In Linguistics and Computation
Eds. Cole, J. et al. Lecture Notes 52, CSLI 1995

A Quarter Century of Computation with Transformational Grammar

Robert C. Berwick and Sandiway Fong
MIT Artificial Intelligence Laboratory
NEC Research Institute, Inc.

1 Introduction: plus ça change...

It seems altogether fitting that a historical celebration of a linguistics department should focus on a bit of history: the history of computation and generative grammar, in particular the history of how computer models of transformationally-based linguistic theories had changed radically, especially recently, to solve computational problems that had haunted them for at least a quarter century. In this article, we would like to show how the promised transparent computational implementations of transformational generative grammar – which made brief appearances in the middle 1960s along with similarly transparent incarnations in psycholinguistic and acquisition models, and then which gave way, in most cases rapidly, to alternative models or "nontransparent" replacements – can now, at long last, be said to have arrived. We can now build efficient parsers for transformational grammar, in its very latest incarnations, and reap all the benefits: transparent coverage of entire linguistic textbooks, top to bottom—one can write Case Assignment or the Empty Category Principle nearly as it appears in a textbook; no special treatment of so-called ungrammatical sentences; easy experimentation and exploration of alternative theory-internal changes; and simple parameter switching to get multiple languages and a better story about language acquisition.

At the same time, we would also like to emphasize the historical continuity of the project—that the transformational generative grammar (TGG) implementations of today face the same problems of overgeneration, constraints, abstraction, arbitrary empty strings, and the like that had to be faced in the 60s. However, we can now at least realize some of the hopes of the earliest researchers of the 60s: we can build a complete, efficient computer implementation, a parser, for the most recent versions of TGG, using an abstract language built on top of PROLOG that remains as close as can be expected to the English (or Italian, or German, ...) one finds in linguistic texts.

The question that naturally arises is: *why* now? Why does TGG parsing work now, and not 25 years ago (the MITRE project (Zwicky et al. 1965), Friedman (1971), or Petrick (1965) systems)? The answer is two-fold. First, we just know more about linguistic representations, and that constrains the system: the Structure Preservation Hypothesis and constraints on landing sites limit the possibilities for inferring moved elements; deletion is more restricted (though infinite regress in adjunction still comes back to haunt us); X-bar theory limits the possibilities for phrase structures and lexical insertion; there are no ordered rules; and so forth. Second, we know more about computation. We can build efficient structural parsers that build the initial scaffolding (like the covering grammars of the 1960s), via multiple-entry type-inference LR machines. The most important trick in all of this is in using the notion of *language abstraction*: the abstract language stating the linguistic theory is automatically transformed via specially-built (but otherwise standard) compilers, into a form the user never sees, that does the actual work and bridges the gap between the abstract representation and efficient execution. In this regard the system is no different from any other programming language, or indeed from many other computational linguistic formalisms. (It is an open question as to whether we could take our updated technology and show that the older theory could have been implemented this way as well—we see no reason in principle why it could not, but for the improvements in the Structure Preservation Hypothesis, bounding deletion, and the like.)

The end result is that computation and TGG went through a U-shaped roller-coaster ride of transparency or faithfulness to the textual rendering of the theory (like the ups-and-downs of a classic learning curve): the initial ("rose") period of the 60s, when researchers tried to build systems where one actually wrote down structural descriptions, structural changes, the factorization of the rules, etc., ordered them, mimicking *Syntactic Structures* or *Aspects* exactly as one can see from the Petrick or Friedman system examples; followed by the middle ("blue") period of the 70s, when this transparency was cast aside in favor of rule systems that respected the representational descriptions and constraints of TGG (S-structure, etc.), but only relatively indirectly (for instance, the Marcus parser); to the current proposal, which is again deliberately transparent.

The implemented system we outline is ambitious: it attempts to incorporate essentially all of Lasnik and Uriagereka's *A course in GB theory* (Lasnik and Uriagereka 1989, henceforth LU). All of the theory in Chapters 1–4 of LU have been implemented. Chapter 5 of LU, covering alternatives to "classical" Binding Theory such as Aoun's Generalized Binding Theory, have not been implemented. Of the open questions posed in Chapter 6 of LU, we have adopted the prohibition against Case-marking of NP traces, and genitive Case realization to bar illicit NP movement. Certain other changes have been made in the interests of consistency or efficient computation, for instance, the use of erasable features as a substitute for gamma marking; we discuss some of these matters as they arise. (For a full description of the implemented theory, see Fong 1991, and Fong and Berwick 1992.) The parser incorporates a current (parameterized) X-bar theory of ten basic structures and 20 adjunct rules, including a full IP (Inflectional) and CP system, though not VP-internal subjects; general movement principles such as raising and lowering (including V raising/lowering); Full Interpretation (FI) and Functional Determination of empty categories; complete Case, Theta, and Binding conditions, along with LF and quantifier movement (QR); and so forth, totaling 25 modular principles that *interact* to give us the surface sentences that are possible. As far we have been able to determine, this is the most comprehensive parser of its kind, implementing the full range of examples found in a modern generative principles and parameters theory.

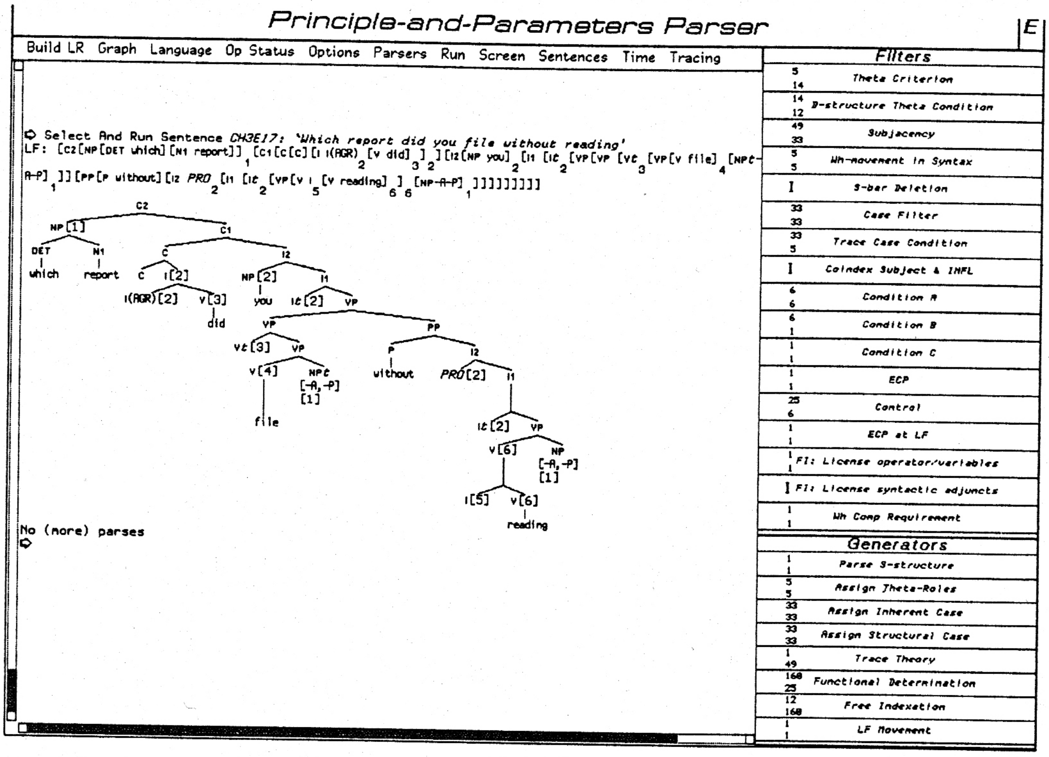
As mentioned, with this parser in hand, we can easily reap some of the long-promised rewards of using a well-worked out linguistic theory: without special rules of any kind or any specially-designed movement constraints and so forth, we can in a second or two parse examples as subtle as parasitic gap sentences (*this is the paper that I gave without reading*) as well as fail in exactly the ways that linguists' (and even ordinary people) predict for sentences like *What do you wonder who likes*, assigning exactly the same structures that people seem to. See Figure 1 for a picture of how the implemented system analyzes the parasitic gap sentence; along the righthand side of the screen is a full list of the principles themselves.

Perhaps most importantly, as promised in current approaches, crosslinguistic variation works: using the *same* parsing algorithm and the *same* grammar, but with small parametric variations and a new lexicon, we can parse, for instance, Japanese instead of English, in this case, the Lasnik and Saito (1984) *wh*-questions that illustrate scrambling and a rather subtle distribution of empty elements that are the meat of a modern theory. As an example, consider the Japanese sentence:

(1) *Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no*
'What are you so angry about the fact that Taro obtained'

Here the subject of the matrix clause (= *you*) has been omitted. *Nani* and *te* ('hand') have been permuted from an assumed canonical indirect object, direct object order—an example of scrambling. The LF for this sentence may be interpreted roughly as, for what x, *pro* is so angry about [the fact that Taro obtained x]. Here, *pro* represents the understood subject of *okotteru* ('be angry'). As we shall see, the implemented system does correctly recover this form (with some subtleties concerning alternative syntactic readings that we return to below). Thus as expected, on this account there is no separate phrase structure grammar for each language

(JapanesePSG or JPSG, GermanPSG, etc.)—just a single UG, plus a handful of switches. Furthermore, again as expected, it did not take long to "implement" the parser for the new language, if indeed one can even call it implementation. Automatic programming is a better name for it.



```
⇨ Select And Run Sentence CH3E17: 'Which report did you file without reading'
LF: [C2[NP[DET which][N1 report]]  [C1[C[C][I I(AGR)  [V did]   ]  ][I2[NP you]  [I1 [It  [VP[VP [Vt  [VP[V file]  [NPt-
                                 1                  2          3 2            2        2            3                4
A-P]  ]][PP[P without][I2 PRO  [I1 [It  [VP[V I  [V reading]   ]  [NP-A-P]  ]]]]]]]]]
    1                         2        2        5             6 6          1

No (more) parses
⇨
```

```
Filters
 5   Theta Criterion
14
14   D-structure Theta Condition
12
49   Subjacency
33
 5   Wh-movement in Syntax
 5
 I   S-bar Deletion
33   Case Filter
33
33   Trace Case Condition
 5
 I   Coindex Subject & INFL
 6   Condition A
 6
 6   Condition B
 1
 1   Condition C
 1
 1   ECP
 1
25   Control
 6
 1   ECP at LF
 1
 1   FI: License operator/variables
 1
 I   FI: License syntactic adjuncts
 1   Wh Comp Requirement
 1

Generators
 1   Parse S-structure
 1
 5   Assign Theta-Roles
 5
33   Assign Inherent Case
33
33   Assign Structural Case
33
 1   Trace Theory
49
168  Functional Determination
25
12   Free Indexation
168
 1   LF Movement
 1
```

Figure 1: A snapshot of the computer output of the principles and parameters parser analyzing a parasitic gap sentence. The input is the sentence "which report did you file without reading." The output is a logical form tree. Along the righthand side of the screen is a full list of the principles used in the parsing system, that remains invariant from language to language.

