gives tenuous evidence for a Spec-right analysis. We do not attempt to resolve this linguistic question here. Given this change, the resulting structures will have their Operators on the right, rather than the left, and will *not* be center-embedded. In addition, suppose the parser does not take advantage of right-hand information, thus eliminating this as a possible source of speedup. What happens to the resulting parsing complexity? Parsing time is significantly improved over the unoptimized version, by about 10 to 15 percent.

left-to-right order. That much seems possible, even plausible, on a clause-by-clause basis.

As illustrated earlier, we implement this modification by introducing dummy nonterminal nodes and associated special checking procedures for them, in effect reversing the phrase structure locally. This compilation is done automatically—not by hand. For example, for each V subcategory, the LR machine will contain in effect a new LR state: The system will add a command to look as far into the input as needed to determine whether to branch to this new state or another V subcategory state. Thus, the action and transition tables of the resulting machine, which we call "optimized," will be far larger than its "unoptimized" counterpart.

The advantages gained by this optimization are significant. Fig. 8.10 displays the basic results (standard error bars are displayed). It compares the total number of LR state transitions to parse the embedding example sentences (ce1)–(ce4), as an unoptimized over optimized ratio, so that any value greater than 1 indicates an improvement over the base, unoptimized case. As one can see from the bar graph, although for a nonembedded sentence the optimized parser operates at essentially the same level as the unoptimized one, the *unoptimized* number of LR state transitions grows astonishingly rapidly with embedding, as we saw

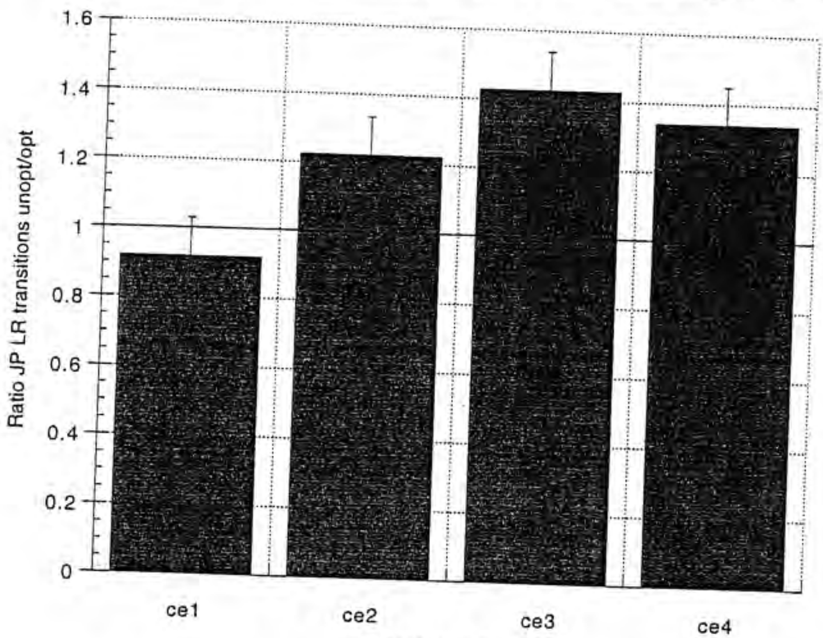

FIG. 8.10. A bar graph showing the reduction in LR states required using a right-hand information compared to the unoptimized base case when parsing the Japanese examples (ce1–ce4). Any ratio greater than 1 (indicated by the horizontal line) indicates an improvement over the base case.

earlier.[21] Thus, for doubly or triply center-embedded sentences, the parser improves by 40% or more. We would expect this improvement to be maintained for more complex sentences, since the number of possible Operator-variable and compositional arrangements increases roughly exponentially (Fong, 1989).

The Japanese right-hand information optimized version is superior to the unoptimized, base-case Japanese version, but it is still not as efficient as parsing English, because over 150,000 (155,125) transitions are needed to handle the most complex center-embedded sentence in Japanese, as opposed to 21,074 for English. Thus, it appears as if a basic left–right efficiency asymmetry is so far confirmed, because using information on the right in this powerful way reduces complexity by so much. The same basic trend also holds, though not as strongly, when we look at the other Japanese sentences (see Fig. 8.12). The effect is more pronounced with more complex sentences.

