Cirrus: an automated protocol analysis tool

KURT VANLEHN VANLEHN@A.PSY.CMU.EDU
STEVE GARLICK GARLICK@A.PSY.CMU.EDU

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

In P. Langley (Ed.) Proceedings of the Fourth International Workshop on Machine Learning. Los Altos, CA: Morgan-Kaufman. 1987. pp. 205-217.

Abstract

Cirrus is a tool for protocol analysis. Given an encoded protocol of a subject solving problems, it constructs a model that will produce the same protocol as the subject when it is applied to the same problems. In order to parameterize Cirrus for a task domain, the user must supply it with a problem space: a vocabulary of attributes and values for describing spaces, a set of primitive operators, and a set of macro-operators. Cirrus' model of the subject is a hierarchical plan that is designed to be executed by an agenda-based plan follower. Cirrus' main components are a plan recognizer and a condition inducer. The condition inducer is based on Quinlan's ID3. Cirrus has potential applications not only in psychology but also as the student modelling component in an intelligent tutoring system.

1. Introduction

Cirrus is belongs to the class of machine learning programs that induce a problem solving strategy given a set of problem-solution pairs. Other programs in this class are Lex (Mitchell, Utgoff, & Banerji, 1983), ACM (Langley & Ohlsson, 1984), LP (Silver, 1986), and Sierra (VanLehn, 1987). The induction problem solved by these programs is: given a problem space (i.e., a representation for problem states and a set of state-change operators) and a set of problem-solution pairs, find a problem solving strategy such that when the strategy is applied to the given problems, the given solutions are generated.

The primary difference among the program in this class is the types of problem solving strategies they are designed to induce. Cirrus is unique in that the strategies it produces are hierarchical plans that are designed to be run on an agenda-based plan interpreter. Section 2 describes Cirrus' representation of strategies and discusses the motivations for choosing it.

A secondary differences among strategy induction programs is the amount of information contained in the problem-solution pairs. Some programs (e.g., ACM) expect to be given only the final state while other programs (e.g., LP) expect to be given a sequence of states leading from the initial to the final state. Cirrus expects to be given a sequence of primitive operator applications leading from the initial to the final state. Section 3 describes this aspect of Cirrus' design. This amount of information seems larger than that given to most strategy induction programs. However, because Cirrus assumes a hierarchical plan-following strategy, there may be many changes to the internal control state (i.e., manipulations of the agenda) between adjacent primitive operator applications. These changes are not visible to Cirrus, so it must infer them. The amount of information given to Cirrus in the problem-solution pairs is really not so large after all, given the difficulty of its induction task.

Another major difference among strategy induction programs is their intended application. There seem to be four major applications for strategy induction programs: (1) as models of human skill acquisition, (2) as a way to augment the knowledge base of an expert system, (3) as the student modelling component of an intelligent tutoring system, and (4) as a tool for data analysis. Usually, a program is useful for more than one type of application. Cirrus is intended to be used for the third and fourth applications, i.e., student modelling and data analysis, but it may perhaps prove useful for the others as well. In section 5, we show how Cirrus

has been employed to construct models of students solving arithmetic problems.

Cirrus' intended applications share the characteristic that the input to Cirrus can be noisy, because it is generated by humans solving problems. People often make slips (Norman, 1981), wherein they perform an action that they didn't intend. A classic example of a slip is an arithmetic error of the type that infest one's checkbook. The existence of slips in the problem-solution pairs means that Cirrus must use induction algorithms, such as Quinlan's (1986) ID3 algorithm, that are fairly immune to noisy data. As far as we know, no other strategy induction program has been used with noisy data.

The main components of Cirrus are a plan recognizer and the ID3 concept inducer. The plan recognizer is a modification of a parser for context-free grammars. Although these components are not particularly novel, their arrangement into a useful system is. Thus, this paper will concentrate on describing the design choices that, we believe, make Cirrus a useful tool. Most of these concern the type of data given Cirrus and the representation of problem solving strategies. These points are discussed in sections 2 and 3. The plan recognizer and concept inducer are discussed in section 4. The performance of Cirrus is illustrated in section 5. Section 6 summarizes and indicates directions for futher research.

2. The representation of student models

Cirrus' design assumes that subjects' problem solving strategies are hierarchical. That is, some operations are not performed directly, but are instead achieved by decomposing them into subgoals and achieving those subgoals. For instance, answering a subtraction problem can be achieved by answering each of its columns. A concommitant assumption is that the same goal may be achieved by different methods under different circumstances. For instance, answering a column can be achieved either by taking the difference between the digits in the column when the top digit in the column is larger than the bottom digit, or by first borrowing then taking the difference between the digits when the top digit is smaller than the bottom digit. These two assumptions, that procedural knowledge is hierarchical and conditional, are common to most current theories of human skill acquisition and problem solving (Anderson, 1983; VanLehn, 1983a; Laird, Rosenbloom, & Newell, 1986; Anzai & Simon, 1979).