Lest no one forget the distinction, this *is* a full-blooded parser. As Figure 1 shows, one supplies a sentence in orthographic form, and the parser returns the proper structural descriptions associated with the sentence: the associated S-structure, LF, and assignment of thematic roles and indices to NPs. Note that this is quite unlike some recent approaches where the computational system acts as a kind of "proof checker" where, given a structural description and LF, the computer will tell one whether or not that particular S-structure, LF, phonetic form triple can be derived from the axioms of the theory via "guided deductions," driven presumably for reasons of computational efficiency because these systems use more powerful theorem provers.

Note that in order to use such systems as parsers, the straightforward approach involves costly analysis-by-synthesis, and indeed timing results for such systems have been so far disappointing (Stabler 1992). We have found such first-order theorem provers and their attendant corner-cutting to be unnecessary. Indeed, their use may introduce a potentially pernicious influence that computer implementation is precisely designed to sidestep, in particular that the linguist knows just what the intended derivation of an example sentence is.[1]

Now of course we all know that linguists are infallible (who could deny it in such surroundings?), but to be sure, the implications of modern linguistic theories can be complex enough that it easy to lose track of what happens if some assumptions change, or even if they do not. In fact, we can document that this is so. For instance, as we shall see, the implemented system – reflecting the linguistic theory exactly – can produce many movement possibilities even in a simple sentence (for example, in the Japanese example above, there turn out to be 106 possibilities for movement and 468 indexings), and it is highly unlikely that a linguist would hit upon them all (present company excepted of course).

As a concrete example, a sentence that is standardly cited as a "Subjacency Violation," **Who does Mary wonder why John hit*? (Example 4:57 in LU) actually admits three structural possibilities that slip by that constraint but are ruled out later down the road by another condition, Condition B (they all involve a base-generated PRO bound to *John*, hence out by Condition B). These possible derivations were evidently never noticed before. Thus, building such a parser can explore unintended avenues of a theory, and so it becomes part of the theory-construction process itself. In general then, the search for *all* possible parses can produce somewhat unexpected results. We note that in such cases the parser is not "wrong" in the sense that it is producing parses that are inconsistent with linguistic theory. (Naturally, the parser falls down where the theory does, but then, so much the worse for the linguistic theory, at least—indeed, as we shall see, this is a benefit of the computer implementation.)

In short, it may well be true that linguistic theory is now deductively rich enough, like mathematics, that it actually *demands* a tireless machine to keep slavish track of all the bookkeeping. We should not be surprised by this. After all, there are now mathematical results, like the 4-color

[1]Other systems of this kind have been fragmentary, or else not really parsers at all in the sense of mapping input sentences in their orthographic form to structural descriptions. The best worked-out example that we know of, Stabler's (1992) formalization of Chomsky's (1986) *Barriers* model, has two admitted shortcomings: first, *Barriers* is but a fragment of a principles and parameters syntactic theory; and second, the resulting formalization is not a parser (it can only check structural descriptions for well-formedness, it cannot map sentences to their structural descriptions efficiently enough).

map theorem, proved only by machine. Beyond this however, the parser is complete enough to serve as a kind of linguists' apprentice, if you will—an oft-cited goal from the 1960s, but even more appropriate today. Such a give-and-take partnership is possible only if the system does in fact parse fast enough to make user interaction bearable, and if the system explores all possibilities, rather than letting the user exclude some cases that would actually reveal gaps in the theory.

So, let us give a roadmap then for the remainder of this article. We shall swiftly review the earlier ages that implemented TGG: first, the Dawn Age (classic transformational parsers); followed by the Dark Age (little or no work on parsing and TGG); and then the Middle Ages (the Marcus parser and its relatives), pointing out the relevant difficulties that moved linguists and computation from one age to the next. We conclude with the Renaissance: the computational representation and control of a transparent implementation of a principle and parameters linguistic theory.

2 The Dawn Age

Dawn is marked by rose – the rosy-fingered dawn, or *rotodactylos* as Homer said – and optimism similarly suffused this period of computation and TGG. It was thought that transformational generative grammar could be used directly as a parser for natural languages. Like all dawns however, this early hope faded. Why? Transparent parsers for TGG used the familiar structural description matching/factorization–structural change format along with the usual base context-free structure grammar, thus yielding highly language-particular and surface and construction-oriented rule systems. A standard example would be the "passive" construction in English:

structural	NP	[+V +Aux],		[+V −Aux],		NP
description	1	2		3		4
structural						
change	4	2	Be+en	3	by	1

Note that this rule is in fact particular to English—it explicitly encodes the left-to-right order of English subjects and objects, the morphology of *be*, and so forth. A typical transformational system for even a fragment of a language, say, English, would consist of well over a hundred such rules; for instance, the MITRE system (Zwicky et al. 1965) contained 134 such rules, and Petrick's question answering system (Plath 1976) many more.

We call parsing systems based on such rules *transparent* because they embed rules directly in a parsing device, without additional source-to-source translation. To take another example consider the rule of subject formation from Petrick (1965) that attaches a *wh*-NP to the Auxiliary position, converting for instance *will who eat ice-cream* into *who will eat ice-cream*, while 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:  or: 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, transformational rules were marked as obligatory or optional, cyclic or postcyclic, embedding or not, and so forth. These and other systems were intended to be used as full-fledged parsers, linguistic apprentices, and so forth—all the conventional noble goals. However, as is well known, there were many computational problems with such parsing systems—so many problems that the transparent, transformational rule-reversal approach was abandoned in computational linguistics. We summarize some of the familiar difficulties here:

1. Rules were ordered in intricate ways. Further, one could delete items, so a forward transformational rule need not be invertible (if an item is deleted, what do we invert it to?). Thus, in general the mapping backwards from an input string to a structural description would be highly nondeterministic.

2. Since rules operated only on structural descriptions (factorized trees), and we are given a sentence not a tree to parse, any parser must first recover some structure to match against. A common solution to this problem (adopted by MITRE, for example), was to build a phrase structure *covering grammar* for a (superset) of initial structural descriptions, and then attempt to run rules (deterministically) in reverse against these.

3. Arbitrary deletion can loop, or, at best, take at least exponential space. As a simple example, consider any transformation that deletes some element. We cannot literally invert this to say that this element

may be inserted between *any* two positions in a tree. To consider another example, the explicit reconstruction of an empty subject, as in Peters' (1973) example *Their sitting down threatens* [*empty*] *to spoil the joke*, where the *empty* position corresponds to *their sitting down*. If we embed this again, as in [*their sitting down promising* [*empty*] *to steady the canoe* $]_2$ *threatens* [*empty*$_2$] *to spoil the joke*, then the second empty position, when reconstructed, contains the original full subject, with *its* reconstructed subject. If the deep structure of such examples is *explicitly* reconstructed, then the empty subject can be embedded over and over again, which leads to a deep structure that is doubly exponentially larger than the surface structure, and inevitable computational intractability. Of course, there are familiar termination problems with the arbitrary Turing machine computations that can be simulated with unrestricted *Aspects* type systems.

To attempt to overcome these problems the MITRE system used a context-free *surface grammar* with 49 rules and 550 subrules to first recover a set of *presumable surface trees*, and from there use reverse transformations to recover *presumable base trees* (that is, representations of thematic relations, or a combination of what is now D-structure and LF). These were then filtered by a final stage where the presumable base or D-structure tree was driven by forward transformations and then checked against the given sentence. Great effort was expended at developing covering grammars and inverse transformations in the MITRE system. To reduce nondeterminism, the inverse transforms were assumed to be obligatory—that is, the system did not follow the alternative paths of both applying and not applying the inverse. (We shall see that the system we describe overcomes this problem and considers both possibilities.) Even so, practical parsing efficiency was not achieved. The MITRE authors note that even with a much smaller grammar for some simple sentences, *The general that Johnson met in Washington had traveled eight thousand miles*, over 48 presumable surface trees were obtained. Their conclusion was simple. To quote them (Zwicky et al. 1965:325):

> It is clear from even these few numbers that if the procedure is to be practical, it will be necessary to incorporate a highly efficient routine for obtaining surface trees and to work on the rapid elimination of spurious ones.

Strikingly, this efficiency proved very hard to achieve. Perhaps the only widely-publicized example of a TGG system that achieved practical speeds was the method of Petrick (Petrick 1965, Plath 1976), developed at IBM that was actually used in a database question-answering system. This method also used a covering grammar, using powerful augmentations of a context-free grammar to reduce the number of surface trees; however, it remained grounded in transformational models of the 1960s.

3 The Dark Age

After the Dawn Age of the early 1960s we enter Dark Age. Not only were methods based on TGG discredited for parsing, but early transparent approaches to psycholinguistics grounded on TGG lost favor. By the early 1970s it was possible for standard texts to proclaim the demise of TGG. Fodor, Bever, and Garrett (1974:368) have this to say:

> ...there exist no suggestions about how a generative grammar might be concretely employed as a sentence recognizer in a psychologically plausible model.

while a comparable text on language acquisition by Maratsos (1978:246) gave the same message:

> If... transformational grammars represent the essence of the linguistic system captured by the child, the phenomenon of language acquisition seems to be an inexplicable mystery.

As is familiar, to fill this apparent explanatory gap, several more "computationally oriented" approaches were devised, augmented transition network grammars being among the most prominent. Alternative linguistic theories were also proposed, presumably more consonant with processing and acquisition considerations. Transformational approaches, save for efforts like that of Petrick, died out.

Like the Dawn, this Dark Age did not last forever, however. By focusing on at least one of the computational problems with TGG, namely, its computational inefficiency, some insights were soon won about how to emerge from the shadows of the past.

4 The Middle Ages: Transparency Lost and Regained

By the middle 1970s, our understanding of both linguistic and computational theory had advanced to the point where parsers for TGG could again be taken seriously. The key to this advance was *constraint*. On the linguistic side, prompted by empirical and learnability considerations, TGG was constrained in a number of directions. Transformations were made optional; extrinsic rule ordering was eliminated; the Structure Preservation Hypothesis ensured that most derived structures could be base generated (by some context-free grammar) anyway; X-bar theory demonstrated how the lexicon could be used to fix most of the individual phrase structure rules of the base; full copies of empty pronominals were replaced by indexing of empty elements not subject to recursion; and individual rules were shown to be subject to island constraints and other locality conditions, unifying some under a single movement rule—the Extended Standard Theory (EST).

