ROBERT C. BERWICK AND KENNETH WEXLER

PARSING EFFICIENCY, BINDING, C-COMMAND AND LEARNABILITY

A principal goal of modern linguistic theory has been to formulate constraints on grammars so as to explain how it is that linguistic knowledge can be *acquired*. The aim has been to narrow the class of possible grammars so that it is as small as possible, consistent with observed variation in natural grammars. But from the very beginning of the modern study of generative grammar, there has been another "functional" motivation that has been used — though less frequently — to constrain the class of possible grammars. This is the demand of parsability or (in its dual sense), of generability. For example, we might require that natural grammars be amenable to "easy" recognition or generation, in some sense. This demand has actually been explicit since the earliest days of the field, as the following quote from Chomsky's *Morphophonemics of Modern Hebrew* [1951] indicates:

> The criteria of simplicity governing the ordering of statements is as follows: that the shorter grammar is the simpler, and that among equally short grammars, the simplest is that in which the average length of derivation of sentences is least.

If we identify "short" grammars as a proxy measure of acquisition simplicity, and "number of derivation steps" as a proxy for recognition or generation complexity, then this passage identifies the two crucial complexity measures for grammars.

An important question is how these measures interact. In this paper we will consider how one constraint that has been advanced in current linguistic theory can be viewed as a constraint that aids parsing, rather than, as is typically (and tacitly) the case, as a constraint that aids learning. This is the constraint of constituent command, or C-command. We will show that C-command helps produce "short derivations," basically by reducing the number of computational steps required for co-indexing. In particular we will show that C-command can be quite readily embedded in an efficient parser for English based on the design of Marcus [1980]. Moreover, the resulting design seems to comport with a natural model of on-line sentence processing. Here we make use

B. Lust (ed.), Studies in the Acquisition of Anaphora, Vol. II, 45—60.

of well-known formal models drawn from compiler theory, as discussed in Aho and Ullman [1972].

To begin, let us first review the basic terminology and facts about C-command and co-indexing. How do quantifiers bind pronouns? In structural terms, an important law of binding states, that quantifiers in their structural position must *C-command* the pronoun that they bind. (See Chomsky [1981], based on work by Higgenbotham [1980]) More precisely, the *trace* of a quantifier, as positioned by a rule of Quantifier Raising (QR) or wh-movement must C-command the pronoun that it binds. We will say that α *C-commands* β iff the first branching category that dominates α dominates β; some variation in this core idea is probably necessary, but this definition will serve here. Finally, for our purposes in this paper it will be enough to assume that it is the quantifier at S-structure that must C-command a bound pronoun. Thus we ignore in the sequel a level of "logical form". So for example, in (1) below the quantifier *every* C-commands *him* and the sentence is good; in contrast, in (2) *every* does not C-command *he* and the sentence is out.

(1) Everyone$_i$ thinks Mary will write a book about him$_i$.
(2) *The woman over there who has met everybody$_i$ in the room thinks he$_i$ is silly.

The binding of pronouns by quantifiers is an interesting parsing problem precisely because it is a case of non-local interaction: the distance between quantifier and pronoun is not limited by a certain number of *terminal* elements or even by a certain number of intervening phrase structure nodes. This is in distinct contrast to the binding of NP's by traces, which must obey Subjacency; here, while the separation between NP and trace can be arbitrary in terms of number of terminal items, the number of cyclic nodes that can intervene is strictly bounded.

Generally speaking, the problem of binding a quantifier to a pronoun could be viewed as one of linking two elements in a parse tree. (For now we shall be deliberately vague about just how this tree is to be represented.) At a certain level of abstraction, the problem of binding is just that of co-indexing. Whatever the computational system assumed, when a pronoun is reached one must locate the quantifier (if any) that serves as its antecedent, and assign the pronoun the index of the quantifier. Plainly, there are many possible ways of constructing a system so that this task can be carried out.

