May 1996

Complexity and the Induction of Tree Adjoining Grammars

Robin Clark
University of Pennsylvania

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-96-14.

Complexity and the Induction of Tree Adjoining Grammars

Abstract
In this paper, I will develop the formal foundations of a theory of complexity that underlies theory of grammatical induction. The initial concern will be the learning theoretic foundations of linguistic locality. That is, I will develop a theory that will place bounds on the amount a learner can draw from an input text. These bounds will limit the amount of variation that could potentially be encoded within a parameter space. A fully developed form of the theory will place a tangible upper limit on what the learner can induce from the input text. The formal theory developed establishes a relationship between the complexity of descriptions and their likelihood; that is, the more complex a structure is, the less likely it is to occur. I will use this result to develop a theory of linguistic complexity. I will rely on this relationship to show that the results developed in the first part of the paper for the parameter setting model also hold for the inductive theory. The final sections of the paper turn to the formal specification of the learning model and a description of the linguistic theory that supports it. This section also describes a pair of heuristic constraints on the learner's search for viable hypotheses. In general, the learner faces a computationally intractable problem in that there are exponentially many grammatical hypotheses for any input text. These constraints, the Adjunction Constraint and the Substitution Constraint, greatly reduce the number of hypotheses that the learner must consider. Furthermore, metrics on the complexity of the learner's descriptions guarantee that the hypothesis space can be tractably searched for the adult grammar.

Comments
University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-96-14.

Institute for Research in Cognitive Science

Complexity and the Induction of Tree Adjoining Grammars

Robin Clark

University of Pennsylvania
3401 Walnut Street, Suite 400C
Philadelphia, PA 19104-6228

May 1996

Site of the NSF Science and Technology Center for Research in Cognitive Science

IRCS Report 96--14

Complexity and the Induction of Tree Adjoining Grammars

Robin Clark
Department of Linguistics
University of Pennsylvania
Philadelphia, PA 19104
rclark@babel.ling.upenn.edu

In this paper, I will develop the formal foundations of a theory of complexity that underlies theory of grammatical induction. The initial concern will be the learning theoretic foundations of linguistic locality. That is, I will develop a theory that will place a bounds on the amount a learner can draw from an input text. These bounds will limit the amount of variation that could potentially be encoded within a parameter space. A fully developed form of the theory will place a tangible upper limit on what the learner can induce from the input text.

I will first turn to a general discussion of locality constraints and linguistic evidence. All theories of learning must contain a theory of how the learner processes and uses the linguistic evidence to form hypotheses of the adult target. I will consider one famous model of learning, that of Wexler & Culicover (1980) and show how their model established a systematic relationship between the input evidence available to the learner, locality constraints on transformational rules and the hypotheses that the learner framed in response to the text. Using their results as a model I will turn to parameter setting and develop several alternative models; as will become evident in what follows, all these models require a theory of linguistic evidence if they are to be serviceable. Our ultimate goal, however, will be to develop a theory that will supplant the standard approach to parameter setting and replace it with a form of induction. To this end, I will summarize the formal theory, itself, beginning in section 2. Crucially, the theory establishes a relationship between the complexity of descriptions and their likelihood; that is, the more complex a structure is, the less likely it is to occur. I will use this result to develop a theory of linguistic complexity. I will rely on this relationship to show that the results developed in the first part of the paper for the parameter setting model also hold for the inductive theory.

In section 3 I turn to an application of the formal theory described in section 2 to language learning. Section 4 turns to the formal specification of the learning model

and a description of the linguistic theory that supports it. This section also describes a pair of heuristic constraints on the learner's search for viable hypotheses. In general, the learner faces a computationally intractable problem in that there are exponentially many grammatical hypotheses for any input text. These constraints, the Adjunction Constraint and the Substitution Constraint, greatly reduce the number of hypotheses that the learner must consider. Furthermore, metrics on the complexity of the learner's descriptions guarantee that the hypothesis space can be tractably searched for the adult grammar.

1 Input Simplicity

One of the most compelling arguments for nativism has been based on the poverty of the stimulus. In its simplest form, the thesis of poverty of the stimulus simply notes that the evidence available to the learner massively underdetermines the knowledge state that the learner ultimately achieves. In general, the learner receives simple grammatical input from the environment. For example, the following utterances were addressed to Adam by various adults in his environment:[1]

(1) it's a movie camera.
no, it takes movies, then you show them later on.
we had a halloween party.
you want to put that on the floor?
what does he have?
fishing rod?
would you ask Cromer if he would like some coffee?
the man on the radio.
over here, dear.
the tape recorder.
the sun! The sun isn't shining in that window.
you simply don't want what?
those are carrots.
that's cats?
red fish?
I been chug. I've been chugging, I guess.
you woke up at fourteen o'clock. I'd like to know what time fourteen o'clock is.

The above examples reveal little of significant deviance or complexity. For the most part, these utterances consist of simple sentences without much embedding, although there are

[1] I've drawn these from two files on Child Language Data Exchange, one early file (adam15) and one late file (adam40).

a few noun phrases, echo questions, repetitions and expansions. Indeed, as Gleitman & Wanner (1982) observe, speech to young learners is "propositionally simple, limited in vocabulary, slowly and carefully enunciated, repetitive, deictic, and usually referring to the here and now" (page 15).[2]

Indeed, the input to the learner is so simple that one might well ask how something so complex as an adult grammar could be accurately acquired from it. This is a rather perverse reinterpretation of the old "motherese" debate of the 1970s. The hypothesis at the time was that adult caretakers fine-tuned their utterances to children in such a way as to aid the process of acquisition and reduce the role of an innate component (see Brown, 1977). Somehow, the learner would use motherese to induce the adult grammar, although proponents of the theory never explained how the learner would managed this. One problem, among many, is that the learner's final state is very rich, capable in principle of detecting and characterizing ambiguities, paraphrases, synonymy, antinomy and so on. If the input is so simple, how can the learner learn something so complex?[3] Supposing that, in general, speech addressed to children is simpler than speech between adults, how do children learn to speak like adults? An appropriate response to this question consists, we believe, of two intimately related parts:

(2) a. In its relevant respects, adult speech to children contains just enough information to allow them to accomplish the task of learning the adult grammar.

b. Children are able to carry out their task on such input because they are structured to do so; they know what to look for.

Thus, if anything, the relative simplicity of adult speech to children increases the need for some sort of innate learning device. In particular, those properties of the target grammar which must be learned from experience must be of sufficient simplicity as to be "witnessed" in the input text. If a grammatical theory is to have the learnability property, then, it must be able to guarantee a connection between those parts of the grammar which are to be learned and the evidence available to the learner, whether the learning is done by pure induction or by some method of parameter setting. This requirement puts very strong constraints on the grammatical theory, as I shall argue throughout this report.

1.1 Degree 2 Learnability

One of the most important applications of formal learning theory to linguistic theory is Wexler & Culicover's (1980) proof of degree 2 learnability. Although the proof concerned

[2]See the reference cited in Gleitman & Wanner (1982) for extensive discussion.

[3]See Wexler & Culicover (1980), particularly their chapter 2, for an excellent discussion of the hypothesis.

the learnability of a standard theory transformational component, it is worth considering the general form of their argument since much of the reasoning remains relevant both to a *P&P* framework and to an inductive model of the type proposed in here. I will therefore give a brief overview of their work.

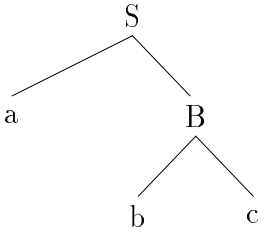
Wexler & Culicover assume that the learner is given a universal fixed set of phrase structure rules; these rules generate base phrase markers from which semantic interpretations are derived and which are input to the transformational component which maps them onto surface strings. The learner is presented with a text consisting of (b, s)-pairs, where b is a base phrase marker generated by the phrase structure rules and s is the surface string. The assumption that the learner is given a base structure may seem odd to some readers and Wexler & Culicover go to some lengths to justify it, for our purposes the issue is tangential as it plays no role in the main theory proposed below. Suffice it to say that the base structure corresponds to the semantic interpretation of the input sentence; I will suppose that the learner is able to construct this interpretation given available context, known word meanings and sensory input. Furthermore, the learner is able to do this accurately with sufficient frequency to exceed noise levels.

The learner, when presented with a (b, s)-pair, applies it's transformational component to b and tests whether the string dominated by the resulting phrase marker matches s. If it does, then the learner is content and moves on to the next (b, s)-pair in the text; crucially, if the learner's grammar has succeeded in matching s given b then it assumes that its current hypothesis is correct. It will only change its current hypothesis in response to a detectable error as defined in (3):

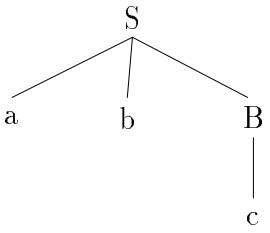
(3) *Error and Detectable Error*
If a transformational component C [the learner's transformational component — RC] maps a base phrase-marker P onto a surface structure that is different from the surface structure obtained when A [the target (adult) transformational component — RC] is applied to P, we say that C *makes an error* on P. If C makes an error on P, and if the surface *sentence* when C does the mapping is different from the surface sentence when A does, we say that C *makes a detectable error* on P. [Emphasis in the original text — RC]

The learner has made an error if the output of the learner's grammar differs from the output of the target adult system on some datum. Notice that the two systems might differ without an observer being able to detect the difference. For example, the two grammars might output the same string with different hierarchical structures as shown in (4):

(4) a. *Adult Structure*



b. *Child Structure*



The trees in (4a) and (4b) both cover the string *abc*, although they assign different hierarchical structures to it. We might imagine that the child's transformational component contains a rule which mistakenly raises b to make it a left sister of the node B. While this is an error, it is not a detectable error since nothing in the external world will alert the learner to its mistake. The definition of *detectable error* singles out errors on which the string (the surface sentence) generated by the learner's grammar does not correspond to the string generated by the target grammar.

Detectable errors were a crucial ingredient in the Wexler & Culicover proof. The learner only changes its hypothesis when evidence from the external world forces it to do so. If the learner's hypothesis is incorrect, then, it will only change its hypothesis if it makes an error on some input datum generated by the target grammar. The learner would never change its hypothesis without the motor of detectable errors to drive it; if its current hypothesis is successfully able to account for the input, the learner will not change its hypothesis to test some other alternative.

Suppose that the learner has made an error on some input datum d. The learner must, then, change its transformational component. It can do so in one of only two ways; it may either *reject* a rule currently in its transformational component or it may *hypothesize* a new rule to include in its transformational component. The *rejection* subroutine selects a rule which applied on the errorful derivation of d and purges it from the set of transformations. Notice that this rule may or may not be responsible for the learner's error; it, nevertheless, was implicated in an errorful derivation.

The *hypothesization* subroutine is slightly more complex. Recall that a standard theory transformation consists of two parts, a *structural description* which describes a set of phrase markers to which a particular *structural change* may apply. Let us assume that the learner has enumerated the set of possible structural descriptions. As we shall see

below, this set is finite, a fact which is crucial for the learnability proof to go through. Similarly, we will assume that the learner has an enumeration of the set of possible structural changes. If the learner selects the hypothesization subroutine, then it will find a structural description which matches the top level of the phrase marker b, where $d = (b, s)$. In standard theory terms, the learner is compelled to hypothesize a transformation which applies in the last cycle of the derivation of d. A flowchart for the learning procedure is given in figure 1.

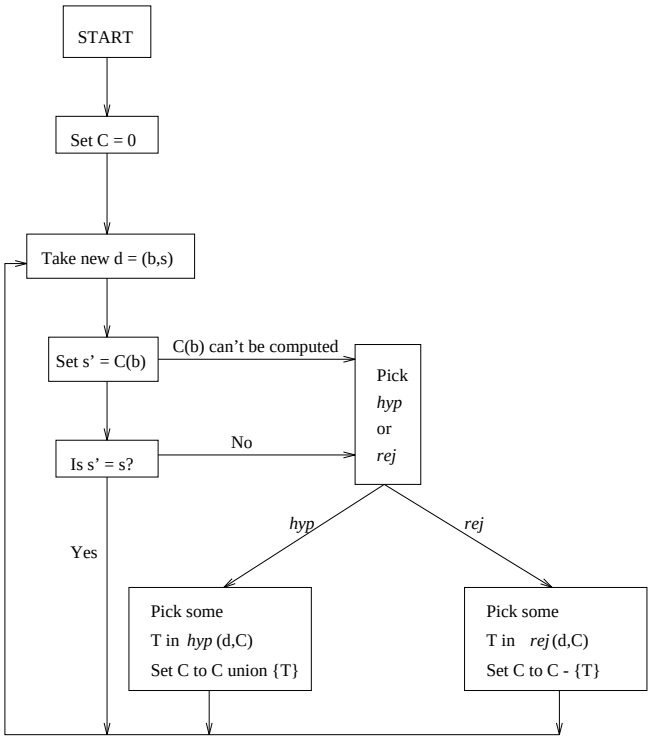


Figure 1: The Wexler & Culicover Learning Procedure

Notice that the learner's grammar and the target grammar might agree up to strings of a high-level of complexity. For example, the learner's grammar could agree with the target up to structures with 20 levels of embedding and then differ from it on structures with 21 levels of embedding or more. Since the learner would only change its hypothesis due to a detectable error, it will change its hypothesis only when it has made an error on one of these highly complex structures. Since these structures are very rare in realistic input texts, the learner is unlikely to encounter such an example. As the probability of encountering the crucial input decreases, the amount of time required for the learner to converge increases. In the worst case, the learner would never encounter the relevant examples, thus making the error that the learner has made effectively undetectable and the amount of time required to converge approaches infinity. In other words, the learner is effectively placed in the situation of being unable to converge since the time required

is so high.

The preceding argument can be made formally rigorous. To do this, let us take the case where the grammar includes rules applying only to structures of twenty levels of embedding. Recall that the learner enumerates the set of possible structural descriptions. For the case at hand, this set includes transformations which apply to trees consisting of a single clause (passive, for example), trees containing a single level of embedding (subject-to-subject raising) and so on up to twenty levels of embedding. The resulting set of transformations will be quite large, consisting of the cross-product of the set of possible structural descriptions (a function of the number of grammatical categories plus acceptable analyses of variables) with the set of possible structural changes (left-adjunction, right-adjunction, deletion and specified insertion). Assuming a uniform probability distribution over data presented to the learner, the odds that the learner will select the correct transformation on a string on which it has made a detectable error is extremely small, though finite. Given the enormity of the hypothesis space, it will take the learner a great deal of time to search it and converge on the adult target.

One way to solve this problem is to reduce the size of the hypothesis space that the learner must search. Reducing the number of possible transformations will result in a smaller search space and more rapid convergence. Notice that doing so should limit the amount of structure that can be mentioned in the structural descriptions of the transformations. Given that the structural change of a transformation is linked in a non-arbitrary way to the structural description, we should be able to guarantee that the complexity of the structures on which the learner makes a detectable error be strictly limited. That is, it must be guaranteed that if the learner makes an error, then it will make an error on a sufficiently simple example. This is the content of the *Boundedness of Minimal Degree of Error* (from Wexler & Culicover, 1980):

(5) *Boundedness of Minimal Degree of Error* (BDE)
For any base grammar B there exists a finite integer U, such that for any possible adult transformational component A and learner (child) 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.

Here *degree* refers to the number of S nodes embedded in the representation. Since simple examples dominate the input, the BDE increases the probability that the learner will make a detectable error if its hypothesis is incorrect. The increased probability of making an error decreases the amount of time that the learner must spend searching the hypothesis space before it converges. Furthermore, since simple structures would not be show the effects of rules of arbitrary complexity, there should be a link between the complexity of structures in the input text and the complexity of rules induced from that text. The learner is not only more likely to converge (the probability of convergence approaches 1

in the limit, as Wexler & Culicover demonstrate), but it is more likely to converge in less time. Furthermore, the learner will tend to select relatively simple transformational rules, leading to a parsimonious transformational component. Since real language acquisition is an automatic process which takes place over a relatively small time period, this property is a crucial one for a psychologically plausible theory of learning.

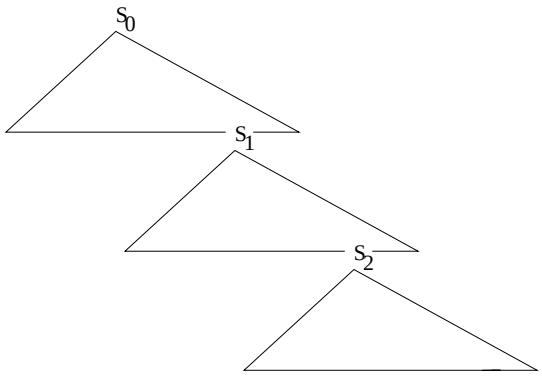
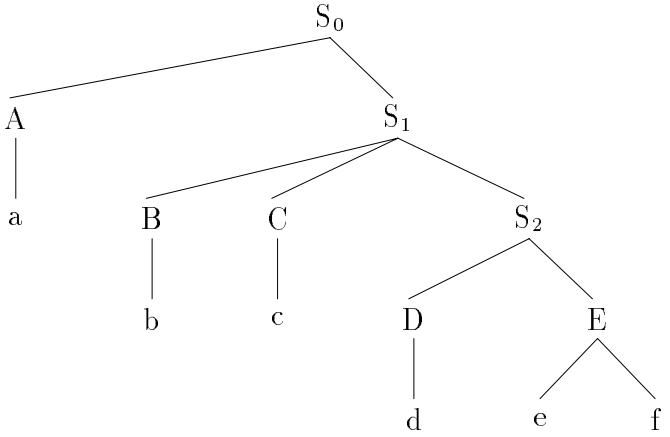


Figure 2: A Degree 2 Structure

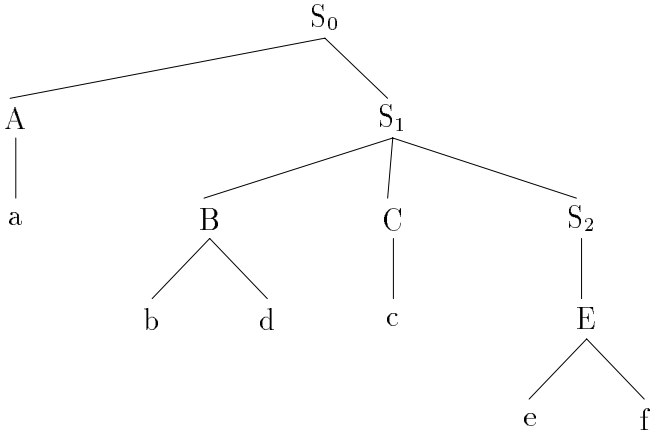
The smaller we can make the constant U in (5) the more the learner's task will be facilitated. Wexler & Culicover propose that U corresponds to phrase-markers of degree 2, as shown in figure 2. To understand how the proof worked, let us consider a concrete example. First, we will assume that rules apply in a strict cycle. That is, where S is a cyclic node, the most deeply embedded clause is the first domain of rule application, the next most deeply embedded S is the next domain and so forth. Let us take the case where the learner's transformational component raises the element d in the following tree from S_2 to S_1 in the following base structure:

(6) Base structure:



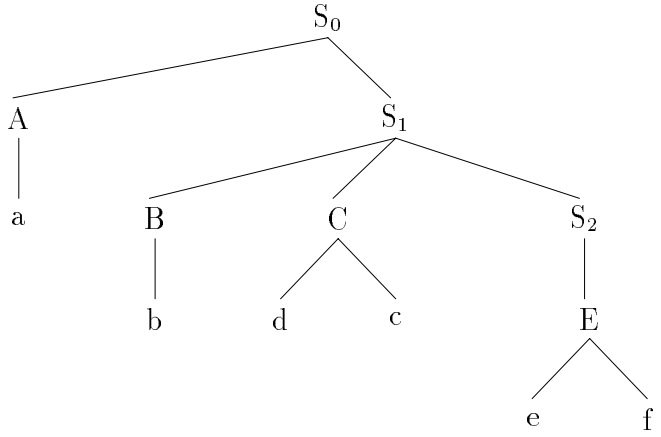
Suppose further that in the adult transformational component, d is adjoined as a right-daughter of B, as follows:

(7) Adult intermediate structure:



while in the child grammar, d is mistakenly adjoined as a left-daughter of C:

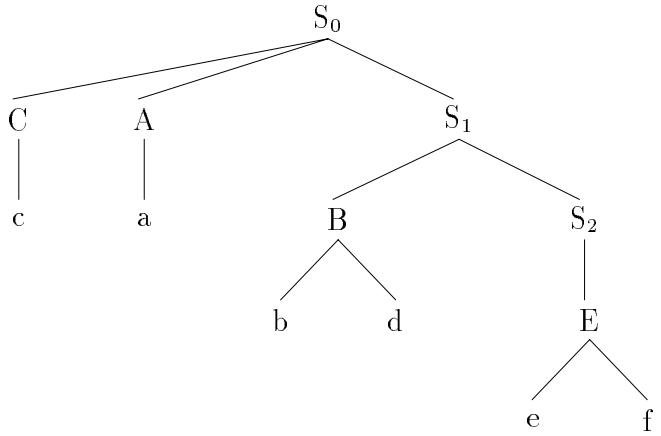
(8) Child intermediate structure:



Notice that both grammars generate the string *abdcef* although they assign different representations to the string. Thus, the error that the learner has made is not yet detectable. Intuitively, the domain in which a raising transformation would apply is the seed for an error which remains concealed from the learner since the output of the bad rule generates the correct string but not the correct hierarchical representation for the string.

