NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

An Overview of Parse Completion

Technical Report PCG-10

Steve Nowlan

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213 U.S.A

July 1, 1987

Running head: Parse Completion

Acknowledgments

This research was supported by the Computer Science Division, Office of the Naval Research, and DARPA under Contract No. N00014-85-C-0678. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

006.3
C28p
No.10
c.3

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) PCG-10	5. MONITORING ORGANIZATION REPORT NUMBER(S) Same as Performing Organization
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Information Sciences Division Office of Naval Research (Code 1133)
6c. ADDRESS (*City, State, and ZIP Code*) Departments of Psychology and Computer Science Pittsburgh, PA 15213	7b. ADDRESS (*City, State, and ZIP Code*) 800 N. Quincy St. Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS p400005ub201/7-4-86 PROGRAM ELEMENT NO: N/A; PROJECT NO.: N/A; TASK NO.: N/A; WORK UNIT ACCESSION NO: N/A
11. TITLE (*Include Security Classification*) An Overview of Parse Completion	
12 PERSONAL AUTHOR(S) Steve Nowlan	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM 86Sep15 TO 91Sep15	14. DATE OF REPORT (*Year, Month, Day*) 87 July 1 15. PAGE COUNT 11
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*) Induction, machine learning, parse completion, grammatical inference, grammar induction

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

See reverse side.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION	
22a NAME OF RESPONSIBLE INDIVIDUAL Alan Meyrowitz	22b TELEPHONE (*Include Area Code*) (202) 696-4302	22c. OFFICE SYMBOL 1143

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Hierarchical knowledge structures are pervasive in Artificial Intelligence, yet very little is understood about how such structures may be effectively acquired. One way to represent the hierarchical component of knowledge structures is to use grammars. The grammar framework also provides a natural way to apply failure-driven learning to guide the induction of hierarchical knowledge structures. The conjunction of hierarchical knowledge structures and failure-driven learning defines a class of algorithms, which we call *Parse Completion* algorithms. This paper presents an overview of the results derived from a theoretical exploration of this class of algorithms.

1 Introduction

Hierarchical knowledge structures are pervasive in artificial intelligence systems. Classic examples include semantic networks [21], scripts [24], and frames [14] and their descendants, which are employed in current knowledge representation technology. Planning and plan recognition systems [7, 6, 23] make extensive use of procedural hierarchical structures. Despite this pervasiveness, very little is understood about how such knowledge structures may be effectively acquired.

We focus on executable hierarchies: those that represent control strategies, plans or procedures [27, 9]. Grammars can provide a uniform representation for such hierarchical control structures. The hierarchy of control is implicit in the rules of the grammar, but becomes explicit in the derivation tree[1] for a particular string.[2] In this context, one can regard the rewrite rules of a grammar as a way of transforming some goal into a group of sub-goals. The idea of creating sub-goals to hierarchically solve a complex problem goes back at least to GPS [17], and is the basis for several models of cognitive architecture [13, 1, 2, 26]. Different types of grammars correspond to different classes of control structure; the activation of a sub-goal may depend only on the presence of its parent goal, or it may also depend on the concurrent activation of other goals with different parents (*i.e.* context free vs. context sensitive grammars). An effective algorithm for inducing grammars may also be a powerful tool for learning hierarchical control structures from experience [27].

Mitchell [15] highlights the importance of biases in induction problems. Failure (or impasse) driven learning [25, 22, 27] is a particular bias that can make many induction problems more tractable. This bias favours using the existing knowledge structures as much as possible to solve a problem. When an impasse is reached, it adds just enough additional control knowledge to bridge the gap and complete the solution. This bias may have psychological validity as well as practical utility [28].

The combination of failure-driven learning and grammar induction yields a class of algorithms

---

[1]The term derivation tree is synonymous with parse tree, and in this context is equivalent to a trace of a procedure. The derivation tree can be a general graph for context-sensitive grammars.

[2]Throughout this paper we will use the term string to represent the end product of a derivation using a grammar. The string is not necessarily a string of characters, but may equally well be a sequence of operations for performing some task.

which we have referred to as *Parse Completion* algorithms. In Parse Completion one attempts to build a derivation tree for some string, using existing rules of the grammar, until no existing rules can be applied. The process may be thought of as a combination of top-down and bottom-up parsing (Fig. 1). When the derivation tree is as complete as possible, new rules are added to the grammar to *fill in* any remaining gaps in the derivation tree.