To calculate the actual computational effort involved in binding a quantifier to a pronoun, we must advance a specific computational model — a parsing model. In this paper we will adopt a variant of the Marcus parser [1980]. In brief, this machine is a restricted two-stack parser that analyzes sentences left-to-right. One stack holds phrases under construction — phrases that are partly built, but whose argument structure has not yet been completely analyzed. This stack is called the active phrase stack. The other stack, the input buffer, holds a finite number of words of the sentence under analysis. These data structures are actually push-down stacks because elements enter and exit them in last-in, first-out order. For example, consider the analysis of the sentence. *The woman who likes the man thinks that he is silly.* At the point where the tokens *he is silly* are in the input buffer, the previous tokens *The woman who likes the man thinks that* will have been already analyzed into three partially built constituents: an $\overline{S}$ node, to which is attached an NP (*The woman who likes the man*); a VP node, to which is attached a Verb *thinks*, and, the most recently created node, another $\overline{S}$ with *that* attached as a Complementizer. Note that all three of these phrases — $\overline{S}$, VP, and $\overline{S}$ — are not yet completely built. The old $\overline{S}$ lacks its VP, the VP lacks its $\overline{S}$ complement, and the new $\overline{S}$ lacks an NP and a VP. Crucially, these not-yet-completed nodes are accessible to the parser according to the recency of their construction: the newest $\overline{S}$ is on the top of the stack, and is most accessible phrase; the VP is the next most recently created phrase, and the $\overline{S}$ that is the root of the sentence is the oldest and hence at the bottom of the stack of uncompleted phrases.

Suppose now that there is no C-command restriction on pronoun-quantifier binding. Given the model sketched above, let us calculate the maximum possible number of computational steps involved in recovering the index of an antecedent quantifier. The worst case is when we have a very deep NP structure attached to an $\overline{S}$, with the $\overline{S}$ a phrase in the stack, and this NP contains a large number of NP's that must be searched to find the right antecedent. For example, suppose we are given a structure like this:

(3) $[_{NP}\,N[_{S}\,NP[_{VP}\,V\,NP[_{S}\,NP[_{VP}\,V\,NP]]]]]$

The correct quantifier phrase could be any of the NP's in this list. If we assume that the time it takes to find the right phrase is proportional to

the number of nodes that must be searched, then evidently the number of steps this could take is at most proportional to $\underline{n}$, where $\underline{n}$ is the number of terminal elements (lexical items, roughly) up to the point where the pronoun was encountered. This is because there can be a tree of width $\underline{n}$ at the bottom where the number of internal NP nodes is also proportional to $\underline{n}$ — as is the case in the example above. More formally, suppose that the constituent structure is generated by a grammar in Chomsky normal form, so that all phrases are generated by context-free re-write rules of the form $A \Rightarrow BC$, $A \Rightarrow a$. In a symmetric tree of depth j before terminals, the length of the terminal string is $\underline{n} = 2^j$. The number of nodes in the tree is $2^{j+1} = 2n - 1$. If half of these are NP's then the number of NP's is proportional to $\underline{n}$.

Now assume that the C-command condition applies. Now any NP's below the first are inaccessible to the pronoun, blocked by the C-command restriction. In other words, if "X" is a not-yet-completed phrase in the stack, the C-command restriction limits the search for antecedents to immediate maximal projections attached to that "X" — call these maximal projections "Y" phrases. But any phrases in turn attached to Y's are inaccessible. Now what does the possible search space look like? The phrase stack in the Marcus parser can be at worst proportional to the depth of the constituent structure tree being built. Besides this factor, we must search at most some constant number of NP's per phrase in the stack (assuming now some fixed upper limit on the number of NP arguments per S). Therefore, given C-command, the number of NP's to be searched will be proportional to j, the depth of the parse tree, rather than its width. For a perfectly symmetrical tree, this amounts to a reduction in search time from $\underline{n}$ to $\underline{\log n}$. This saving is nearly achieved in the NP case described above. For a right-branching (linear) tree, there is no gain, since the width of such a tree is proportional to its depth. We conclude that at least in certain cases, the C-command restriction allows a substantial reduction in the number of computational steps required for co-indexing. This is, then, a case where a linguistically-motivated restriction can also be justified on computational grounds.