Another assumption is that goals and the methods for achieving them are schematic and that they must be instantiated by binding their arguments to objects in the problem state. Again, this is a nearly universal assumption among current theories of skill acquisition.

Cirrus employs a simple knowledge representation that is consistent with the above assumptions and introduces very few additional assumptions. The representation is based on two types of operators. *Primitive operators* change the state of the problem. *Macro operators* expand into one or more operators. Table 1 shows a common subtraction procedure in this format.

Both types of operators specify the type of *goal* for which they are appropriate. Goals are just atomic tokens, such as C (for subtract Column) or F (for borrow From) or R (for Regroup). Goals take arguments, which are shown as subscripts in our notation. Thus, the schematic goal C_i represents the general idea of processing a column, and C_2 represents the instantiated goal of processing the tens column (columns are numbered from right to left). The table's notation for macro operators shows the goal to the left of an arrow, and the subgoals to the right of the arrow. The notation indicates argument passing by placing calculations in subscripts of subgoals. Thus, the sixth macro operator means that the goal of processing a regrouping a column (R_i) can be achieved by achieving two subgoals: borrowing from the next column to the left (F_{i+1})

Table 1: Operators for subtraction

Macro Operators		
1.	Sub_i --> C_i Sub_{i+1}	Column i+1 is not blank
2.	Sub_i --> C_i	Column i+1 is blank
3.	C_i --> CT_i	B_i is blank
4.	C_i --> $-_i$	Not $T_i < B_i$
5.	C_i --> R_i $-_i$	$T_i < B_i$
6.	R_i --> F_{i+1} A_i	true
7.	F_i --> S_i D_i	Not $T_i = 0$
8.	F_i --> R_i F_i	$T_i = 0$

Primitive Operators	
CT_i	Copy T_i into the answer of column i
$-_i$	Take the difference in column i and write it in the answer
A_i	Add ten to the top of column i
S_i	Put a slash through the top digit of column i
D_i	Decrement the top digit of column i

and adding ten to the current column (A_i).

Macro operators may have associated with them a condition that indicates when the the operator is appropriate for achieving its goal. No assumptions are made about the necessity or sufficiency of the given conditions. In table 1, conditions are shown after the subgoals. They are described in English, with the convention that T_i and B_i refer to the top and bottom digits, respectively, of column i. Inside Cirrus, conditions are represented as Lisp predicates.

This representation has been used for many types of procedural knowledge. It is a variant of the notation used in GPS, Strips, and their successors (Nilsson, 1980). To illustrate this generality with a classic example, a macro operator for personal travel might be:

$Travel_{a,b}$ --> $Travel_{a,Airport(a)}$ $TakePlane_{Airport(a),Airport(b)}$ $Travel_{Airport(b),b}$ if Distance(a,b)>500

which says that when the distance between points *a* and *b* is more than five hundred miles, then a good way to travel between them is to take a plane, which requires travelling from *a* to the airport near *a*, and travelling from the airport near *b* to *b*.

Although theorists often agree on the hierarchical, conditional nature of procedural knowledge, they usually disagree on how the deployment of this knowledge is controlled. For instance, VanLehn (1983, 1986) assumes that subjects use a last-in-first-out stack of goals, and that they attack subgoals in the order that those subgoals are specified in the macro operators. On the other hand, Newell and Simon (1972), in their discussion of GPS and means-ends analysis, assume that subjects use a goal stack, but they do not always execute the subgoals in a fixed order. Rather, the subjects use a *scheduler* to choose which subgoal of a macro operator to execute first. In principle, the scheduler's choice may be determined by features of the problem state, the goal stack, or whatever memory the solver may have of past actions. The strategy for making such choices is generally assumed to be task-specific, problem-independent knowledge (e.g., that the third subgoal of macro operator five in table 1 should never be scheduled as the initial subgoal).

Yet another control regime is espoused by Anderson and his colleagues (Anderson, 1983; Anderson, Farrell, & Saurers, 1984; Anderson & Thompson, 1986). They assume that instantiated goals are stored in a goal-subgoal tree. Some goals are marked "done" and others are marked "pending." A scheduler may pick

any pending goal from the tree for execution. This tree regime allows the solver to work on the subgoals of one goal for a while, shift to working on another goal's subgoals, then come back to the first goal's subgoals. The stack regimes of Sierra (VanLehn's program) and GPS do not allow such flexibility.

VanLehn and Ball (1987) studied the performances of 26 subjects executing various subtraction procedures and found that a third of them execute their procedures in such a way that the flexibility of the tree regime is necessary for modelling their behavior. For instance, some subjects would first make all the modifications to the top row required by various borrows in the problem, then answer the columns, starting with the leftmost column and finishing with the units column. This execution order requires instantiating all the borrow goals, executing some of their subgoals (the ones that modify the top row), then executing the column-difference subgoals. This requires the ability to temporarily suspend execution of the subgoals of a goal, which the stack regimes do not have.