On the computational side, the corresponding constraint was *determinism* as advanced in the Marcus parser (Marcus 1980). Marcus avoided the problems with the MITRE system by eliminating the backwards nondeterminism inherent in the older system: he assumed that local surface cues were enough to fix the inverse application of transformational rules, building an EST-type S-structure. The parser was deterministic in the sense that a single structure was built—hence each operation was deterministic, if one considered the operation of the parsing engine as the operation of an automaton (this was borne out by subsequent analysis by Berwick (1985) and more carefully by Nozohoor-Farshi (1987) in which it was established that the languages recognized by the Marcus-type devices were deterministic context-free languages).

Note how these assumptions remove the problems of the earlier TGG systems: deleted elements are assumed to be recoverable from structural cues; by positing that this is just deterministic context-free parsing a large search space of possible trees to map back to is eliminated. Further, by using explicit phonetically null categories that could not be successively embedded, degenerate examples of the Peters' type were avoided. Of course, the trick in all of this is to show that this is sufficient to cover the entire linguistic theory.

Using the older linguistic terminology, the Marcus parser used a set of structural descriptions and inverse structural changes that were strongly constrained, in the following sense. The structural descriptions, the triggering patterns for structural changes, were constrained to examine only (i) the features of the topmost node in a pushdown stack (say, a VP or NP) and its daughters, along with (ii) the features of perhaps one additional node in this stack, plus (iii) the features three input buffer cells, namely the current input word or phrase of the sentence being analyzed, along with two additional words or phrases. This locality condition on structural descriptions had the effect of enforcing c-command and other locality conditions as required by the EST theory (see Berwick 1985 for details). Put another way, the architectural features of the system constrained structural descriptions, as argued by Marcus (1980). Structural changes were also constrained to be deterministic, as mentioned: they could not remove features or structure, but simply monotonically add to it; further, they also were subject to the same locality restrictions as structural descriptions.

We can now compare an old-style transformational rule, say, passive, with the corresponding structural description–structural change format (if–then rules) in the Marcus parser, and note where the constraints make a difference.

We repeat below the English passive rule as it might be written in an older transformational system:

structural description	NP 1	[+V +Aux], 2		[+V −Aux], 3		NP 4
structural change	4	2	Be+en	3	by	1

Under the constrained formulation, the object NP is base-generated as before (reflecting its underlying thematic role), and then, by the Structure Preservation Hypothesis, can move into an empty Subject position, leaving behind a phonetically null element. The movement does not violate any locality constraints. (The creation of the *by* phrase remains a bit of a mystery on this account.) More importantly from the parsing standpoint, the structure after the movement has occurred can serve as a deterministic triggering surface pattern for a corresponding rule inverse.

This is just what the Marcus parser does. In the Marcus version, this single rule is broken down into three rules, to pick up one piece at a time the left-to-right "proper analysis" of the structure *after* the transformation. The passive rule itself depends on prior marking of the verb as passive, which in turn depends on detecting an auxiliary form of the verb *be* plus a verb with *ed* morphology, as in *Mary was kissed*, roughly as follows (the exact rule formulations have been slightly modified for readability):

Rule 1:
passive-auxiliary:

trigger:	input cell 1 root *be*	cell 2 verb marked *en*
action:	attach *be* form as passive mark verb as *passive*	

Rule 2:	
trigger:	verb marked *passive*
action:	run Rule 3
Rule 3:	
trigger:	verb marked *passive*
action:	insert NP trace
	as object (link to Subject)

Note that the division into separate rules is required by the way the parser works left to right: the Subject NP must first be parsed as if it were a declarative, and only then is the auxiliary verb and verb with passive morphology picked up. This is the first part of the proper analysis tree factorization (in reverse). (In fact, the whole rule is properly conditioned with features beyond the feature proxy *en* so that only passivizable verbs will be marked as such, this detail being suppressed here.) Next, when the verb such as *kissed* is actually in the first buffer position, the parser creates a VP (not shown here), and then inserts a phonetically null NP, a trace, into the first buffer position. This NP is then parsed just as if it were an ordinary object noun phrase, by the same rule that parses noun phrases. By the determinism hypothesis, it is assumed that there is a unique action for every trigger (where the triggers may be prioritized essentially by specificity).

However, by slicing up the rules in this way the parser does lose some of the modularity of the EST system itself. Not only has one rule been split across three, but the division of the EST theory into an X-bar base (forming D-structure) plus transformations has been partially obscured. In the original Marcus parser, much of the information about phrase structure or X-bar constraints was encoded in a particular sequence of *packet* activations that turned on or off whole rule groups at a time. In our example, the rules to parse specifiers of Inflection phrases (or IP projections in current terminology) are collected into a single group, a *Parse-Subject* packet. Rules can only trigger if the corresponding packet they are in is activated (by some previous rule); this activation sequence corresponds roughly to the Specifier-Head-Complement order of X-bar phrase structure. Similarly, the expansion of INFL itself is guided by a linear sequence of packet activations.

Modularity is obscured (and transparency violated) because there is nothing in the parser's rule-writing framework to guarantee this division into X-bar rules plus transformations. For instance, Rule 3 that actually inserts a trace into the NP object position resides in its own packet, apart from other verb complement rules. Similarly, the division into three separate rules is demanded in part by the exigencies of left-to-right analysis of the input. More generally, one can see that this parser is construction-specific: the rules to handle a sentence that in modern terms is putatively derived from the same constraints, such as *It was believed that the ice-cream was eaten*, requires an entirely different set of four or five completely different rules. In this way, commonalities among syntactic phenomena are obscured rather than illuminated.

Further, ideally the restrictions of movement phenomena generally are motivated in the Marcus parser specifically by determinism and the functional architecture of the parsing machine. This is both a plus and a minus. On the plus side, the locality restrictions on stack and buffer access conspire with determinism to guarantee a linear time parse (this much follows from the formal results about the class of deterministic context-free languages parsable by this device); on the minus side, basic linguistic constraints like c-command and subjacency are thereby implicitly "hidden" from the operation of the rules themselves. Efficiency seems to have been purchased at the price of a loss in theoretical transparency.

Expanding on this last point, and turning to more recent analyses of so-called "passive constructions," we find that passive is not a unitary phenomenon. Passive includes movement of the object; inserting a *be* and altering the main verb's morphology, and adding the *by* phrase. Chomsky noted a decade ago that all of these components can occur on their own. Specifically, the object can be in subject position even without a verb, as in *the destruction of the city* yielding *the city's destruction*); the *by* phrase can appear by itself, as in *the book understandable by everyone*; passive morphology can occur by itself, as in *the melted ice-cream*. Further confirmation comes from examining languages that have different word orders than English, but still exhibit passive-like sentences, say, languages like Japanese, German, or Dutch, where the verb comes at the end, revealing that even the Subject—Verb—Object Passive Rule triggering pattern was illusory: in Dutch we have *Kees zei dat Jan Marie kuste* ('Kees said that Jan Mary kissed') and the corresponding passive form *Kees zei dat Marie door Jan gekust werd* ('Kees said that Mary by Jan kissed was'); (see Kolb and Thiersch 1990:252-253 for these examples).

What the Marcus parser has done, then, is to take presumably deeper underlying principles and "compile" them out via deductive chains, yielding particular surface sequence patterns. Over the past decade, linguistic research in the EST tradition has eliminated language-specific rules in favor of a small set of universal principles plus parametric variation, perhaps restricted to the lexicon; we shall turn to these in the next section. Thus, while the deterministic Marcus parser design achieved a considerable success in terms of efficiency and faithfulness to the input-output representations of linguistic theory within its own era, it does not employ current theoretical vocabulary or mirror precisely the fundamental "atoms" of current *principles and parameters* theory. As a result, cross-linguistic parser construction is difficult. For example, the parser architectural constraint that helps ensure the locality condition known as the Specified Subject Condition actually depends on the specific Subject-Inflection order of items in English. Similarly, the English parser rules rely on the head-first character of phrases: given a preposition, verb, determiner, and so forth, then the parser can easily predict the existence of a prepositional phrase, verb phrase, noun phrase, and so forth. Indeed, this almost makes the parser a *left-corner parser*, using the same strategy for English as Head-Driven Phrase Structure Grammar. Building a parser for a head-final language, such as Japanese or German, entails major surgery; indeed, it is not clear that such a project can be carried out at all.[2]

In short, with the advent of constraints in the TGG theory of the 1970s, determinism and faithful construction of TGG representations TGG proved to be a major step forward out of the Dark Ages for TGG parsing. What remained was to build a system that was efficient and faithful to more recent grammatical theory in both input/output representations *and* operating principles. That is our story for the Renaissance of TGG parsing, described next.

5 The Renaissance: A Principles and Parameters Parser

As is well known, the past decade has seen a shift in transformational generative grammar from homogeneous, language-particular accounts of rules such as passive to a highly modular, non-homogeneous, and parameterized deductive system of universal principles. Following current practice, we shall call such approaches *principles and parameters theories*. On this view, there is no "rule" of passive, but rather a system of (declarative) constraints that interact to yield surface forms in English that may be described as passive. If we think of the principles as axioms, the passive construction emerges as a theorem. But the deductive chain is much longer than in a simple if-then rule system like the Marcus parser, where there is a direct, one-step connection between passive sentences and rules. Let us briefly review this notion here, and then see how to make use of it in a parser.

Assuming some familiarity with the familiar "Y-diagram" of linguistic representations in transformational theory linking phonetic form (the ortho-

[2]The only large-scale German Marcus-style parsers that this author knows of, with many hundreds of rules, were constructed for the MIT Athena Foreign Language Instruction Project. This parsing system proved to be extremely unwieldy for both German and Japanese.

graphic input), LF, S-structure, and thematic representation (D-structure), as well as the basic components of this theory, we shall merely note here some of the (universal) principles involved that "force" the derivation of a passive sentence:

(1) X-bar theory: Languages allow just two basic tree shapes or *parameterizations* for their constituent phrases: (i) function–argument order, as in English, where heads begin constituents; or (ii) or the mirror image, argument–function form, as in Japanese or German.

(2) Thematic theory: Every verb must assign a thematic role to its "arguments" that says, roughly, who did what to whom; and every Subject and Object must receive exactly one such role.

(3) Case Theory: Overt or pronounced Subjects or Objects must receive *Case* where by *Case* we mean an abstract version of what one would find in a Latin grammar. The Subject position receives, or is assigned, nominative case from the inflection of a verb; the Object of the verb receives accusative case; the Object of a preposition receives oblique case, and so on.