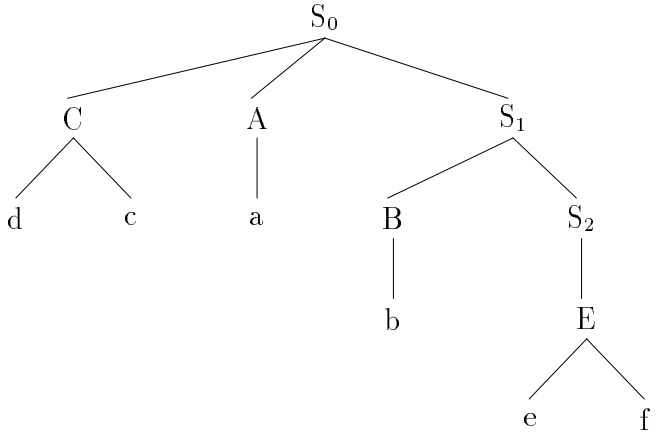
The BDE requires that the error reveal itself on a phrase-marker of the appropriate complexity; that is, the error must be made manifest on a degree 2 structure. Clearly, the error in (8) can be revealed on a degree 2 structure. Suppose that both the child and the adult transformational components contain a rule which raises the constituent C and makes it a left-daughter of S_0. The output of the adult transformational component will be as in (9):

(9) Adult output structure:



The output of the child's transformational component will be as in (10):

(10) Child output structure:



Notice that the adult's grammar has generated the string *cabdef* while the child's grammar has generated the string *dcabef*. Since the strings are not equal, the child's error has been revealed and the child must change its hypothesis.

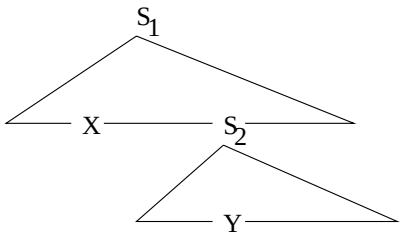
There is an important interplay between rule application and detectable errors in the above example. The movement rule applied within a restricted syntactic domain, the subtree dominated by S_1, in both the child grammar and the adult grammar. The child's error is revealed within the superordinate domain of rule application, the tree dominated by S_0. Wexler & Culicover present a number of constraints which serve to limit the application of grammatical processes to domains of complexity less than degree 2, that is, the constant specified by the constant U in the BDE. They argue that degree 2 trees are the smallest that can contain raising rules plus a domain of application which will reveal the learner's errors. Notice that degree 2 trees correspond, in an interesting way, to the domain of classical subjacency as defined in Chomsky (1973):

(11) *Subjacency*
No rule may relate X, Y in the structure:

$\ldots X \ldots [_\alpha \ldots [_\beta \ldots Y \ldots _\beta] \ldots _\alpha] \ldots X \ldots$

where α, $\beta \in \{S,NP\}$.

The definition in (11) restricts rule application to relating positions X and Y in trees of the following type, for example:



If we add one more cyclic domain to the above tree, we have a classic degree 2 structure of the same type as in figure 2 on page 8. Thus, a degree 2 tree was the least tree to contain a cyclic node dominating a domain for subjacency. If Wexler & Culicover were on the right track, then, notions like *cyclicity* and *subjacency* which were useful for syntactic analysis would have their ultimate grounding in a theory of learnability. In other words, an appropriately constrained and elaborated learning theory could potentially provide an explanation of why certain domains were relevant to syntactic operations.

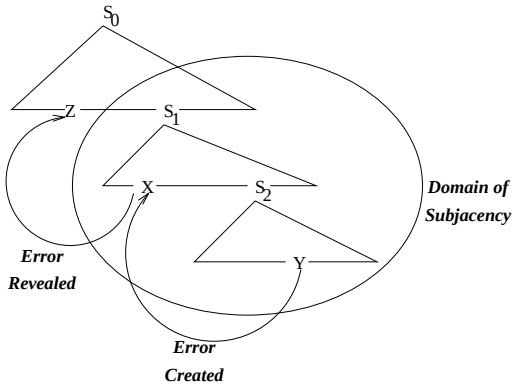


Figure 3: The Relationship between Subjacency and Degree 2 Structures

The most crucial aspect of the Wexler & Culicover model is the relationship between the complexity of structures that the learner must encounter to acquire the grammar and the probability of converging. Tightly constraining the form and domain of application of the grammatical rules means that there are fewer possible rules and less complex the structures for "witnessing" the effects of these rules. I take the notion of witnessing structure to be fundamental to language learning. In the present case, we can think of a witnessing structure to be one in the derivation of which a particular transformation (or other grammatical rule) applied. Notice that I do not require witnessing structures to be unequivocal; a single datum could witness a number of rules and, furthermore, it could be ambiguous. A learnable text, however, must be such that the learner is driven inexorably toward the target. Thus, a population of witnessing structures is unequivocal although any one structure might be indecisive. The BDE allows for an important reduction in grammatical complexity of both witnessing structures and possible linguistic rules. This, in turn, means that the learner can converge to the adult transformational component

given a sparser text. Finally, the fact that a degree 2 learner can use a smaller text than a degree 3 learner (for example) means that it can save computational resources and converge in less time (or using less memory) than the degree 3 learner. In other words, there is a strict relationship between the grammatical complexity of the input text, Universal Grammar and computational resources. It is the business of formal learning theory to elucidate this relationship.

1.2 Models of Parameter Setting

At this point, the reader no doubt feels a certain amount of impatience. The Wexler & Culicover proof held for Aspects style transformational grammars, after all, and *P&P* theories are, at least on the surface, radically different from the earlier framework. In this section I will develop arguments that *P&P* theories also must have bounds comparable to Wexler & Culicover's BDE; that is there must be a bound on the complexity of the texts from which they can be learned. In order to make the argument, we must show that it holds across various different interpretations of the *P&P* framework. I will therefore first turn to various interpretations of the notion of parameter that have been proposed in the recent literature. In one model, parameters are taken to be general properties, not connected to particular lexical items or functional categories. Another approach limits parameters to functional categories. Other approaches connect parameters to particular lexical items. Although these models are superficially quite different, they all presuppose some notion of boundedness of witnessing structures if they are to be viable models of learning.

Consider the theory in which parameters are not associated with particular lexical items or functional categories. In this case, we can assume that parameters are stated as general grammatical properties, as in (12):

(12) a. Heads precede their specifier. {yes, no}

b. Heads precede their complements. {yes, no}

On the face of it, this theory predicts that grammatical properties should remain uniform across categories. Thus, languages that are head-final should be OV, post-positional and so on. In other words, head-final languages which have prepositions (for example, German) should either not exist or be highly marked.

A theory which associates parameters with lexical items does not predict that all categories should behave in a uniform fashion with respect to parameter setting. Under this theory, individual lexical items can specify their values for parameter settings, although there may be preferred default values. It would be possible for most verbs in a language to be head-initial, for example, while a few might be head final. This is a situation which, to our knowledge, does not arise. Indeed, much of the attested syntactic variation can

be encoded on functional categories. Thus, we might speculate that properties like placement of the head in a VP are not properties of verbs but properties of the functional categories that co-occur with verbs. This theory predicts that lexical categories like verb should behave in a unified manner within a language while functional categories, like adpositions, could show some variation. One example is Dutch, which has both prepositions and postpositions.

For present purposes we can assume, without loss of generality, that all of these theories presuppose the same general learning procedure however much they may differ in the way that information is packaged. In particular, we can outline the procedure as follows:

(13) *Learning Procedure*

a. A single example, σ_i, is presented to the learner.

b. The learner attempts to analyze σ_i using its current hypothesis (or hypotheses) outputting the result to an output buffer **B**.

c. The learner analyzes the contents of **B** and updates its current hypothesis (or hypotheses).

The procedure in (13) is repeated indefinitely. The learner converges if, after finite time, it does not alter its hypothesis; for a *P&P* framework, a hypothesis will consist of a sequence of parameter settings. It converges on the target grammar if it has converged on a sequence of parameter settings and this sequence is identical to the target parameter settings (strong convergence) or if the learner's hypothesis specifies the same language (weak convergence), where *language* is taken to mean a set of strings.

There are different methods of fleshing out the procedure in (13). In an error-driven learner, for example, the learner does not change its hypothesis unless the buffer **B** contains a failed parse. Another type of learner might change its hypothesis even when **B** contains a well-formed representation; for example, it might search the hypothesis space which spans the input example with the smallest representation possible.

Equally, there are different methods for changing hypotheses. One method might select a parameter for resetting at random or select a parameter whose value was crucial to arriving at the representation in **B**. The algorithm of Gibson & Wexler (1994) is error-driven and greedy; that is, it selects a single parameter for resetting if and only if resetting the parameter allows it to parse the input example. This is a special case of deductive learning (see Clark, 1992), where the learner attempts to update its hypotheses on the basis of some rational choice. The learner of Kapur (1992) uses indirect negative evidence to accept or reject parameter settings. The learner defined in Brill & Kapur (1994) and Kapur & Clark (in press) use calculations of relative entropy to set parameters.

Finally, different algorithms will use computational resources in different ways. For example, the buffer **B** might have room for only one input datum or it might contain a sequence from the input text; in the latter case, the learner would have a memory.

Similarly, the learner might be allowed to consult and revise other data-structures; one example would be lookup tables that contain information about frequencies in the input text. The learner might equally be required to converge after a bounded amount of time; that is, it might be forced to conjecture its definitive hypothesis after it has seen a prespecified sequence of the input text or after a certain number of clock ticks.

Despite the differences in the above algorithms, they all assume that the learner is presented with the data sequentially (although it may have some memory for past examples) and that the learner has limited computational resources (memory, time or both). These assumptions are sufficient to demonstrate that all theories of parameter setting must develop a theory of locality analogous to degree 2 learnability. In particular, no theory can assume that the learner has access to arbitrarily complex input examples. In the sections that follow we will develop a precise mathematical theory of complexity. At this point it is worthwhile to consider a less formal argument, one that is conceptually related to the discussion in section 1.1.

Let us begin with the assumption that learners, in order to fix the value of a parameter, must be exposed to its effects on the input text. This informal statement, although plausible, is actually quite difficult to formalize. On the most general interpretation of the notion of parameter, the values that parameters may take on are not *informationally encapsulated*. Parameter settings would be informationally encapsulated if there were contexts where the setting did not interact with any other element of the grammar. These contexts would be "dead giveaways" in that they would tip the learner off as to the correct value for a given parameter. But this claim would be tantamount to saying that parameters are construction specific; such an approach would negate the theoretical value of adopting the *P&P* framework. Instead, parameter settings interact with grammatical principles, other parameter settings, lexical information and so on to generate a text. Let us consider a concrete example. Suppose that the learner has encountered a sentence which can be analyzed as having the order SVO; the examples in (14) show such examples drawn from two different languages, English and German:

(14) a. John saw Bill.

b. Peter kauft Brötchen
Peter buys bread
"Peter buys bread."

The sentence in (14a) is drawn from a language that is genuinely SVO; the parameter settings that specify the grammar of English result in structures that are head-initial. The sentence in (14b) is drawn from a language that is not SVO but is, rather, underlyingly SOV; the parameters settings that specify the grammar of German result in root V2 structures, creating the illusion that the language is SVO (until you know the rest of the grammar, of course). A learner encountering an apparent SVO structure for the first time

has no prior information about the correct parameter settings. Thus, as far as the learner knows, (14b) might be drawn from a genuine SVO language like English and (14a) might be drawn from a language that is not underlyingly SVO, but has V2 phenomena. Taken in isolation, the sentences in (14) are ambiguous as to the mechanisms which generated them.

With the above in mind, let us formulate the following (adapted from Clark, 1994):

(15) *Generalized Parameter Expression*
A string ω with representation τ expresses the value v_i of a parameter p in a grammar G just in case p must be set to v_i in order for G to represent ω with τ.

The definition in (15) relatives the expression of a parameter value to a single grammar, G. I will assume that the learner comes equipped with at least one hypothesis about the target grammar. When it encounters an input datum, in the form of a string, ω, it attempts to assign it a grammatical representation. If the learner succeeds in doing so using G, then it must be because some variable property (a parameter) in G was fixed to the correct value, v_i. This is what it means to express a parameter. In general, then, a set of parameter settings in a system of parameters $\mathcal{P}$ is learnable from a text just in case each setting in the set is expressed in that text; in other words:

(16) *Parameter Expressability*
For all parameters $\mathrm{x_i}$ in a system of parameters $\mathcal{P}$ and for each possible value $\mathrm{v_j}$ of $\mathrm{x_i}$, there must exist a datum $\mathrm{d_k}$ in the input text such that a syntactic analysis τ of $\mathrm{d_k}$ express $\mathrm{v_j}$.

Notice that nothing in the definition in (15) requires parameter expression to be unambiguous relative to a single input datum. Thus, the examples in (14) are completely ambiguous as to their parameter expression. Assuming a non-V2 language, they express pure SVO; if the grammar is V2, then they express other parameters settings. A datum is unambiguous given a fully specified grammar G and a representation τ. The text presented to a learner consists of a sequence a strings. Thus, each datum in the text can be ambiguous as to its parameter expression. The learner will only be able to set parameters gradually by considering a number of examples. The child exposed to German is driven to the hypothesis that the target is SOV on the basis of a number of different factors, including the presence of strings with OV order, as in:

(17) wir tun hier Bilder malen.
we aux here pictures painted
"We painted pictures here."

A grammar with parameters settings that work for an underlyingly SVO word order

will fail on (17) and similar examples while a grammar allowing for V2 structures with underlyingly SOV order will succeed. The learner will eventually be driven to the correct parameter settings even though any one input datum is ambiguous with respect to parameter expression.

The above scenario suggests, in turn, that there is a statistical component to parameter setting. In order to specify the value of a parameter, the learning must consider classes of data which express parameter values ambiguously, preferring to set parameters to values that are expressed most frequently. On this view, the learner processes each new datum, keeping track of the (ambiguous) parameter expression it finds as a result of its processing; those values which are most often expressed are most likely to be selected by the learner.[4] Crucially, the learner cannot set a parameter on the basis of a single exposure to a datum. We can simulate this effect by setting thresholds on parameter setting as in (18). Given a particular parameter $\mathrm{p_m}$ which is to be set to a particular value $\mathrm{v_n}$, there is some basic threshold frequency $\Phi_{(m,n)}$ that must be met in order to set $\mathrm{p_m}$ to $\mathrm{v_n}$. Letting $f_{(m,n)}(\mathrm{s_i})$ be the actual frequency perceived in the input text, we can formalize this intuition by:[5]

(18) *Frequency of Parameter Expression*
Given an input text σ_i, a target parameter sequence p_a and a learning system $\mathcal{L}$, $\lim_{T\to\infty} \overline{\mu}_0(\mathcal{L}, T) = 1$ if, for all parameter values v in positions m in the target p_a, $f_{(v,m)}(\sigma_i) \geq \Phi_{(v,m)}(p_a)$.

Here, I intend the threshold represented by $\Phi_{(v,m)}(\mathrm{p_a})$ to be the number of times the learner must encounter a construction which expresses the value v of parameter m in the input text in order to correctly set the parameter. Notice that parameters that are expressed in "simple" structures are likely to be expressed with fairly high frequency. Consider, for example, those parameters which express the relative order of a head and its complements. The minimal tree on which these parameters can be expressed is quite simple:

(19)

The subtrees in (19) are frequent in parsing the input text, so that the learner should quickly master the relevant parameters; these parameters will pass threshold relatively early. The minimal tree upon which specifier-head relations can be expressed is slightly more complex, as shown in (20):

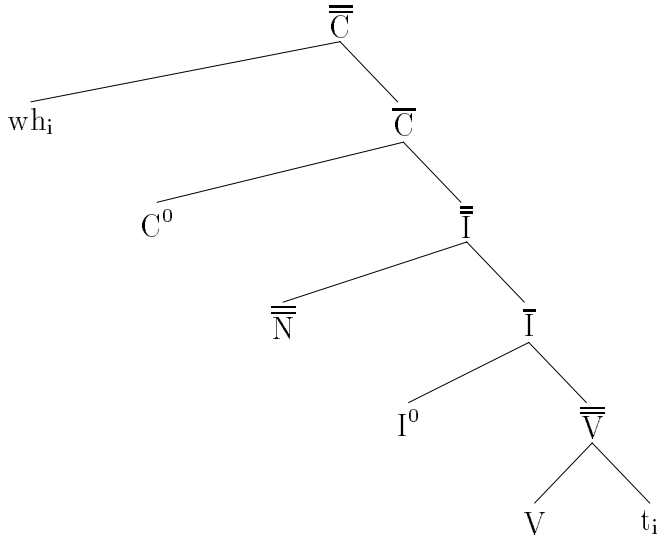
[4]See Clark (1992) or Clark & Roberts (1993) for discussion of a learning model with these properties.

[5]The limit, $\lim_{T\to\infty} \overline{\mu}_0(\mathcal{L}, T) = 1$, expresses convergence to the target sequence of parameter settings as time, T, goes to infinity, given a learning system $\mathcal{L}$ and a test for correctness $\overline{\mu}_0$. We can take $\overline{\mu}_0(\mathcal{L}, t)$ to be the number of parameters correctly set by $\mathcal{L}$ at time t.

(20)

The minimal tree which would exhibit non-string vacuous wh-movement is still more complex:

(21)



The intuition underlying both (16) and (18) is that parameters which are expressed by small, "simple" structures, like head-complement and specifier-head order in (19) and (20), will be expressed with high frequency, since these structures are likely to be embedded in larger structures or simply occur on their own. Thus, the learner is likely to set these parameters rapidly since it will have been exposed to the effects of the target parameter setting at a level which exceeds threshold fairly early on. Parameters which are expressed in more complex structures, like non-vacuous syntactic application of "Move α" as in (21) will be expressed less frequently since these are not as likely to be embedded within larger structures or occur on their own.[6] More complex parameters will achieve threshold frequency later than simple ones.

As an empirical hypothesis we might suppose that a parameter which can be expressed on a simple structure can, likewise, be formulated relative to such a simple structure; that is, we can use some boolean combination of a small number of simple theoretical predicates (like "government" or "Case marking") to formulate any property that is subject to cross-

[6] I assume here, as is standard in the acquisition literature, that the learner's input is made up, for the most part, of simple grammatical sentences.

linguistic variation. In other words:

(22) Complexity of linguistic expression is directly proportional to complexity of formulation.

Pretheoretically, there is no reason to suppose that (22) should be true. Indeed, one might expect it to be false since a number of predicates might be required to accurately specify the domain of the linguistic property. Consider, in this light, the difference between "Move α" and transformations in the standard theory (Chomsky, 1965). In the latter theory, the structural descriptions of transformations could be quite complicated; this was often done to restrict their domain of application. The change to "Move α" crucially assumed that the computational mechanisms of the ideal speaker/hearer were appropriately constrained so that locality conditions need not be mentioned in linguistic rules, the statement of which could then be simplified. In short, the ideal speaker/hearer was so constructed as to apply rules in a particular way, one which excluded violations of the domain constraints. Below, I will give formal justification for (22), but for the moment I will take it as an empirical hypothesis about the optimal theory of the human language component.

Intuitively, a system of parameters will be learnable just in case each possible parameter value in the system can be expressed on a structure that is simple enough that the learner is likely to encounter that structure with a frequency that is greater than $\Phi_{(m,n)}(p_a)$, the threshold frequency for setting parameter m to value n given the target sequence p_a. One way to capture this intuition is to stipulate the *Boundedness of Parameter Expression* (Clark, 1992):

(23) *Boundedness of Parameter Expression (BPE)*
For all parameter values v_{i} in a system of parameters $\mathcal{P}$, there exists a syntactic structure τ_{j} that express $\mathrm{v_i}$ where the complexity $\mathrm{C}(\tau_{\mathrm{j}})$ is less than or equal to some constant U.

The BPE is conceptually related to the BDE of Wexler & Culicover (1980; see (5) on page 7, above). Both the BPE and the BDE attempt to place an upper bound on the complexity of the data that the learner must see in order for the learner to converge on the target. If the constant U in (23) is sufficiently small, then each parameter value $\mathrm{v_i}$ in $\mathcal{P}$ is expressed on a structure that is simple enough that the learner is likely to encounter the relevant data with frequency greater than $\Phi_{(m,v_i)}(p_a)$ for the target p_a.