However, VanLehn and Ball also showed that a simpler control regime, based on maintaining an agenda of goals (i.e., an unordered set of instantiated goals) rather than a tree, is sufficient for modelling their data. Moreover, the agenda regime can do anything that the stack regimes can do, because appropriate scheduling strategies will make an agenda act exactly like a stack. Motivated by this finding, the design of Cirrus assumes that the subject's problem solving strategy is executed by an agenda regime.

The agenda assumption implies that subjects' have two types of procedural knowledge, an operator set (e.g., the one shown in table 1) and a scheduling strategy. The job of a scheduling strategy is to select a goal from the agenda given the current state of the agenda and the current state of the problem. The remainder of this section discusses ways to represent scheduling strategies and motivates the choice of the representation that Cirrus employs.

In principle, the scheduling strategy can be very complicated. Blackboard architectures (Nii, 1986), for instance, approach the scheduling problem with nearly the same complicated, powerful techniques as they employ for attacking the base problem itself. However, one goal of Cirrus is to infer the subjects' scheduling strategies from data, so an unwarranted assumption of complexity and power only makes its job harder without producing better quality models of the subjects. The optimal choice for representation power is a point just slightly beyond that which seems to be minimal, given the data at hand. Thus, when new data are analyzed, Cirrus has a good chance of succeeding and yet the cost of infering the scheduling strategy is kept reasonable.

To find this optimal point is tricky, because there is a tradeoff in how powerful the scheduling strategy is and how much of the data can be modelled. GPS represented scheduling strategies with a total order on the types of goals (Newell & Simon, 1972, pp. 418 and 436). However, some of the subjects' scheduling choices could not be predicted within this framework.

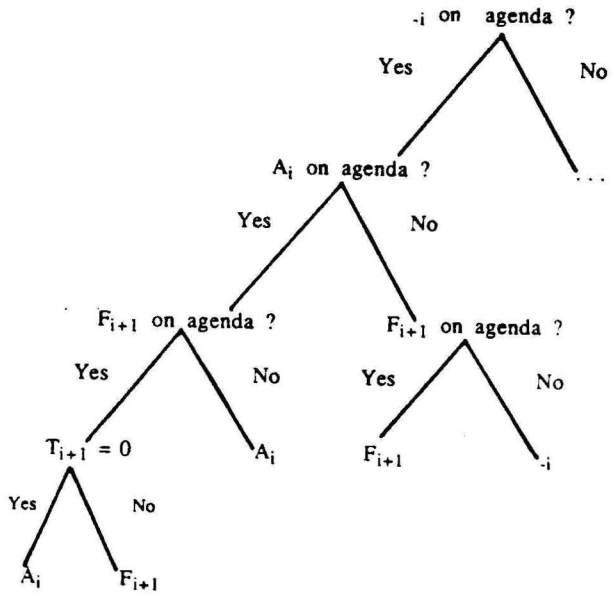
VanLehn and Ball (1987) found that a total order is far too inflexible and that a partial order on goal types allowed much more of the data to be captured. For instance, the two ordering relations, $A_i >\!\!\cdot_i$ and $R_i >\!\!\cdot_i$, represent the strategy that the difference in a column must come after the adding of ten to its top digit, but the relative order of the two regrouping subgoals, A_i and F_{i+1}, doesn't matter. VanLehn and Ball found that the best fitting partial orders still did not predict the observed scheduling choices very well. On average, the best fitting partial orders left about a third of the agenda choices underdetermined.

An underdetermined choice means that either the subject truely has no consistent scheduling strategy, or that the subject has a consistent strategy but Cirrus' representation is not adequate to represent it. In order to

avoid the latter case, the scheduling strategy representation should be just powerful enough to represent any subject's strategy. Since it seems unlikely that students' are guessing a third of the time, the Cirrus design uses a representation for scheduling strategies that is slightly more powerful than a partial order on goal types.

Cirrus represents a scheduling strategy with a decision tree, such as the one shown in figure 1. The tree's tests are predicates on the state of the agenda and the state of the problem. The leaves of the tree contain goal types. The scheduler simply traverses the tree, guided by the tests, until it reaches a leaf. Usually, the leaf will contain only one goal type and there will be only one goal of this type on the agenda. If such is the case, the scheduler just selects that goal instance from the agenda; otherwise, the choice is underdetermined. The decision tree shown in figure 1 represents the scheduling heuristics $A_i>\text{-}_i$ and $F_{i+1}>\text{-}_i$, but in addition, it represents the heuristic that one should do F_{i+1} first unless the top digit in column i+1 is zero. The latter heuristic can not be represented by a partial order on goal types.

Figure 1: A decision tree for a scheduling strategy



In summary, the assumptions built into the design of Cirrus entail that the representation of knowledge has two components: a set of operators, and a decision tree. The operators represented knowledge about how to decompose goals into subgoals and when such decompositions are appropriate. The decision tree represents the subject's strategy for when to work on what types of goals.

3. Cirrus' inputs and outputs

Cirrus takes as inputs (1) a set of problem-solution pairs generated by someone solving problems, (2) a vocabulary for describing problem states, and (3) a set of operators. The latter two inputs are sometimes referred to as a problem space. Cirrus produces a decision tree that represents the subject's scheduling strategy. This section describes these inputs and outputs, and discusses the design issues behind them.