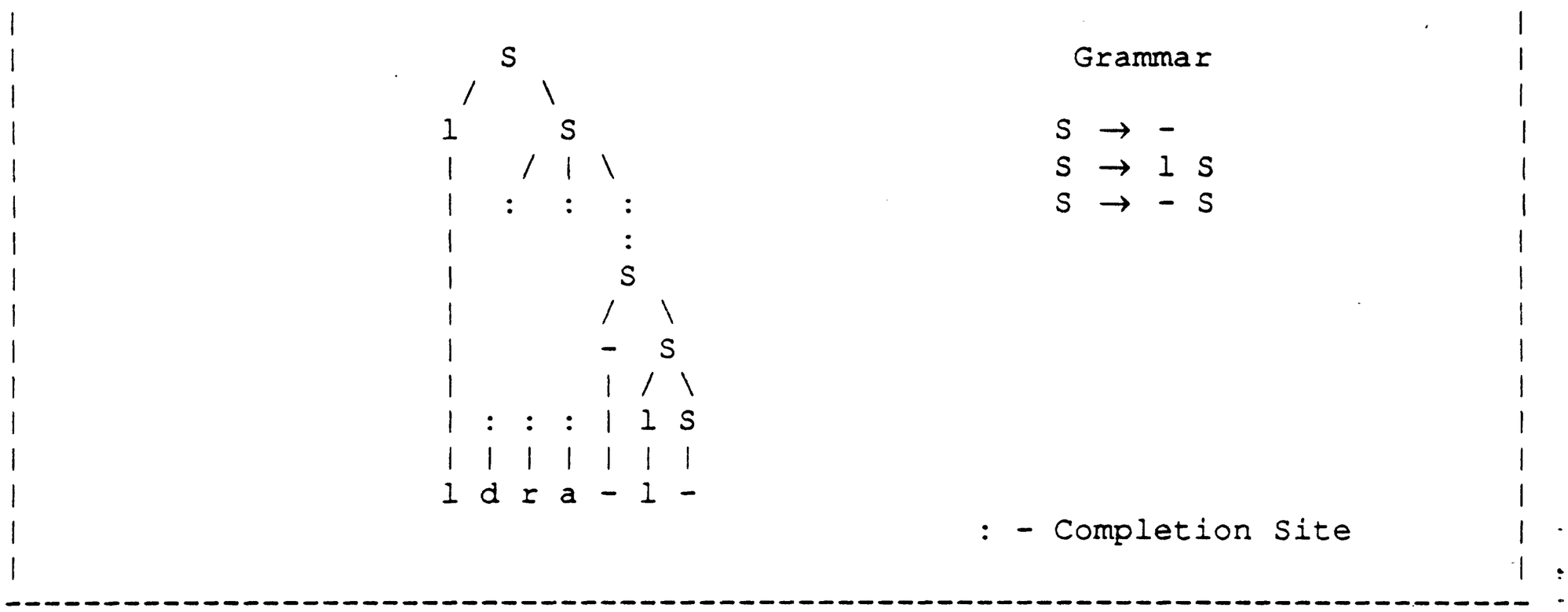

Figure 1: Incomplete derivation tree prior to Parse Completion

Parse Completion is not a new idea. Specializations of it have been used in programs that learn plans [12], procedures [27, 9], programs from examples [3, 4], and models of cognitive skills [26]. Although the idea has been used by a variety of researchers, there has been no attempt to characterize the space of parse completion algorithms, or to systematically examine where biases [15, 27] may be most effectively introduced.

This paper is concerned not just with the problem of effective grammar induction, but with the problem of *designing* an algorithm for a particular induction problem. Algorithm design is an art, and like most arts, its mechanics are not well understood. This appears to be even more true in the area of algorithms for AI problems, where the problems themselves are usually ill-defined and lack mathematical structure that may be effectively exploited.

2 The Parse Completion Design Space

Parse completion is a particular approach to induction problems. An induction problem is the discovery of expressions in some representation language (generalizations) such that each is (1) consistent with the examples and (2) preferred by learning biases. The set of expressions is partially ordered by a *more-specific-than*[3] predicate [16]. The induction problem is to discover some expression that encompasses all positive examples and no negative ones, by searching the tangled hierarchy of expressions.