We should point out that there are apparent exceptions to the C-command constraint. One is that if we substitute an expression that, intuitively at least, picks out a particular object in the world, then co-indexing can violate C-command. For instance, *the man* and *he* can be successfully co-indexed in the sentence below:

(4) The woman over there who likes *the man*$_i$ in the room thinks that *he*$_i$ is silly.

But just as intuition suggests, these examples do not seem to fall under the operator-bound variable paradigm as do quantifiers. Evidently, referential indexing does not abide by the same rules as quantifier-pronoun indexing. This has also been noted by Hornstein [1984], who points out that name-like indexing (i) violates C-command; (ii) operates across sentences in a discourse (e.g., *Pick a number*$_i$ *Divide it*$_i$ *by three*); and (iii) does not enter into scope ambiguities.

Putting these exceptions to one side for now, we will show that the C-command restriction is intimately related to a model of on-line semantic interpretation, drawn from the theory of programming language analysis. Let us consider how the Marcus parser would handle the sentence, *The woman thinks that he is silly.* At the point where *he* is in the input buffer, the phrase stack will hold the following elements: the top of the stack will hold an $\overline{S}$ phrase, with *that* attached as a Complementizer: the next phrase down in the stack will be the VP corresponding to the matrix S, with the Verb *thinks* attached; and the bottom phrase on the stack will be the matrix S phrase, with the NP *the woman* attached. Recall that these three phrases are in the stack *because they are not yet completely built* — the matrix S does not yet have its VP attached to it; the VP does not yet have its arguments attached to it; and the embeded $\overline{S}$ does not have its NP or VP attached to it. Let us call phrases that have not yet been built *open* phrases. If we use a context-free re-write notation, then we can define this notion formally. Given a rule $X \Rightarrow Y_1\ Y_2 \ldots Y_n$, then X is open iff some Y_i has not yet been attached to X. Intuitively, we have not yet "run off" the end of the rule that builds a complete "X". In contrast to the open $\overline{S}$, VP, and matrix S phrases then, any phrases attached to these open phrases *must* be complete. For instance, the NP *the woman* is a complete NP and is attached to the matrix S; the Verb *thinks* is a complete V and is also attached to its proper mother.

Each phrase in the active node stack is a maximal projection of a Head lexical (X^0) item, in the sense of $\overline{X}$ theory. Semantically, we can think of each X^0 item as an "operator" that must be supplied with its proper arguments — its "operands" — in order to be successfully interpreted. So, for example, a Verb might require NP, $\overline{S}$, or PP arguments; a Noun may require a PP or $\overline{S}$, and a Preposition, an NP

argument. On this view, there is a strong correspondence between syntax and semantics: just where a phrase is syntactically open, it is also semantically incomplete, in the sense that its operands have not yet been completely supplied. Once a phrase has been completely built, however, it is semantically interpreted — leaving aside the question of just what this comes to — and then attached as a single, opaque object to its proper dominating phrase. In the example above, once *the woman* has been analyzed as a complete NP, it is interpreted as such and then attached to the matrix S. Crucially, this means that the NP *now acts as a single, opaque unit*; we may imagine that the semantic interpretation process returns a single "value" (corresponding to the result of interpretation) as the object actually attached to the S. This natural construal of on-line semantic interpretation means that if a phrase is in the stack (and hence is open) then all of its daughters will be accessible since the mother phrase itself has not yet been interpreted and so has not yet been rendered opaque. In contrast, any sub-constituents of the daughter nodes will not be so accessible, at least not to *syntactic* analysis. (For example, we leave open the possibility that referential indexing operates according to an entirely different system, in line with our earlier observation.) But now note that this restriction is simply the core notion of C-command once again. Any NP underneath an NP that is in turn attached to an S will be invisible for operator binding. We see then that in addition to its functional import for parsing, C-command has a natural interpretation as a reflex of on-line semantic interpretation.