Notice, crucially, that as we place tighter time bounds on the learner, the length of the texts on which the learner is required to converge become shorter; we would expect that the constant U in (23) would likewise decrease. This reflects the idea that the simpler a structure on which a parameter value is expressed, the likelier the learner is to encounter that structure when parsing the input sequence. This, in turn, implies that the hypothesis that the learner is computationally bounded will interact with the theory of linguistic variation and typology. If parameter values must be expressed on extremely

compact structures, then the set of parameters we can incorporate in our theory of UG will be tightly constrained, thus placing a limit on the kind of linguistic variation we can observe in principle. If this is correct, then complexity limits on learning will translate to substantive constraints on linguistic theories.[7]

If the complexity of parameter expression has U as an upper bound, we could in principle construct a text where the representation of each datum is bounded by U:

(24) *Minimal Text*
Let σ_{min} be a set of sentences drawn from the language L_i such that, when parsed according to the grammar for L_i, all grammatically admissible trees τ_j of complexity $\text{C}(\tau_\text{j}) \leq U$ are exemplified once in σ_{min}; σ_{min} is a minimal text for L_i.

The idea is that a minimal text contains one example of each type of grammatical construction of complexity less than the constant U given by (23). Notice that a minimal text is finite, since arbitrary embeddings are ruled out by the complexity bound U. Given a minimal text, we can define a *fair text* in the following way:

(25) Let r be the threshold frequency, $\phi_{(v,m)}(\text{p}_\text{a})$, for all parameters, m, and values, v, in some system of parameters p_a and let σ_i be a minimal text for a language L_j. The text that results from concatenating σ_i to itself r times is a fair text for L_j.

In other words, a fair text can be constructed from a minimal text in such a way that each construction is repeated enough times to guarantee that the thresholds for parameter setting is exceeded. Thus, no information is withheld from the learner in a fair text. We can define the learnability property as follows:

(26) A system of parameters p_a is learnable if and only if there exists a learner φ such that for every language L_i determined by p_a and every fair text σ_j for L_i, $\varphi(\sigma_\text{j})$ converges to L_i.

So a system has the learnability property just in case there is some learner that learns the languages determined by that system from any arbitrarily selected fair text. The complexity bound U established for the constraint in (23) should serve to limit the complexity of the input text; in particular, given U we can establish an upper bound on both the sample size and the time required by the learner. This is so since U established a limit on the size of the minimal texts. As U grows, the minimal texts for each language will also grow. But the size of a fair text is just $|\sigma_{\text{min}}|^{r+1}$, so the fair texts will also grow.

[7] See Osherson & Weinstein, 1992, for a substantive discussion of this point. Their results are summarized in Clark (1994).

Assuming, as seems reasonable, that the time to converge is a function of the size of the text φ learns on, then the time-complexity of learning is also a function of U. Recall that U is a bound on parameter expression; no parameter can contain more information than can be expressed by a phrase marker of complexity at most U. Thus, U also limits the information that can be encoded by any one parameter. Finally, since cross-linguistic variation is determined by the different parameter values, U also limits the amount of variation that is possible across languages.

In the sections that follow, I will develop a general theory of the complexity of linguistic representations. This theory will formalize the intuitive argument given above; in particular, the simpler the structure on which a parameter is expressed, the more frequent that structure should be in the input text and the sooner the learner should converge to the correct setting for that parameter. The theory itself rests on the foundations of information theory, recursive function theory and statistics. I turn now to a brief exposition of the formal background of the theory.

2 The Complexity of Linguistic Descriptions

In the previous section, I established an informal, intuitive connection between the complexity of a linguistic representation and its likelihood. The simpler a representation is, the more likely it is to occur. One way to think about this connection is to consider the relationship between probability distributions and data compression. This might, at first, seem like an unlikely direction to follow in pursuit of a theory of the relationship between parameters and linguistic evidence. Arguably, though, grammars and theories in general can be seen as data compression devices. In particular, a grammar is a finite means of creating an infinite set (a language); in other words, a grammar is a very efficient means of compressing a language.

2.1 Randomness

Let us begin with a simple thought experiment. Suppose that you were at one end of a transmission channel (say a radio receiver). Every second or so, a broadcaster at the other end transmits a binary digit which you write down. You are required to guess whether or not the sequence of binary digits being transmitted over the channel is random or has some order to it. Naturally, if the sequence is ordered you will have to discover an ordering principle which generates the sequence; we might not require, though, that you discover the same ordering principle as the broadcaster used to generate the sequence. Notice the similarity between your position and the position of the learner in the Gold framework (see also the discussion of (13) on page 14, above). At each time step, you are given a datum and you make a hypothesis about the entire sequence you have seen to date by producing a theory of the sequence or the symbol "random". Notice that you may well have to keep a record of the entire sequence seen to date.

Consider the beginning of a sequence. The broadcaster sends a '0' followed by a '1'. At this point, it is impossible to detect whether or not the sequence is random. Suppose that the sequence, after one hundred turns is:

(27) 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1

Clearly, the sequence has a great deal of structure. One might reduce it to the instruction:

(28) Print the sequence '0 1' fifty times.

Suppose, next, that the above sequence continued with another '1' instead of the expect '0'. Clearly this will complicate the description of the sequence, although not by very much since we can modify the instruction in (28) to "Print the sequence '0 1' fifty times; print '1'." Again, the instruction is much shorter than the string, but it perfectly encodes the structure of the string.

The instruction in (28)—as well as its modified version—can do so because it can exploit structure in the sequence in (27). The subsequence '0 1' happens too often for the string to be random. Clearly, (28) exploits this property in reducing the sequence to a print instruction. In general, we might suppose that, in a random sequence, if the probability of a '1' is p and the probability of a '0' is $1 - p$ then '1' should appear p percent of the time and '0' should appear $1 - p$ percent of the time. Notice that this is true of the sequence in (27), assuming that '1' and '0' have the same probability, $p = 0.5$. Suppose that we generalize the test to include subsequences. That is, if '1' appears with probability p and '0' appears with probability $q = 1 - p$, then the sequence '0 1' should appear qp percent of the time, '1 0' should appear pq percent of the time, '1 1' should appear p^2 percent of the time and '0 0' should appear q^2 percent of the time. Clearly, the sequence in (27) fails this test for randomness since the sequences '1 1' and '0 0' never occur. We can test the string for the relative frequency of increasingly long sequences using basically the above argument; if the string is truly random then these sequences should occur with the expected frequency. We can call these expected frequencies *laws of randomness*.

If we generalize the above approach to randomness, we can imagine a number of other "laws of randomness". For example, an infinite sequence of 0s and 1s should contain infinitely many 1s if it is truly random. We can effectively test whether an infinite sequence is random if increasingly long finite initial segments obey the laws of randomness. Notice, though, that some strings appear to obey the laws of randomness even though there is a very simple method of describing the string.[8] Consider the string in (29):

[8]See the discussion of Mises-Wald-Church collectives and Martin-Löf random sequences in Li & Vitanyi (1993).

(29) 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0
1 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0

The above string passes many of the tests for randomness. In fact, though, the string in (29) is the binary expansion of $\sqrt{2}-1$ (see Cover & Thomas, 1991). Thus, returning to our thought experiment, you could at some point during the transmission guess "the string I am receiving is the binary expansion of $\sqrt{2}-1$" and successfully predict the next binary number in the sequence. Although the sequence appears random at first glance, there is, in fact, a simple description which will generate the sequence. The example raises the interesting problem of how to decide when a given string is random; in particular, effective tests for randomness (proportion of sequences like "00", "10", "01" and "11" in the string, and so forth) are not guaranteed to give the correct answer. Thus, the randomness of a string may not be decidable (see Li & Vitanyi, 1993 for some discussion) which brings up the interesting relationship between Kolmogorov complexity and Gödel's incompleteness theorem (Chaitin, 1975; 1987). These considerations suggest that randomness can be defined in the following way:

(30) A string (or any other object) is *random* if it is the same size as its best description.

By *best description* I mean some method whereby the structure of the object can be recoverd—a recipe for building a copy of the object, if you will. Another way of putting it is that a sequence is random if no program which computes an initial segment of the sequence is shorter than the initial segment itself. Intuitively, a nonrandom object shows regularities that can be exploited by an optimal description; thus, the description will be smaller than the object. If the object is random, though, then knowing about part of the object tells you nothing about any other part. There are simply no exploitable regularities so a program for computing the object will have to specify each part separately and, hence, will be at least as big as the object itself. In other words, the sequence cannot be compressed:

(31) A random sequence is incompressable.

Linguistic representations have a great deal of structure which could be exploited by such a data compression device. For example, if all phrases have heads and properties of heads determine the internal structure of phrases, then a program which includes knowledge of headedness ($\overline{\text{X}}$-theory) and argument structure (the θ-Criterion and the Projection Principle) could produce compact descriptions of tree structures since it could automatically replace much of the structure by appeal to these modules of grammar. These concerns about compactness of descriptions suggest that we should briefly consider the mathematics of data compression.

2.2 Data Compression

Data compression involves assigning a short description to a source object. The description should be such that the object can be retrieved in a finite number of steps; in other words, the description cannot lose information about the object. On the other hand, the best description of an object will be one that exploits all the predictable structure available and, so, is significantly smaller than the object it is trying to describe.[9]

We can view data compression as a coding problem. That is, given that a random variable X can take its value in a set $\mathcal{X}$, we can write an encoding function C from $\mathcal{X} \rightarrow \mathcal{D}^*$, where $\mathcal{D}^*$ is the set of strings of finite length on an alphabet $\mathcal{D}$. For example, $\mathcal{X}$ could be the suites of a deck of cards, so $x \in \{\text{Clubs, Diamonds, Hearts, Spades}\}$, $\mathcal{D}$ could be a binary alphabet and:

(32) $C(\text{Clubs}) = 00$ $\quad$ $C(\text{Hearts}) = 10$
$C(\text{Diamonds}) = 01$ $\quad$ $C(\text{Spades}) = 11$

The suite of a card drawn at random could be encoded, then, as a sequence of binary numbers of length 2. Notice, however, that the chance of drawing a card of any one suite is the same as any other, 1 in 4 assuming a fair deck. In general,codewords of varying length are desirable if not all values of X are equiprobable. As a general principle of organization, we might want to reserve short codewords for frequent items and allow infrequent items to be associated with longer codewords. The following quantity is of some interest, then:

(33) The *expected length* $L(C)$ of a source code C for a random variable X and a probability distribution $p(x)$is given by:

$$L(C) = \sum_{x \in \mathcal{X}} l(x)p(x)$$

where $l(x)$ is the length of the codeword for x.

The most efficient code would be one where the expected length is lowest. Anticipating somewhat, we will be interested in taking the source code to be descriptions of the object x (a syntactic representation, for example). The best descriptions (the equivalent to codes in the above sense) will be those that have the lowest expected length.

There are a number of different types of codes. I will briefly review some of the different kinds here, as they will play a role in understanding later results. We turn first to *non-singular* codes:

[9] In this section, I will rely on the presentation of data compression to be found in Cover & Thomas (1991). See, in particular, their chapter 5.

(34) A code is *non-singular* if every element of the range of the random variable X maps to a different string in $\mathcal{D}^*$:

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j).$$

The coding relation in (34) is a true function since each value that X takes on is uniquely related to a distinct codeword. This ensures decodability but may require adding punctuation between codewords if we wish to concatenate codewords reporting a sequence of outcomes of X, as in the case where X takes on letters of the alphabet as its value and we wish to transmit an encoded English text.

If we are encoding sequences of values of X, we will need to define the *extension* of a code as follows:

(35) An *extension*, C^*, of a code C is a mapping from finite length strings over $\mathcal{X}$ to finite length strings over $\mathcal{D}$, defined by:

$$C(x_1x_2\ldots x_n) = C(x_1)C(x_2)\ldots C(x_n)$$

where $C(x_1)C(x_2)\ldots C(x_n)$ is the concatenation of the code words for $x_1x_2\ldots x_n$.

Recall, for example, the code in (32) for encoding the suites of cards drawn randomly from a fair deck. Suppose we wish to encode the drawing of a club followed by a heart. The $C(\text{Clubs Hearts}) = C(\text{Clubs})C(\text{Hearts}) = 0010$. We can now define a *uniquely decodable* code:

(36) A code is *uniquely decodable* if its extension is non-singular.

If a code is uniquely decodable, then any code string has only one possible source string associated with it; strings of codewords are unambiguous. Notice, however, that finding the source string associated with a code string may require looking at the entire code string. If this is so, then it may be quite slow to decode an entire encoded sequence. A *prefix code* or *instantaneous code* allows for instant decoding without reference to future elements of the encoded string. Such a code can be defined as follows:

(37) A code is a prefix or instantaneous code if no codeword is a prefix of any other codeword.

A prefix code can easily be decoded without reference to possible continuations of the codeword precisely because the end of the codeword can be immediately detected; it is a "self-punctuating" code.

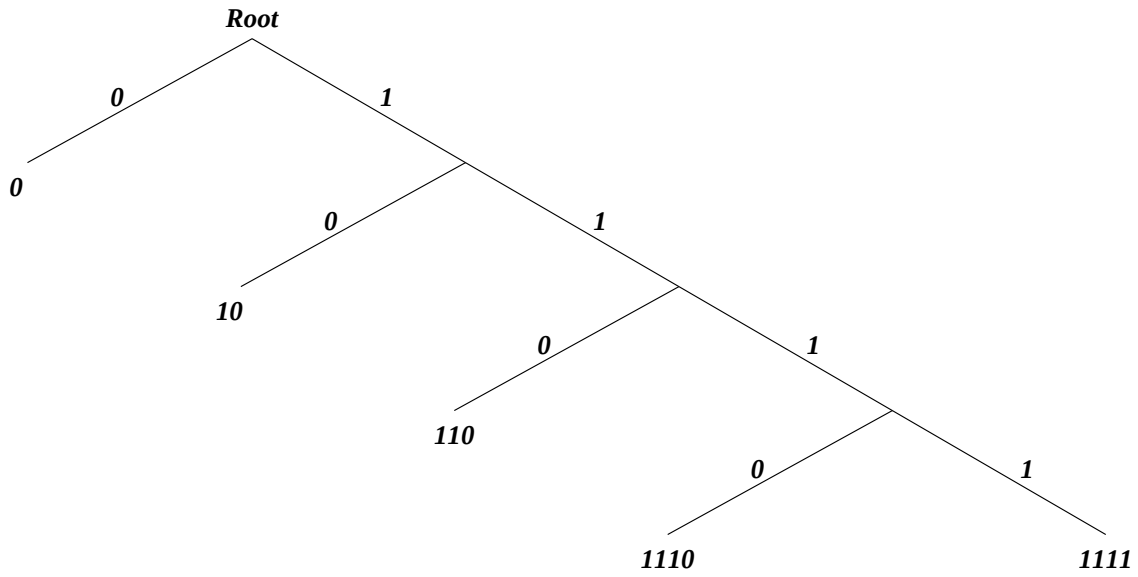


Figure 4: A codetree for a prefix code

Let us consider an example of a prefix code to illustrate the principle. To take an artificial example, suppose that we have discovered a flaw in the management of the local race-track; there is a split second between the end of the race and the close of betting, so that if we could place a bet in that brief moment of time, we could always beat the track. In this case, optimal coding is crucial since the time window within which we can place a bet is so brief that every millisecond counts. Imagine that five horses—Red, Orange, Black, Indigo, and Green. We can easily generate five codewords to create a prefix code for the five horses, as shown in figure 4. Notice how the code tree is constructed. Only the leaf nodes are labeled; each leaf is labeled by a codeword. Left-branches are associated with a "0" while right-branches are associated with a "1". Taking a left-branch results outputs a codeword whose end is signaled by "0". Only one codeword, "1111", lacks this property. Its end is signaled by its length. Thus, the code is self-punctuating. Let us associate the horse with codewords via the encoding function E : HORSES $\longrightarrow$ CODEWORDS:

(38) $E(\text{Red}) = 0$
$E(\text{Orange}) = 10$
$E(\text{Black}) = 110$
$E(\text{Indigo}) = 1110$
$E(\text{Green}) = 1111$

Suppose that the sequence "11111100101110" is transmitted over the channel. This sequence can be unambiguously decomposed into the codewords "1111" followed by "110" followed by "0" followed by "10" followed by "1110". Adopting the convention that or-

der in the sequence corresponds to order across the finish line, then we can interpret the string as indicating that Green was first, followed by Black in second place, Red in third, Orange in fourth and Indigo in last place. A little experimentation should show that any sequence of the codewords in (38) can be unambiguously segmented.

Notice that the code in (38) is not necessarily optimal. Recall, however, that we needed to report only the winner of the race and that we had only a very brief time to transmit the report and place the bet. In order to optimize our resources, we would want to assign the shortest codeword to the most likely winner, and so on. Notice the association between shortness and probability and recall the discussion above concerning randomness and description length; the association between description length and probability apparent here. Suppose that we have the following probabilities of winning:

(39) $\Pr(X = \text{Red}) = \frac{1}{2}$
$\Pr(X = \text{Orange}) = \frac{1}{4}$
$\Pr(X = \text{Black}) = \frac{1}{8}$
$\Pr(X = \text{Indigo}) = \frac{1}{16}$
$\Pr(X = \text{Green}) = \frac{1}{16}$

Now the code given in (38) is optimal. The most likely winner, Red, is associated with the shortest code word since $E(\text{Red}) = 0$ which is of length 1. Analogously, the least likely winners, Indigo and Green, are both associated with codewords of length 4.

Before continuing, we should note that an optimal prefix code like the one in (38) can be constructed via an algorithm; it requires no special, transcendental computational properties to devise a prefix code for the values of a random variable. A good example of an algorithm for constructing prefix codes is the one discovered by Huffman. Here, the values of the random variable are ranked from most likely to least likely, as in (39). The two least likely values are processed and assigned a codeword that differs only in the last bit. The process of combining the least likely values is repeated until values sum to 1. See Cover & Thomas (1991) for fully worked examples as well as proofs that the algorithm is optimal.

For present purposes, it is important to develop a sense of the relationship between codes and probabilities. In order to firm up this relationship, let us first note the existence of the so-called *Kraft inequality* (see Cover & Thomas, 1991, for proof):

(40) *Kraft Inequality*
For any prefix code over an alphabet of size D, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy the inequality:

$$\sum_i D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

Applying the Kraft inequality to our toy code in (38) we see that each codeword length associated with a horse is crucially related to the probability that the horse will win according to the distribution in (39). Thus, there is an interesting relationship between probabilities and codeword lengths in an optimal prefix code. This relationship can be best understood by considering the *entropy* of the random variable ranging over the things we wish to encode. Entropy is a measure of the degree of uncertainty of a random variable. Let X be a random variable ranging over an alphabet $\mathcal{X}$ with probability mass function $p(x) = \Pr\{X = x\},\ x \in \mathcal{X}$. Then:

(41) The *entropy* $H(X)$ of a discrete random variable X is defined by:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

In the discrete case, entropy measures the expected number of bits required to report what value the random variable X has taken on in an experiment. Note that in our horse race example in (39), the entropy is: $-(\frac{1}{2}\log\frac{1}{2} + \frac{1}{4}\log\frac{1}{4} + \frac{1}{8}\log\frac{1}{8} + \frac{1}{16}\log\frac{1}{16} + \frac{1}{16}\log\frac{1}{16}) = 1.875$ bits.

Naturally, there is a tight relationship between entropy and optimum codes. Intuitively, the best code is one which is just long enough to transmit a message and no longer. If a code is too short (below the number of bits required by entropy), then information is lost. If it is too long, then there are redundancies (and, hence, wasted effort) in the system. In fact, the following is a theorem (see Cover & Thomas, chapter 5, for a proof and discussion):[10]

[10] Note that $H_D(X)$ is the entropy of X calculated with a base D log.

(42) Let $l_1, l_2, \ldots l_m$ be the optimal codeword lengths for a source distribution $\mathbf{p}$ and a D-ary alphabet and let L be the associated expected length of the optimal code ($L = \sum p_i l_i$). Then:

$$H_D(X) \leq L < H_D(X) + 1.$$

The theorem in (42) just says that entropy of the source provides a bound on the length of the optimum codewords for encoding that source. Indeed, many data compression schemes rely on the relationship between entropy, probability and prefix codes to approach optimum compression. Returning to the code in (38) and the probability distribution in (39) we see that $L = \sum p_i l_i = ((\frac{1}{2} \times 1) + (\frac{1}{4} \times 2) + (\frac{1}{8} \times 3) + (\frac{1}{16} \times 4) + (\frac{1}{16} \times 4)) = 1.875$ which is the same as the entropy of the distribution; thus, the code given in (38) is optimal relative to the probability distribution in (39).