In a problem-solution pair, the solution is represented by an *encoded protocol*, which is a sequence of primitive operator applications. For instance, the first few steps of solving a subtraction problem might be encoded as $[A_1, -_1, S_2, D_2 ...]$. This sequence represents the actions of adding ten to the top digit in the units column, then taking the units column difference and entering it in the answer, then putting a slash through the top digit of the tens column, then decrementing the top digit of the tens column by one and writing the result above the tens column.

Such encoded protocols may be hand-generated by a human reading a transcript of an experiment, or they may be produced automatically by, for instance, recording the user's selections from a menu-driven interface to an experimental apparatus or an intelligent tutoring system.[1] Other strategy induction programs, such as ACM (Langley & Ohlsson, 1984), assume that only the final state (e.g., the answer to a subtraction problem) is available. The advantage of that assumption is that it is much easier to hand-encode a final state than to hand-encode a whole protocol. However, we believe that the state of the art in personal computing and user interfaces is such that automatic generation of encoded protocols will become so common that hand-encoding will soon become a lost art. Thus, there will be no penalty for assuming that encoded protocols are the input.

The second input to Cirrus is a vocabulary of attributes for describing problem states. These attributes are used as tests in the decision tree that Cirrus builds. An attribute can be any discrete-valued function on a state. If all the functions are predicates, then Cirrus will build only binary trees; however, it can handle functions with any type of range.[2] An attribute is a Lisp function with three inputs: the external problem state (e.g., a partially solved subtraction problem), the agenda, and the list of the operator applications up until this point. The latter argument is needed in order to implement attributes such as "Has there been a borrow in this problem yet?" or "Was the last primitive operator a decrement?"

As usual when dealing with induction programs, the user must experiment with attribute vocabularies until the program begins to generate sensible results. This is why we consider Cirrus a *tool* for data analysis, and not an automated psychologist. The art of analysing protocols lies in intuiting what the subject thinks is relevant. Such intuitions are encoded as attibute vocabularies, and Cirrus is used to do the bookkeeping involved in checking them against the data. The ultimate goal of protocol analysis is a model that is consistent with the data, parsimonious and psychologically plausible. Cirrus can evaluate the consistency of the model, but only a human can evaluate psychological plausibility. Parsimony could in principle be measured by Cirrus, but that is left to the human at present.

When the combination of human and Cirrus have found an attribute vocabulary that seems to work for a sample of the subject population, then it may be reasonable to assume that it will continue to yield good

---

[1]The tests of Cirrus reported here use hand-encoded protocols from the VanLehn and Ball (1987) experiments.

[2]Only values that actually occur during the tree building are placed in the tree, so attribute functions will work even if they have infinite ranges. Of course, one would probably want to include a mechanism that breaks large ranges into intervals, in order to enhance the generality of the induced scheduling strategy.

models for subjects who have not yet been tested. If so, then Cirrus can now be used as an automated model builder. Such model builders are used in intelligent tutoring systems in order to track the learning of a student so that the system can offer appropriate instruction. Cirrus can be used as the student modelling module in such a system, but it must first be parameterized for a task domain by empirically deriving the appropriate attribute vocabulary.

The output of Cirrus is a subject model, which consists of an operator set and a decision tree, as previously described. The subject model will be consistent with the set of protocols that it is given. Here, "consistency" means that the model will generate exactly the same primitive operator applications as the subjects, if the model is deterministic. If the model is underdetermined, say, because some of the leaves in the decision tree have more than one goal type in them, then one of the possible solution paths generated by the model for each problem must correspond to the subject's protocol.

Currently, Cirrus does not build the whole subject model, but only the decision tree. The operator set is given to it. This has not been a problem in the initial application because we have a theory of how arithmetic is learned (VanLehn, 1983a; VanLehn, 1983b) that predicts the operator sets that subjects will have. For subtraction, for instance, the theory predicts that the subjects will have one of 30 operator sets (VanLehn, 1983b). In principle, Cirrus could itterate through these 30 possibilities and choose the one that maximize the fit and parsimony of the subject models. However, this choice is currently done by the user.

In task domains where no theory is available, Cirrus could be augmented to accept even less information about the operator sets. It could take only the goals and subgoals of operators as input, and use standard machine learning techniques (Langley et al., 1980) to induce the conditions of the operators.

If one is studying skill acquisition, then making a plausible assumptions about the goals and subgoals of operators is not as hard a problem as it may seem. It has often been observed that instruction tells the student what to do (i.e., what subgoals go with each goals) but not when to do it (Anderson, 1983; VanLehn, 1983b). In cases where the instruction does mention the bodies of operators, then one can generate a plausible set of operator sets by introducing perturbations (e.g., deleting subgoals from operators) into the operator set that the material teaches.