The goal of parse completion is to build a complete derivation tree for some string starting from an existing grammar. If the existing grammar is powerful enough to parse the string then a complete derivation tree may be built. The interesting case occurs when the existing grammar is inadequate to parse the string. If we combine a top-down and bottom-up parse, we will produce a partial derivation tree that contains a number of *gaps* (Figure 1). Each gap becomes a *completion site*, a point at which additional rules must be added to the grammar to complete the derivation tree. The new grammar will be a generalization of an old grammar, since it will be able to parse at least one string that the old grammar could not parse.

Although the above may appear to be a tight description of an algorithm, there are in fact a wide variety of design choices to be made within this general framework. Each choice may produce an algorithm with dramatically different characteristics. We wish to explore these choices in some systematic fashion.

At each completion site there usually exist many different ways in which the grammar may be generalized to allow the derivation to continue. Each of these new grammars represents an alternate path within the tangled generalization hierarchy of grammars. Our first decision point is whether to consider all or just one of these alternate generalizations. This is a least-commitment versus most-commitment distinction: a most-commitment algorithm will select just one alternative, and continue its search in a depth first fashion, back tracking if necessary. Most grammar induction algorithms fall in this category [8, 5, 19]. A least-commitment algorithm attempts to explore all of the generalization alternatives in parallel, without committing itself to one particular path. In this sense it is more like a breadth first search. The best known example

---

[3]More-specific-than(x,y) is true iff the *denotation* of x (*i.e.*, all possible instances of x) is a subset of the denotation of y.

of a least-commitment induction algorithm is the version space algorithm [16].

Least-commitment algorithms are memory intensive compared to their most committment counterparts, and are thus regarded with disfavour for most machine learning applications. However, if the induction domain itself is ill understood, then a least-commitment algorithm can offer valuable information about this domain. If we are interested in the impact of certain design choices on an induction algorithm, then we need to know more than just the final solution obtained by an algorithm. We would also like some idea of the blind alleys explored, and those avoided. This ability to see more than just a narrow view is one advantage of a least-commitment algorithm.

There are many other design choices available. A grammar may be generalized in two different ways: by introducing new rules into the grammar, or by merging old non-terminals in existing rules. Each approach defines a partial order over the set of grammars consistent with a set of examples, and in both cases the partial order is a strict suborder of the partial order based on the predicate *more-specific-than*. The partial order defined by merging old non-terminals has been investigated elsewhere [29, 11, 20].

The parse completion algorithm provides an effective means to deal systematically with the different alternatives possible within the paradigm of generalization through the addition of rules. The approach taken is to classify rules added to a grammar in terms of the format of the right hand side (RHS). One can think of parsing a string as involving two processes. The first process partitions a string into several contiguous substrings. Each partition element is then labelled with some symbol from the grammar, either a terminal or non-terminal. The partitioning and labelling process is repeated on each partition element labelled with a non-terminal, until all elements are labelled with terminals. At each stage the number of elements in the partition and the labels assigned to each element correspond to the RHS of some rule in the grammar. Conversely, a new rule RHS can be formed by taking a partition of a string and labelling its elements (Figure 2).

We can generate a variety of different algorithms from the parse completion framework by giving specific functions for generating partitions and labelling the elements. At one extreme, we could restrict the partitioning and labelling so that only rules already existing within our grammar could be generated. Under this restriction the parse completion algorithm becomes a

Partition	Labels	Rules
(d r a S)	-	S → d r a S
(d r a) (S)	A,-	S → A S A → d r a
(d r) (a S)	A,B	S → A B A → d r B → a S
(d) (r a) (S)	A,B,-	S → A B S A → d B → r a
(d r) (a) (S)	A,B,-	S → A B S A → d r B → a

Figure 2: Partitions and labellings for the derivation tree in figure 1.
"-" represents an unlabelled partition.

simple top-down parser. With different restrictions new rules of varying power may be added to an existing grammar.

The RHS of rules may be classified according to whether they contain terminals, non-terminals that have previously appeared in the grammar, new non-terminals, and various combinations of the above. In addition, classification may be based on the length of the rules, and the order of the RHS constituents (*i.e.*, new rules may only be formed by extending the RHS of an existing rule). Using these classification schemes, a partial order of RHS formats may be defined [18]. (This partial order of RHS formats is distinct from the partial order of grammars in the generalization hierarchy mentioned previously.) The algorithm is designed so that a particular point in this partial order may be selected, or the program may be permitted to move through the partial order itself. In this latter mode, the weakest class of RHS format[4] is tried first, and stronger RHS formats are tried only if the weaker ones fail to allow the parse to succeed. In this manner the program searches automatically for useful combinations of RHS formats. RHS formats were derived for several interesting types of grammars including Left Linear, Right Linear, Greibach Normal Form, and Chomsky Normal Form [18].