2.3 Machines, Programs and Descriptions

I have so far focussed on statistical aspects of descriptions. I turn, in this section, to some central notions of computation theory that will allow us to connect statistical properties like randomness and compressability to symbolic descriptions. Basic to the work to be discussed below is the notion of a Turing machine. We can visual a Turing machine as consisting of a read/write head (the *cursor*, positioned on an infinite paper tape, marked off into squares. Each square may be blank or may contain a symbol. A Turing machine has a single data structure, a string of symbols, and a very restricted set of operations. It may move a cursor left or right on the string, it can read the symbol of the string at the current cursor position and it may write a symbol at its current cursor position. The string acts both as a data structure and as a memory device for the Turing machine. Although the architecture is quite simple, Turing machines are powerful computational device, capable of performing any algorithm and simulating any programming language.[11]

Formally, a Turing machine consists of a quadruple $M = \langle K, \Sigma, \delta, s \rangle$. K is a finite set of states and $s \in K$ is a special state, called the *initial state*. Σ is a finite set of symbols, disjoint from K, called the *alphabet* of M. Σ must contain the special symbols $\sqcup$, corresponding to a blank on the tape, and $\triangleright$, the *first symbol*. δ is a transition function that maps pairs from $K \times \Sigma$ to $(K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$. Here, h is the *halting state*, “yes” is the *accepting state* and “no” is the *rejecting state*. Finally, $\{\leftarrow, \rightarrow, -\} \notin K \cup \Sigma$ are cursor directions; $\leftarrow$ for “left”, $\rightarrow$ for “right” and $-$ for “stay”.

The function δ is the program of the Turing machine; it specifies the current state $q_i \in K$, the current symbol being read by the cursor $\sigma \in \Sigma$ and a triple $\delta(q_i, \sigma) = \langle q_j, \rho, D \rangle$.

[11] Space prevents a more complete discussion of Turing Machines and computation theory here. For a more detailed exposition, see Papadimitriou (1994). I will rely on Papadimitriou's formalism for the discussion here.

The triple $\langle q_j, \rho, D\rangle$ specifies the state, q_j, that the Turing machine will enter upen reading σ, the new symbol ρ that the Turing machine will write over the old symbol σ, and D is a member of the set $\{\leftarrow, \rightarrow, -\}$ of cursor moves. The machine starts at the symbol $\triangleright$ on the tape in the state s. From this initial configuration, the behavior of the Turing machine is fully specified by the function δ. It continues to move through its computations until one of the three halting states (h, "yes", "no") has been reached. If the machine halts in the "yes" state on string x then it accepts x, if it halts in the "no" state on x then it rejects x. Finally, if it halts in the state h leaving the string y on the tape, then we will say that y is the output of the machine on x.

A final possibility is that the machine doesn't halt at all on x, but continues forever. In that case, we will write $M(x) = \nearrow$ for the Turing machine M does not halt on x. Let us note that there is no general method that will allow us to predict whether or an arbitrary Turing machine will halt on a string. This is the famous *halting problem.* I refer the reader to Rogers (1967) for a proof and theoretical discussion. For our purposes, it is sufficient to note the existence of the halting problem. The proof, however, crucially relies on the ability of a Universal Turing machine to simulate other Turing machines. Intuitively, a Universal Turing machine may be thought of as being *programmable* in the same way that a personal computer is programmable. We can imagine that each δ function can be enumerated by an infinite list. The index of a Turing machine M will be the number associated with M's δ function on the list. A Universal Turing machine can be given a pair (i, x) where i is the index of a Turing machine and x is a string. It finds the δ function in the ith position on the list and simulates the Turing machine, M_i on the string x:

(43) If $M_{\mathcal{U}}$ is a universal Turing machine then:

$$M_{\mathcal{U}}(i, x) = M_i(x)$$

where M_i is the ith Turing machine in the enumeration.

The notion of universal Turing machine will play a role in our discussion of descriptive complexity, below.

We should note that the architecture for the Turing machine described above, with a single infinite tape and a read/write head, is not the only possible architecture for a Turing machine. Consider, for example, a k-string Turing machine, where $k \geq 1$ is an integer. As above, a k-string Turing machine $M = \langle K, \Sigma, \delta, s\rangle$ consists of a set K of states, an alphabet Σ, an initial state s and a set δ. Like the 1-string machine above δ determines the next state, but unlike the 1-string machine it also determines the symbol overwritten and the cursor movement by looking at the current state and symbol for each of the k strings. Thus, δ is a transition function that maps pairs from $K \times \Sigma^k$ to $(K \cup \{h, \text{"yes"}, \text{"no"}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$. For example, $\delta(q_i, \sigma_1, \ldots, \sigma_k) = (q_j, \rho_1, D_1, \ldots, \rho_k, D_k)$ means that the machine M is in state q_i when it reads σ_1 in the first string, σ_2 in the second string, and so on. It then enters state q_j, writes ρ_1 in the first string, moving the first cursor

in direction D_1, writes ρ_2 in the second string, moving the second cursor in direction D_2 and so on. All the strings begin with the reserved symbol $\triangleright$. The output of the machine can be read from the kth string, if the machine halts. Notice that the 1-string Turing machine described above is just a special case of a k-string machine so that this new characterization is a generalization of the old one.

As an example, consider the δ function for a 2-string machine which decides palindromes. Intuitively, the machine starts by copying its input onto the second tape, positioning its first cursor at the start of the first string and its second cursor at the end of the second string. It then steps through the first string from left to right and the second string from right to left, comparing symbols as it does so. As long as the two symbols match, it continues.

$$
\begin{aligned}
(44)\quad & \delta(s, 0, \sqcup) = (s, 0, \rightarrow, 0, \rightarrow) \\
& \delta(s, 1, \sqcup) = (s, 1, \rightarrow, 1, \rightarrow) \\
& \delta(s, \triangleright, \triangleright) = (s, \triangleright, \rightarrow, \triangleright, \rightarrow) \\
& \delta(s, \sqcup, \sqcup) = (q, \sqcup, \leftarrow, \sqcup, -) \\
& \delta(q, 0, \sqcup) = (q, 0, \leftarrow, \sqcup, -) \\
& \delta(q, 1, \sqcup) = (q, 1, \leftarrow, \sqcup, -) \\
& \delta(q, \triangleright, \sqcup) = (p, \triangleright, \leftarrow, \sqcup, \rightarrow) \\
& \delta(p, 0, 0) = (p, 0, \leftarrow, \sqcup, \rightarrow) \\
& \delta(p, 1, 1) = (p, 1, \leftarrow, \sqcup, \rightarrow) \\
& \delta(p, 0, 1) = (\text{“no”}, 0, -, 1, -) \\
& \delta(p, 1, 0) = (\text{“no”}, 1, -, 0, -) \\
& \delta(p, \sqcup, \triangleright) = (\text{“yes”}, \sqcup, -, \triangleright, -)
\end{aligned}
$$

The structure of the program in (44) should be fairly easy to see. While the machine is in the s state, it copies the string on the first tape to the second tape. When it hits a blank on the first tape, it enters state q and repositions its first cursor to the beginning of the first string, leaving its second cursor at the end of the second string. When it reads $\triangleright$ in the first string, it has hit the beginning of that string and can now compare the two strings. It enters the p state and begins the comparison. If the two strings disagree at any point, it enters the “no” state and stops. If it reads $\sqcup$ in the first string and $\triangleright$ in the second string, then it has gone through the entire string. It enters the “yes” state and halts. Otherwise, it simply continues the comparison.

It is interesting to note that adding strings does not change the set of functions computed by the Turing machine. A 1-string machine accepts the same languages that a 2-string machine accepts. Programming a 2-string machine can be simpler than programming a 1-string one, however. This fact will be relevant to our discussion of complexity below. In the palindrome example, a 2-string machine is relatively easy to program since it can copy the input string and then compare the two strings point by point. A 1-string machine is fairly laborious to program since it must move back and forth from the be-

ginning to the end of the string, comparing symbols one by one. This requires a larger number of states as well as more instructions. It is important to note, however, that an n-string universal Turing machine can simulate the output behavior of an m-string machine, where $m \neq n$, so that an external observer would be unable to guess whether the machine had m or n strings.

Although the structure of a Turing machine is quite simple, it is a powerful computational device. The palindrome language is context-free, for example. In fact, Turing machines can compute the recursively enumerable (R.E.) sets. Although the exact character of the set of natural languages is as yet unknown, recent research indicates that it is likely to be mildly context sensitive. As figure 5 shows, the set of mildly context sensitive languages lies well within the set of R.E. languages. Thus, a Turing machine has more than enough power to compute the representations for natural language sentences.

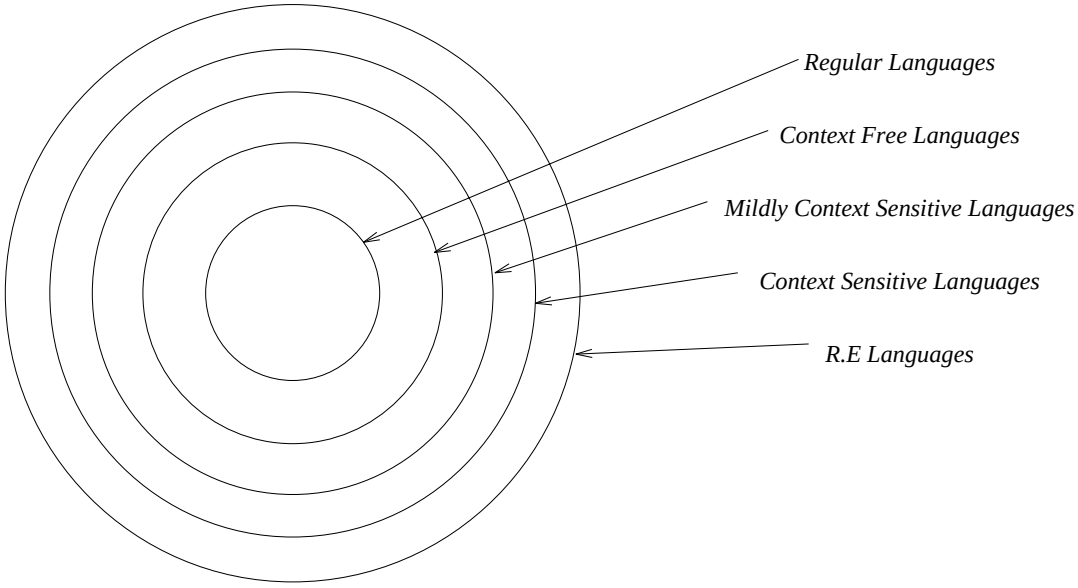


Figure 5: The Chomsky Hierarchy

Despite its computational power, the vocabulary for programming Turing machines is quite restricted, consisting of a finite set of states, a finite vocabulary, and a finite set of cursor moves. We can, if we so choose, apply the methods discussed in section 2.2 to the symbols in $K \cup \Sigma \cup \{h, \text{“yes”}, \text{“no”}, \leftarrow, \rightarrow, -\}$ to encode Turing machines. That is, we can think of a code, $C : \mathcal{X} \rightarrow \mathcal{D}^*$ where the random variable $\mathcal{X}$ ranges over the symbols used in the δ function and $\mathcal{D}^*$ is the set of strings of finite length on a vocabulary $\mathcal{D}$. Indeed, let us assume that $\mathcal{D}$ is the set $\{0, 1\}$. The function C would take δ functions into strings of binary digits. In (45) I've given a prefix code for specifying δ functions over the vocabulary $\{0, 1\}$; note that q_i refers to states in K:

(45) $E(\sqcup) = 0$
$E(0) = 10$
$E(1) = 110$
$E(\triangleright) = 1110$
$E(\text{“yes”}) = 11110$
$E(\text{“no”}) = 111110$
$E(h) = 1111110$
$E(\rightarrow) = 11111110$
$E(\leftarrow) = 111111110$
$E(-) = 1111111110$
$E(s) = 11111111110$
$E(q_0) = 111111111110$
$E(q_1) = 1111111111110$
$E(q_n) = 1^n111111111110$

Like the code in (38) on page 26, the code in (45) is self-punctuating; thus, we can concatenate codewords into longer strings without introducing ambiguity into the string. A receiver could easily translate a long bit string encoded according to (45) back into a program for a Universal Turing machine.

Consider, for example, the program for recognizing the palindrome language in (44). Recall that the program is for a 2-string Turing machine. This means that all of the rules are of constant length, since each rule maps a triple (the state of the machine and the current symbols in each string) to a quintuple (the new state, the symbols written in each string and the cursor direction for each cursor). The receiver can exploit this fact and the fact that each symbol in the program is encoded unambiguously by the prefix code. Consider the first rule:

(46) $\delta(s, 0, \sqcup) = (s, 0, \rightarrow, 0, \rightarrow)$

Stripping away the δ, the parentheses, the commas and so forth, we need only encode the following sequence of symbols:

(47) $s0 \sqcup s0 \rightarrow 0 \rightarrow)$

We can now step through the sequence in (47) and associate each symbol with its codeword. This results in the sequence:

(48) 11111111110 10 0 11111111110 10 11111110 10 11111110

The sequence in (48) can be concatenated to yield a single binary number:

11111111110100111111111101011111110101111111 0

This process can be repeated for each rule in (44) to encode it. Finally, all of the binary numbers in the program can be concatenated to yield a single binary number. Notice that this number could be taken as the index in the enumeration of Turing machines used by a universal Turing machine.

Notice how the encoding procedures discussed in section 2.2 can dovetail with the theory of computation described above. Nothing, however, guarantees that the code given in (45) is optimal. This property would hinge on the actual statistical distribution of the symbols in Turing machine programs. Notice that we could imagine that a random variable, $\mathcal{X}$, was ranging over the symbols for encoding programs for a particular universal Turing machine. We could then investigate the statistical distribution of these symbols in the programs and develop an optimal prefix code. The result could be used to transmit compressed versions of these programs or, indeed, act as the index for each encoded program.

We could imagine, for example, generating potential Turing machine programs by flipping a coin. One could theorize about the likelihood that a sequence of coin tosses results in a well-formed program. Notice, significantly, that the longer the sequence is, the less likely it is to result in a well-formed program; complex objects become increasingly unlikely. This intuition, we will argue, has significance for issues of language learnability; the more complex a linguistic structure is, the harder it should be to learn, the less often the learner should encounter it and, thus, the longer it should take the learner to master it. The views we have been developing in this section combines the symbolic aspects of computation theory (Turing machines) with statistical properties (optimal prefix codes). We will begin a serious investigation of this relationship, one that makes rigorous the above intuitions, in the next section.

2.4 Descriptive complexity and algorithmic information theory

We now turn to the inherent descriptive complexity of an object. There are general methods for estimating the amount of information associated with an object, whether the object is a phrase-marker, a strand of DNA or a lump of coal, as we shall see. These methods rely on the tools outlined in the preceding sections; in particular, the notion of effective tests for randomness, statistical methods for data compression and computation theory. We will now turn to the general theory of descriptive complexity.

The basic intuition underlying the theory is that, given a description language D, the complexity of an object should correspond to the length of the shortest description in D. The description language must be powerful enough to describe computations. In fact, we will mainly be interested in linguistic representations, so that we would like a description language that is powerful enough to describe possible linguistic representations. D can be thought of as a programming language for linguistic representations. Clearly, linguistic representations are a subset of the R.E. languages; we might consider restricting

the automata associated with D. For example, we might allow D to be a language for programming linear bound automata. For present purposes, however, we need not worry about this. Let us take as given a particular universal Turing machine, call it $\mathcal{U}$, with D as its programming language. Clearly, $\mathcal{U}$ will be able to compute labeled bracketings (for example) as well as a great deal of other things. Thus, although $\mathcal{U}$ is likely to be more powerful than we require, it can certainly do the job we need it for. If x is a program written in the language D, then $\mathcal{U}(x)$ denotes the result of running $\mathcal{U}$ on x. For present purposes, I will conflate the description of an object with the object itself; thus, if x is a description of the object y in D then we will write $y = \mathcal{U}(x)$, even though the output of $\mathcal{U}(x)$ is a description of y and not necessarily y itself.

We define *Kolmogorov complexity* as follows:

(49) The *Kolmogorov complexity* $K_{\mathcal{U}}(x)$ of a string x with respect to a universal computer $\mathcal{U}$ is defined as:

$$K_{\mathcal{U}}(x) \;=\; \min_{p\,:\,\mathcal{U}(p)=x} l(p)$$

where $l(p)$ denotes the length of the program p.

In other words, the Kolmogorov complexity of an object x is the length of the shortest program, p, for $\mathcal{U}$ that allows $\mathcal{U}$ to compute a description of x. There is a great deal to be said about the use of the term *shortest* in this context. For the moment, let us note that that the shortest program is one that cannot be compressed any further. This should immediately recall our discussion of data compression in section 2.2, above. We noted there that there was an important relationship between probability and codeword length (see the discussion of the Kraft inequality on page 28).

It should be emphasized that x itself can be anything we can describe. For example, we might estimate the complexity of Marcel Duchamp's "Nude Descending a Staircase" by scanning the picture and performing our calculations on the resulting binary file. This would be a pixel by pixel encoding of the image of the painting. A receiver could reconstruct a fair copy of Duchamp's painting from our encoding, so x will do as a fair estimate of the information content of the painting. Furthermore, suppose that the most common color in Duchamp's painting is a sort of reddish brown; then that is the most likely hue associated with a pixel drawn at random from the encoding and, by the reasoning associated with data compression, that should be the color encoded by the shortest codeword in the description of the painting. The shortest program can do no better than the optimal compression of the best description of the painting. It could easily do worse; it could assign the shortest codeword to the least likely color in the painting, for example. The resulting ndescription would be gratuitously complicated by wasted bits due to a bad choice of codewords.

In our case, x ranges over programs for computing linguistic representations; let us

call this set X:

(50) $X = \{x \ : \ x$ is a program which computes a well-formed labeled bracketing for a sentence of English $\}$

In other words, $\mathcal{U}(x)$ will be the representation of a grammatical utterance. For present purposes, I will suppose that these representations are labeled bracketings. These bracketings may be as detailed as the reader likes; they may contain feature decompositions and so on. All that we require is that x be a set of instructions that cause the machine $\mathcal{U}$ to print a particular labeled bracketing. Crudely, all the programs in X might be of the following form:

(51) Print z
where z is a labeled bracketing.

We would like to use the above definition of complexity as a measure of the relative complexity of programs in X; that is, $K_{\mathcal{U}}(x)$ would be an estimate of the information content of the linguistic representation output by the universal Turing machine $\mathcal{U}$ when given the input x. We might think of x as being the best description of the linguistic representation available in the description language D.

It might seem as though the above definition of complexity is of only limited interest, since it is defined relative to a particular universal Turing machine, $\mathcal{U}$. In fact, Kolmogorov complexity is machine independent as shown by the following theorem (see Cover & Thomas, 1991, for a complete proof):

(52) *Universality of Kolmogorov Complexity*
If $\mathcal{U}$ is a universal computer, then for any other computer $\mathcal{A}$,

$$K_{\mathcal{U}}(x) \ \leq \ K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

for all strings $x \in \{0,1\}^*$, where the constant $c_{\mathcal{A}}$ does not depend on x.

Briefly, suppose that $\mathcal{A}$ is a Turing machine and that $K_{\mathcal{A}}(x)$ is the complexity of x relative to $\mathcal{A}$. Since $\mathcal{U}$ is a universal Turing machine, it can simulate any other Turing machine. In particular, it can simulate $\mathcal{A}$. Let $c_{\mathcal{A}}$ be the Kolmogorov complexity of the program, y that $\mathcal{U}$ uses to simulate $\mathcal{A}$. We can compute a description of x on machine $\mathcal{U}$ using the program we used to compute x on machine $\mathcal{A}$ plus y, the simulation program. Thus, the Kolmogorov complexity of x relative to $\mathcal{U}$ is bounded from above by the Kolmogorov complexity of x relative to machine $\mathcal{A}$ plus the Kolmogorov complexity of y. The absolute Kolmogorov complexity of x relative to $\mathcal{U}$ may well be less than this amount, but it can never exceed $K_{\mathcal{A}}(x) + c_{\mathcal{A}}$.

In other words, our complexity calculations are independent of the architecture of

the universal computer $\mathcal{U}$ we have chosen; any other choice would lead to a variation in the complexity bounded by a constant term and, thus, well within the same order of magnitude of our estimate of complexity. Given the result in (52), we can drop reference to the particular machine we use to run the programs on.

One might still object that the constant c can be extremely large. In particular, we could imagine the case where an incredibly large and complex structure is encoded using a single of bit, say "0". Let us assume that the descriptions are both self-punctuating (see figure 4 on page 26 and the accompanying discussion) and is otherwise optimal. In this case, the Turing machine, call it $M_{\mathcal{T}}$, is designed to print out that structure when it encounters the codeword "0"; otherwise, it is resembles the optimal machine for the descriptions. The theorem in (52), in fact, guarantees that $M_{\mathcal{T}}$ differs from the optimal machine by at most a constant.