(4) Movement theory: Any phrase can move anywhere (*move-α*), subject to locality constraints. When it does, it leaves behind a phonetically silent element, a trace, linked to itself.

Given these axioms, then we can derive a passive sentence such as *the ice-cream was eaten* as follows:

X-bar theory sets the basic function-argument order of English

↓

$\emptyset$ *was eaten the ice-cream*

↓

Eaten is an adjective, and so does not assign Case

↓

Ice-cream must receive Case

↓

Ice-cream (allowably) moves to subject position
where it receives nominative case

↓

Leave behind an empty category, linked to *ice-cream*
(so that *eat* can meet thematic constraints and
make *ice-cream* the thing eaten)

↓

the ice-cream was eaten trace

Just as with the earlier TGG systems, the key question now is how are we to build a parser to invert the derivation and map from sentence to its underlying structural representation (both S-structure and its so-called Logical Form)? Further, how can we do this efficiently? Note that this problem is already different from that of earlier transformational theory, for two important reasons: first, the system is formulated as a system of declarative constraints on representations; second, the constraints themselves are deterministic (given one input tree they produce only one output).

Pursuing this proposal, we can divide principles into one of two classes: *generators* and *filters*. Generators produce or hypothesize possible structures. For example, consider X-bar theory. Given a string of words, say, *eat the ice-cream*, this theory theory would say that *eat* is possibly the beginning of a verb phrase, with *the ice-cream* as its argument. Similarly, movement creates possible structures. Given a valid X-bar structure, move-α can displace various noun phrases like *ice-cream* to create new ones.

Filters weed out possible structures. For example, if the structure *John is proud ice-cream* is input to Case Theory it would be filtered out as a violation (it should be *proud of ice-cream*, where *of* assigns case to *ice-cream*). Our actual system contains 17 filters and eight generators, as shown in Table 1.

Given this generator-filter model, the simplest way to build a parser is as a cascaded sequence of principle modules, with the input sentence piped in at one end and one or more Logical Forms emerging out the other: the sentence must run a gauntlet of constraints. For example, consider the example sentence, *Mary was kissed*. We can imagine passing it first through the X-bar module, producing several output possibilities depending on word and structural ambiguities, as is typical of context-free parsing generally. All the usual techniques for efficient processing, such as lookahead, can be useful here.

Continuing, let us suppose that the hypotheses output from the X-bar component, tree structures, are fed into the next module down the line, say Case Theory. Case Theory now acts as a declarative filter on the output trees, eliminating some of them. Next, Movement expands the possibilities once more, generating all possible structures with displaced phrases; continuing, after several running through several other constraint modules, the final output Logical Form will emerge (if one is possible at all).

Of course, this is still mere conceptual handwaving. We still face the problems of twenty years ago: (1) how can we generate just the right trees to begin with; (2) how can we invert the effects of transformations? In response to these demands, we adopt a more sophisticated parsing approach that *combines* some of the features of analysis-by-synthesis ("generate-and test") and analysis-by-analysis ("invert from the sentence"), relying on the declarative determinacy of filters (they produce only single outputs) and the

Filters
Theta Criterion
D-structure Theta Condition
Subjacency
Wh-movement in Syntax
S-bar deletion
Case Filter
Trace Case Condition
Agreement of Subject and Inflection
Condition A (anaphors bound in governing category)
Condition B (pronominals free in governing category)
Condition C (referring expressions free)
Empty Category Principle
Control Theory
Empty Category Principle at LF
Full interpretation: license operator-variables
Full interpretation: license syntactic adjuncts
Wh Comp requirement
Generators
Build quasi-S structure
Assign thematic roles
Assign inherent Case
Assign structural Case
Move-α
Functional determination of empty categories
Free indexation
LF movement

Table 1: The principles (filters and generators) used by the principles and parameters parser.

restrictiveness of generators (so-called empty positions are highly restricted in nature).

These are by no means trivial problems. Combining the possibilities of adjunction and empty categories in a simple X-bar theory implies that one can associate with a single lexical token, a verb say, with a countably infinite number of well-formed adjoined VP structures with empty categories at the leaves. To take a concrete example, on common assumptions each terminal element of the following basic clausal structure may be empty, that is, $CP \overset{*}{\Rightarrow} \lambda$:[3]

(2) $[_{CP}$ Spec $[_{C'}$ C $[_{IP}$ NP $[_{I'}$ I $[_{VP}$ $[_{V'}$ V NP$]]]]]]$

Recursion through CP will then lead immediately to nontermination problems, since we can generate an arbitrarily long string of empty elements.

How do we get around such problems? In brief we use a covering grammar to generate a superset of hypotheses that satisfy X-bar theory and any base-generated empty elements, while incorporating the constraints from the actual sentence elements (thus carrying out analysis-by-analysis of the sentence as well). These candidate structures are then passed through the filters and generators, perhaps in a parallel (interleaved) way.

The reason for starting with X-bar theory is simple. Many, if not most, of the constraints depend on particular structural configurations. For instance, Case is often assigned only under a particular local structural arrangement—the element receiving case is an immediately adjacent sister to a verb or a preposition. These logical dependencies must be respected in any principles and parameters parser design.

The theory components themselves – the principles – are encoded using terms and predicates as close to the linguistic theory as possible; we give examples below. Thus the parser aims to be as transparent as possible, within the constraints of attaining efficient parsability. In fact, both goals seem to be fairly well attained, with average parsing times on the order of a few seconds or less for the several hundred example sentences in the Lasnik and Uriagereka (1989) reference book that we used.

In the sequel, we first describe how the principle-based parser works via a parasitic gap example sentence, while discussing the operation of the covering grammar. We then continue with an illustration of the power

[3]The NP may be empty to account for empty categories; by head movement, both V and I may be lexically empty; C may be filled by an empty complementizer; the Specifier of C need not be filled at all. Notational conventions: We adopt a two-level system for all categories except verbs; hence CP=C2, IP=I2; C′=C1, I′=I1; C0= lexical head, etc., in our structures. For VPs we have a 3-level system, to attach indirect and direct objects to V2 and V1, respectively. For readability we sometimes use CP, VP, and so forth instead of C2, V2, and on.

of the system by demonstrating how simple it is to switch to a different language, in this case, Japanese.

6 How the Parser Works

To see how the parser actually works together with the linguistic section we work through the parasitic gap example given in the first section in detail, showing how the 25 principles interact to yield precisely the correct output analysis. As mentioned earlier, the parser produces correct parses in the sense that it succeeds or blocks as described in the LU textbook for each example.

Recall that for this sentence the parser builds an LF with a controlled PRO indexed to *you* as the subject of *reading*, while a pure variable fills the position after *reading* (marked −A(naphoric) −P(ronominal) via the functional determination of empty categories, as adopted here); an NP 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 Inflection, so as to receive tense, and then the Verb-Inflection complex raised and adjoined to C yielding Subject-Auxiliary inversion. All this detail is captured by the 25 principles shown in Figure 1 and Table 1.

We first give an overview of the computation, and then go back and cover in detail how the parser works (refer back to Figure 1.) Basically, there two main parsing stages: (1) S-structure recovery via a covering grammar; and (2) the application of the remaining filters and generators.

In Stage I a *single* quasi-S-structure is recovered by a special-purpose LR machine, 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 and so forth). 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 *John seems to be happy* and *John wants to be happy*, inserting a *generic* empty category as the subject of the complement CP. Later principles must fix these empty categories as a trace in the first case but not the second. 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. The LR machine we use has multiple table entries to handle ambiguity in natural languages, including the possibility that an empty category might or might not be inserted in some position.

Free (optional) Movement then fans this single output to 49 candidates, which are whittled down to five by locality and case constraints; Free Indexing expands these back to 36, and then the Theta Criterion, Control, and Condition B cut these back to just a single final LF (we shall see in detail how this is done next).

(3) *Which report did you file without reading?*

(4) 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]]]]]]]]]]]]]

(5) Final LF output:

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

Between Stage I's Parse S-structure output and the single, final LF output above 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 the computer snapshot. Numbers going in denote structures input to a principle module, and numbers out are those that make it through (either generated or filtered).

Briefly, the cascade of the remaining 24 principles runs like this:[4] Full Interpretation of syntactic adjuncts at S-structure and Sbar-deletion do not 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 essentially builds all possible chains by computing the set cross-product of possible links between existing empty categories and partial chains as the parser walks a candidate tree structure, extending partial chains or not as the parser compositionally traverses the tree structure it

[4]The principles are ordered as shown in the figure, but not the computer snapshot; note that 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 notice.

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 non-trace NP head for the chain.

This nondeterminism reflects the complete optionality of the underlying linguistic system. All movement is optional. Hence, for each empty category the parser can either decide or not for it to participate in an already existing partial chain, do nothing, be 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, violating 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 non-trace element (hence, an empty Operator can head a chain, for relative clauses).[5]

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 a trivial 1-element chain, or do nothing at all with it. If it has started a chain (it is the tail), then on composing the subtree in resides in with the subtree of the next highest maximal clause, the VP, we find that there are no additional empty NP candidates in the subtrees below the VP to select, so the VP simply inherits as its (partial) chains the base partial NP trace chain. Compositionally moving one more tree up, to the IP, the parser can now examine the subtrees of IP and take note of the candidate empty NP that is the subject of *reading*. Thus the parser may now freely select to extend its partial chain by using this free candidate or not (if that candidate were already the part of another chain, it would not be on the list of free candidates). Suppose it does. Similarly, a few compositional steps later, the parser may freely select to end the chain with the non-trace head *which report*. Thus the parser can and does output the following (ultimately failed) possibility, where *which report* is linked to both the object of *file* and the subject of *reading*. (Note the coindexing; here, the object of *file* is in effect an intermediate trace, and the object of *reading* is not part of any movement chain.) Note that some (movement) indices are now in place. Of course, we must ensure that this Chain Formation algorithm in complete, so that among all these possibilities we generate at least all the proper chains as well.

(6) $[_{C2}\ [_{NP}\ [_{Det}\ \textit{which}]\ [_{N1}\ \textit{report}]]_1\ [_{C1}\ [_{C}\ [_{C}\ ]\ [_{I}\ (\mathrm{Agr})\ [_{V}\ \textit{did}]]]\ [_{I2}\ [_{NP}\ \textit{you}]_2$
$[_{I1}\ [_{It}\ [_{VP}\ [_{VP}\ [_{Vt}\ [_{VP}\ [_{V}\ \textit{file}]\ [_{NP}\ \textit{NP-t}[\text{-A-P}]]_1]]\ [_{PP}\ [_{P}\ \textit{without}]\ [_{I2}\ [_{NP}$
$\textit{NP-t}[\text{-A-P}]]_1\ [_{I1}\ [_{It}\ [_{VP}\ [_{V}\ \mathrm{I}\ [_{V}\ \textit{reading}]]\ [_{NP}\ \textit{NP-}[\text{-A-P}]]]]]]]]]]]]]$

[5]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.

