LEARNING STRUCTURAL DESCRIPTIONS OF GRAMMAR RULES FROM EXAMPLES

Robert C. Berwick
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This paper describes a LISP program that can learn English syntactic rules. The key idea is that the learning can be made easy, given the right initial computational structure: syntactic knowledge is separated into a fixed interpreter and a variable set of highly constrained pattern-action grammar rules. Only the grammar rules are learned, via induction from example sentences presented to the program. The interpreter is a simplified version of Marcus's parser for English [1], which parses sentences without backup. The currently implemented program acquires about 70% of a simplified core grammar of English. What seems to make the induction easy is that the rule structures and their actions are highly constrained: there are only four actions, and they manipulate only very local parts of the parse tree.

1. INTRODUCTION

An important goal of modern linguistic theory is to show how learning a grammar can appear to be so easy, given the poor quality of the data children receive. This paper reports on a currently running LISP program which, by computationally embodying some theories of transformational grammar, can learn syntactic rules in the manner of Winston's blocks world program [2]. The program proceeds by examining example sentences to modify its descriptions of grammar rules that make up part of its knowledge about language.

The key idea is that learning syntactic transformations is easy, given the right initial computational structure. This program uses as initial structure a simplified version of Marcus's PARSIFAL [1], a parser for English which is an interpreter for grammar rules of a particularly simple production rule form. The basic operation of the interpreter is taken as fixed, corresponding to an intial set of computational abilities. Only grammar rules are learned. Because the rules themselves are so simple, and the operation of the interpreter so constrained, bugs have a diameter-limited location. Further, the parser itself is strictly deterministic; that is, already-built portions of the parse tree are assumed correct, and there is no backup. As shown below, these assumptions are crucial in the operation of the learning algorithm.

More specifically, the Marcus interpreter uses the following data structures: A parse tree, a syntactic representation of the input sentence. The lowest, right-most node of the tree under construction is called the **current active node**, denoted C. A buffer of three (to five) cells that holds words from the input sentence or as yet not-completely analyzed phrases. Phrase structure rules that are used to turn on and off logically grouped sets of grammar rules (for example, the rule S-->NP+VP would first activate all grammar rules that start sentences, then turn off that group and activate noun phrase rules). The phrase structure control system was designed by Shipman [3]. Production rules (also called grammar rules) of the form: **IF** *<pattern>* **THEN** *<action>*. Each *<action>* does the actual work of building the parse tree, attaching words or phrases from the buffer onto the parse tree, moving new words into the buffer, and so forth. *<Patterns>* determine if the given action is to fire; if the pattern given in a grammar rule matches the pattern in the buffer, the specified *<action>* takes place.(Patterns use common lexical features like *Noun phrase, singular* or *Verb, transitive.*)

----

*This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643, and in part by NSF Grant MCS77-04828.

The learning program acquires only the patterns and actions of the grammar rules. One of the accomplishments of this research has been to simplify the original Marcus parser to a point where there are only four valid actions: **ATTACH** first buffer item to C; **SWITCH** first and second buffer items; **INSERT** a specific lexical item into the first buffer slot; and **INSERT-TRACE** into first buffer slot. (*Traces* are not further discussed in this paper but function as in Chomsky's theory; see Fiengo [4] for discussion.)

Learning proceeds by induction on the *<patterns>* and *<actions>*, but with an important constraint: children (and this program) do not appear to receive negative data examples on what is not a sentence (see Brown and Hanlon [5] and Anderson [6] for discussion). On the other hand, children (and this program) do appear to receive reinforcement on what is a **semantically meaningful** sentence. Therefore, the current program does assume a lexicon and selectional restrictions on the phrase-structure categories (for example, that *Mary* is a noun and can fit under a noun phrase). More advanced versions of the program will probably have to assume a known case-frame representation for the input sentence given (Fillmore [7]), but this has not yet been found to be necessary; a recent result obtained by Wexler [8] proves mathematically that, given a transformational grammar to be learned and only surface sentences as input, a recursive learning procedure for the grammar does *not* exist (this was shown by Gold [9]), but that such a procedure *does* exist if surface sentences are paired with some representation of the underlying meaning of the sentence.