Notice, though, that $M_{\mathcal{T}}$ has had to pay a price for giving an overly simple description to an extremely complex object. All of its other descriptions are now one bit too long (at least). What would otherwise be the simplest possible program for $M_{\mathcal{T}}$ has been gratuitously complicated because a complex object has superseded its place in the encodings for programs. But then all the programs for $M_{\mathcal{T}}$, except for one, will be systematically greater than the true Kolmogorov complexity of the objects they describe. There is more to be said about this example, and I will return to it below, since it suggests that we should select a reference machine that allows for an enumeration of programs that matches the probability of the objects that the program describes. In particular, we must consider probability distributions in fixing a good interpretation for our description language D. I will return to the problem of fixing a reference machine below.

Having seen that Kolmogorov complexity is invariant up to a constant across computing machines, let us turn, briefly, to some general results that bound the complexity of descriptions. Let us first define *conditional Kolmogorov complexity* as in (53):

(53) *Conditional Kolmogorov Complexity*
If $\mathcal{U}$ is a universal computer then the conditional Kolmogorov complexity of a string of known length x is:

$$K_{\mathcal{U}}(x|l(x)) = \min_{p\,:\,\mathcal{U}(p,l(x))=x} l(p)$$

The definition in (53) is the shortest description length if $\mathcal{U}$ has the length of x made available to it. From the above definition, it is a fairly routine matter to prove the following:

(54) *Bound on conditional Kolmogorov complexity*
$K(x|l(x)) \;\leq\; l(x) + c$

In this case, the length of the string x is known before hand. A trivial program for describing x would, therefore, be merely "Print the following $l(x)$ bits: $x_1 x_2 \ldots x_{l(x)}$". That is we simply transmit the description along with a print instruction. The length of the above program is therefore $l(x)$ plus the print instruction, c. Hence, $K(x|l(x))$ is bounded from above by $l(x) + c$. This means that the conditional complexity of x is less than the length of the sequence x. Notice that the conditional complexity of x could be far less than $l(x)$; we have guaranteed that the complexity of an object will never exceed its own length.

What happens if we don't know the length of the program x? In this case, the end of the description of x will have to be signaled or computed somehow. This will add to complexity of the description, but by a bounded amount. Thus, the following is a theorem (see Cover & Thomas, 1991 for a formal proof):

(55) *Upper bound on Kolmogorov complexity*
$K(c) \leq K(x|l(x)) + 2 \log l(x) + c$

The addition term, $2 \log l(x)$, comes from the punctuation scheme that signals the end of x. Notice the utility of self-punctuating codes in this context.

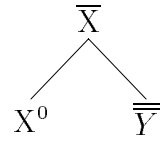
We have seen in (54) and (55) that we can estimate the inherent descriptional complexity of an object by the expedient of using programs which compute a description of the object and that this metric is universal. Once a program that computes a description of the object has been discovered, it is an upper bound on the actual Kolmogorov complexity of that object. Can we ever discover the actual Kolmogorov complexity of the object? It is perhaps surprising to realize that we can't. Recall that we are measuring complexity relative to programs for a universal Turing machine, $\mathcal{U}$. Suppose that we were to enumerate the possible programs in lexicographic order (starting from the shortest program and proceeding in alphabetical order). We could then run each program on $\mathcal{U}$. Suppose that $\mathcal{U}(p_i) = y$ (that is, $\mathcal{U}$ halts on p_i, yielding a description of y); we can enter $l(p_i)$ as an estimate of $K(y)$. But there may be programs shorter than p_i such that $\mathcal{U}$ has yet to halt on these programs. In particular, suppose that there is a program p_j such that $l(p_j) < l(p_i)$ and $\mathcal{U}(p_j)$ has not yet halted. It could be that $\mathcal{U}(p_j)$ will eventually halt with $\mathcal{U}(p_j) = y$. If so, then $l(p_j)$ is a better estimate of $K(y)$ than $l(p_i)$. If we could know that $\mathcal{U}(p_j) = y$ then we could find the actual Kolmogorov complexity of y. But this entails that we know that $\mathcal{U}(p_j)$ halts, which in turn entails that we have a solution to the Halting Problem.[12] Since the Halting Problem is unsolvable, we cannot guarantee that we have arrived at the true Kolmogorov complexity of an object once we have a program which computes its description. In other words:

(56) An upper bound on the Kolmogorov complexity of an object can be found, but a lower bound cannot.

[12] I refer the reader to section 2.3 for a brief discussion of the halting problem.

As with the result in (52) on page 36, the reader may be concerned that the actual utility of the theory for linguists is relatively low. Not only could the result be washed out by a large constant, but we cannot place a lower bound on the complexity assigned to any given representation; in other words, there might be a less complex description available, but we cannot find it.

Some comments, then, are necessary to justify our interest in Kolmogorov complexity. First, an upper bound is of some linguistic interest. In particular, we want to estimate the relative complexity of structures which express linguistically variant properties, our goal being to establish a theory of linguistic evidence. This suggests that we should attempt to establish the least structure for what we have been calling parameter expression; this is the smallest structure which exhibits a linguistically variant property. Suppose we have found that a variant property, say the relative order of a head and its complement, can be expressed on a representation of the form (order irrelevant):

(57) 

The result in (56) suggests that there might be a more compact description of the structure on which this relation is expressed. Let τ be the program that prints a description of the structure in (57); we know from (56) that that this structure cannot be any more complex than $K(\tau)$. Thus, we might define the complexity of a linguistically variant property as in (58):

(58) The complexity of a variant property p is $\min_\tau(K(\tau))$ where τ computes a representation which expresses p

Naturally, for all variant properties, we can only estimate their complexity from above. Note that, in a sense, Degree 2 learnability was also an upper bound; some transformations could be learned from structures that were smaller than degree 2, but no structure greater than degree 2 was required to learn any transformation. Notice that I am not proposing a flat upper bound on the complexity of variant properties. Some properties can be learned from quite simple evidence and others might require more complex evidence. We will suggest that there is a non-trivial relationship between complexity, in the sense of (58), and order of acquisition.

Second, we can constrain our theory using observed probability distributions. This point is related to the discussion of the perverse universal Turing machine $M_{\mathcal{T}}$, above, which reserved its simplest representation for a highly complex structure. We will explore this point more fully below. For now, it is sufficient to recall that, as stressed above, the best description for an object cannot beat the best data compression and that the best data compression must exploit the probability distribution of a random variable ranging

over the set to be compressed. This suggests that we can estimate the Kolmogorov complexity of a variant property by observing the real distribution of that property in natural texts, particularly adult input to learners.

Finally, our interest should be in the relative complexity of constructions given a machine which we have optimized for computing linguistic representations. For our purposes, it is interesting to note that one property can be expressed on a simpler structure than another property, given a fixed computational architecture. We assume here that the Turing machine selected is as close to optimal as we can make it for linguistic purposes. Having fixed such an architecture, call it $\mathcal{A}$, we can study the relative complexity of properties $p1$ and p_2, expressed on structures described by π_1 and π_2 respectively, by comparing the values of $K_{\mathcal{A}}(\pi_1)$ and $K_{\mathcal{A}}(\pi_2)$. Again, using empirical estimates of the relative likelihood of the constructions at hand should serve to constrain the our descriptions.

Having fixed a plausible architecture $\mathcal{A}$ within which to study the complexity of structures, the size of the constant $c_{\mathcal{A}}$ in (52) is no longer of concern to us. The relative complexity of structures which express linguistically variant properties would allow us to rank structures relative to our choice of computational architecture. If simpler structures are, indeed, more likely to occur in texts, then we would expect a learner to reflect these differences in relative frequency. This would follow from the frequency of parameters expression, discussed above (see (18) on page 17). That is, the more frequently the learner encounters examples of a variant property, the more likely it is to learn it. Frequency, then, should be reflected in order of acquisition. Studying the Kolmogorov complexity of the structures which express variant properties is, then a powerful tool in explaining developmental sequences.

2.5 Kolmogorov complexity and probability

The definitions and theorems presented in section 2.4 allow any program for the universal Turing machine to count as a possible description. A number of interesting results hold if we require that the programs be prefix codes (see the discussion of (37) on page 25 as well as the general discussion in section 2.2). It should be unsurprising that there is an interesting relationship between Kolmogorov complexity proper and data compression, given that Kolmogorov complexity is concerned with optimum description length. Presumably, the shortest description of an object is already in its most compressed form (otherwise, it wouldn't be the shortest description). Let us assume that we encoded the programs for our universal Turing machine $\mathcal{U}$ using a prefix code. The theorem in (59) can be seen as the complexity analog of the Kraft inequality in (40) on page 28:

(59) For any computer $\mathcal{U}$:

$$\sum_{p\,:\,\mathcal{U}(p)\text{ halts}} 2^{-l(p)} \;\leq\; 1.$$

Recall that the Kraft inequality stated that the sum of D, the size of the alphabet, raised to the codeword lengths in a prefix code must be less than or equal to 1. The idea is that prefix codes are optimum when codewords are assigned in such a way that they match the probabilities associated with the encoded random variable. Short codewords should absorb most of the probability in order to optimize the code. The theorem in (59) shows that the halting programs for our machine $\mathcal{U}$ must form a prefix code. Notice that 2 is used as the base in (59) since we are using a binary encoding for our reference Turning machine.

Recall, further, that there is a systematic relationship between optimal codeword lengths and entropy (see (42) on page 29). The entropy of the random variable being encoded bounds optimum codeword lengths from below while the entropy plus 1 (one full bit to absorb any decimal remainder) bounds it from above. From (59), the fact that the halting programs form a prefix code, we would expect that entropy of a random variable X ranging over an alphabet $\mathcal{X}$ should provide a useful bound on the Kolmogorov complexity of objects described by $\mathcal{X}$. This is indeed the case, as the following rather imposing looking theorem states:

(60) *The relationship between Kolmogorov complexity and entropy*
Let the stochastic process $\{X_i\}$ be drawn in an independent identically distributed fashion according to the probability mass function $f(x)$, $x \in \mathcal{X}$, where $\mathcal{X}$ is a finite alphabet. Let $f(x^n) = \prod_{i=1}^{n} f(x_i)$. Then there exists a constant c such that

$$H(X) \leq \frac{1}{n} \sum_{x^n} f(x^n) K(x^n|n) \leq H(X) \; + \; \frac{|\mathcal{X}| \log n}{n} + \frac{c}{n}$$

for all n. Thus

$$E \frac{1}{n} K(X^n|n) \longrightarrow H(X).$$

That is, the average expected Kolmogorov complexity of length n descriptions should approach entropy as sample size grows. Intuitively, our theory must specify those aspects of an object which cannot be predicted; thus, we could never give a description of an object that was below entropy. Our description should not be much greater than entropy, however, since entropy bounds the best compression of descriptions, by the Kraft inequality. Note that, for our purposes, the random variable is ranging over structures which express linguistically variable properties as they occur in texts presented to a learning machine.

We have so far noted a relationship between Kolmogorov complexity, prefix codes and entropy. The relationship may seem both surprising and deeply suggestive. Recall that Kolmogorov complexity is defined relative to symbolic objects, namely Turing machine programs; we can, in fact, think of these programs as programs for a physical computer

if we like. Entropy is a statistical notion, a measure of the amount of uncertainty in a system. Nevertheless, as (60) shows, there is a systematic relationship between entropy and Kolmogorov complexity.

In order to firm up this intuition, consider the following thought experiment. Suppose we started feeding a computer randomly generated programs. Sticking to the binary programming language we have been using for Turing machines, we might generate these programs by tossing a coin and using "1" for *heads* and "0" for *tails*. In general, these programs will crash (halt with no output), but every once in a while one of them will halt with a sensible output. Thus, the following quantity is well-defined:

(61) The *universal probability* of a string x is

$$P_{\mathcal{U}}(x) \;=\; \sum_{p\,:\,\mathcal{U}(p)=x} 2^{-l(p)} \;=\; \Pr(\mathcal{U}(p) = x),$$

which is the probability that a program randomly drawn as a sequence of fair coin tosses $p_1, p_2, \ldots$ will print out the string x.

Notice the similarity between the definition in (61) and the Kraft inequality in (40). Given the relationship between optimal prefix codes and probability, we would expect that there should be a tight relationship between universal probability and Kolmogorov complexity. Indeed, the following is a theorem (see Cover & Thomas, 1991):

(62) $P_{\mathcal{U}}(x) \;\approx\; 2^{-K(x)}$

That is, we can approximate the universal probability of x by using its Kolmogorov complexity. Intuitively, this is because the high probability things are encoded by short strings, as we have seen. Thus, simple objects are much more likely than complex ones. Suppose that for each variable linguistic property (parameter) we take the Kolmogorov complexity of the smallest structure which expresses it. Those parameters associated with low complexity should be more likely to be expressed in the input text, since they will have relatively high universal probability by (62). As noted above, we would expect that parameters with low Kolmogorov complexity to be set relatively early. This follows from the interaction between the Frequency of Parameter Expression (see (18) on page 17) and Boundedness of Parameter Expression (see (23) on page 19). We initially formalized this via minimal texts and fair texts. The intuition was that the simpler the parameter was, the more frequently it would be expressed and, therefore, the more likely it was to be set correctly. Notice, though, that the properties of these texts follow directly from the complexity theory outlined here. In particular, if we take the constant U in the definition of Boundedness of Parameter Expression to be a function of the (average) Kolmogorov complexity of the parameters in the system, then the frequency of expression of the parameters will follow from the Kolmogorov universal statistic. Thus, the theory

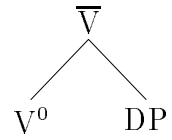
of Kolmogorov complexity, and its association with universal probability, formalizes the informal argument made in section 1.2.

3 Descriptive Complexity and the Learning Model

We began by considering the relationship between input simplicity and learnability. The input to the learner is, by and large, simple and grammatical. It seems, then, that linguistically variant properties must find their expression within the limits of such simple structures. If thiser were not so, then the learner would have fewer chances of encountering the correct form and would, therefore, be less likely to master the form quickly and correctly. If the expression of the variant property becomes too complex, then the learner might not have any chance to master that property within realistic time limits. In order to formalize these intuitions, we appealed to the theory of descriptive complexity. As we have seen, this theory captures the underlying intuition in an elegant and general way, making a direct connection between the "symbolic" complexity of an object and its likelihood.

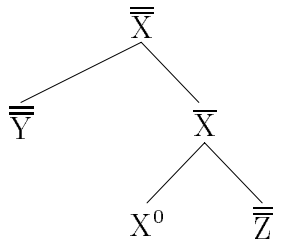
Notice, though, that there are two ways for an object to be simple. It can be small or it can be predictable. A simple verb-object sequence can be expressed on a compact tree, for example:

(63)



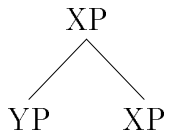
The tree in (63) is extremely simple, although even it contains some predictable properties. The verb, for example, contains information about its lexical requirements which make the presence (and perhaps certain semantic properties) of the object predictable. Thus, the optimal representation of the structure in (63) would no doubt be even smaller than shown here.

The preceding example illustrates, then, the other way in which an object can be simple. In particular, if some of the structure is predictable from general principles then a representation could be highly articulated yet still receive a relatively low Kolmogorov complexity. In present terms, predictability is a property of the reference machine. For example, the computational architecture could be optimized to exploit $\overline{X}$-theory. The following structure would then have a relatively low descriptive complexity:

(64) 

Assuming that features of the head are transmitted throughout its projections, the properties of $\overline{X}$ and $\overline{\overline{X}}$ are predictable. Similarly, many of the properties of the complement, $\overline{\overline{Z}}$ can be predicted from features on X^0. Analogously, given Spec-head agreement, the best representation of the relations in (64) would allow properties of the specifier, $\overline{\overline{Y}}$, to be recovered from features on the head. Thus, the structure in (64) could receive a very compact representation given that our reference machine is optimized to exploit $\overline{X}$-theoretic relations and Spec-head agreement. Note that the resulting representation would look rather different from the one given in (64), since redundancies would be eliminated. We should not, however, make the mistake of reifying representations like (64), precisely because so much of the representation is predictable from first principles. Thus, we would expect the optimal representation to look quite different.

Given the above reasoning, certain configurations will be rather more complex. Consider, in particular, adjunction structures:

(65) 

Although XP has been copied so that its description need not be repeated in its entirety, no core principles of $\overline{X}$-theory allow us to state any precise relationship between XP and YP. That is, the description of YP cannot be predicted from properties of XP. Thus, we would expect the Kolmogorov complexity of structures like (65) to be rather higher than the simple structure in (64).

Similar comments can be applied to constructions involving $\overline{A}$-movement. Consider, for example, the following partial bracketing:

(66) [who$_i$ will$_j$ [John [t_j [see t_i]]]]

Notice that the construction in (66) actually involves two distinct processes. First, there is movement of the *wh*-phrase to the specifier position of CP. Second, there is movement of the auxiliary to C^0. Let us consider, first, the movement of the *wh*-phrase. Notice that its surface position is not one which is selected for; that is, any maximal projection could, in principle, be moved to this position. Thus, there is little that is predictable about this

position, given the structure.[13] In order to receive an interpretation, the wh-phrase must form a chain with its trace. Once again, the exact site of the trace cannot be predicted from first principles, but is a contingent property of the example at hand. Therefore, its exact location must be stipulated in the representation of (66); this stipulation will increase the Kolmogorov complexity associated with the example. Furthermore, although much of the properties of head movement can be derived from first principles, we would expect the subject-auxiliary inversion in (66) to involve a small increase in complexity.

If the above reasoning is correct then, given a choice of a variety of representations for a given example, those that do not involve $\overline{\text{A}}$ movement processes will on average receive a lower Kolmogorov complexity than those which include such movements. One can, on this basis, imagine a language processing device built to select an output representation (or set of representations) by selecting the least descriptively complex structure from a set of grammatically possible descriptions of an input string. No doubt arguments like the above can be repeated to rank the complexity of a wide variety of constructions. Let us turn, rather, to the notion of selection discussed in Clark (1992) and Clark & Roberts (1993).

We have been assuming a model of language learnability grounded in selection of the most fit hypotheses from a pool of available options. This selection is performed according to the metric given in (67):

(67) *The Fitness Metric*

$$\frac{(\sum_{j=1}^{n} v_j + c\sum_{j=1}^{n} e_j) - (v_i + ce_i)}{(n-1)(\sum_{j=1}^{n} v_j + c\sum_{j=1}^{n} e_j)}$$

The fitness metric compares representations of an input datum produced by different grammars and ranks them according to the following criteria:

1. The number of violations, v_j of core principles associated with a representation.

2. The "elegance", e_j, of a representation measured in terms of the number of nodes in the phrase-marker that spans the input string.

The constant c in (67) weights the importance of the "elegance" relative violations of grammatical principles. In general, we have assumed that $c < 1$, so that avoiding violations of grammatical principles takes precedence over elegance.

We are now in a position to give some substance to the notion of "elegance" in (67). Our goal is to select that hypothesis which, on the whole, tends to assign the least complex

[13]This point is made quite forcefully in Brill & Kapur (1993). Their study of the relative entropy around the verb in V2 and non-V2 languages shows a marked increase in entropy before the verb in V2 languages. Given the results in sections 2.4 and 2.5, we would expect an increase in complexity in this construction. Analogous comments hold for wh-movement and the residual V2 found in examples like (66).

grammatical analysis possible, up to grammatical violations, to an arbitrarily selected sentence. This suggests immediately that Kolmogorov complexity should form the basis for the elegance metric:

(68) *The Revised Fitness Metric*
Let $\mathcal{A}$ be the reference Turing machine for computing complexity, X be a population of grammatical hypotheses and τ_n the structural description generated by hypothesis $x_n \in X$ and let $v(x_n)$ be the number of grammatical violations generated by $x_n \in X$ on a given input. Then the fitness of an individual hypothesis x_i drawn from X is given by:

$$\frac{(\sum_{j=1}^{n} v(x_j) + c\sum_{j=1}^{n} K_{\mathcal{A}}(\tau_j)) - (v(x_i) + cK_{\mathcal{A}}(\tau_i))}{(n-1)(\sum_{j=1}^{n} v(x_j) + c\sum_{j=1}^{n} K_{\mathcal{A}}(\tau_j))}$$

where $c < 1$.

The revised metric in (68) uses Kolmogorov complexity as part of the basis for its choice. The same complexity metric forms the basis for the theory of locality of parameter expression.

In order to make the above metric more concrete, let us turn to some specific examples. Consider the case where the learning algorithm has been presented with a string with VO order such as:

(69) John hit Mary.