Optimization 3: The Effects of Scrambling and *pro*-drop

Part of the complexity of Japanese is the result of free scrambling and *pro*-drop. To explore this, we ran a series of computer experiments on a quasi-Japanese grammar, J*, which was just like Japanese except scrambling and *pro*-drop were barred. The changes were again simple to make: One change was automatic, just turning off a parameter value, and the second involved 3 lines of hand-coding in the X-bar schemas to force the system to look for a lexical NP in direct (and indirect) object positions. In this case, we looked at two possibilities: one with *just* scrambling and *pro*-drop turned off, and *no* head-final optimization; and one without scrambling and *pro*-drop and *with* the head-final optimization. This second test can be regarded as near as one can get to an English-like Japanese language, but without scrambling (because English used this optimization as well).

The results are instructive. Eliminating *just* scrambling and *pro*-drop results in efficiency gains that parallel those of the head-final optimization—roughly, a factor of 1.3 to 1.5 improvement. However, by *combining* the two factors, we get an interactive effect. This is the best optimization of all, in fact finally comparable to English: The most deeply center-embedded sentence takes just 27,938 LR transitions. (Plainly then, this complexity metric does *not* account for the unacceptability of English center-embedded sentences.)

Figure 8.11 displays the results. Without scrambling, and hence no movement at all compared to English, the head-final quasi-Japanese was for the most part parsed 5 to 8 times more efficiently than unoptimized Japanese.

How are we to interpret this last result? As before, with a short sentence, there is little difference between optimization methods, but over a range of

[21] We should point out that in all cases, about two thirds of these transitions occur before the LR machine reaches a point in the search space where the solutions are "clustered" enough that the remaining solutions do not take so much effort.

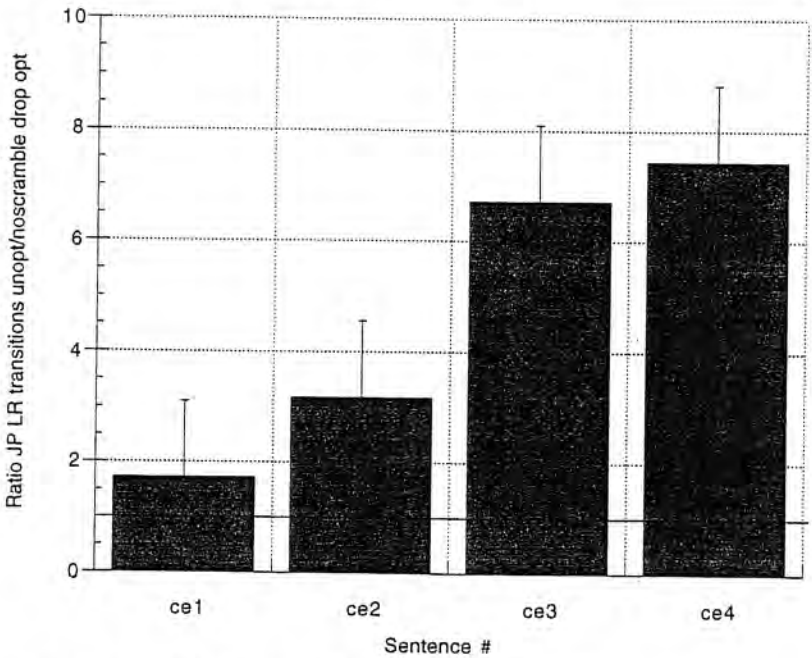

FIG. 8.11. A bar graph showing the improvement in total LR transitions when parsing Japanese examples (ce1–ce4), as a ratio compared against the base-case parser.

sentences, and with longer sentences, the dominance of the optimization becomes clear. Evidently, given the framework of assumptions we have made, the head-final character of Japanese does not hurt the most; rather, it is the interaction between the features of scrambling and *pro*-drop that does. We can confirm this by looking at the LR transitions for the sentences in Fig. 8.6, across methods, summarizing our tests (see Fig. 8.12).