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 five will pass through both the Trace Case Condition (traces cannot 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).[6] 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 five possibilities to 36 by computing all possible indexings between any remaining unindexed NPs. Again, the process works essentially by forming the compositional cross-product of possible indices: if we have three indices i, j, and k, then either they are all unequal; i and j equal, k, unequal, etc.—precisely the problem of partitioning n elements (the NPs) into m sets (the indices). Features of the empty elements are fixed by the theory of Functional Determination, a theoretical choice point that can in fact be altered; the parser would then simply be using a different grammatical theory.

Finally, Control Theory cuts these 36 structures down to ten (ruling out uncontrolled PRO); passing through the Theta-Criterion and Binding Conditions A, B, C, the Empty Category Principle, LF movement, and Full Interpretation at LF, the parser eliminates all but one of the viable candidates, striking out (via Condition B for the most part) cases where PRO is bound in its minimal governing category where it should not be. Out of many hundreds of possible interactions, the single right result emerges, as displayed earlier.

This sketch does not say *how* the Stage I and Stage II processes are carried out in detail. While there is not enough space for a full description here (see Fong and Berwick 1992), we can give at least some of the details about how the quasi-S-structures are computed and how linguistic principles are represented and applied. Let us cover these topics in turn.

6.1 Stage I: Building Phrase Structure

The key idea to build the quasi-S-structures is to take some of the ideas of determinism from earlier systems – namely, using as much of the surface structure cues as possible to deduce the distribution of empty categories – and build that into a bottom-up, shift-reduce LR parser that is small

[6]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 the Trace Case Condition, saving some work as suggested earlier since fewer candidates will have survived.

enough to work efficiently, while at the same time producing only a relatively small number of candidate trees. Currently, we do this by building a covering grammar for S-structure, that varies from language to language; each language uses about 30 rules.

Figure 2 shows the major components of this covering grammar. As the figure shows, the covering grammar is formed from two parts: (1) the phrase structure for D-structure, as given by the instantiated X-bar rules for a given language, and certain empty category rules and adjunction rules, introducing empty NPs (*empty was followed John*) and empty elements for Adverbial adjunction ($[_{VP}$ *was followed John*$][_{Adv}$ *why*$]$; and (2) addition of movement at S-structure, including Argument and Nonargument movement, scrambling (via adjunction), and limited Head movement.

$$
\begin{array}{ll}
\text{S-structure rules} & = \\
\text{D-structure rules} & \left\{ \begin{array}{l}
\text{X-bar rules: } \overline{\overline{\mathbf{X}}} \rightarrow Spec, \overline{\mathbf{X}} \\
\mathbf{X} = N, Spec = Det \\
[_{NP}\ [_{Det}\ \textit{the}]\ [_{N'}\ \textit{destruction of the city}]] \\
\\
\text{Empty category rules: NP} \rightarrow \lambda \\
[_{IP}\ [_{NP}\ \textit{NP-ec}[\pm\text{A}\pm\text{P}]\ [_{VP}\ \textit{was followed John}]] \\
\\
\text{Adjunction rules: VP} \rightarrow \text{VP, Adv} \\
[_{VP}\ [_{VP}\ \textit{was followed John}]\ [_{Adv}\ \textit{why}]]
\end{array} \right. \\
+ & \\
\text{Movement rules} & \left\{ \begin{array}{l}
\text{CP} \rightarrow \text{Adv}, \text{C}'; \qquad \text{Adv} \rightarrow \lambda \\
[_{CP}\ [_{Adv}\ \textit{why}][_{C'}\ \textit{was John followed}]]
\end{array} \right.
\end{array}
$$

Figure 2: Components of the phrase structure grammar for S-structure are constructed by taking the D-structure grammar and adding adjunction, empty categories, and movement.

In more detail, the X-bar rules are binary branching and use unordered righthand sides. We assume for this parser that subjects are in Spec of IP. Parameters are incorporated by adding constraints on the 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 follows, where we leaved undefined the obvious auxiliary predicates. A schema compiler turns this form into an actual context-free rule by grounding through the lexicon (the parser compiler lets 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, for V= *persuade* in English the system automatically adds the rules,

(7) VP→ NP CP

To factor in NP movement at S-structure is actually now easy because empty NPs are already nondeterministically generated by existing covering grammar rules: all Argument positions (A-positions, subject and object) and Specifier of COMP already admit empty noun phrases at D-structure. To cover scrambling (for Japanese), we adopt an adjunction analysis of NPs at VP and IP, adding a single context-free rule. Other cases of movement, such as verb lowering and adverbial adjunction, are added in a case-by-case way.

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 certain parameters such as [±**specInitial**, ±**headInitial**], and the lack of *wh*-movement in syntax will bar certain movements in Japanese. Summarizing the system for the two grammars we have:

Language	X-bar prototype rules	Empty category rules	Adjunction rules	Others
English	12	5	8	6
Japanese	(same)	4	7	2

Precisely because the resulting grammar is small – one of the properties of principles and parameters 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, canonical LR(1) parsing that allows optimal early error detection, a must for a system that has to dispose of bad candidate structures as quickly as possible. This is one of the key techniques that was unavailable to TGG parsers two decades ago. Of course, the LR parser itself must be extended to handle ambiguity via the use of multiple action entries and backtracking, but this is now also a standardly available technology. The important point is that with a small grammar size, we can now achieve a balance between overgeneration and efficiency: the covering grammar and associated parser does select few enough candidate S-structures to make the system usable.

6.2 Stage II: Representing and Using Linguistic Principles

Given some (small) set of quasi-S-structures, Stage II of the parser next applies the remaining generators and filters to arrive at zero or more logical forms for the sentence. This is done by encoding the principles almost directly in PROLOG and then applying them to the quasi-S-structures. The exact order in which this is done has obvious implications for processing efficiency. First, principle application obviously must be subject to the *logical* dependencies of the theory, for instance, the Case Filter cannot apply until Case Assignment has been carried out.

The reader will note, however, that the principle ordering as specified does some unnecessary work in that Case Assignment and the Case Filter are applied to 49 structures, when in fact only 33 of these will survive Subjacency. Thus it would save computational work to order Subjacency ahead of the Case operations in this case. Plainly, following standard practice in artificial intelligence search techniques, the operative ordering principle is to delay hypothesis generation (expansion of the search tree) as long as possible, and apply constraints as early as possible. For a given sentence, up to an order of magnitude change in parsing time is possible, given an optimal principle ordering. We have explored this kind of reordering in three ways; see Fong and Berwick 1991 for details.

In one approach, the user may statically reorder the principles. In a second method, the system dynamically selects a tentative optimal ordering for a given sentence based in part on surface sentence cues and a voting scheme potentially based on the previous history of a given ordering's success rate. The implications of this for human sentence processing, namely, a systematic theory of individual processing time variation, has not escaped our attention but goes far beyond the scope of this article. Finally, we have systematically investigated *interleaving* principles, that is, the simultaneous application of two or more principles. For instance, we could enforce simultaneously all X-bar constraints, licensing of syntactic adjuncts, and a condition that traces cannot have Case. While this particular combination improves computation time, in general, contrary to intuition and some published accounts, combining *all* principles does not always improve performance, for more complex reasons that cannot be discussed here but roughly have to do with the computational complexity of each principle vs. the nondeterminism in building a set of S-structures. The current system can in part automatically determine which interleaving possibilities will prove efficacious.

Returning now to our main theme of representing linguistic principles, earlier we stated that a primary objective of this parser is to represent linguistic principles directly and transparently. To be concrete, how is Case Assignment carried out?

In the theory we have implemented, there is both *structural* and *inherent* Case assignment: assignment according to tree configuration and assignment according to the inherent properties of a lexical head item. For example, Structural Case Assignment is standardly done by assignment under government and adjacency (if that holds in a language). To implement this, the parser walks through a candidate structure, and, when it encounters a configuration of Case Assignment, that is, a maximal projection configuration CF with components Assigner (the structural Case assigner), Case (the Assigner assigns this Case) and an NP (the Case receiver), it first checks that (i) the Assigner governs the NP in the configuration; (ii) the NP has the feature *np*; (iii) the Assigner is in fact a Case Assigner, as determined previously; and (iv) the Assigner is adjacent to the NP, if the Case Adjacency parameter holds. If all this is true, then structural case assignment is carried out, by making the NP have the feature Case, and checking that the NP's morphology is compatible with the Case just assigned.

Let us expand on this English statement to see how it is implemented. The actual declarative statements that implement Structural Case Assignment are completely straightforward. 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 NP:

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

Now we must declaratively state the two predicates used above. We can directly translate the English given earlier for the first, `sCaseConfig`, adding a constraint (not discussed here) that the NP does not have the feature of *indirect Object*. In this statement, `cat` simply holds when a node has a category, possibly with features as given in lower case, and `max` holds when that category is maximal. Note that it is precisely here where the "matching" process implicit in interpreting declarative statements comes to the fore: the system "finds" the category C of configuration CF by starting out with C as a variable, without a value given to C, as if we were attempting to find the truth value of a Boolean statement in logic. The interpreted statement returns whatever sequence of assignments to C that makes the relation `cat(CF, C)` hold; similarly, in turn, once that stream of values is set, those values are tested by the predicate `max(C)`.

```
sCaseConfig(CF,Assigner,Case,NP) :-              % Case assign in config. CF
   cat(CF,C),                                    % when CF is a category C
   max(C),                                       % and it is maximal
   governs(Assigner,NP,CF),                      % and it governs a node NP
   cat(NP,np),                                   % andnode has feature np
   \+ NP has_feature indirObj,                   % and the NP is
                                                 % not an indir. obj.
   caseAssigner(Assigner,Case),                  % and the Assigner
                                                 % is a Case Assigner
   adjacent(Assigner,NP,CF)if caseAdjacency.     % and the Assigner is
                                                 % adjacent to the NP if the
                                                 % caseAdjacency parameter
                                                 % is true.
```

Putting aside for the moment exceptional Case marking and the requirement to transmit Case in certain situations, then we can state the second predicate very simply as well:

```
assignCase(Assigner,Case,NP) :-                  % Case assignment holds
     NP has_feature case(Case),                  % when NP has feature Case
     morphCaseAgreement(NP,Case).                % and that Case agrees
                                                 % morphologically with NP,
                                                 % if any.
```

We should stress that this declarative way of putting matters does not "make" or "assign" the NP the feature Case. Rather, the system will non-deterministically generate both possibilities where the NP does not have the feature Case, *and* where it does, in an attempt to match against this filter's logical template. Only in this last case will the `assignCase` predicate be true, as required to pass the filter. In this way, the candidate trees are successively generated and pruned.