This approach to semantic interpretation is called "on-line" because we do not wait for the whole sentence to be analyzed before we attempt to interpret it. Rather, we interpret each completed argument and each completed operator-operand structure as it appears. This was the method adopted by Marcus [1980] and is in fact the *usual* method employed in the interpretation of programming languages (see, e.g., Aho and Ullman [1972]). Plainly, on-line interpretation is advantageous for real-time processing: why should one wait when partial semantic analysis can proceed? In addition, it lends itself naturally to a simple, modular multi-processing scheme. As each operand or operator-operand structure is completely built, the result is handed off to a separate semantic component, which then returns the result of interpretation. In the literature on programming languages two basic approaches are advanced as to how this inter-leaved processing might be done. In the first, we assemble the entire right-hand side of some expansion $X \Rightarrow Y_1$

. . . Y_n, and then output a semantic representation of the phrase X. This is a bottom-up method. The second is the dual of the first: we first predict that a phrase of type X will be found, then confirm this against the actual input. There are certain advantages to the second approach. First, one can determine what the required semantic routine will be before all the arguments are assembled — a boon for inter-leaved processing, if operations can be carried out in parallel. Second, the process is goal-directed, so one has expectations as to what the remaining arguments should be. If these expectations are not confirmed, then we know exactly where things have gone astray. For example, if it is known that a VP of a particular type should expand as a Verb plus two NP arguments, then when just one NP is encountered we know what the mistake is.

Interestingly, the Marcus parser incorporates both approaches so that it can exploit all of these advantages. It predicts maximal projections wherever possible, so that inter-leaved semantic processing can occur more easily. For example, if it sees the "leading edge" of a Noun Phrase, such as *the*, then it will create an NP phrase, even though this phrase has not yet been completely built. This allows one to retrieve, concurrently, any necessary semantic routines required to process NP's.

We see then that one can view C-command as a constraint on representations that has several computational implications. First, it aids in parsing, in the best case reducing the search for an antecedent by an exponential factor. Second, it fits comfortably into a model of on-line semantic processing that allows for concurrent semantic interpretation. Not surprisingly, linguistic representational constraints — "data structures" — impact on computations that use these representations — "algorithms". The investigation of the interaction between data structures and algorithms is one way that linguists can use their knowledge of formal structures to develop a more computationally-based linguistic theory.

THE INTERACTION OF PARSABILITY AND LEARNABILITY

In the previous sections we've seen how a grammatical constraint like C-command can be motivated from the standpoint of parsing efficiency. However, from another point of view the argument is a peculiar one. It assumes that the language faculty has been designed with parsing efficiency in mind — that this functional demand, above all others, was

more highly valued in the evolutionary design of language. While this outcome is certainly possible, it is by no means obvious. There are other "functional" demands on language, notably, the demand that natural languages be learnable. Indeed, since the inception of generative grammar learnability has been taken as criterial for the *naturalness* of languages: a natural language is precisely one that is learnable given the usual environmental conditions available to the child.

Given this setting, there is no *a priori* reason why efficient parsability should be first among the functional demands shaping language. In fact, there is no reason to believe that the demands of learnability and parsability are even compatible. After all, learnability has to do with the scattering of possible grammars with respect to evidence available to the child — a property of a family of grammars. Parsability, on the other hand, is typically a property of a single grammar. This implies that a family of grammars could be easily learnable, but not easily parsable. Certainly this is so, at least in principle. One can imagine a finite family of grammars, $G = G_1, G_2, \ldots, G_n$, where each G_i generates a non-recursive language L_i. Suppose that all the sentences of L_1 begin with a characteristic terminal element, say *a*, whereas sentences of L_2 always start with *b*, L_3 sentences with *c*, and so forth. Plainly any target L_i (or G_i, the associated grammar) is easily identifiable from very simple data: just one positive example sentence from the target language suffices; if the example begins *d* . . . then we know that the grammar to guess is G_4. But since each L_i is not even recursive, there is *no* algorithmic parsing procedure for any language in the family, let alone an efficient one. So easy learnability and easy parsability cannot be coextensive, at least in this sense. Examples like these place direct appeals to the sole functional demand of parsability on treacherous ground.