4. Computational Details of Cirrus

Cirrus performs the following operations in order:

1. *Plan recognition.* Cirrus parses each protocol using the given operator set as if it were a context-free grammar. This converts the protocol, which is a sequence of primitive operator applications, into a *parse tree* whose leaves are the primitive operator applications and whose interior nodes are macro operator applications.

2. *Tree walking.* The parse tree is traversed depth-first in order to infer what the state of the agenda must have been at each cycle of execution. The output of tree walking is a *micro state protocol*, which is the original protocol augmented by all the agenda states and state changes that happen between the changes in the external problem state. This protocol represents what one would see if one could see the subject's internal actions as well as the external ones.

3. *Attribute encoding.* In preparation for building a decision tree, Cirrus collects all instance of agenda selections. These are represented initially as the selected item paired with the total state (i.e., the problem, the agenda, and the preceding operator applications) at the time of the selection. Each total state is converted to an *attribute vector* by running all the attribute functions on it. The output of this stage is a set of pairs, each consisting of an agenda selection and the attribute vector for the total state at the time of the agenda selection.

4. *Decision Tree construction.* The last stage is to build a decision tree that will sort the set of pairs by the goal type of the agenda selection. Cirrus uses Quinlan's (1986) ID3 algorithm. Although we included Quinlan's Chi-squared test (op. cit, pg. 93) hoping that it would prevent ID3 from trying to predict the occurrence of noise (i.e., slips), we realize now that it is inappropriate for this purpose. A slip tends to show up as a single instance of an agenda selection that is not of the same type as all the other agenda selections in the set. The Chi-squared test is not an appropriate test for detecting an exception. It is good for taking a hetrogeneous set and showing that any way one discriminates it, the resulting partitions are just as hetrogeneous as the original set, so partitioning the set is pointless. However, with a set that is homogeneous with one exception, virtually any discrimination helps, so slips almost always remain undetected by the Chi-squared test.

The use of a parser for plan recognition deserves some discussion, for it is not as common an AI technique as decision tree building, which is the only other nontrivial operation performed by Cirrus. If an ordinary context-free parser built the parse trees, then a parse tree would obey the following constraints (for easy reference, we say that A is a "suboperator" of B if A appears just beneath B in the parse tree):

1. If two operators are suboperators of the same macro operator, then their order in the parse tree is the same as their order in the right side of the macro operator's rule.
2. If two operators are suboperators of the same macro operator, and they are adjacent in the parse tree, then the portion of the protocol that each covers must *not overlap*. (This is guaranteed by the fact that a parse tree is a tree, in that operators have just one parent in the tree.)
3. If two operators are suboperators of the same macro operator, and they are adjacent in the parse tree, then the portion of the protocol that each covers must be *contiguous*. There can not be primitive operator applications between them.

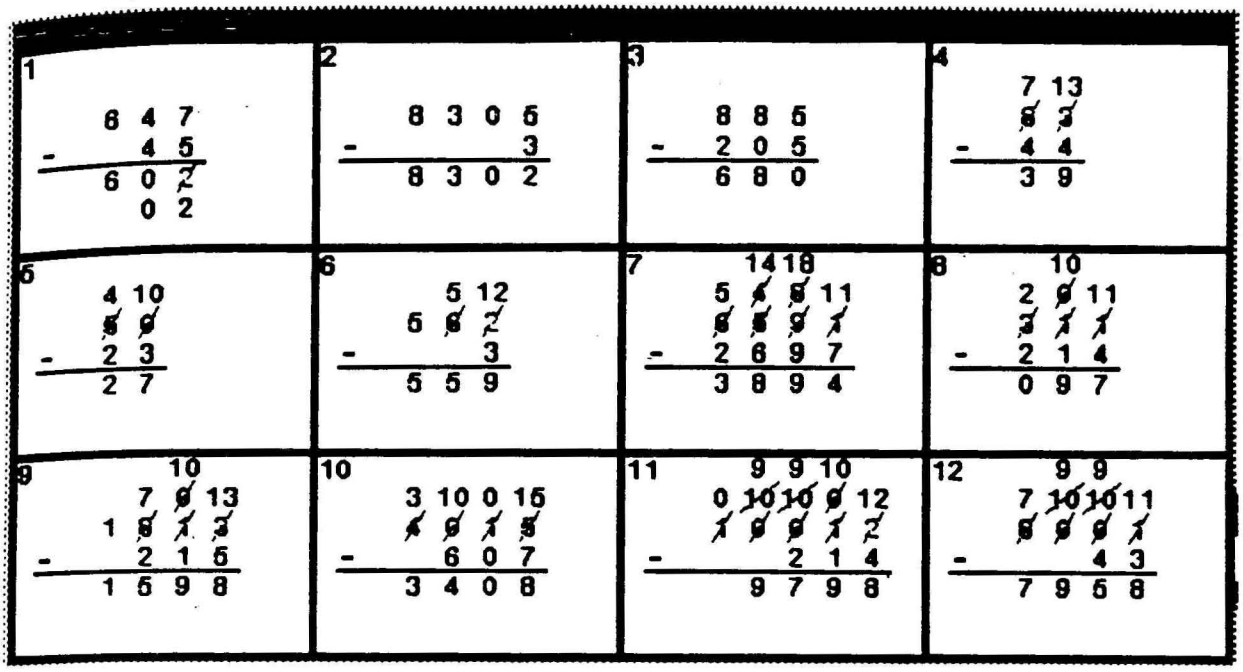
If the control regime were the deterministic stack regime used by Sierra, then this kind of parsing would be appropriate. If the control regime were the scheduled stack regime used by GPS, then relaxing the first constraint would be necessary. Because Cirrus uses an agenda control regime, both the first and third constraints must be relaxed. The third constraint must be removed because the agenda regime makes it possible to stop working on one macro operator's subgoals, work on some other macro operator's subgoals, then go back to the original macro operator's subgoals. This implies that suboperators of the same macro operator need not abut.