7 Linguistic Coverage

To give some idea of the range of examples that the parser can handle, Table 2 gives a partial list of the sentence types from LU that the parser can successfully parse or reject, as appropriate, along with the chapter and example sentence number from LU. About 300 sentence example types can be processed, corresponding to most examples of current interest in linguistic theory, from quantifier raising, to passivization, to exceptional case marking, to superiority effects (*Who will read what*/vs. **What who will read*). As far as we know, this is the most comprehensive system of its kind.

(1:12a)	Someone likes everyone	*Quantifier raising*
(1:15a)	Who that John knows does he like	*Movement & Condition C*
(1:15b)	He likes everyone that John knows	
(1:18)	It is likely that John is here	*Simple Case theory*
(1:19)	*It is likely John to be here	
(1:21)	I am eager for John to be here	*Exceptional Case Marking*
(1:22)	*I am eager John to be here	*(Ecm)*
(1:24a)	I believe John to be here	
(1:24b)	I believe John is here	
(1:25)	*I believe sincerely John to be here	*Case Adjacency*
(1:35a)	I want to be clever	*Optional vs. obligatory*
(1:35b)	*I believe to be clever	*Ecm*
(1:36a)	John was persuaded to leave	*Ordinary passivization*
(1:48)	John was arrested by the police	
(1:36c)	*John was wanted to leave	*Exceptional passivization*
(1:49a)	I believe John to be intelligent	
(1:49b)	*It was believed John to be intelligent	
(1:49c)	John was believed to be intelligent	
(1:52a)	*I am proud John	*Genitive Case realization*
(1:52b)	I am proud of John	
(1:53a)	I wonder who you will see	Wh-*movement*
(1:59c)	*What does Bill wonder who saw	*and Subjacency*
(2:19b)	Their pictures of each other are nice	*Simple Binding theory*
(2:26a)	John likes Mary's pictures of him	
(2:26b)	*John likes Mary's pictures of himself	
(2:45a)	Who does he think Mary likes	*Strong crossover*
(2:88)	*(I am proud of) my belief to be intelligent	*Government of* PRO
(2:91)	The men think that pictures of each other will be on sale	
(2:103)	*The men think that Mary's pictures of each other will be on sale	
(3:17)	Which report did you file without reading	*Parasitic gaps*
(3:18)	*Which book did you file the report without reading	
(3:19)	*Who filed which report without reading	
(3:20)	*The report was filed without reading	↙*Resumptive pronoun*
(3:46)	*The article which I filed it yesterday without reading is (over) here	
(4:3)	*John is crucial to see this	*ECP*
(4:4)	John is certain to see this	
(4:7a)	Who do you think that John saw	*That-trace effect*
(4:7b)	Who do you think John saw	
(4:8a)	*Who do you think that saw Bill	
(4:8b)	Who do you think saw Bill	
(4:18)	*John is believed is intelligent	*Raising and*
(4:20)	*John seems that it is likely to leave	*Super-raising*

(Table 2 continued on next page)

(4:21b)	Who will read what	*Superiority*
(4:21c)	*What will who read	
(4:35a)	Why did you read what	*Compl./Noncompl. asymmetries*
(4:35b)	*What did you read why	
(4:45a)	Who believes the claim that Mary read what	*Subjacency and* LF
(4:45b)	*What do you believe the claim that Mary read	
(4:57)	*Who does Mary wonder why John hit	*Ecm and movement*
(4:58)	*Why does Mary wonder who John hit	

Table 2: Some of the examples from LU that are analyzed correctly by the English version of the principles and parameters parser.

8 Parameterizing the parser: Japanese

While the substantial coverage of a set of linguistic examples for English may be impressive, the "proof of the pudding" for the principles and parameters parser is, of course, whether one can easily switch from one language to another by varying just a small number of parameters. In fact this appears to be possible. In this section we show how the parser can easily be reparameterized to handle the *wh*-movement sentences found in *On the nature of proper government* (Lasnik and Saito 1984), listed in Table 3 (sentence numbers in the table refer to the original cited article). In passing, we shall show how the parser may be used in practice, to debug the principles as implemented, and actually "discover" valid linguistic derivations that might otherwise not have been found.

To begin, these Japanese sentences display many familiar typological Japanese–English differences. Let us review these here.

- *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, as is familiar. The X-bar system compiles out schemas with heads last rather than first, as mentioned in the previous section. (To a first approximation, this does not alter the LR machinery in terms of the amount of ambiguity as measured by alternative state changes, but see Fong and Berwick 1992 for additional discussion.) We also assume without further discussion the existence of a VP node in Japanese.

(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)*	
(63)	Kimi-wa nani-o sagashiteru no	*(Why are you looking for what)*	*Multiple-*whs *in Comp*

Table 3: *Wh*-movement examples in Japanese from Lasnik & Saito

- *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). Typical examples cited by Hoji (1985) of *John gave Mary a book* are these:

  (8) (i) $[_{\mathrm{IP}}$ *John-ga* $[_{\mathrm{VP}}$ *Mary-ni hon-o ageta*$]]$
  (ii) $[_{\mathrm{IP}}$ *hon*$_i$*-o* $[_{\mathrm{IP}}$ *John-ga* $[_{\mathrm{VP}}$ *Mary-ni* t_i ageta$]]]$
  (iii) $[_{\mathrm{IP}}$ *Mary*$_i$*-ni* $[_{\mathrm{IP}}$ *John-ga* $[_{\mathrm{VP}}$ t_i *hon-o ageta*$]]]$
  (iv) $[_{\mathrm{IP}}$ *John-ga* $[_{\mathrm{VP}}$ *hon*$_i$*-o* $[_{\mathrm{VP}}$ *Mary-ni* t_i *ageta*$]]]$

  Movement can account for this. Suppose the canonical order is "subject" followed by "indirect object" followed by "direct object," as in (8)-i. 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 above, or to a sentence-initial position (by *S-adjunction*) as in the second. 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 immediately below 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(argument) to a non-Argument 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.) [7]

- *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 can not 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 the second sentence below (taken from Makino and Tsutsui 1986).

[7]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 indirect object greatly increases computation time. A tighter set of constraints on scrambling seems called for.

(9) (i) Taro-wa sono mise-de nani-o kaimashita ka
(What did Taro buy at the store?)
(ii) Pen-o kaimashita
(He bought a pen)

Standardly, the omitted subject is actually represented in syntax by a phonologically empty pronoun.[8] Again following conventional practice, we represent the binary option that determines whether this is available or not as the *pro*-drop parameter. For the initial parses described in this section, we do not extend this ellipsis to non-subject positions, though again this may be easily changed, simply by saying that *pro* may occupy other NP positions (though as far as we can determine informally with a great loss in efficiency).

- *No visible Wh-movement.*

  Following much current work in linguistic theory (Huang 1982), we assume that Japanese LF looks like English LF: there is no *wh*-movement in syntax, but there is movement at LF. Thus, the option of whether to allow *wh*-movement between D- and S-structure is a language parameter. As we review below, it is this distinction that enables one to explain a variety of facts including why the counterpart of a sentence such as (6) in Table 3 (which is well-formed in Japanese) is ill-formed in English.

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 in Japanese of double Subjects, Case marking, passives, causatives, and so forth. Rather, it is sufficient to demonstrate what we set out to show: to cover the examples shown in Table 3 with just a handful of parameter switches. These parameters are shown in Table 4, with English–Japanese differences starred.[9] This table shows the actual PROLOG code parametric differences that were entered in order to get the parser to parse Japanese rather than English—in effect, just four binary switches to make the system Head final; allow *pro*-drop everywhere; eliminate adjacency for Case marking and thus admit scrambling; and eliminate *wh*-movement in syntax for Japanese.

It is remarkable that the same set of principles for English can then recombine in different ways to handle the Japanese examples. The important

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

[9]Some of these parametric variations lead to implicational universals: for instance, example (187) in Lasnik and Saito (1984) states that if a language has syntactic *Wh*-movement then it obeys the *Wh*-Comp requirement at S-structure.

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

Table 4: 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 asterisks.

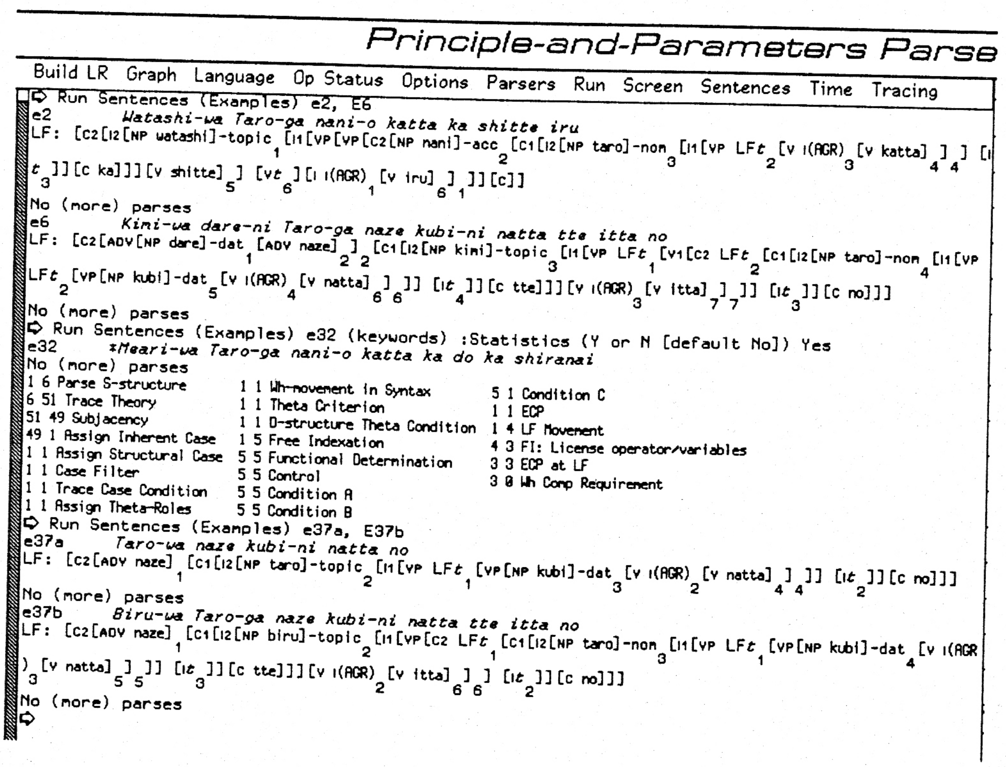
point here again is that the system gets (by design) precisely the parses required, and blocks ungrammatical sentences by the same means as well. Figure 3 shows the system in operation for the first five sentences from Table 3, while Figure 4 displays the LFs for the sixth example sentence. Let us review these.