It looks then as though the relationship between learnability and parsability cannot be settled by some simple *a priori* argument, without examining in detail just what "learnable" and "parsable" languages look like, under more realistic assumptions. Suppose, for example, that the class of easily learnable languages (or their associated grammars) turned out to define just languages that were efficiently parsable with respect to those grammars. In other words, suppose that the class of natural languages was smaller than the class of efficiently parsable languages, under a reasonable definition of efficient. In this case the functional demand of parsability would not contribute much to an

account of why natural languages are the way they are. (Of course, there still could be a methodological or practical advantage to studying efficient parsability — it could be the right way to build algorithms, or the right way to think about certain problems). Conversely, if parsability imposed constraints above and beyond those demanded by linguistic theory, then we could use parsability as part of our explanatory repertoire — given that natural languages (or grammars) were observed to adhere to those extra demands. Of course, it might turn out that learnability and parsability are simply incompatible: in some respects, natural languages would be easily learnable, but some of these properties would cause difficulties for parsing, and vice-versa. This outcome would not be unusual for a biological system subject to multiple functional demands; indeed, such an outcome is more often than not the usual one. Finally, it could be that just those constraints that make languages learnable make languages parsable and vice versa. In some ways this would be the most striking — hence the most interesting — outcome.

As we have said, there is no way to settle this matter without studying specific theories of language learning and parsing. So, in order to be concrete, in the rest of this paper we will pair a detailed theory for the learning of a transformational grammar with a detailed theory for the parsing of a transformational grammar. This is not to say that we regard these models as the last word on learning and parsing: rather, that comparisons demand worked out models, and these two are among the most detailed that have been advanced. The parsing theory we have in mind is roughly the one we've deployed earlier: a variant of the Marcus parser. The learning theory is the Degree 2 theory of Wexler and Culicover [1980]. The Marcus parser defines a class of languages (and associated grammars) that are easily parsable; Degree 2 theory, a class of languages (and associated grammars) that is easily learnable. What is the relationship between these two classes?

To begin our comparison, we must sketch just what class of "easily learnable" languages Degree 2 theory defines. The aim of the theory is to define constraints such that a family of transformational grammars will be learnable from "small" data; the learning procedure can get positive (grammatical) example sentences of depth of embedding of two or less (sentences up to two embedded sentences, but no more). The key property of the transformational family that establishes learnability is dubbed *Bounded Degree of Error* (BDE). Roughly and intuitively, BDE is a property related to the "separability" of languages and

grammars given simple data: if there is a way for the learner to tell that a currently hypothesized language (and grammar) is incorrect, then there must be some simple sentence that reveals this — all languages in the family must be separable by simple sentences.

To move from this intuitive account will require a bit more detail about the envisioned learning procedure. The way that the learner can tell that a currently hypothesized grammar is wrong given some sample sentence is by trying to see whether the current grammar can map from a deep structure for the sentence to the observed sample sentence. That is, we imagine the learner being fed with a series of base (deep structure)-surface sentence (denoted "b, s") pairs. (See Wexler 1982 for details and justification of this approach, as well as a weakening of the requirement that base structures be available; see Berwick (1980, 1982) for an independently developed computational version.) If the learner's current transformational component, T_1, can map from *b* to *s*, then all is well; if not, and T_1 (b) = s^* does not equal *s*, then a *detectable error* has been uncovered.

With this background under our belts, we can provide a precise definition of the BDE property:

> A family of transformationally-generated languages I possesses the BDE property iff for any base grammar B (for languages in L) there exists a finite integer U, such that for any possible adult transformational component A and learner component C, if A and C disagree on any phrase-marker *b* generated by B, then they disagree on some phrase-marker *b* generated by B, with *b* of degree at most U.
>
> Wexler and Culicover 1980 page 108.