Relaxing these constraints is a minor change to the parsing algorithm, but it drastically increases the search space for parse trees. Consequently, we added more assumptions to Cirrus. It is assumed that (1) the values of arguments can not be changed once the goal has been instantiated, and (2) the values of a subgoal's arguments are calculated exclusively from the values of the goal's arguments, regardless of the rest of the total state at the time of instantation. This makes the parser appropriate for attribute grammars (Knuth, 1968) and affix grammars (Koster, 1971) in that it can use the argument values of the goals in order to constrain the search. This vastly improves the performance of the parser, at least for subtraction.

One constraint that the parser does not use, although it could, is provided by the conditions on the macro operators. These could be tested during parsing in order to prune parse trees where macro operators were applied inappropriately. However, since we intend that Cirrus eventually be able to induce these conditions or modify existing ones, we designed the parser to ignore them.

Lastly, the parser is very similar to the one used in Sierra (VanLehn, 1987). Sierra can learn new macro-operators from protocols. Thus, it is possible to modify Cirrus so that when it is given an operator set that is incomplete, it can discover new operators. It is not clear whether this is a useful feature in the applications for which Cirrus is intended.

Figure 2: Paul's problem solutions (above) and protocols (below)



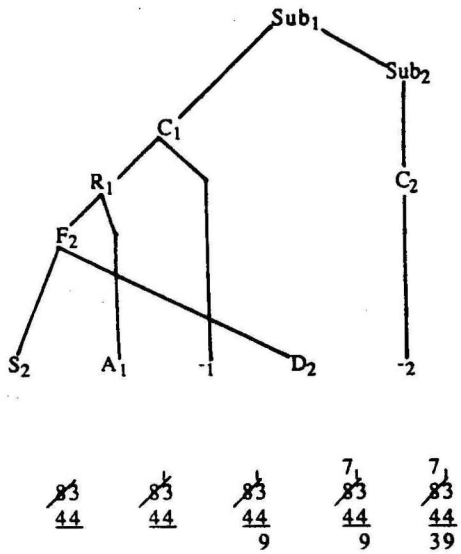
1. $-_1\ -_2\ CT_3$
2. $-_1\ CT_2\ CT_3\ CT_4$
3. $-_1\ -_2\ -_3$
4. $S_2\ A_1\ -_1\ D_2\ -_2$
5. $S_2\ A_1\ -_1\ D_2\ -_2$
6. $S_2\ A_1\ -_1\ D_2\ CT_2\ CT_3$
7. $S_2\ A_1\ -_1\ D_2\ S_3\ A_2\ D_3\ -_2\ S_4\ A_3\ -_3\ D_4\ -_4$
8. $S_2\ A_1\ -_1\ D_2\ S_3\ A_2\ -_2\ D_3\ -_3$
9. $S_2\ A_1\ -_1\ D_2\ S_3\ A_2\ -_2\ D_3\ -_3\ CT_4$
10. $S_2\ A_1\ -_1\ D_2\ -_2\ S_4\ A_3\ D_4\ -_3\ -_4$
11. $S_2\ A_1\ -_1\ D_2\ S_5\ D_5\ A_4\ S_4\ D_4\ A_3\ S_3\ D_3\ A_2\ -_2\ -_3\ CT_4\ CT_5$
12. $S_4\ D_4\ A_3\ S_3\ D_3\ A_2\ S_2\ D_2\ A_1\ -_1\ -_2\ -_3\ -_4$

5. Analyzing a subtraction protocol

This section illustrates the operation of Cirrus by describing the analysis of a protocol from Paul, a third grade student who has just learned subtraction. Paul worked a twelve-item test while we recorded his writing actions. Figure 2 shows his solution to the problems, first as a worked problem and second as a sequence of primitive operator applications. During the encoding process, we filtered out one recognizable slip. Paul

wrote his answer to problem 1 illegibly, so he went back and rewrote it. These actions are not included in the encoded protocol. Although this particular instance of "editing" the data is quite clearly appropriate, part of the process of understanding a protocol may involve conjecturing that certain actions are slips and changing them to what the user thinks the subject intended to do. If this makes Cirrus generate a dramatically better analysis, the conjecture is supported.