- Wh-*movement at* LF.

  As shown at the top of the snapshot in Figure 3, *nani* has moved at LF to a position that has scope over the embedded 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, the parser moves this item 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 to the left, immediately after the C2 bracket in the figure.

  The key point is why has this (correct) parse been produced with *nani* moved, rather than any others? Remember that the system will nondeterministically try out *all* optional possibilities. So the


Principle-and-Parameters Parse
Build LR Graph Language Op Status Options Parsers Run Screen Sentences Time Tracing

```
⇨ Run Sentences (Examples) e2, E6
e2        Watashi-wa Taro-ga nani-o katta ka shitte iru
LF: [c2[i2[NP watashi]-topic [i1[vP[vP[c2[NP nani]-acc [c1[i2[NP taro]-nom [i1[vP LFt [v i(AGR) [v katta] ] ] [i
                            1                          2                  3         2         3            4 4
t ]][c ka]]][v shitte] ] [vt ][i i(AGR) [v iru] ] ]][c]]
 3                    5     6          1        6 1
No (more) parses
e6        Kimi-wa dare-ni Taro-ga naze kubi-ni natta tte itta no
LF: [c2[ADV[NP dare]-dat [ADV naze] ] [c1[i2[NP kimi]-topic [i1[vP LFt [v1[c2 LFt [c1[i2[NP taro]-nom [i1[vP
                        1           2 2                     3         1          2                   4
LFt [vP[NP kubi]-dat [v i(AGR) [v natta] ] ]] [it ]][c tte]]][v i(AGR) [v itta] ] ]] [it ]][c no]]]
   2                5         4          6 6     4                    3         7 7     3
No (more) parses
⇨ Run Sentences (Examples) e32 (keywords) :Statistics (Y or N [default No]) Yes
e32      *Meari-wa Taro-ga nani-o katta ka do ka shiranai
No (more) parses
1 6 Parse S-structure         1 1 Wh-movement in Syntax           5 1 Condition C
6 51 Trace Theory             1 1 Theta Criterion                 1 1 ECP
51 49 Subjacency              1 1 D-structure Theta Condition     1 4 LF Movement
49 1 Assign Inherent Case     1 5 Free Indexation                 4 3 FI: License operator/variables
1 1 Assign Structural Case    5 5 Functional Determination        3 3 ECP at LF
1 1 Case Filter               5 5 Control                         3 0 Wh Comp Requirement
1 1 Trace Case Condition      5 5 Condition A
1 1 Assign Theta-Roles        5 5 Condition B
⇨ Run Sentences (Examples) e37a, E37b
e37a      Taro-wa naze kubi-ni natta no
LF: [c2[ADV naze] [c1[i2[NP taro]-topic [i1[vP LFt [vP[NP kubi]-dat [v i(AGR) [v natta] ] ]] [it ]][c no]]]
                 1                     2          1                3         2          4 4     2
No (more) parses
e37b      Biru-wa Taro-ga naze kubi-ni natta tte itta no
LF: [c2[ADV naze] [c1[i2[NP biru]-topic [i1[vP[c2 LFt [c1[i2[NP taro]-nom [i1[vP LFt [vP[NP kubi]-dat [v i(AGR
                 1                     2             1                   3          1                4
) [v natta] ] ]] [it ]][c tte]]][v i(AGR) [v itta] ] ] [it ]][c no]]]
3           5 5     3                    2         6 6     2
No (more) parses
⇨
```

Figure 3: Sample sentences from Lasnik & Saito, as shown in a computer snapshot.

real trick is ruling out the *bad* possibilities, not generating the good one. Among these failed possibilities are candidates where *nani* is not moved at LF, candidates where the incorrect phrase structure is built, and so forth. The good option is where *nani* is moved at LF, leaving behind an LF trace. We can divide our question then into two parts: first, Why is the correct phrase structure built? and second, Why does only the right LF-movement possibility survive the principle gauntlet?

First, as we described earlier, the correct branching phrase structure is built simply because the LR(1) parser has the rules to build it *at least* that way. Although the LR machine may build *more* structures than ultimately required (to be precise, in this case, five others), among them will be just those that obey the branching conditions of the language in question. This is forced by the construction of the covering LR(1) machine given the parameters `headFinal` and `whInSyntax`. Note also that the covering grammar will not predict *any* empty categories in this example, simply because they cannot be introduced into the quasi-S-structure at all; they are just not part of the S-structure covering grammar for Japanese (putting to one side inflection-verb Head movement).

Second, the correct movement is forced by the fact that Japanese does not have *wh*-movement in syntax and English does, plus a universal constraint (posited by Lasnik and Saito 1984), the Wh-Comp Requirement, that both English and Japanese must meet: a $+wh$ Comp must have a $+wh$ head and a $-wh$ Comp must not have a $+wh$ head. In English, this constraint must be met at both S-structure and LF, while in Japanese, lacking *Wh*-movement in syntax, it holds just at LF. A Comp is $+wh$ if marked by its head or, in English, by a Q(uestion)-operator, if the specifier is $+wh$. We must modify this constraint slightly in our X-bar system, since the parser has both a Spec and Head position: since the *wh* element is moved into Spec, the revised constraint says that a $+wh$ Comp must have a $+wh$ Spec, and a $+wh$ Head, if there is one.

In the current Japanese example, the embedded CP is marked $+wh$ by its head *ka*, as indicated in the lexicon. *Nani* must move because by assumption *wh* elements are operators and to be licensed, an operator must bind something, either an S-structure or LF-trace. Only the second option is possible in Japanese. So the movement must occur or else the operator won't be licensed. The movement is licensed because it passes the Wh-Comp Requirement, as modified above. Thus, structures where *nani* is not moved do not meet this restriction: if the Spec of CP is left empty, the Wh-Comp Requirement is unmet. In fact, two of three final candidates are blocked in just this way, leaving the single correct parse shown. Of course, the LF trace left behind must also meet the Empty Category Principle at LF, and it does: it is lexically governed as a complement of the verb *katta*, 'buy,' as may be seen in Figure 3. So all is well.

- *Multiple wh-elements at* LF.

Consider example (6) in Figure 3. There are two *wh*-elements: (1) *dare* ('who') the indirect object of *itta* ('said' as in "said something to somebody"), and (2) *naze* ('why'). As the second parse in Figure 3 shows, both elements will move at LF to a sentence-initial position, with *naze* moving Comp-to-Comp; this is the LF given by Lasnik and Saito (1984). The usual ambiguity arises. We can gloss the sentence meaning as a double question: "for which x, x a person, *and* for what reason y, you said y (a reason such that [taro was fired]) to x."

Why does this sentences pass in Japanese but not in English? Again the reason is the lack of *wh*-movement at S-structure in Japanese leaving behind an intermediate trace governing the original trace of *naze*.

- *Complement & noncomplement asymmetries; Scrambling and unexpected parses; Trace deletion.*

  Finally, consider example (39a) (and the corresponding illicit 39b) from Figure 3, repeated here, where a complement *wh* but not a noncomplement *wh* can be extracted from a complex NP:

(10) (i) Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no
(What are you so angry about the fact that Taro obtained)

(ii) * Taro-ga naze sore-o te-ni ireta koto-o sonnani okotteru no
*(*Why are you so angry about the fact that Taro obtained)*

This example illustrates several Japanese typological differences with English. The subject of the matrix clause 'you' has been omitted. *Nani* and *te* ('hand') have been scrambled, with the direct object marked *-o* now appearing in front of the indirect object *te*. Our relaxation of the Case Adjacency parameter and the rule that allows adjunction of NP to VP, plus transmission of Case to the scrambled NP will let this analysis through. The LF for this sentence might be glossed something like this:

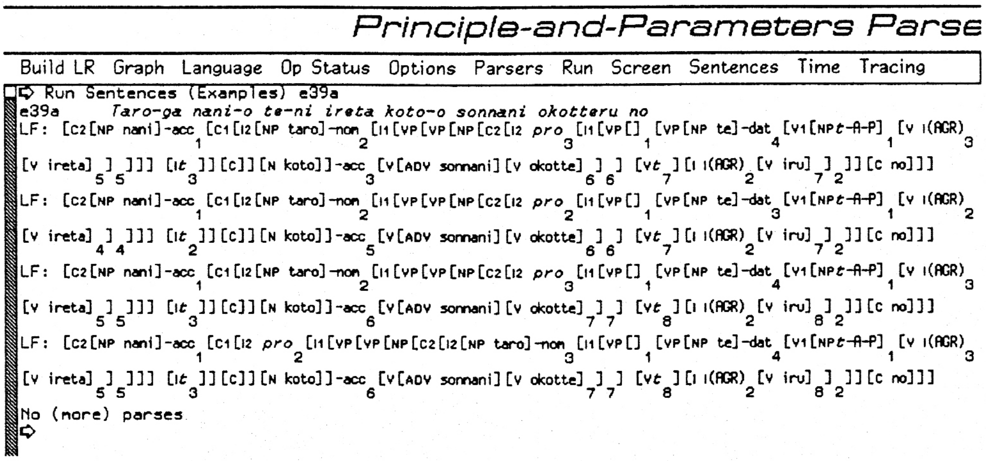
(11) for what x, *pro* is so angry about the fact that Taro obtained x

In this example *pro* denotes the understood subject of *okotteru* ('be angry'). Note the LFs actually returned by the system in Figure 4. As one can see, the system does correctly recover this form, as the last LF in the snapshot. However, it also recovers three *additional* LFs:

(12) for what x, Taro is so angry about the fact that *pro* obtained x

The parser has also admitted the LF where the embedded subject *Taro* is interchanged with the matrix subject *pro*.