[4]In this context weakest means least general, or smallest. This is in contrast to the standard logical definition of "weaker" in which weaker classes are supersets of stronger classes.

In investigating these RHS formats,[5] the allowed rule formats could be divided into those which added additional structure to the grammar, and those that served to generalize the class of strings accepted by the grammar. Additional structure is introduced by rules which contain terminals or new non-terminals. It is perhaps easiest to see this by considering the class of Right Linear grammars for a moment. There is a finite state machine corresponding to each grammar in this class. Adding a rule with only terminals on the RHS corresponds to adding a new transition from some state to the final state. Adding a rule with a new non-terminal corresponds to adding a new state to the machine. In both cases we are creating a new path in the machine from the start state to the final state *which contains no loops or repeated states*. This new path contains at least one state or one transition that is unique from any other path in the machine which contains no loops or repeated states. It is in this sense that we say structure has been added to the machine, or the corresponding grammar. On the other hand, adding a rule containing an old non-terminal corresponds to adding a transition between two existing states. We have generalized the machine in the sense that we have combined the first part of one existing set of paths with the second part of another existing set of paths. If in this process we have created a loop in the machine, then we have converted a finite language into an infinite language containing an arbitrary number of repetitions of some substring. These notions of adding structure versus generalizing the grammar can be extended to context free grammars (see [18]).

The RHS formats for Right or Left Linear grammars and Chomsky Normal Form grammars can both be shown to define a partial order over the set of grammars consistent with the examples. (Note that this is a partial order over the grammars themselves, not over the RHS formats.) Furthermore, these partial orders are well defined and finitely bounded,[6] and can therefore be used to define a version space-like structure for these grammar classes (see [18]). This structure is useful in deriving a number of properties about induction algorithms for these grammar classes.

One important result that can be derived is that under certain conditions, and given only a set

---

[5]Only Right Linear and Chomsky Normal Form were studied in detail, as these are representative of the classes of Linear and Context Free grammars. The other grammar types may be converted to one of these two forms by a simple mechanical transformation (see [10] for proofs and definitions of these grammar types).

[6]The bounds however are quite large, even for simple grammars, hence computation of the entire partial order is often not practical.

of positive example strings, a least committment induction algorithm will always converge to a single grammar in bounded time (for proofs see [18]). The key idea is that it is not possible to arbitrarily introduce rule forms which add structure and rule forms which generalize. Unless a certain minimal amount of structure is present first, the algorithm will over-generalize and fail to converge.

The important question is how much structure is necessary to prevent over-generalization. One can show that if the examples are ordered so that the shortest ones come first, and if the learner starts out by adding only rules that introduce structure, then there is a well defined point at which no further structure need be added, and one can begin to introduce rules which generalize.[7] Identifying the point at which no further structure need be added is equivalent to defining a space complexity bound for the language being induced.

The derivation of the above results also suggested that certain *felicity conditions* [26] can be defined which will also permit convergence. These conditions require the teacher to identify to the student whether a particular example is an example of a new concept in the domain, or merely a generalization of concepts the student has seen before (for details see [18]).

3 Conclusion

Our purpose was to explore the class of Parse Completion algorithms. In pursuing that purpose, we have produced a well defined design space for this class of algorithms. This design space is defined by a partial order over the RHS formats of new rules which may be added to complete a parse. One of the most interesting divisions based on RHS formats divided rules into those which added additional structure to the grammar, and those which generalized existing structure. This particular division led to the discovery of biases under which an induction algorithm can be designed which will always converge to a single, most specific, context free grammar from a finite set of positive example strings. Certain additional conditions must also be met to guarantee convergence. These conditions can be expressed as a complexity bound on the language, which uniquely identifies the point at which no additional structure needs to be added to the grammar. These conditions can also be expressed as felicity conditions, which require the teacher to distinguish examples that introduce a new concept from examples that only serve to

---

[7]These results are based on the Pumping Lemmas for regular and context free languages and are discussed in [18].

generalize existing concepts. Perhaps the most important point to be learned from this study is that a systematic attempt to understand an induction *domain* can lead to useful insights for designing induction algorithms for that domain.

References

[1] Anderson, J. R.
The Architecture of Cognition.
Harvard University Press, Cambridge, MA., 1983.

[2] Anderson, J. R.
Skill Acquisition: Compilation of Weak-Method Problem Solutions.
Technical Report ONR-85-1, Office of Naval Research, 1985.