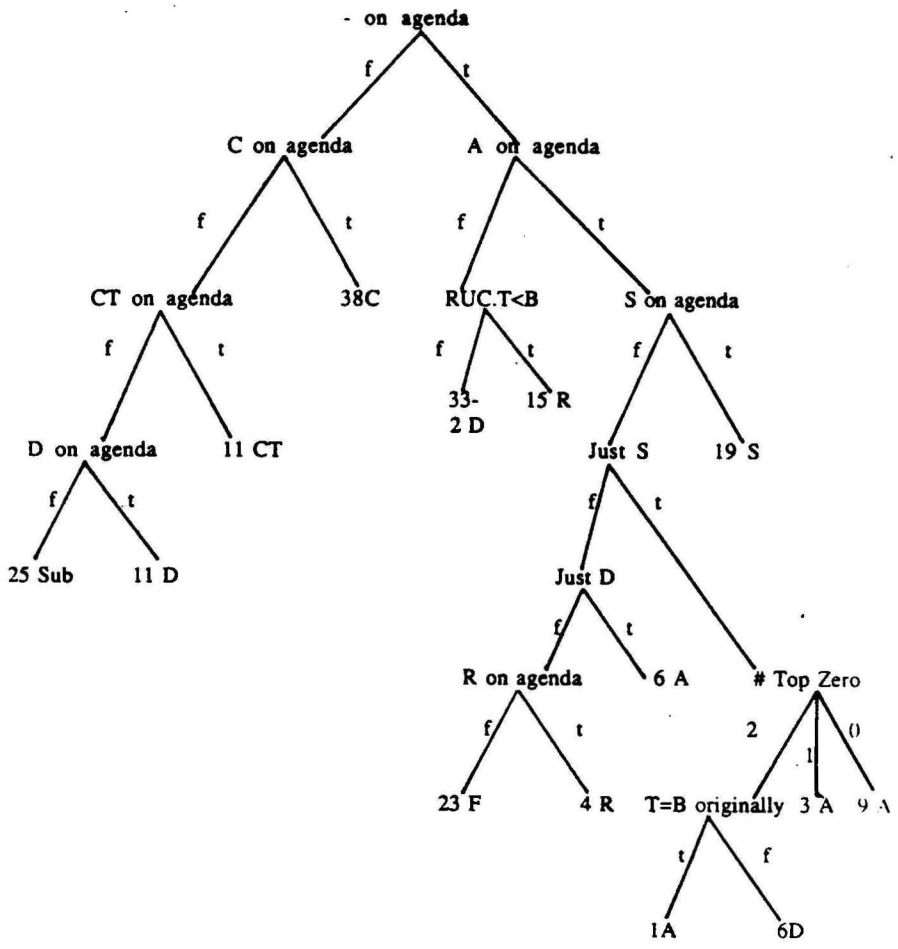
Figure 3: The parse tree for Paul's problem four



The protocols are parsed, using the grammar of table 1. This yields a set of parse trees. Figure 3 shows the parse tree for problem 4. The parse trees are walked, the attribute vectors are collected, and a decision tree is produced. Figure 4 shows the decision tree that represents Paul's scheduling strategy. The leaves of the tree show how many agenda selections of each type have been sorted to that leaf.

Paul's most common strategy for borrowing is to do the scratch mark, then the add ten and column difference, then return to complete the borrowing by doing a decrement (see figure 3). However, on some columns, Paul does the standard strategy of scheduling the decrement so that it immediately follows the scratch mark. These two strategies have much in common, and most of the decision tree is devoted to representing these commonalities. The most interesting part of the tree is the subtree just to the right of the node labelled "Just S" This subtree encodes the sub-strategy for what to do just after a scratch mark has been performed. It represents Cirrus' best guess about the conditions that trigger Paul's two strategies. The attributes chosen have to do with the existence of zeros in the top row of the problem. Apparently, for borrow-from-zero problems (problems 11 and 12), Paul prefers to use the strategy he was taught rather than the strategy he invented. On the basis of this intuition, the user would probably want to augment the attribute vocabulary with an attribute that indicates a column requires borrowing from zeros, because the attributes currently available to Cirrus are such that it must use an implausible attribute, T=B originally, in order to

Figure 4: Paul's decision tree.
"Just X" means that action X was the most recently executed primitive operator.
"RUC.T<B" means that the top digit is less than the bottom in the rightmost unanswered column.
"T=B originally" means that the column's digits were equal in the initial problem state.
"# top zeros" counts the number of zeros in the top row.



differentiate the tens column of problem 11 from the others.[3] This symbiotic exploration of hypotheses about cognition is exactly what Cirrus is designed to expedite.