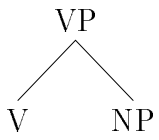
I assume, for simplicity, that the learner has information (perhaps gleaned from deductions based on information from its extra-linguistic environment) that *John* is the AGENT, *Mary* is the PATIENT and *hit* is a verb which relates an AGENT and a PATIENT. Let us consider the subtree which contains the verb and object. We put aside, without loss of generality, functional projections like AgrO.

One class of representations of the VP in (69) violates the word order in the example by placing the object before the verb. This class of representations will be rated lower by the fitness metric in (68) than those which correctly represent the word order due to the penalty imposed by the violations term, $v(x_j)$.[14]

We turn our attention to representations where the word order is correctly represented, that is, where the verb precedes the object. This class will include a minimal representation like the following:

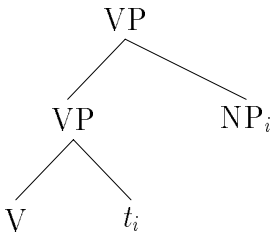
[14]Note, in particular, that the mirror images of (70), (71) and (72) may be available. Their ratings will be uniformly degraded by the violations term; otherwise, the discussion of these examples would exactly parallel the discussion of (70), (71) and (72), below. We can therefore safely omit the discussion of the mirror images without loss of any essential information.

(70)



We assume that the tree in (70) shows regular Case and thematic relations as well as a simple $\overline{\text{X}}$-structure. The class will also include representations like the following:

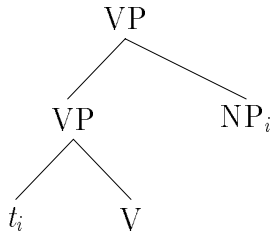
(71)



The structure in (71), like the one in (70), captures the basic word order facts correctly and obeys any expectations the learner might have about the basic Case and thematic relations in the language. In particular, everything that must be said in the description of the representation in (70) must also be said in the description of (71). However, the complete description of (71) must include information about the displacement of NP$_i$; this must include the locus of NP$_i$ as well as the site of the trace, information which cannot be completely recovered from first principles which can be prewired into the reference machine. Thus, the complexity of (71) with respect to our reference machine $\mathcal{A}$ should be than the complexity of (70). In other words, if P_{70} is the program which computes the representation in (70) and P_{71} is the one which computes the representation in (71), then $K_{\mathcal{A}}(P_{71}) > K_{\mathcal{A}}(P_{70})$. It follows that the fitness function, which will minimize the Kolmogorov complexity relative to the reference Turing machine, will give a higher fitness rating to (70) than to (71), thus preferring any grammar which allowed the minimal representation in (70).

Note that there is a further class of representations which superficially obey the word order of the example while including either violations of the learner's grammar or added complexity in the description. Such an example is shown in (72):

(72)



Suppose that P_{72} is the program that outputs the tree in (72). Because the description of

the representation in (72) includes all the information about chains that must be included in the description of (71), $K_{\mathcal{A}}(P_{72})$ is at least as great as $K_{\mathcal{A}}(P_{71})$.

In general, if our architecture is optimized to exploit headedness, then minimizing the descriptive complexity of representations will tend to prefer representations where movement has been avoided, since properties of the displaced constituent cannot be locally predicted; instead, representations must be made more complex to include non-local relationships. Thus, the descriptive complexity term in (68) mimics the principle of economy (Chomsky, 1995). Movement can, however, be forced if local relationships like thematic marking, Case and agreement conspire to force it. For example, movement can be forced if failure to move creates a violation of some local requirement. Notice that the violation term in (68) captures this, as well as filtering out other mismatches between surface word order and representations. In short, the violation term in (68) can be viewed as a generalization of the principle of greed (see, again, Chomsky, 1995). I do not wish to suggest that the fitness metric in (68) can supplant the principles of greed and economy, only that the fitness metric embodies a learning theoretic reflex of these principles. One might suggest, however, that greed and economy reflect general computational principles that pervade cognitive systems, learning and parsing included.

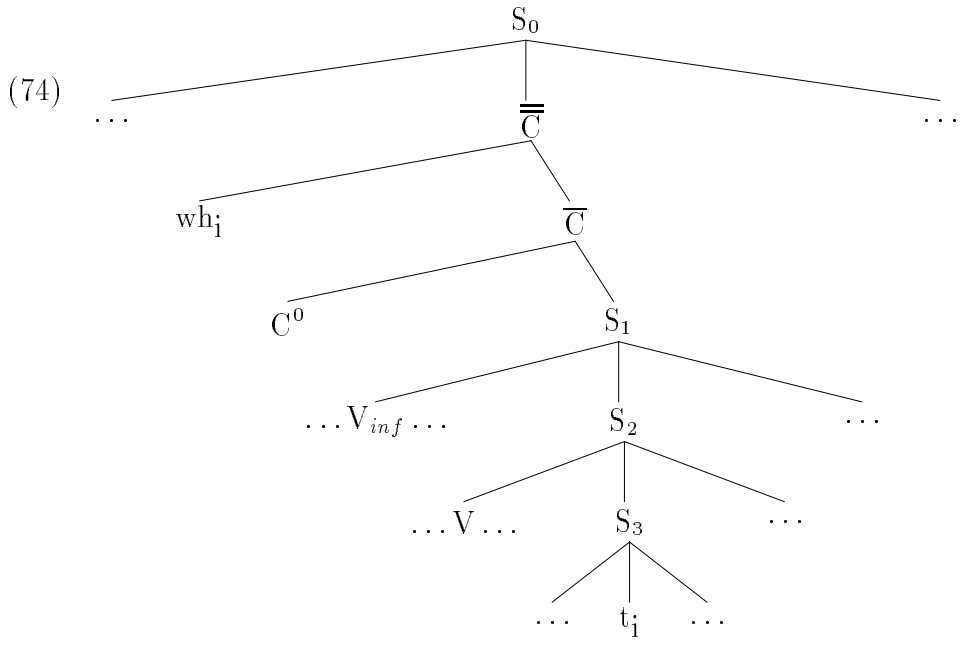
4 Constrained Induction

I have so far assumed that classical parameter setting provides the most pexplanatory approach to language learnability. Our analysis, however, has concentrated on statistical properties of the input text that underlie learning; nothing in our discussion has actually hinged on parameter setting or even parameters, except insofar as they encode linguistic variation. In this section, I will abandon the classical framework of parameter setting and explore some implications of an inductive approach to language learnability.

Our goal, here, is to explore further the connection between linguistically variable properties expressed in the input text and the learning process. In the preceding sections I argued that parameters, if they exist, have a tight connection to statistical properties of the input text. Put tersely, linguistically variant properties that cannot be expressed on simple structures cannot be learned, given the computational bounds on the learner. Standard P&P theory, however, has had nothing to say about the relationship between variation and texts. Notice, however, that arbitrary complexity could be built into a parameter. Nothing rules out parameters of the form in example (73):

(73) A special form of agreement is used in the main clause when the main verb governs an embedded wh-question, the specifier of which $\overline{\text{A}}$-binds a trace contained in the complement of a raising verb which occurs in a clause which is a complement to an infinitive non-factive verb.

This parameter could only receive expression on a structure of relatively great depth:



Our discussion, above, gives us sound reasons to suppose that the parameter in (73) would be difficult to learn since examples corresponding to structures of the form shown in (74) are unlikely to occur in natural texts.

I should note that a learnable parameterized system exists wherein (73) can be set. For example, imagine a system where (73) is the only parameter. If structures of the form in (74) were the most likely to occur, then the parameter could be easily set. Notice, though, that the architecture would then be optimized around structures like (74). This follows from the discussion of the relationship between Kolmogorov complexity and probability in section 2.5, above. Indeed, the descriptive complexity of an item provably converges to its entropy in the limit (see the discussion of (60) on page 41, above). But a text where structures of the form in (74) have a high probability would look quite different from the text actual learners encounter in nature. Thus, we have sound empirical reasons for supposing that (73) is not a possible variant property, given Universal Grammar. Statistical properties of real texts can, then, be used as a check against which theories of language variation can be tested.

Thus, we must show how learning is connected to texts. One conceptual problem with P&P theories is that they seem to loosen the connection between learning and the real world. Inductive learning is less prone to this criticism since induction is grounded in experience. I do not deny that an unconstrained inductive learner would stand little chance at converging to the target grammar. A constrained form of inductive learning, however, might provide a suitable framework for the acquisition of linguistically variable properties. Let us turn now to a discussion of such a framework.

The first step in defining an inductive procedure is to specify a grammatical formalism

over which induction takes place. For present purposes, I will select a tree rewriting framework, Tree Adjoining Grammar (TAG) which can formally model operations like "adjoin" (Chomsky, 1995). Furthermore, since the TAG formalism allow grammars that are mildly context-sensitive in their generative capacity, it seems to accord well with what is known about the computational power required to parse natural languages.

Recall that traditional $\overline{X}$-theory is a refinement on context free phrase structure rules:

(75) A context free phrase structure grammar is a quadruple $G = \langle V, \Sigma, S, P \rangle$ where:

(i) V and Σ are finite sets such that $V \cap \Sigma = \emptyset$, V is the set of nonterminal symbols and Σ is the set of terminals;

(ii) $S \in V$ is the start symbol;

(iii) P is a finite set of production rules of the form:

$$\alpha \longrightarrow \omega$$

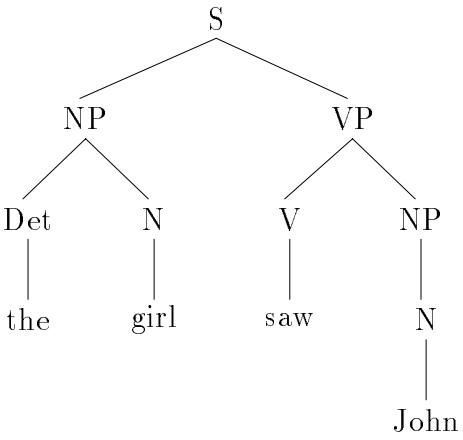
where *alpha* is a single non-terminal symbol and ω is a string of symbols drawn from $\{V \cup \Sigma\}^*$.

The definition in (75) are of the familiar form shown in (76):

(76) S $\longrightarrow$ NP VP
NP $\longrightarrow$ (Det) N
VP $\longrightarrow$ V (NP)
N $\longrightarrow$ John
N $\longrightarrow$ girl
Det $\longrightarrow$ the
V $\longrightarrow$ saw
V $\longrightarrow$ slept

These rules can be used to define a set of trees:

(77)



Unlike context-free grammars, TAGs are a tree rewriting system. While context-free rules rewrite nonterminal symbols, TAGs take trees as their fundamental units. One might, for example, specify the set of trees shown in figure 6. The trees themselves can be combined by two operations: *substitution* and *adjunction*.

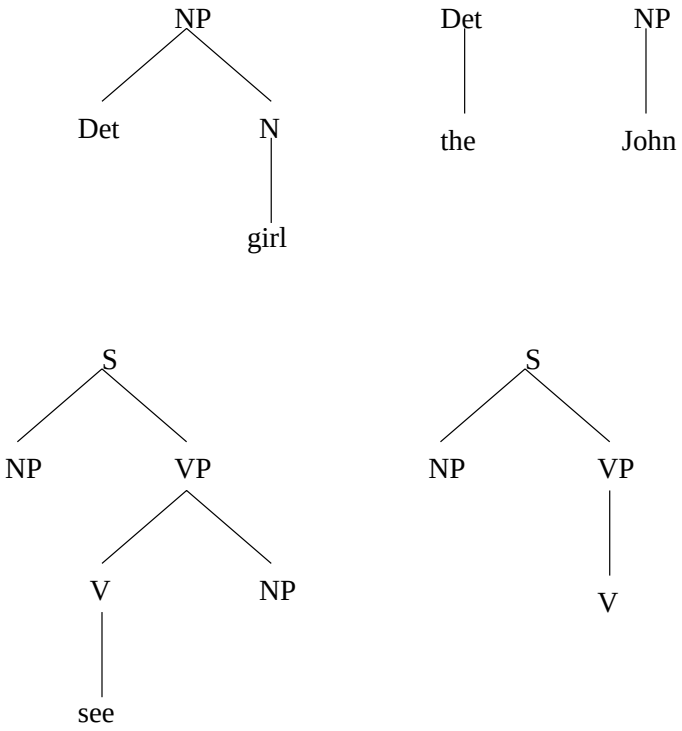
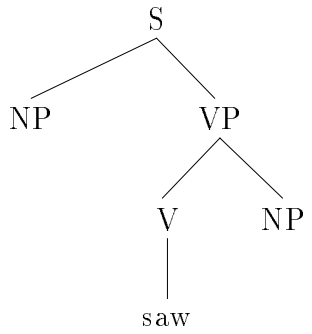


Figure 6: A Set of Basic Trees for a Simple Tree Adjoining Grammar

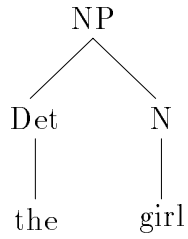
Substitution is the simple identification of the root node of one tree with a frontier node of another tree. For example, consider the trees in (78a) and (78b). The root node

of (78b) can be identified with the subject NP node of (78a) to derive the structure in (78c):

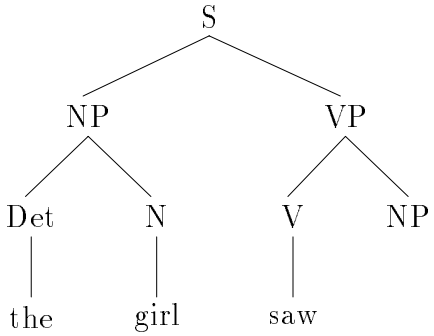
(78) a.



b.

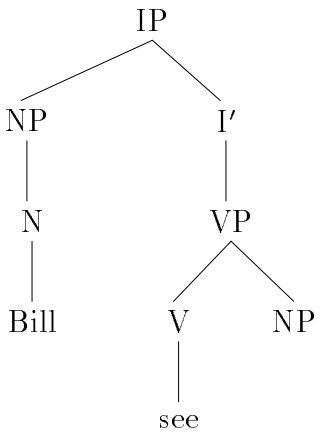


c.



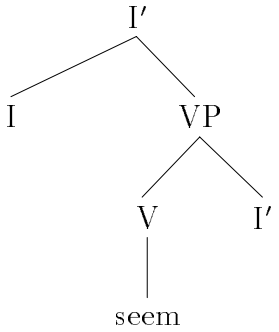
Adjunction, like substitution, takes two trees as input and returns a single tree. Unlike substitution, however, adjunction involves the non-frontier nodes of one of the trees. In order to show how adjunction works, it is best to work through an example. Consider the tree in (79):

(79)



Notice the node I′ which, according to the usual interpretation of $\overline{\text{X}}$-theory, is not well-formed since it lacks a head. I will assume that primary trees need not be headed. Similarly, consider the tree in (80):

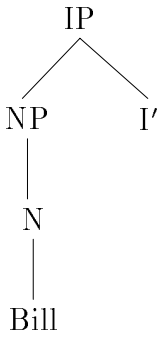
(80)



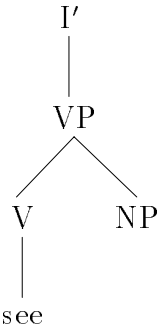
This tree is rooted by I′, a token of which also occurs as a frontier node.

We can combine the trees in (79) and(80) by dividing (79) into two subtrees by "cutting" it at the I′ node. This operation creates two trees, one of which has an I′ node as a leaf and the other of which has an I′ node as its root:

(81) a.

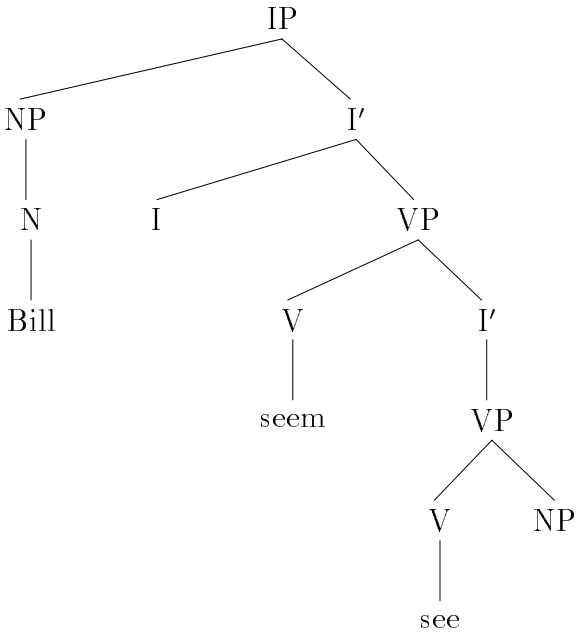


b.



The two trees in (81) can be combined with the tree in (80) by identifying the root I′ of the tree in (80) with the leaf I′ node in (81a) and identifying the root I′ node of the tree in (81b) with the leaf I′ node of (80). This produces the structure shown in (82):

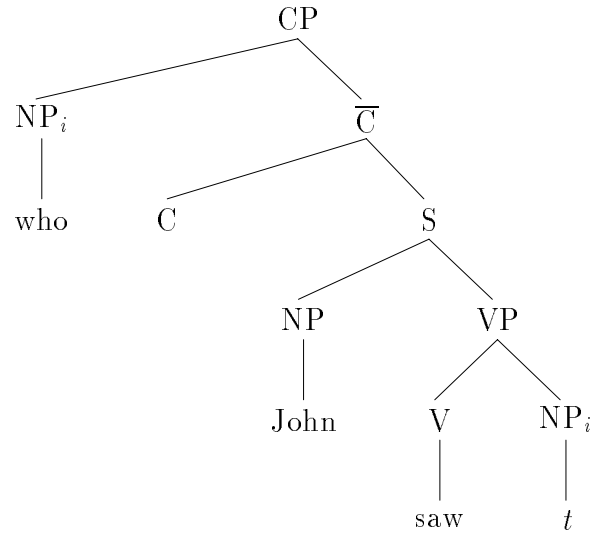
(82)



Further adjunction and substitution rules could adjust the structure in (82) to include the infinitive marker *to* and so forth. Notice that thematic relations can be defined relative to a single basic tree with adjunction and substitution preserving those relations. In general, then, we can hypothesize that words are associated with basic trees in the lexicon.

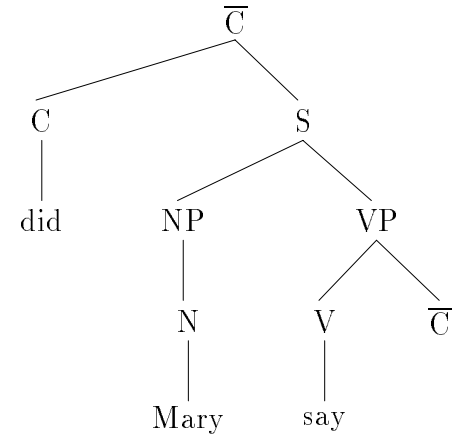
The system presented above contains only the minimal operations of Tree Adjoining Grammar. We assume, further, that nodes can be annotated for obligatory adjunctions. Consider, for example, the tree in (83):

(83)

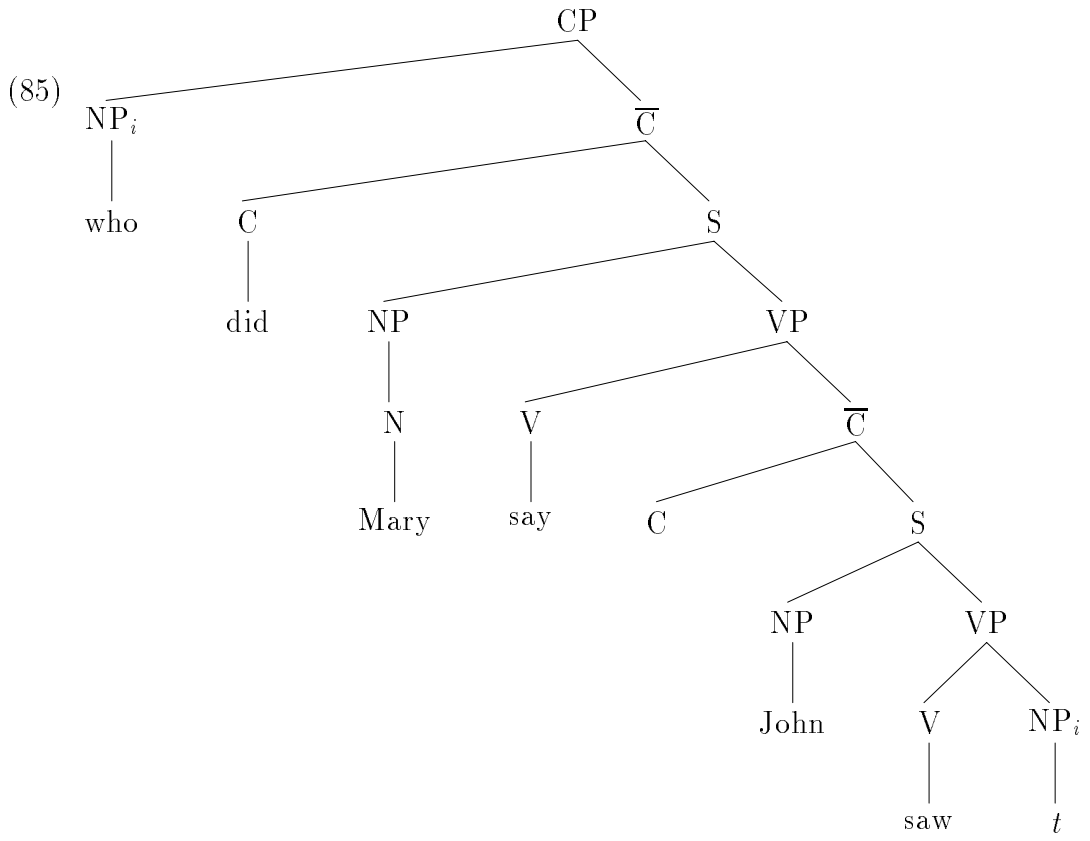


The structure in (83) could be combined with the structure in (84):

(84)



The result of adjoining (83) and (84) is shown in (85):



The tree in (85) has some unusual features from the point of view of standard $\overline{\text{X}}$-theory. In particular, the verb *say* takes a non-maximal projection as its complement. For the sake of discussion, let us accept this result.[15] The TAG formalism presented here has been greatly stripped down. More elaborated formalisms can be found in Joshi (1987), Vijay-Shanker (1987), Weir (1988) or Rambow (1994). Kroch (1987), Frank (1992) and Frank & Kroch (1995) discuss the linguistic ramifications of the TAG formalism in the analysis of NP-movement, WH-movement, bounding theory and binding theory.