[3] Biermann, A. W.
On the Inference of Turing Machines from Sample Computations.
Artificial Intelligence (3):181-198, 1972.

[4] Biermann, A. W.
The Inference of Regular Lisp Programs from Examples.
IEEE Transactions on Systems, Man, and Cybernetics SMC-8(8), August, 1978.

[5] Biermann, A. W. and Feldman, J. A.
A survey of results in grammatical inference.
In S. Watanabe (editor), *Frontiers of pattern recognition.* Academic Press, New York, 1972.

[6] Fahlman, S.
A Planning System for Robot Construction Tasks.
Artificial Intelligence (5):1-49, 1974.

[7] Fikes, Hart, and Nilsson.
Learning and Executing Generalized Robot Plans.
Artificial Intelligence (3):251-288, 1972.

[8] Fu, K. and Booth, T.
Grammatical Inference: Introduction and survey.
IEEE Transactions on Systems, Man, and Cybernetics (5):95-111, 1975.

[9] Genesereth, M. R.
The role of plans in intelligent teaching systems.
In *Intelligent Tutoring Systems.* Academic Press, New York, 1982.

[10] Hopcroft, J. E. and Ullman, J. D.
Introduction to Automata Theory, Languages, and Computation.
Addison Wesley, Reading, MA., 1979.

[11] Horning, J. J.
A study of grammatical inference.
Technical Report CS-139, Stanford University, Computer Science Department, 1969.

[12] Knoblock, C. and Carbonell, J.
Learning by Completing Plans.
Technical Report, Computer Science Department, Carnegie-Mellon University, In Preparation.

[13] Laird, J., Rosenbloom, P. S. and Newell, A.
Chunking in Soar: The anatomy of a general learing machine.
Technical Report, Computer Science Department, Carnegie-Mellon University, 1985.

[14] Minsky, M.
A framework for representing knowledge.
In P. Winston (editor), *The psychology of computer vision*, pages 211-277. McGraw-Hill, New York, 1975.

[15] Mitchell, T. M.
The need for biases in learning generalizations.
Technical Report CBM-TR-117, Rutgers University Computer Science Department, 1980.

[16] Mitchell, T. M.
Generalization as search.
Artificial Intelligence (18):203-226, 1982.

[17] Newell, A., Shaw, J. C., and Simon, H. A.
Report on a general problem-solving program for a computer.
In *Information Processing: Proceedings of the International Conference on Information Processing*, pages 256-264. UNESCO, Paris, 1960.

[18] Nowlan, S. J.
A Design Space for Parse Completion Algorithms.
Technical Report, Computer Science Department, Carnegie-Mellon University, In Preparation.

[19] Osherson, D., Stob, M. and Weinstein, S.
Systems that Learn.
MIT Press, Cambridge, MA, 1985.

[20] Pao, T. W.
A solution of the syntactical induction-inference problem for a non-trivial subset of context-free languages.
Interim Report 69-19, Moore School of Electrical Engineering, University of Pennsylvania, 1969.

[21] Quillian, M. R.
Semantic Memory.
Semantic Information Processing.
MIT Press, Cambridge, Mass., 1968.

[22] Riesbeck, C. K.
Failure-driven reminding for incremental learning.
In *Proceedings of IJCAI 7*, pages 115-120. 1981.

[23] Sacerdoti, E.
Planning in a Hierarchy of Abstraction Spaces.
In *International Joint Conference on Artificial Intelligence 3*, pages 412-422. 1973.

[24] Schank, R. C., and Abelson, R. P.
Scripts, plans, goals and understanding.
Lawrence Erlbaum, Hillsdale, N.J., 1977.

[25] Schank, R.
Dynamic Memory: A Theory of Learning in Computers and People.
Cambridge University Press, Cambridge, 1982.

[26] VanLehn, K.
Human procedural skill acquisition: Theory, model and psychological validation.
In *Proceedings of AAAI-83*. Los Altos, CA, 1983.

[27] VanLehn, K.
Learning one subprocedure per lesson.
Artificial Intelligence 31(1):1-40, January, 1987.

[28] VanLehn, K.
Towards a Theory of Impasse Driven Learning.
In H. Mandel and A. Lesgold (editors), *Advances in Intelligent Teaching Systems Research*. Academic Press, New York, In Preparation.

[29] VanLehn, K. and Ball, W.
A Version Space Approach to Learning Context Free Grammars.
Machine Learning Journal , In Press.