Although experimentation with attribute sets in inevitable with any induction algorithm, this particular example indicates a problem that seems to be unique to ID3. The desired attribute, which should distinguish borrow-from-zero columns from other kinds of columns, is a conjunction of two existing attributes, $T_i<B_i$ and $T_{i+1}=0$. We had thought that ID3 would capture this conjunction as a subtree with two tests, say, $T_i<B_i$ on the upper node and $T_{i+1}=0$ on the lower node. However, the $T_i<B_i$ test is always true at this point in the tree, so it has no discriminatory power; hence, ID3 would never choose it. This seems like a major problem

[3] A similar situation exists at the leaf under the false branch of RUC.T<B. Although the leaf is shown as containing two types of agenda selections, 33 occurrences of take-difference and 2 occurrences of decrement, Cirrus actually placed a large subtree here filled with implausible attributes. Again, straightening out this subtree would require experimentation with the attribute vocabulary.

with ID3. Its only cure may be to augment the original attribute vocabulary with all possible conjunctions of attributes, a combinatorially infeasible solution. This suggests that ID3 may not be the right choice for an induction algorithm, a possibility we wish to explore in further research.

6. Directions for further research

The only way to evaluate the utility of Cirrus as a tool is to use it as such. This is the first item on our agenda for future research. Our hope is that the combination of Cirrus and human analyst is a more productive combination than human analyst alone. In particular, we will be reanalyzing the rest of the VanLehn and Ball protocols in the hope that Cirrus will help uncover patterns that were overlooked by the unaided human theorists.

As mentioned earlier, we intend to have Cirrus induce the conditions on macro-operators instead of having the user supply them.

There is nothing in our theory of skill acquisition that stipulates that ID3 is the right model of scheduling or macro-operator conditions. (Indeed, we find the decision tree format a difficult one to understand -- a common complaint about decision trees (Quinlan, 1986).) We would like to try a variety of concept formation techniques. We are particularly interested in exploring techniques based on models of human concept acquisition (Smith & Medin, 1981). If people learned their conditions and strategies by induction, then using the same inductive mechanisms, with the same built-in biases, seems like a good way to infer those conditions.

Lastly, there is a sense in which Cirrus is a theory of problem solving. If Cirrus fails to analyze a protocol, then the protocol lies outside its theory of problem solving.[4] There are two ways analysis can fail: (1) The parser may be unable to parse some of the protocol. This indicates either that the operator set is wrong (i.e., a parameter to the theory has the wrong value -- a minor problem) or that the subject is using a more powerful plan following regime. It would be interseting to use Cirrus as a filter on large quantities of protocol data, looking for subjects who are using a more powerful control regime. (2) The second kind of failure occurs when the decision tree builder builds unintelligible trees. This could be the fault of the attribute vocabulary (again, a minor problem with an incorrectly set parameter), or more interestingly, the theory of scheduling strategies could be wrong. We actually believe that the latter may be the case, and look forward to finding evidence for it by uncovering protocols that Cirrus can not analyze.

Acknowledgments

This research was supported by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract Numbers N00014-86-K-0349 and N00014-85-C-0688.

[4]Cirrus' theory of problem solving is discussed in (Garlick & VanLehn, 1987). See Bashkar and Simon's (1977) for an example of a protocol analyzer with a strong theory built into it.

References

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R., & Thompson, R. (1986). Use of analogy in a production system architecture. Paper presented at the Illinois Workshop on Similarity and Analogy. Urbana, Illinois.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8,* 87–129.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86,* 124–140.

Bhaskar, R., & Simon, H. A. (1977). Problem solving in a semantically rich domain: An example from engineering thermodynamics. *Cognitive Science, 1,* 193–215.

Garlick, S., & VanLehn, K. (1987). *Deriving descriptions of the mind: A rationale for serial models of cognition* (Technical Report PCG-6). Pittsburgh, PA: Carnegie Mellon-University, Department of Psychology.

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory, 2,* 127–145.

Koster, C. H. A. (1971). Affix grammars. In J. E. Peck (Ed.), *ALGOL 68 implementation.* Amsterdam: North Holland.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning, 1,* 11–46.

Langley, P., & Ohlsson, S. (1984). Automated cognitive modeling. In *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 193–197). Austin, TX: Morgan Kaufmann.

Langley, P., Neches, R., Neves, D., & Anzai, Y. (1980). A domain-independent framework for learning procedures. *International Journal of Policy Analysis and Information Systems, 4,* 163–197.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. B. (1983). Learning problem-solving heuristics by experimentation. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nii, P. (1986). The blackboard model of problem solving. *Artificial Intelligence, 7,* 38–53.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review, 88,* 1–15.

Quinlan, J. R. (1986). The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Silver, B. (1986). Precondition analysis: Learning control information. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Smith, E. E., & Medin, D. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

VanLehn, K. (1983). Human skill acquisition: Theory, model, and psychological validation. In *Proceedings of the Third National Conference on Artificial Intelligence* (pp. 79–85). Washington, DC: Morgan Kaufmann.

VanLehn, K. (1983). *Felicity conditions for human skill acquisition* (Technical Report CIS-21). Palo Alto: Xerox Palo Alto Research Center.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31,* 1–40.

VanLehn, K., & Ball, W. (1987). *Flexible execution of cognitive procedures* (Technical Report PCG-5). Pittsburgh, PA: Carnegie-Mellon University, Department of Psychology.