For our purposes, it is sufficient to consider the two basic operations of adjunction and substitution without worrying about more complex operations or the details of the proper linguistic treatment of particular constructions. We will require only that the learner return a set of basic trees, when exposed to an appropriate text (see section 1.2). We can divide the general learning problem into two sub-problems:

[15] See Kornai & Pullum (1990) for some discussion on the interpretation of heads and maximal projections, however.

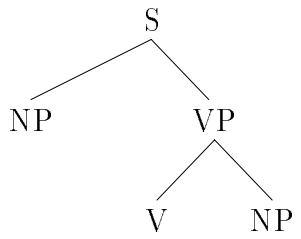
(86) a. *The Parsing Problem*
Given a particular string drawn from the input text, what is the best parse (or set of parses) for that string?

b. *The Generation Problem*
Given a set of parsed examples, what is the best set of basic trees for generating that set?

The problems in (86) are so interconnected that it might seem that they are equivalent. They are, however, logically distinct and one can, upon reflection, imagine two devices, one of which proposes parses for strings while the other device searches for optimal generators (that is, basic trees) for the set proposed by the former device. Clearly, the operation of the former will be affected by the hypotheses of the latter. Similarly, the latter is constrained to consider only the representations proposed by the former device. In other words, the states of the two devices are interrelated so that we can construe the learning problem as a form of *co-evolution*.

Consider, first, the parsing problem in (86a). Here the problem is to find an optimal representation for a string; that is, the processor must find a representation of the input string that minimizes both grammatical violations and descriptive complexity in the sense discussed above. We can assume that this simple parsing device is equipped with a simple set of basic trees as shown in figure 7. The trees in figure 7 show various possible dependencies between elements in the grammar. Not all of the dependencies will be exemplified by the target grammar so that some of the initial primary trees will go unused and will, as a consequence, atrophy (see below). Furthermore, the trees in figure 7 are context free in the sense that they represent only immediate dominance; if the initial state of the learner allows for only substitution, then the initial grammar will be a context free grammar. The learner's initial grammar could include information beyond the level of context free structures. For example, the structure of a transitive verb could be stipulated as in (87):

(87)



I have suppressed morphological information in (87), although certain functional heads could be associated with the lexical projections. Part of the task of the learner would be to associate the nodes of the basic trees with the appropriate feature information. For the sake of discussion, I will put aside such considerations.

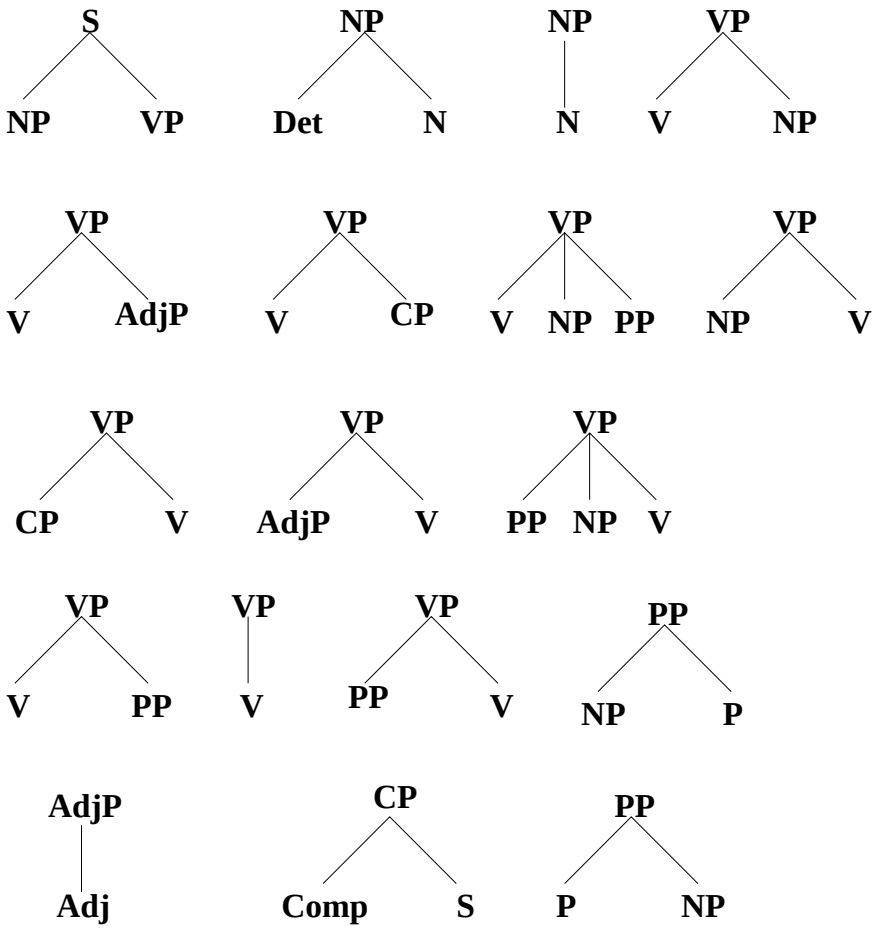
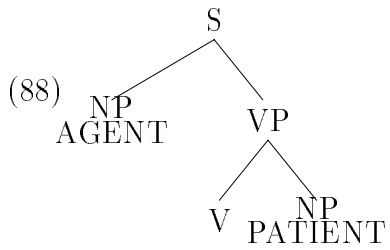


Figure 7: A Fragment of the Initial State of the Parsing Device

In addition, the learner might have certain biases about the association of semantic roles with grammatical functions. Thus, the structure in (87) might be further articulated to include such information:



The structure in (88) includes information about the arguments associated with the main verb. We could assume, following a great deal of recent work in language acquisition (see, in particular, Grimshaw, 1981; Pinker, 1984; Clark, 1982, among many others),

that the learner's initial set of basic trees includes a number of default semantic role assignments. The exact assignments may require a more sophisticated analysis of the situation presented to the learner; see Dowty (1991) for one such proposal.

Let us suppose, then, that the learner is presented with an input string, σ, plus some representation of the situation, a *sensorium*, s. For the sake of fidelity, let us stipulate that the sensorium is a complete representation of the learner's information state at the time of its exposure to the input sentence. This information state would consist of sensory inputs, beliefs and so on, which are available to the learner at the time of utterance of s (see Gärdenfors, 1988; Landman, 1991 for a discussion of information states in the intended sense). The sentence/sensorium pair described here is conceptually distinct from the (b, s) pairs of Wexler & Culicover (1980). The latter are a pairing of the deep structure with a semantic representation, while sentence/sensorium pairs a surface phonetic representation with a state of the learner. Viewed in this light, the learner must at least partially solve a number of daunting problems before syntactic learning can begin:

(89) a. The input, σ, must be segmented into words.

b. Words must be assigned denotations.

We will have nothing to say about the segmentation problem described in (89a). The denotation problem in (89b) must be approached using a number of different learning strategies. For example, the acquisition of concrete nouns might require the association of perceptual constants in the sensorium with segmented words. Thus, reliable presence of "daddy" qualia in the environment might lead to an association with the word *daddy*. There is essentially nothing new in this account, which owes its essentials to John Locke.

The acquisition of denotations for words of other grammatical categories might well require other strategies, including inferences based on their syntactic distribution. For example, knowledge that *toy* is noun and *broke* is a verb would trigger the learner to conclude that *some* has a determiner denotation in the following:

(90) Some toy broke.

since only determiners could combine with nouns to create a phrase of the correct type. Similarly, the learner could use an item's distribution in a text to observe that it both occurred in verb-like positions and was highly predictive of a following noun phrase, and, thus, was likely to be a transitive verb.

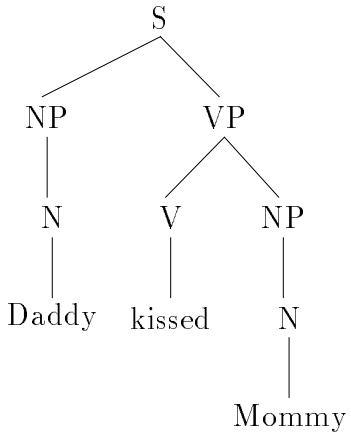
Consider the following simple example:

(91) Daddy kissed Mommy.

Assuming that the learner has reliably determined that *Daddy* and *Mommy* behave se-

mantically like names, the parsing device could at least assign structure to these two elements. The assumption that (91) is a clause would further assign the entire string to the category S. The set of basic trees in figure 7 would warrant the following structure:

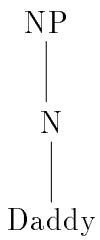
(92)



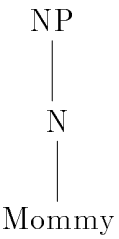
The grammar may well make other potential representations available. These alternative representations would be rated for the goodness of fit according both to the need to minimize violations of grammatical requirements and to the need to minimize the descriptive complexity of the representation, as we have seen above.

The best representation produced by the parsing device would then be turned over to the component which searches for the best generators for the input text (see (86) on page 57, above); let us refer to this component as the *shredder*. Since both *Daddy* and *Mommy* are names, the shredder can return at least the following basic trees:

(93) a.



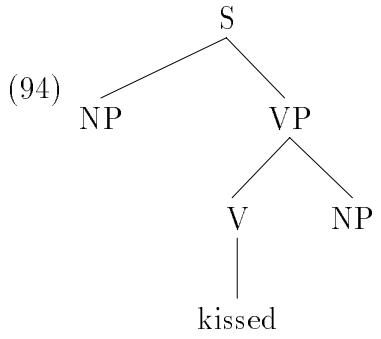
b.



If the trees in (93a) and (93b) are removed from the tree in (92), the remaining structure

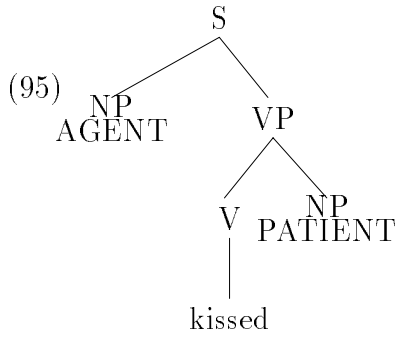
is as shown in (94):

(94)



Further inferences would associate thematic roles with the NPs in (94), analogous to the structure shown in (88), above:

(95)



The structure in (95) could be returned to the parsing device to be used in the grammar.

With continued exposure to the input text, more and more basic trees will be added to the grammar used by the parser. One would expect that the grammar would continue to expand, although some hypotheses might be abandoned by the learner early as not leading to productive hypotheses about the target grammar. Clearly, some mechanism for weeding out unused basic trees is needed. A number of such methods exist (see, for example, Holland, 1986); such systems tend to involve competitive bidding with reinforcement of successful rules:

(96) a. Individual rules have a certain amount of capital available to them.

b. In order to apply in a derivation, rules must post bids; the highest bidder is permitted to apply.

c. Bids are subtracted from the successful rule's store of capital.

d. Successful derivations receive reinforcement in the form of capital.

e. The reinforcement associated with a successful derivation is distributed among the rules that applied in that derivation.

The system described in (96) is essentially a "bucket brigade" algorithm as described by Holland (1986). In our case, rules are actually the basic trees in a TAG. The trees will be associated, initially, with a certain amount of capital that can be spent in bidding to apply in a derivation. Successful trees will, in other words, pay to be included in a derivation. The resulting representation will be evaluated by the fitness metric, described above, which prefers representations which both minimize violations of grammatical principles and minimize the descriptional complexity of the representation relative to Universal Grammar. The best representation will be reinforced by the system and the reinforcement is distributed among the trees that were used in the derivation. Basic trees which participate in unsuccessful derivations will tend to lose capital over time and fall into bankruptcy, at which point they are removed from the grammar. Basic trees which tend to participate in successful derivations will also tend to gain capital (or at least break even); they are therefore allowed to remain in the grammar. The basic form of the learning device is shown in figure 8. The learning device in figure 8 consists of two stages. In the parsing phase the learner attempts to fit an input string with a linguistic representations, as discussed above. Its output is regulated by the revised fitness metric on page 46 and the operation of its internal grammar is regulated by the bucket brigade algorithm described above. Its output is passed to the shredder. The shredder takes representations proposed by the parser and breaks them down into basic trees, which are then returned to the parsing device for use in its internal grammar. The global goal is to find the optimal set of basic trees that will cover the input text.

Let us turn, now, to a more precise characterization of the shredder. Recall that TAGs allow for two basic operations: substitution and adjunction. As a result, when given an input tree the shredder will have an enormous number of potential basic tress which it can return. The shredder could break the tree a number of ways at the frontier. The trees in (97b) and (97c) show two possible factorizations of the tree in (97a):

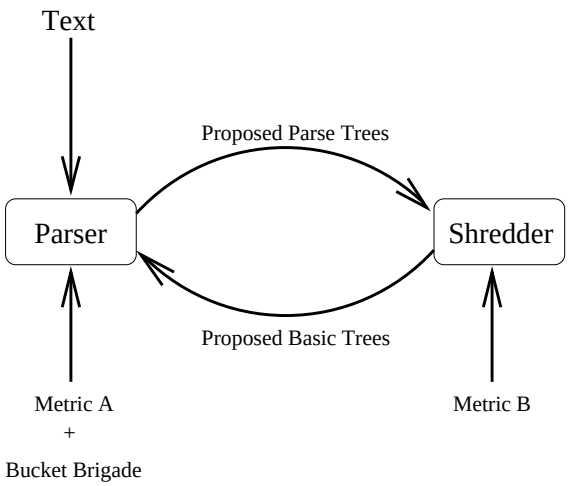
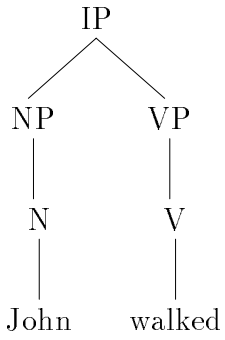
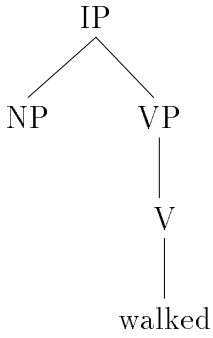


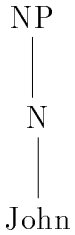
Figure 8: A Tandem Learning Device

(97) a.

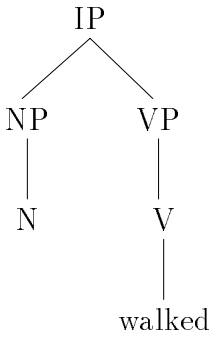


b.





c.



Corresponding to the adjunction operation, the shredder could also break the tree internally in a large number of ways internally. We would expect, then, that the number of possible factorizations of a parse tree would grow exponentially as a function of the number of nodes in the original tree. The shredding operation must be constrained to rule out an enormous number of formally possible factorizations

Before turning to a discussion of the constraints on factorization, let us consider the general character of metric B in figure 8. In principle, the shredder could propose each tree built by the parser as a basic tree; this would be the identity factorization of the parse tree. If this were to happen, the size of the grammar would be a linear function of the length of the input text, assuming that repetitions in the text do no create new basic trees. But then the grammar would be as complex as the input text, which would defeat the purpose of having a grammar. Notice that the grammar can be viewed as a form of data compression over texts. This view suggests that the learner attempts to minimize the number of basic trees in its grammar relative to the length of the text. This, in turn, implies that the learner attempts to maximize the probability of occurrence of each basic tree relative to the text.

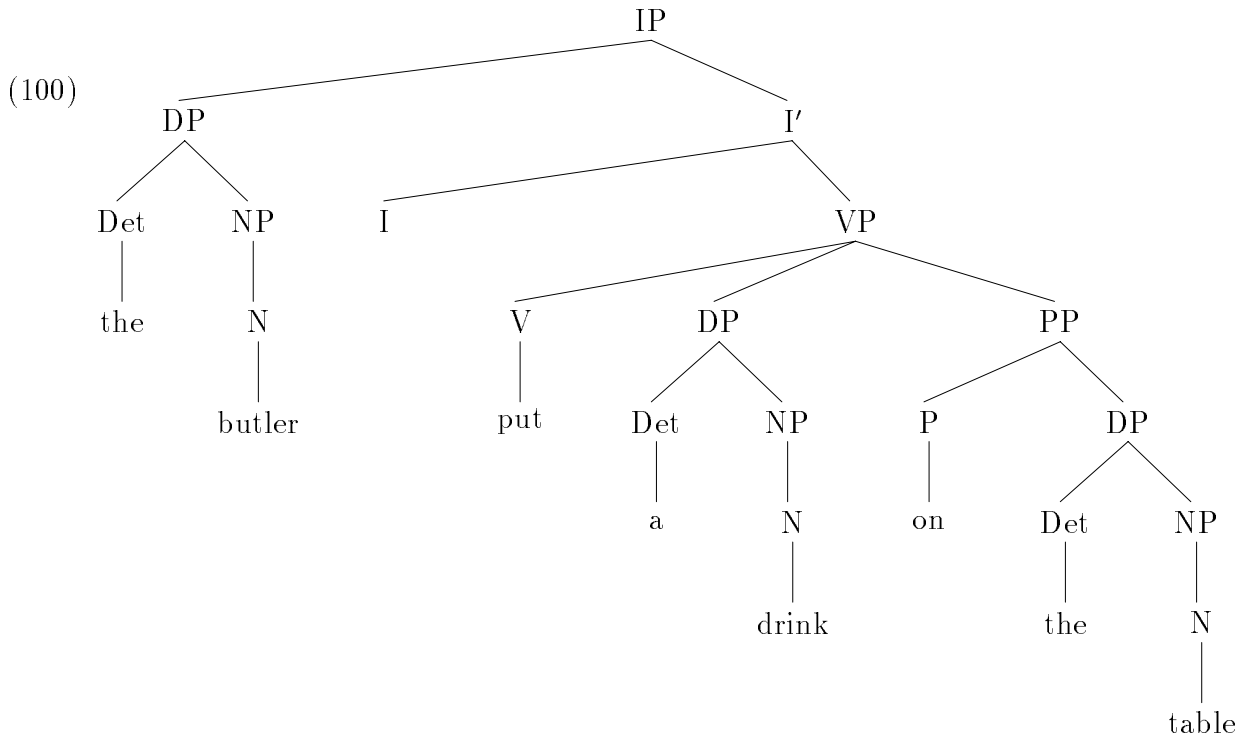
(98) *The Compression Principle*
The output grammar is the smallest set of basic trees that will cover a minimal text for the language, given a fixed lexicon.

Although the Compression Principle provides a general requirement on the learner, it is of little use in specific cases where a single tree must be factored into a set of basic trees. Let us turn, now , to some heuristics which would limit the possible factorizations of particular trees. The first heuristic is suggested by the example in (97), above. The problem is that there is a large number of ways to break a tree down such that it can be reassembled via substitution alone. Notice, however, that many of the breaks will be at non-maximal projections, as shown in (97c). One heuristic would constrain the shredder to factor the tree at maximal projections:

(99) *The Substitution Constraint*
Break input trees at maximal projections.