If we substitute "2" for *U* in the theorem, we get the Degree 2 constraint. Note that the BDE property is defined not just with respect to possible adult target languages, but also with respect to the distribution of the learner's possible guesses. So for example, even if there were just ten target transformational languages (grammars), the BDE property must hold with respect to those languages and any intervening learner languages (grammars). It is this joint relationship between targets and guesses that subsumes Chomsky's picturesque "scattering of grammars with respect to data" phrase — the learner's guesses are part of the scattering process.

BDE, then, is our criterial property for "easy" learnability. Just those families of transformational grammars that possess the BDE property (with respect to a learner's guesses) are learnable.

What about parsability? What criterial property could we embrace as representative of "easy" parsability? As a working hypothesis, we will adopt the parsing model described in the previous section. That is, we will assume a language is easily parsable iff it can be successfully analyzed using the machine we have described. But what kind of parser does this model define? As it turns out, there is a ready-made definition of the parsing model we have suggested, one where parsing decisions are made by looking at a local stack and buffer context: such machines are called *Bounded context parsers* (BCP). Naturally enough, a language (grammar) that is parsable using a BCP machine is called *BCP parsable*. While our aim here is not to give a formal account of BCP machines and language theory (see Szymanski and Williams 1976), we should point out that our parser is indeed BCP in at least one crucial respect: local parsing decisions are made by examining strictly *literal* contexts around the current locus of parsing contexts.[1] For example, suppose the machine is parsing the sentence, *John was kissed by Mary*. At the point in the parse where the machine is looking at the token immediately after *kissed*, it will have already assembled *John* into a Subject NP and attached it to the S, as well as created a VP and attached *was kissed* to it as the Main Verb. The input buffer will contain the tokens *by Mary* and the final punctuation mark. Importantly, its next move will be to drop a trace into the input buffer signaling its recognition that *kissed* demands an Object and none is present in the input. That decision will be made by consulting the local environment of the parse — the S and VP nodes, with the attached Verb, and the three input buffer items. Further, these items are recorded exactly as written by the linguist — as the nodes S, VP, V, and so forth. No additional "coding" is carried out. It is this literal use of the parse tree context that distinguishes bounded context parsing from other, more general methods of this sort.[2]

The BCP model was central to our explanation of c-command. Remember that our account hinged on the notion of *constituent completeness*: the "interior" of any phrase attached to a node on the active node stack was assumed to be opaque to further access. Not only does this define the C-command predicate in a natural way, it effectively limits the amount of left-hand parsing context that is available. If the parser's rule vocabulary can refer to just the unalloyed constituent names provided by the $\overline{X}$ theory — V, $\overline{V}$, VP, N, NP, and so forth — then this is in fact a necessary requirement for such a parser to work. Otherwise, the parser might have to refer to an unbounded string of

left-hand context symbols in order to make its correct move (since the left-hand portion of the string already seen at some point can be arbitrary in size). The BCP constraint rules out this possibility by fiat.

The BCP class also makes sense as a proxy for "efficiently parsable" because all its members are analyzable in time linear in the length of their input sentences, at least if the associated grammars are context-free. If the grammars are not context-free, then BCP members are parsable in at worst quadratic (*n* squared) time. (See Szymanski and Williams 1976 for proofs of some of these results.) The reliance on a literal encoding of local context, via the same non-terminal symbols that the grammar uses, also makes sense. (See Berwick and Weinberg 1984 for an argument that uses this property to show that "locality conditions" like Subjacency must exist.) Finally, deterministic operation is a reasonable (though strong) assumption. So for now, let us take the BCP property as our representative of easy parsability.

We can now at least formalize our problem of comparing learnability and parsability. The question now becomes: What is the relationship between the BDE property and the BCP property? One approach would be to show that the BCP property implies the BDE property, in the sense that if one has a collection of transformational languages defined according to Wexler and Culicover's model, and if each of those languages (along with possible guesses) meets the BCP property, then that collection will also meet the BDE condition. That is, the family will automatically fulfill all the constraints required to meet the BDE, and hence be learnable.