2. THE LEARNING PROCEDURE

The learning program starts with an interpreter, a lexicon, simple phrase-structure rules, selectional restrictions, but **no** grammar rules. The program is then given input sentences to parse. If it gets stuck in a parse-- if no current rule patterns match or if all current rules cause selectional errors-- then the program attempts to build a new grammar rule that will apply at that point. It does this by trying each of its possible actions in turn: **attach**, **switch**, **insert**, **insert-trace**. (This ordering was found empirically.) The first action that succeeds in satisfying the selectional restrictions is saved along with the current machine state (buffer plus current active node) as the pattern; this becomes the new rule. If no possible generated rule has worked, the active phrase structure rule is assumed to be optional. Finally, rules with common **actions** within a phrase-structure group have their **patterns** continually generalized via intersection.

3. AN EXAMPLE: AUXILIARY-INVERSION

Suppose that at a certain point the program has all and only the grammar rules necessary to build a parse tree for **Mary did hit the ball**. The program now gets as input, **Did Mary hit the ball?** No rule currently known can fire, for in the phrase structure packet S activated at the beginning of a sentence, the only rules have the pattern [=Noun Phrase][=Verb], and the buffer holds the pattern [**=Did: auxerb**][**=Mary: Noun**]. A new rule must be written, so the program tries each of its possible rule actions in turn. Attach fails because of selectional restrictions; **did** can't be attached as a noun phrase. But **Switch** works, because when the first and second buffer positions are switched, and the buffer now looks like [**=mary**][**=did**], an existing rule for parsing declarative sentences can match. The rest of the sentence is parsed as if it were a declarative. Finally, the **switch** rule is saved along with the current buffer pattern as a trigger for the next case of auxiliary inversion. It is crucial to notice that the debugging is strictly local: the error is assumed to lie exactly where the error first occurred, and not in some other rule. At most one new rule is added or one old rule modified with each example sentence, a kind of incremental debugging that is analagous to Sussman's *debugging almost right programs* [10]. In this regard it is important to point out that Wexler has proved [8] that local debugging is apparently a necessary condition for the learning of a transformational grammar.

The currently implemented LISP version of this procedure has acquired about 70% of a "core-grammar" of English originally developed for the Marcus parser, as well as some new rules; acquired rules include unmarked-order, auxiliary inversion, imperative, simple there-insertion, to-infinitive, do-support, and some passives. On the other hand, rules for parsing the complicated complement structure of English have yet to be learned, nor is it clear how they might be. Future work will consider a straightforward way to learn the phrase structure rules themselves, by generalizing templates of phrase structure rules according to Chomsky's X-bar theory (Jackendoff [11]). The relationship between the local debugging constraints assumed by the learning procedure and those constraints found necessary by Wexler [8] will also be investigated.

ACKNOWLEDGEMENTS

Special thanks are of course due to Mitch Marcus, whose thesis and advice forms the foundation of this work, and to the linguistic expertise of Candy Sidner and Beth Levin.

REFERENCES

[1] Marcus, M., "A Computational Account of Some Constraints on Language," in proceedings of *Theoretical Issues in Natural Language Processing* (July 1978; TINLAP-2), pp. 236-246.

[2] Winston, P., "Learning Structural Descriptions from Examples," in P. Winston, editor, *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975.

[3] Shipman, D., and Marcus, M., "Towards Minimal Data Structures for Parsing," *Proc. IJCAI-79*, Tokyo, Japan, 1979.

[4] Fiengo, R., "On Trace Theory," *Linguistic Inquiry* 8:1 (1977), pp. 35-61.

[5] Brown, R., and Hanlon, C.,"Derivational Complexity and Order of Acquisition in Child Speech", in J.R. Hayes, ed., *Cognition and the Development of Language*, New York: John Wiley and Sons, 1970.

[6] Anderson, J. R., "Induction of Augmented Transition Networks", *Cognitive Science*, 1, (1977) pp.125-157.

[7] Fillmore, C.J., "The Case for Case", in Bach, E., and Harms, R.T., editors, *Universals in Linguistic Theory*, New York: Holt, Rinehart, and Winston, 1968.

[8] Wexler, K., "Transformational Grammars are Learnable from Data of Degree Less than or Equal to Two," *Social Sciences Working Papers*, ¢139, School of Social Sciences, University of California, Irvine, 1977.

[9] Gold, E.M., "Language Identification in the Limit," *Information and Control*, 10 (1967) pp. 447-474.

[10] Sussman, G., *A Computational Model of Skill Acquisition*, American Elsevier, 1975.

[11] Jackendoff, R., *X-bar Syntax: A Study of Phrase Structure*, Cambridge, Mass: MIT Press, 1977.