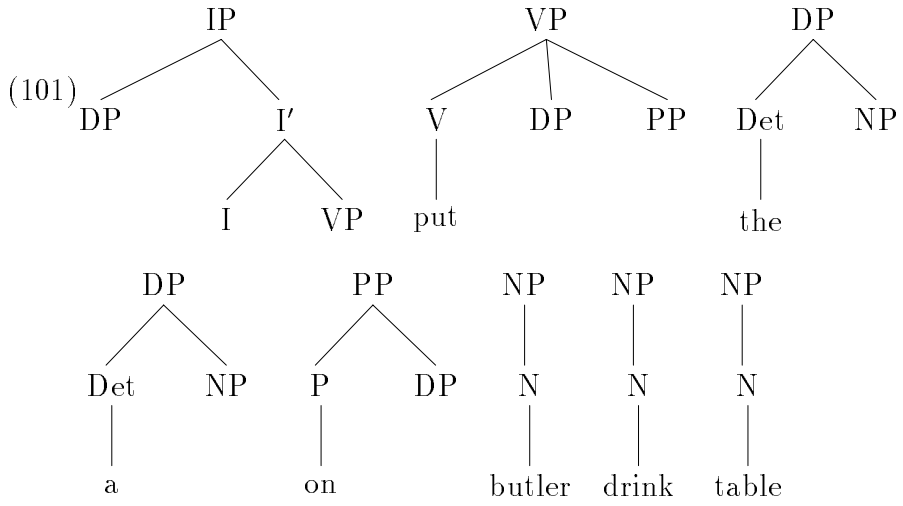
To see the intended effect of (99), consider the representation in (100), below:

(100)



According to the Substitution Constraint in (99), the tree in (100) can be factored into the following subtrees:

(101)

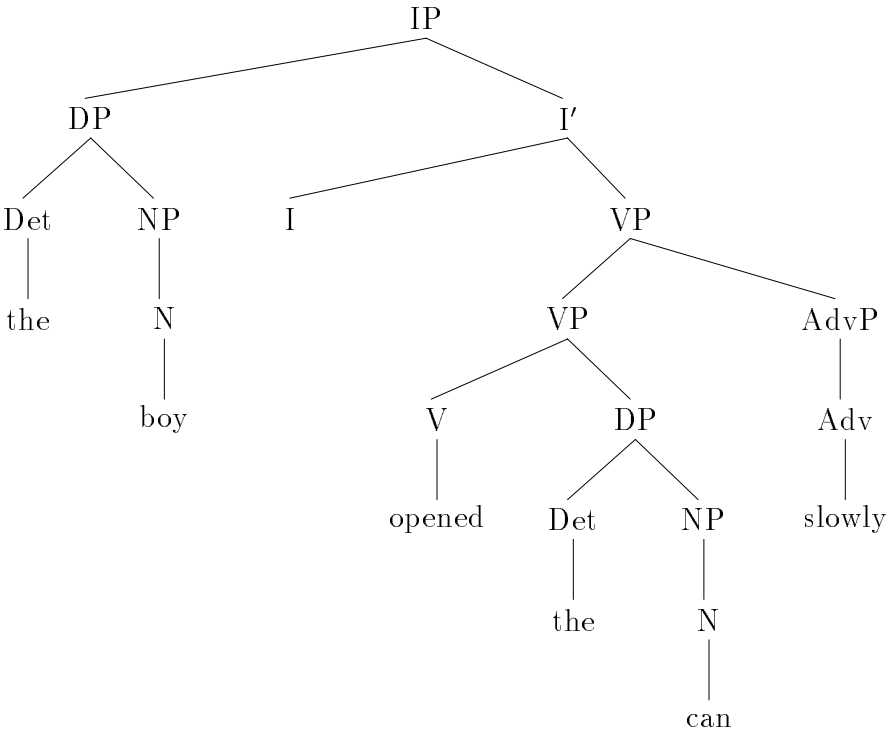


If we further require that breaks occur only at maximal projections, then the set in (101) is an exhaustive factorization of the representation in (100). Furthermore, the trees containing lexical elements could be taken as lexical entries. Recall, however, that raising and wh-movement involve adjunction of nodes at $\overline{\text{C}}$ and $\overline{\text{I}}$, so some exception to the

Substitution Constraint will have to be made for these cases.

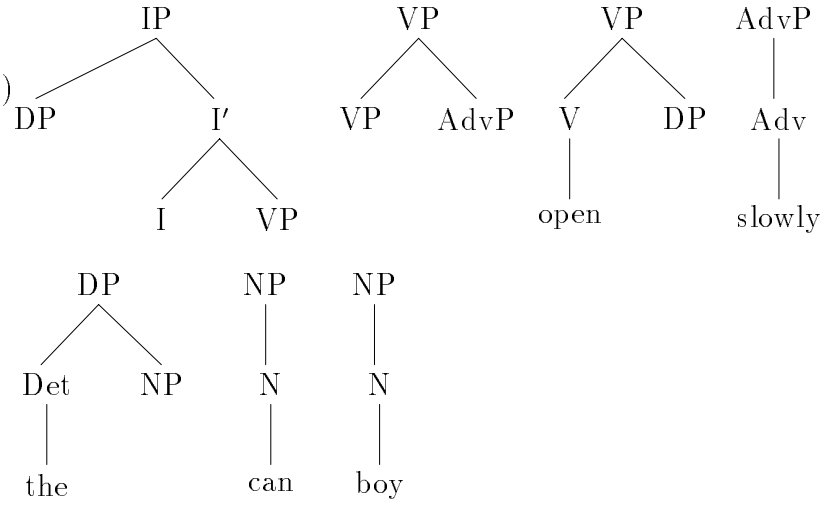
Consider, next, a clause containing an adverbial modifier, as shown in (102):

(102)



Factoring the tree in (102) according to the substitution constraint yields the following set of basic trees:

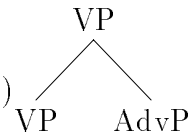
(103)



Notice, in particular, that the above factorization has permitted the discovery of the

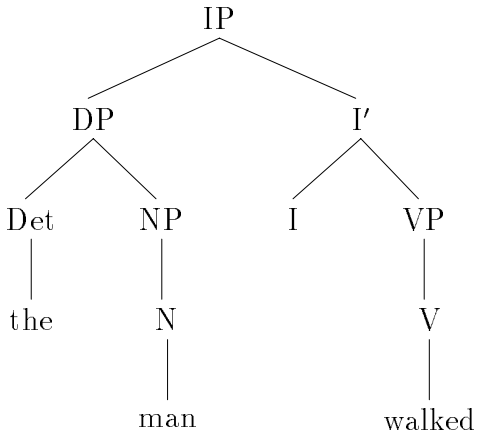
following basic tree:

(104)

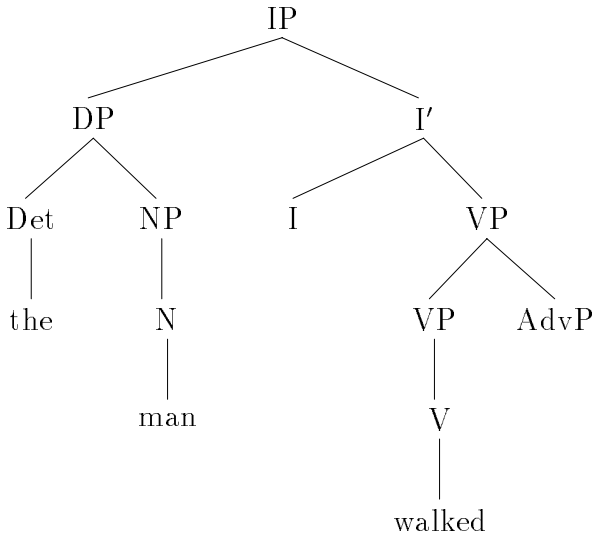


This result is interesting since it allows for the learning of some adjunction structures. Thus, the tree in (104) can combine with the tree in (105a) to form the tree in (105b):

(105) a.

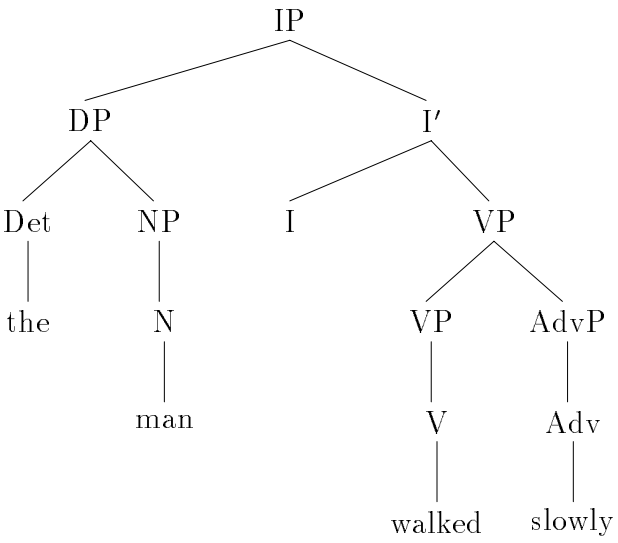


b.



Further substitution at the AdvP node would yield:

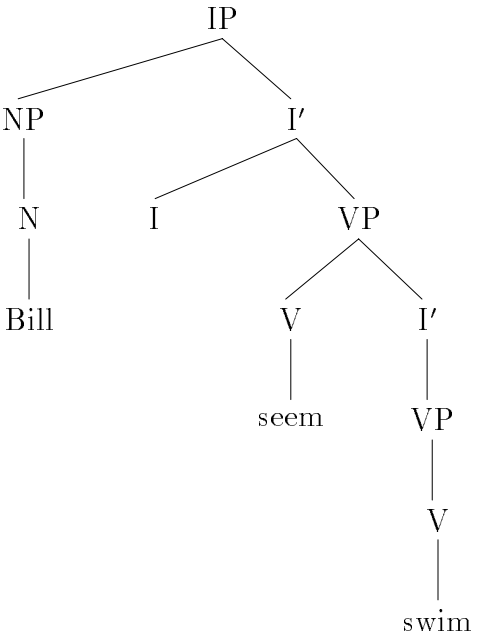
(106)



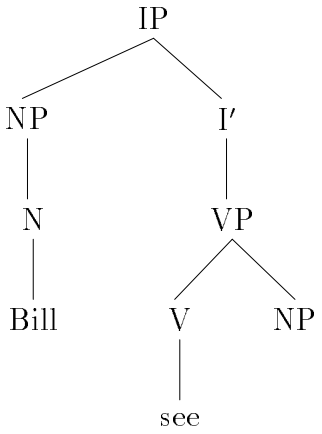
Thus, the Substitution Constraint in (99) not only reduces the search space for factoring trees but it also generalizes across both substitution and adjunction.

Let us turn, now, to the general problem of factoring trees that involve adjunction in their derivation. The basic problem is that the learning algorithm must reconstruct a connected basic tree that has been "interrupted" by another subtree. For example, the tree in (107) must be factored into the trees in (108):

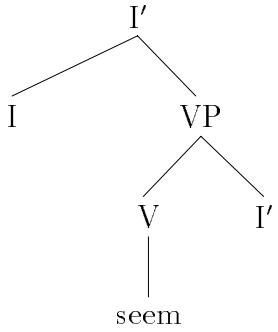
(107)



(108)a.



b.



Recall that wh-movement is treated analogously so that we can confine our discussion to raising without loss of generality.

We need a restrictive procedure that will lead to the automatic recovery of tree-internal adjunctions. Recall that adjunction can occur when one basic tree has a node in its frontier of the same category as its root node. If the root matches a node in another tree that the latter may provide a site for a tree-internal adjunction of the former tree into the latter. This suggests that a tree, T_i, may be factored into two subtrees when there is a subtree rooted at a node N_j of category A just in case N_j dominates a node of N_k also of category A. In that case, T_i is factored into two subtrees, T_m and T_n, with the following properties:

(109)a. T_m is that subtree of T_i rooted in node N_j. The node N_k occurs in its frontier, but the material dominated by N_k has been deleted.

b. T_n can be factored into two subtrees. The first consists of the root and material dominated by the root, exclusive of material dominated by N_j. The second consists of the node N_k and all material dominated by N_k. T_n, then, is the first subtree with the second subtree inserted in the position of node N_j. Let us refer to this tree as the complement of T_i relative to T_m, or $T_i - T_m$.

Notice that the subtrees in (108) can be factored from the tree in (107) by finding subtrees corresponding to those described in (109a) and (109b), since these properties simply describe the inverse of the adjunction operation.

Nevertheless, the procedure outlined above is still overly general since it will provide too many candidate basic trees. One need only consider multiple clausal embeddings, prepositional embeddings or noun phrase embeddings to see this:

(110) a. [$_{\mathrm{IP}}$ John [$_{\mathrm{I'}}$ thinks [$_{\mathrm{IP}}$ Mary [$_{\mathrm{I'}}$ said [$_{\mathrm{IP}}$ Bill [$_{\mathrm{I'}}$ was late]]]]]]

b. John put the money [$_{\mathrm{PP}}$ in [the jar [$_{\mathrm{PP}}$ on the shelf]]]

c. [$_{\mathrm{DP}}$ a friend of [$_{\mathrm{DP}}$ [$_{\mathrm{DP}}$ John's] cousin]] arrived.

The above procedure will allow for basic trees corresponding to the following:

(111) [$_{\mathrm{I'}}$ thinks [$_{\mathrm{IP}}$ Mary [$_{\mathrm{I'}}$ said I′]]]

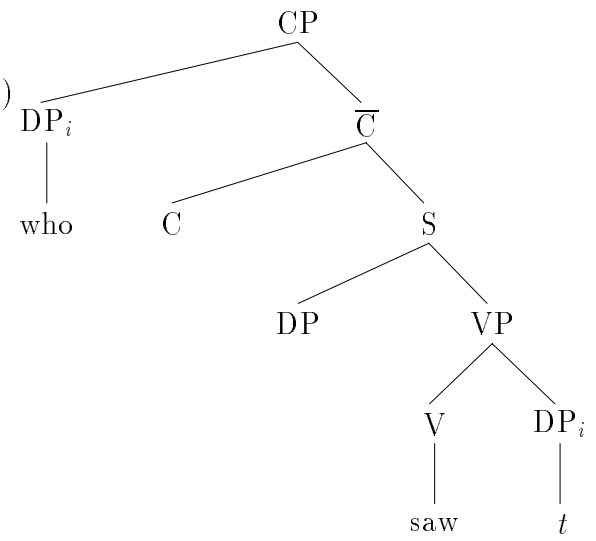
The tree in (111) includes material from the domains of two distinct lexical heads: *think* and *say*. Although the tree in (111) is too particular to survive the bucket brigade algorithm, the shredder should not propose it as a basic tree in order to minimize the complexity of the tree factoring process. Notice that, in general, tree-internal adjunction tends to separate dependent elements in one of the basic trees. As a constraint on the shredder, we might require the following:

(112) *The Adjunction Constraint*
The output of the shredder must consist of a single lexical head and its dependents.

The constraint in (112) will prevent the shredder from proposing the tree in (111) when it factors the representation in (110a).

The heuristic in (112) prefers the factorization of trees into semantic and morphological domains. Consider, for example, the minimal tree for wh-movement:

(113)



The wh-phrase occurs in the same basic tree as the head that it is dependent on for its semantic, morphological and grammatical functions. Similar remarks apply to the basic trees needed for raising structures.

Both the Adjunction Constraint in (112) and the Substitution Constraint in (99) prefer factorizations of parse trees into minimal basic trees, in accordance both with locality constraints long noted in the generative literature and with the approach to complexity described earlier. The heuristic constraints on the shredder, as well as the fitness functions on both the parser and the shredder drive the learner to discover minimal basic trees of the type that can be learned from extremely simple input texts of the type noted in our discussion of the adult input to children, above. In particular, these constraints interact to guarantee that the Boundedness of Parameter Expression (see (23) on 19, above) is satisfied; notice, however, that nothing in our system requires parameters in the sense familiar from *P&P* theories. We can reformulate the BPE as follows:

(114) All linguistic variation must be expressed on a syntactic syntactic structure of descriptive complexity bounded by a constant U

Given the relationship between descriptive complexity and probability (see section 2.5, above), the principle in (114) implies that learnable variation will occur with sufficient frequency in the input text. This implies the inductive counterpart of the Frequency of Parameter Expression (see page 17).

The conspiracy between complexity and the search heuristics provide a restrictive account of possible language variation that is based on induction as opposed to parameter setting. We hope that this approach will encourage a serious reconsideration of the role of induction in grammar learning.

References

1. Brill, E. & S. Kapur (1993). "Information-theoreitc solution to parameter setting". ms. University of Pennsylvania.

2. Brown, R. (1977). "Introduction" in C. Snow, & C. Ferguson (eds) *Talking to Children: Language Input and Acquisition.* Cambridge: Cambridge University Press.

3. Chomsky, N. (1973). "Conditions on transformations" in S.R. Anderson and P. Kiparsky (eds) *A Festschrift for Morris Halle.* New York: Holt, Rinehart and Winston.

4. Chomsky, N. (1995). *The Minimalist Program.* Cambridge, MA: The MIT Press.

5. Clark, E. (1982). "The young word maker: a case study of innovation in the child's lexicon." in E. Wanner & L. Gleitman (eds) *Language Acquisition: The State of the Art.* Cambridge: Cambridge University Press.

6. Clark, R. (1992). "The Selection of Syntactic Knowledge" *Language Acquisition*, 2, pp. 83-149.

7. Clark, R. (1994). "Kolmogorov complexity and the information content of parameters" IRCS Report 94-17. Institute for Research in Cognitive Science, University of Pennsylvania.

8. Clark, R. & I. Roberts (1993) "A Computational Approach to Language Learnability and Language Change," Linguistic Inquiry 24: 299 - 345.

9. Cover, T. & J. Thomas (1991). *Elements of Information Theory.* John Wiley & Sons, Inc., New York.

10. Dowty, D. (1991). "Thematic proto-roles and argument selection". *Language.* 67, pp. 547-619.

11. Frank, R. (1992). *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives.* PhD Thesis, University of Pennsylvania.

12. Frank, R. & A. Kroch (1995). "Generalized transformations and the theory of grammar." *Studia Linguistica.* 49, pp. 103-151.

13. Gärdenfors, P (1988). *Knowledge in Flux.* Cambridge, MA: The MIT Press.

14. Gibson, E. & K. Wexler (1994). "Triggers". *Linguistic Inquiry*, 25, pp. 407-454.

15. Gleitman, L. & E. Wanner (1982). "The state of the state of the art" in E. Wanner & L. Gleitman (eds) *Language Acquisition: The State of the Art.* Cambridge: Cambridge University Press.

16. Grimshaw, J. (1981). "Form, function, and the language acquisition device" in C. L. Baker & J. McCarthy (eds) *The logical problem of language acuqisition.* Cambridge, MA: The MIT Press.

17. Holland, J. (1986). "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems". in R. Michalski, J. Carbonell & T. Mitchell (eds) *Machine Learning: An artificial intelligence approach. Vol II.* pp. 593-623. Los Altos: Morgan Kaufmann.

18. Joshi, A. (1987). "An introduction to Tree Adjoining Grammars." in A. Manaster-Ramer (ed) *Mathematics of Language*, pp. 87-115. Amsterdam: John Benjamins.

19. Kapur, S. (1992). *Computational Learning of Languages.* PhD Thesis, Cornell University.

20. Kapur,S. & R. Clark. (in press). "The automatic construction of a symbolic parser via statistical techniques" in J. Klavans & P. Resnik (eds) *The Balancing Act.* Cambridge, MA: The MIT Press.

21. Kornai, A. & G. Pullum (1990). "The X-Bar theory of phrase structure". *Language*, 66, pp. 24-50.

22. Kroch, A. (1987). "Subjacency in a Tree Adjoining Grammar" in A. Manaster-Ramer (ed) *Mathematics of Language.* Amsterdam: John Benjamins.

23. Landman, F. (1991). *Structures for Semantics.* Dordrecht: Kluwer Academic Publishers.

24. Li, M. & P. Vitányi (1993). *An Introduction to Kolmogorov Complexity and its Applications.* New York: Springer-Verlag.

25. Papadimitriou, C. H. (1994). *Computational complexity*, Addison-Wesley, Reading, MA.

26. Pinker, S. (1984). *Language learnability and language development.* Cambridge, MA: Harvard University Press.

27. Rambow, O. (1994). *Formal and computational aspects of natural language syntax.* PhD Thesis, University of Pennsylvania.

28. Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*. New York, McGraw-Hill.

29. Vijay-Shanker, K. (1987). *A study of Tree Adjoining Grammars*. PhD Thesis, University of Pennsylvania.

30. Weir, D. (1988). *Characterizing mildly context-sensitive grammar formalisms*. PhD Thesis, University of Pennsylvania.

31. Wexler, K. & P. Culicover (1980). *Formal Principles of Language Acquisition*. Cambridge, MA: The MIT Press.