Before we address where the additional ambiguity comes from, we should account for the expected derivation itself. *Nani* ('what') undergoes both S-structure and LF movement. At S-structure, the direct object "scrambles" by VP-adjunction, leaving the trace NP_1. At LF, *nani* undergoes *wh*-movement into the specifier position of CP. This would leave an illicit intermediate trace in the adjoined VP, as indicated here:



```
⇨ Run Sentences (Examples) e39a
e39a      Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no
LF: [C2[NP nani]-acc [C1[I2[NP taro]-nom [I1[VP[VP[NP[C2[I2 pro [I1[VP[] [VP[NP te]-dat [V1[NPt-A-P] [V I(AGR)
                    1                   2                      3        1              4            1         3
[V ireta] ] ]]] [It ]][C]][N koto]]-acc [V[ADV sonnani][V okotte] ] ] [Vt ][I I(AGR) [V iru] ] ]][C no]]]
          5 5     3                    3                          6 6    7           2        7 2
LF: [C2[NP nani]-acc [C1[I2[NP taro]-nom [I1[VP[VP[NP[C2[I2 pro [I1[VP[] [VP[NP te]-dat [V1[NPt-A-P] [V I(AGR)
                    1                   2                      2        1              3            1         2
[V ireta] ] ]]] [It ]][C]][N koto]]-acc [V[ADV sonnani][V okotte] ] ] [Vt ][I I(AGR) [V iru] ] ]][C no]]]
          4 4     2                    5                          6 6    7           2        7 2
LF: [C2[NP nani]-acc [C1[I2[NP taro]-nom [I1[VP[VP[NP[C2[I2 pro [I1[VP[] [VP[NP te]-dat [V1[NPt-A-P] [V I(AGR)
                    1                   2                      3        1              4            1         3
[V ireta] ] ]]] [It ]][C]][N koto]]-acc [V[ADV sonnani][V okotte] ] ] [Vt ][I I(AGR) [V iru] ] ]][C no]]]
          5 5     3                    6                          7 7    8           2        8 2
LF: [C2[NP nani]-acc [C1[I2 pro [I1[VP[VP[NP[C2[I2[NP taro]-nom [I1[VP[] [VP[NP te]-dat [V1[NPt-A-P] [V I(AGR)
                    1          2                               3        1              4            1         3
[V ireta] ] ]]] [It ]][C]][N koto]]-acc [V[ADV sonnani][V okotte] ] ] [Vt ][I I(AGR) [V iru] ] ]][C no]]]
          5 5     3                    6                          7 7    8           2        8 2
No (more) parses
⇨
```

Figure 4: Japanese examples illustrating ambiguous logical forms and scrambling.

(13) $[_{\text{CP}}\ [_{\text{NP}}\ \textit{nani}]\text{-acc}_1 \ldots [_{\text{I1}}\ [_{\text{VP}}\ [_{\text{VP}}\ [_{\text{NP}}\ [_{\text{CP}}\ [_{\text{IP}}\ \textit{pro}\ [_{\text{I1}}\ [_{\text{VP}}\ \text{NP-}t_1 \ldots$

This structure should be ruled out: the intermediate LF trace is barred by the Empty Category Principle because it is in a non-argument position, and it is neither lexically governed as the complement of a verb nor antecedent governed by *nani* (too many barriers intervene, with two CPs and an NP, at least, blocking antecedent government). This is just what we discovered when the parser was first run on this example sentence.

To repair this example, observe that this is precisely the situation where a trace deletion approach would work. We therefore implemented something along these lines, permitting LF trace deletion in just this configuration. Thus we find the form as actually shown in the top line of every LF in Figure 4, with the LF trace being subsequently deleted, as indicated by [VP $[\ ]_1$]. This element is thus no longer subject to the Empty Category Principle, and the structure passes. In effect, we have optional deletion of LF traces only.

Turning now to the ambiguity the parser discovered, the basic prediction seems to be correct. The sentence happens to be ambiguous with respect to the two basic interpretations.[10]

[10]This was pointed out by Pesetsky, and confirmed by Saito. However, presumably the use of *wa* rather than *ga* and intonational pauses could be exploited by a computer (or a person) as a surface cue to rule out more general ambiguity in this example and others like it. See Fong and Berwick (1991) for a discussion of how to integrate sentence surface cues into the principle-based system.

For completeness, here are the three variants that correspond to the first three LFs reported by the parser. S. Miyagawa (p.c.) informs us that the last two, given proper context, are in fact possible.

1. *pro* is coreferent with *koto* ('fact'):
   for what x, Taro is so angry about the fact that the fact obtained x
   This interpretation can be eliminated by imposing selectional restrictions on the possible "agents" of *okotteru* (let us say that they must be animate).
2. *pro* is coreferent with *taro*:
   for what x, Taro is so angry about the fact that Taro obtained x
3. *pro* is free in the sentence:
   for what x, Taro is so angry about the fact that (someone else) obtained x

To summarize, using a very simple and automatic parameterization, exactly as suggested by theory, we can accommodate a range of Japanese constructions that differ markedly from English. Without any rule reprogramming or changes in the principles or parsing algorithm we can obtain a parser for Japanese that works as the English system did (in particular, there are no difficulties with Head final constructions). This is the promise of the principles and parameters theory, now met by a concrete implementation.

9 Conclusions

Coming full circle then, just as it seems altogether fitting that a historical celebration of a linguistics department should be in part historical, it seems particularly fitting that just in time for this celebration we have been able to develop a full, efficient parser for TGG. The barriers have at last come down: it took a quarter-century of research into constraints on TGG—in particular, restrictions on movement and empty categories along with a modular, declarative design—and a quarter-century of research on computation—in particular, LR parsing and its integration with an efficient system for nondeterministic, declarative computation (here, PROLOG). What lies in store a quarter-century ahead remains hidden. No doubt our current views will look as curious to us then as the MITRE system does now. However, if we are on the right track, then the same lessons of constraint and efficiency that guided work in the 60s until the current day will continue to apply in full force into the foreseeable future—more than enough of an historical moral with which to conclude.

coupled with the Wh-Comp Requirement. Assuming the particle *no* marks the Comp as +*wh*, then for CP to receive a +*wh* Spec, both *dare* and *naze* must move, as in the previous example, and the candidates where these elements do not move are ruled out.

All other options explored by the parser are ruled out. If the intermediate trace is not present, or if *dare* as an NP moves into Spec *first* and *naze* is then adjoined to it, then the base trace is not properly governed, with CP or NP as barriers, respectively. The (declaratively stated) Empty Category Principle blocks these possibilities. Thus only the winning output structure shown can run the entire constraint gauntlet.

Note that the corresponding English sentence would be ruled out: example 4:35b in the previous section is similar (**what did you read why*). Because English has movement at S-structure, the LR machine (*not* a later operation) will put *what* in Spec of CP, and place an empty category after *read*. *Why* must move as an operator as before, but the LF trace of *why* cannot be properly governed because it is adjoined to the NP *what*; this NP acts as a barrier for proper government, ruling out the possibility of antecedent government. The LF trace cannot be lexically governed either by *read* since it is not a complement. In this way, the parser captures a basic difference between English and Japanese in the same way, while maintaining a single Empty Category Principle.

- *Nonadsorption/semantic parallelism and the Wh-Comp requirement.*

  Sentence example (32) in the table illustrates another Japanese grammatical effect. Evidently the question element *ka do ka* ('whether or not') and the *wh* phrase *nani* ('what') cannot both appear in the same (lower) Comp at LF. The relevant condition is one of "semantic parallelism," which we have implemented by a feature unification check. We simply add the feature *yn* in the lexicon to items such as *ka* or *ka do ka*, but not to *nani*, and require this feature to be compatible with the element in Spec. The example output in Figure 3 shows that in fact three candidates are produced by the parser but none of them meet the Wh-Comp Requirement, so the sentence is blocked.

- *Comp to Comp movement at LF.*

  Like example sentence (6), example sentence (37b) ('Why did Bill say that John was fired', literally, 'fired'='cut off at the neck') shows how *naze* can move at LF out of its position adjoined to the verb *natta*, first to the lower Spec of CP, and then to the Spec of the matrix CP,

References

Berwick, Robert C. 1985. *The acquisition of syntactic knowledge.* Cambridge, MA: MIT Press.

Chomsky, Noam. 1986. *Barriers.* Cambridge, MA: MIT Press.

Fodor, Jerrold, Thomas Bever, and Merrill Garrett. 1974. *The psychology of language.* New York: McGraw-Hill.

Fong, Sandiway. 1991. *Computational properties of principle-based grammatical theories.* Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Fong, Sandiway and Robert C. Berwick. 1991. The computational implementation of principle-based parsers. In M. Tomita (ed.), *Current issues in parsing technologies,* pp. 9–24. Norwell, MA: Kluwer.

Fong, Sandiway and Robert C. Berwick. 1992. *Cartesian computation.* Cambridge, MA: MIT Press.

Friedman, Joyce. 1971. *A computer model of transformational grammar.* New York: American Elsevier.

Hoji, H. 1985. *Logical form constraints and configurational structures in Japanese.* Ph.D. dissertation, Department of Linguistics, University of Washington, Seattle.

Huang, James. 1982. *Logical relations in Chinese and the theory of grammar.* Ph.D. dissertation, Department of Linguistics and Philosophy, Massachusetts Institute of Technology.

Kolb, Hans and Craig Thiersch. 1990. Levels and empty categories in a Principles and Parameters approach to parsing. In H. Haider and K. Netter (eds.), *Representation and derivation in the theory of grammar.* Dordrecht: Kluwer.

Lasnik, Howard and Mamoro Saito. 1984. On the nature of proper government. *Linguistic Inquiry* 15:2.235–289.

Lasnik, Howard and Juan Uriagereka. 1989. *A course in GB syntax.* Cambridge, MA: MIT Press.

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

Maratsos, Michael P. 1978. New models in linguistics and language acquisition. In M. Halle, J. Bresnan, and G. Miller (eds.), *Linguistic theory and psychological reality,* pp. 247–263. Cambridge, MA: MIT Press.

Marcus, Mitchell P. 1980. *A theory of syntactic recognition for natural language.* Cambridge, MA: MIT Press.

Nozohoor-Farshi, R. 1987. Context-freeness of the language accepted by Marcus' parser. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics,* pp. 117–122.

Peters, P. Stanley. 1973. On restricting deletion transformations. In M. Gross, M. Halle, and M. Schutzenberger (eds.), *The formal analysis of natural language,* pp. 372–384. The Hague: Mouton.

Petrick, Stanley Roy. 1965. *A recognition procedure for transformational grammars.* Ph.D. dissertation, Department of Foreign Languages, Massachusetts Institute of Technology, Cambridge, MA.

Plath, Warren J. 1976. Request: A natural language question-answering system. *IBM Journal of Research and Development* 20:4.326–335.

Stabler, Edward P., Jr. 1992. *The logical approach to syntax.* Cambridge, MA: MIT Press.

Takezawa, T. 1987. *A configurational approach to case marking in Japanese.* Ph.D. dissertation, Department of Linguistics, University of Washington, Seattle.

Zwicky, Arnold, Joyce Friedman, Barbara Hall, and Donald Walker. 1965. The Mitre syntactic analysis procedure for transformational grammars. In *AFIPS Fall Joint Computer Conference,* pp. 317–326. Washington, DC: Spartan Books.