Such a result would be intriguing from a number of standpoints. For one thing, it says that once we have efficient parsability, then learnability follows. For another, since the BCP property has been studied from a formal point of view, it might be easier to determine whether a class was learnable by first looking at its parsability properties. Finally, it gives a clue to what class of languages is learnable via Degree 2 theory (though it gives only sufficient conditions). For example, since $a^n b^n c^n$ is BCP (in an extended sense), it should be in the class of Degree 2 learnable languages — which it is. (See Wexler 1982) More carefully, we would want to show that the class of languages which have the BCP property is Degree-2 learnable. The actual demonstration of this result would seem to require some care, and is a topic for future research. However, we can sketch the logic of how such a demonstration would go. This will indicate some of the problems involved in studying the parsing-learnability connection.

Intuitively, the BCP and BDE properties are related in the following way. The BCP property implies that parsing decision effects must be "localized" in the sense that any potential errors (parsing mistakes) are detectable within a small, finite radius of each parsing decision. Analogously, the BDE property amounts to the constraint that any transformational effect be localized. The two properties differ in that they refer to derivations in opposite directions: BCP refers to a locality effect during parsing, going in the direction from surface sentences to deep structures, and BDE refers to the derivation of a surface sentence from its deep structure. So one must be careful in relating the two. One way to demonstrate that BCP implies BDE would be to show that if BDE fails to hold, then BCP fails to hold. Taking the contrapositive, this will show that BCP implies BDE.

Suppose then that BDE fails to hold for a Wexler and Culicover family of transformational languages (and their associated grammars). Our job then becomes to show these languages are not BCP. If a family of languages doesn't meet the BDE condition, then there must exist a detectable error but no detectable error on small data. That is, there must be a target language L_A (the adult language) such that for some sentence s in L_A, with associated base structure b, and some learner guessed transformational component C, A(b) = s and C(b) = s′ not equal to s, but no b′ of bounded degree such that A(b′) does not equal C(b′). What we want to show is that this implies that the languages in question are not BCP.

The way that we might do this is as follows. We will establish that any detectable error (in the Wexler and Culicover sense, a "learning" error) is also a detectable parsing error, and vice versa. The failure of the BDE property will then show that there must be an unbounded (detectable) parsing error for some pair of languages drawn from the family of languages in question — that is, some sentence that will be parsed (incorrectly) as belonging to the grammar for L_C rather than L_A, given only bounded context parsability. Thus BCP will fail to hold.

Now, to establish this last point will take some formalization of just what is meant by a parsing error. We will sketch only part of how the argument would go. To prove that any detectable (W and C) error is also a detectable parsing error, suppose we have such a detectable learning error. By definition, this means that we have some surface sentence s with associated base structure b such that some (adult) sequence of transformations $T_1, T_2, \ldots, T_n$ applied to b yields, s, i.e., $T_nT_{n-1} \ldots T_1(b) \rightarrow s$. In shorthand, $T_A(b) = s$. By definition of

detectable error, we have that there is some sequence of (child) transformations applied to *b* that yields a different surface sentence, *s'*. Now we apply some additional assumptions of the Wexler and Culicover system. In their model, all transformations apply obligatorily and deterministically. That is, at any step in a transformational derivation, at most one transformation can apply. (As Wexler and Culicover note, this implies that there will be at most one surface sentence for every distinct deep structure.) Now, transformations are invertible, but perhaps not uniquely so. Let us denote an inverse transformational sequence (in the adult) by $T^{-1}{}_A$. Since transformational inverses are not unique, $T^{-1}(s)$ could map to many base structures, but it is at least true that for some T^{-1}, $T^{-1}{}_A(s) = b$.