OPERATIC CONCLUSION

So, we have come to the operatic conclusion: English and Japanese have met, but are they really one and the same? Given our limited set of test sentences, our results must be tentative. Nonetheless, we can make several points:

- One *can* parse Japanese by parametrically varying a grammar, much as expected. The limits of the method are theory-bound: We can accommodate just as much as we understand about Japanese syntax, in principle.
- A single parser suffices for distinct languages; the grammar is parameterized, but not the parser. Japanese sentences appear at first much more complex

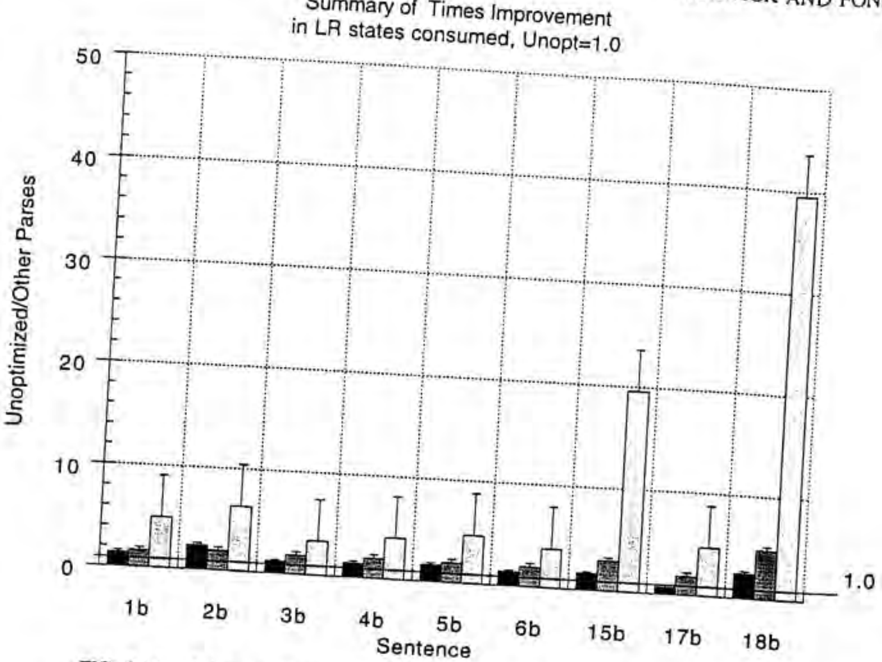

FIG. 8.12. A bar graph showing the improvement in total LR transitions when parsing Japanese examples in Fig. 8.6, compared against the original base-case unoptimized parser, across the experiments described here.

to parse than corresponding English sentences. However, the complexity appears to come more from the possibilities introduced by scrambling and the omission of NPs interacting with Head-final properties. Unoptimized, the system is too slow. More efficiency is obtained if one can lift information from the right for use in parsing with an LR machine. Thus, basic left–right differences between English and Japanese do show up in the LR machine (even in the exact details of the LR machine states, which we have not pursued in this chapter). From a purely computational viewpoint, the lifting optimization is the best, because one can still parse Japanese. From a heuristic standpoint, it suggests that strategies limiting what may appear in a scrambled position or dropped in a certain context will aid such an LR-based device more than switching to a parser presumably geared for a different branching direction.

• The principle-based system affords a new and generally straightforward way to explore different grammatical theories, structural assumptions, and parsing methods, and their computational consequences in a precise way, without extensive hand coding. All of the experiments we tried took no more than a few lines of modification. Of course, the difficult part is to come up with a universal set of principles in the first place—so that in fact, English looks just about like Japanese, and vice versa. This we have done, to a first approximation, and it is the real accomplishment of this research.

Like all productions, this linguistic opera is ongoing and demands a theatrical sequel. Perhaps minimalism is desirable not only in theatrical performance, but in grammar. We believe that notions of economy of derivation would help, not hurt, computational effort, as has sometimes been maintained. Because *Butterfly* itself is in Italian, perhaps we should add a third language—or a fourth. The Met will have to wait a little longer for the last note.