Now we define a detectable parsing error. Again we are given a surface string, base pair, (*b, s*). We will say that there is a detectable parsing error just in case the learner's parser component (the inverse transformational component) applied to the surface string does not yield the right base structure, i.e., $T^{-1}{}_C(s)$ does not equal *b*. There are two ways that this can occur. Either (1) there is no inverse possible at all, because $T^{-1}{}_C(s)$ is blocked at some point, or else (2) $T^{-1}{}_C(s)$ does not equal *b*, for all possible nonblocked invertible sequences.

Now suppose that what we want to show, that any detectable learning error implies a detectable parsing error, is false. This means we have for all (b, s) pairs that $T_A(b) = s \neq T_C(b)$. But since there is no detectable parsing error for *s*, it is not the case that all $T^{-1}{}_C(s) \neq b$. But this means that for some T^{-1}, $T^{-1}{}_C(s) = b$. Taking inverses, we have that $T_C(b) = s$ for some learning component transformational sequence. But this contradicts our original assumption, that there was a detectable learning error.

Establishing the rest of what we want to show seems more difficult at present, and we leave it for future work. In any case, what we want to illustrate here is not the final result but the method of study. We can assess the relative strengths of parsability and learnability in this case, but only because we have advanced specific models for each. These characterizations are still quite specific, being grounded in particular linguistic theories. The results are therefore quite unlike the formal learning theories of Gold (1967), or more recently, of Osherson, Stob, and Weinstein (1982) nor are they like the general results obtained from the analysis of the parsability of formal languages. Rather, they hold of a narrower class of languages that are already known to be

linguistically relevant. In this respect, what the results lose in terms of invariance over changes in linguistic theories, they gain in terms of specificity. This lack of invariance does, however, leave the door open for future work. It remains to be seen how to compare the learnability of the grammars posited in more recent theories of transformational grammar — namely, the "parameter setting" model advanced as the Government-Binding theory — and the parsability of the grammars of that theory. What is clear is that an analysis like the one we have just made awaits a crisp, formal characterization of learnability and parsability that is particular to the Government-Binding theory itself.

NOTES

[1] The BCP model must be extended to handle non context-free languages. This is done in Berwick 1982.

[2] In fact, it was by weakening the literal coding demanded by BCP methods that Knuth 1965 was led to a more general method of left-to-right deterministic parsing, LR(k) parsing.

REFERENCES

Aho, A., and J. Ullman: 1972, *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Englewood Cliffs.

Berwick, R.: 1980, 'Computational analogs of constraints on grammars', *Proceedings of the 18th Annual Meeting of the Association for Computational Linguisitics*, Philadelphia, pp. 49–54.

Berwick, R.: 1982, *Locality Principles and the Acquisition of Syntactic Knowledge*, unpublished doctoral dissertation, MIT, Cambridge.

Berwick, R., and A. Weinberg: 1984, *The Grammatical Basis of Linguistic Performance*, MIT Press, Cambridge.

Chomsky, N.: 1951, *Morphophonemics of Modern Hebrew*, unpublished masters thesis, University of Pennsylvania, Philadelphia.

Chomsky, N.: 1981, *Lectures on Government and Binding*, Foris Publications, Dordrecht.

Gold, E.: 1967, 'Language identification in the limit', *Information and Control* **10**, 447–474.

Higginbotham, J.: 1980, 'Pronouns and bound variables', *Linguistic Inquiry* **11**, 679–708.

Hornstein, N.: 1984, *Logic as Grammar*, MIT Press, Cambridge.

Knuth, D.: 1965, 'On the translation of languages from left to right', *Information and Control* **8**, 607–639.

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

Szymanski, T., and J. Williams: 1976, 'Noncanonical extensions of bottom-up parsing techniques', *SIAM Journal on Computing* **5**, 231–250.

Wexler, K.: 1982 , 'Some issues in the theory of learnability', in C. Baker and J. McCarthy (eds.), *The Logical Problem of Language Acquisition*, MIT Press, Cambridge, pp. 30–63.

Wexler, K., and P. Culicover: 1980, *Formal Principles of Language Acquisition*, MIT Press, Cambridge.