ACKNOWLEDGMENTS

This chapter describes research done at the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Support for the work described in this chapter was provided in part by NEC Research Laboratories, Inc., by NSF Grant DCR85552543 under a Presidential Young Investigator Award to Professor Robert C. Berwick, by a grant from the Kapor Family Foundation, and by Mitsubishi Electrical Laboratories, Inc. We thank Howard Lasnik, Alec Marantz, Shigeru Miyagawa, David Pesetsky, and Mamoro Saito for valuable discussions and valiant attempts to tell us about Japanese, as well as to Reiko Mazuka, Janet Fodor, the other participants of the Duke University conference on Japanese Sentence Processing, and especially Amy S. Weinberg for careful lepidoptery. We may not have heard everything they have said, but we have tried hard to listen.

REFERENCES

Chomsky, N. (1981). *Lectures on government and binding*. Dordrecht: Foris.

Chomsky, N. (1986). *Barriers*. Cambridge, MA: MIT Press.

Crocker, M. W. (1992). *A logical model of competence and performance in the human sentence processor*. Unpublished doctoral dissertation, University of Edinburgh, Edinburgh, Scotland.

Fong, S. (1989). The computation of free-indexing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics* (pp. 105–110). Association for Computational Linguistics.

Fong, S. (1991). *Computational properties of principle-based grammatical theories*. Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Fong, S., & Berwick, R. C. (1991). The computational implementation of principle-based parsers. In M. Tomita (Ed.), *Current issues in parsing technologies* (pp. 9–21). Dordrecht: Kluwer.

Fong, S., & Berwick, R. C. (1994). *Cartesian computation*. Cambridge, MA: MIT Press.

Frazier, L., & Rayner, K. (1988). Parameterizing the language processing system: Left- vs. right-branching within and across languages. In J. Hawkins (Ed.), *Explaining language universals* (pp. 247–279). Oxford: Blackwell.

Hasegawa, N. (1990). Comments on Mazuka and Lust's paper. In L. Frazier & J. de Villiers (Eds.), *Language processing and language acquisition* (pp. 207–224). Dordrecht: Kluwer.

Hosokawa, H. (1991). Syntactic differences between English and Japanese. *Georgetown Journal of Languages and Linguistics, 1*(4), 401–414.

Hoji, H. (1985). *Logical form constraints and configurational structures in Japanese.* Unpublished doctoral dissertation, University of Washington, Seattle.

Johnson, M. (1989). Use of knowledge of language. *Journal of Psycholinguistic Research, 18*(1), 105–128.

Kuno, S. (1973). *The structure of the Japanese language.* Cambridge, MA: MIT Press.

Lasnik, H., & Saito, M. (1984). On the nature of proper government. *Linguistic Inquiry, 15*(2), 235–289.

Lasnik, H., & Uriagereka, J. (1988). *A course in GB syntax: Lectures on binding and empty categories.* Cambridge, MA: MIT Press.

Makino, S., & Tsutsui, M. (1986). *A dictionary of basic Japanese grammar.* Tokyo: The Japan Times.

Mazuka, R. (1991). Processing of empty categories in Japanese. *Journal of Psycholinguistic Research, 20*(3), 215–232.

Mazuka, R., & Lust, B. (1988). Why is Japanese not difficult to process? A proposal to integrate parameter setting in Universal Grammar and parsing. In J. Blevins & J. Carter (Eds.), *Proceedings of the New England Linguistics Society, 18* (pp. 333–356). Amherst: University of Massachusetts.

Mazuka, R., & Lust, B. (1990). On parameter setting and parsing: Predications for cross-linguistic differences in adult and child processing. In L. Frazier & J. de Villiers (Eds.), *Language processing and language acquisition* (pp. 163–206). Dordrecht: Kluwer.

Petrick, S. R. (1965). *A recognition procedure for transformational grammars.* Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge.

Stabler, E. P., Jr. (1992). *The logical approach to syntax: Foundations, specifications and implementations of theories of government and binding.* Cambridge, MA: MIT Press.

Takezawa, T. (1987). *A configurational approach to case marking in Japanese.* Unpublished doctoral dissertation, University of Washington, Seattle.

Zwicky, A., Friedman, J., Hall, B., & Walker, D. (1965). The MITRE syntactic analysis procedure for transformational grammars. *Proceedings of the 1965 Fall Joint Computer Conference 27* (pp. 317–326). Washington, DC: Spartan Books.