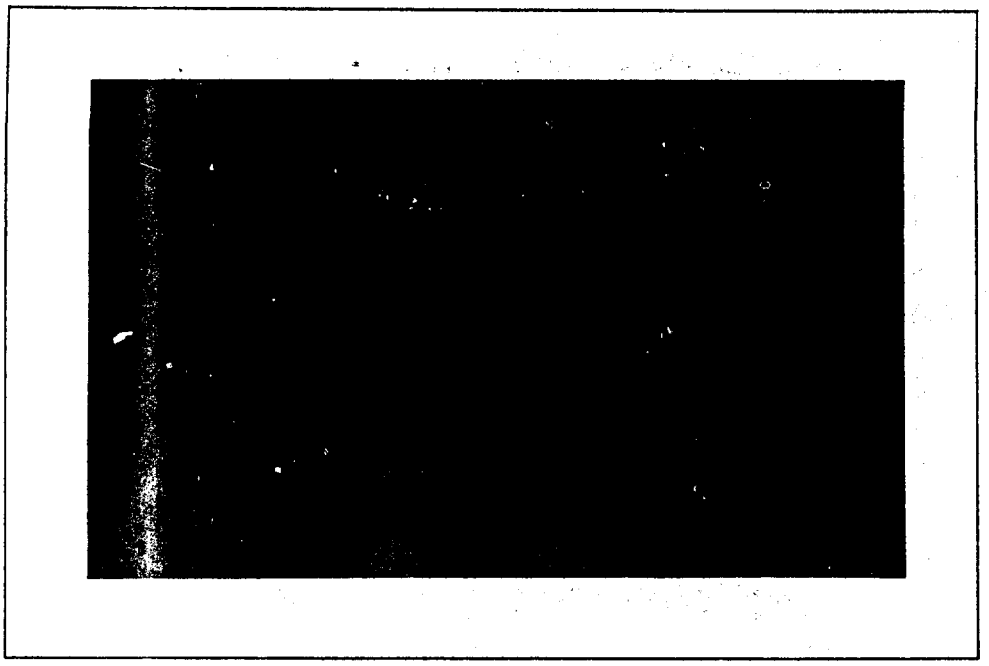
(2)



The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

2

INDUCTION OF PARTIAL ORDERS BEATS CLASSIFICATION: IMPROVEMENTS TO THE CIRRUS PROTOCOL ANALYSIS SYSTEM

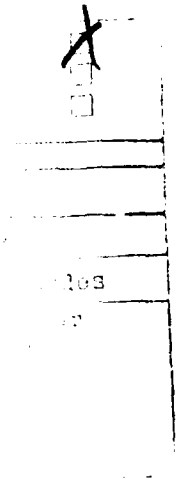
Technical Report AIP - 31

Bernadette Kowalski & Kurt VanLehn

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA. 15232

14 January 1988

This research was supported by the Computer Sciences Division, Office of Naval Research and DARPA under Contract Number N00014-86-K-0678; Reproduction in whole or in part is permitted for purposes of the United States Government. Approved for public release; distribution unlimited.



A-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

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) A I P - 31	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213	7b ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 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 p4000ub201/7-4-86 PROGRAM ELEMENT NO: N/A; PROJECT NO: N/A; TASK NO: N/A; WORK UNIT ACCESSION NO: N/A

11 TITLE (Include Security Classification)
INDUCTION OF PARTIAL ORDERS BEATS CLASSIFICATION: IMPROVEMENTS TO THE CIRRUS PROTOCOL ANALYSIS SYSTEM

12 PERSONAL AUTHOR(S)
Kowalski, Bernadette & VanLehn, Kurt

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	1988 January 14	14

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Machine Learning; Artificial Intelligence; Protocol Analysis

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

In this paper, we discuss Cirrus, a semi-automated data analysis and model formation system. Cirrus starts with a half-order model for a data set, and induces the remaining knowledge necessary to turn the model into one which can reproduce subjects' protocols. This knowledge takes the form of scheduling strategies for selecting tasks from an unordered agenda.

The first implementation of the system, Cirrus-I, used a standard concept formation approach, with ID3 as the induction algorithm. Drawbacks of Cirrus-I led to the development of Cirrus-II, which is based on induction of a partial order. This approach appears to be applicable to many standard classification problems. Moreover, results from computational experiments on a pilot data set indicate that Cirrus-II outperforms human analysts.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b TELEPHONE (Include Area Code) (202) 696-4302 22c OFFICE SYMBOL N00014

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

1. Introduction

Cirrus is a semi-automated data analysis and model formation system. Like Dendral and other systems (Lindsay, Buchanan, Feigenbaum & Lederberg, 1980), it is given a half order model, which is a model that is so general that it is useless by itself. Cirrus specializes the half-order model to fit a given data set. As with Dendral, Cirrus assumes that the data is noisy, so a perfect fit is not expected. The resulting model should nonetheless be an accurate postdiction of the given data, as well as an accurate predictor of future data.

Cirrus analyzes protocol data. For Cirrus, a protocol is assumed to be a sequence of operator applications, and the entity producing the protocol is assumed to be an expert doing routine problem solving. In many task domains, there is good evidence that human experts have acquired a large set of procedures, and that routine problems are solved by selecting a procedure and executing it (VanLehn, in press; Chi, Glaser & Rees, 1982). However, the execution process is not as simple as that used to execute procedures in computer programs. Humans exhibit a great deal of flexibility in how they apply their procedures (VanLehn & Ball, 1987). One type of flexibility is that subjects will often permute the order of a sequence of actions on different occasions. Sometimes they might do A then B then C, and on other occasions, they might do B then C then A. This flexibility in action ordering is presently a source of some mystery in cognitive psychology. As a first step towards understanding the flexibility of expertise, it seems helpful to have an accurate description of the conditions under which each permutation is observed. *Cirrus was designed to yield exactly such a description.*

The half-order theory given to Cirrus is a non-deterministic procedure. The procedure can be executed, even in its original underspecified form, by an agenda-based interpreter. The basic

cycle consists of (1) selecting and removing a task from the agenda, (2) executing it, which may cause creation of new subtasks, and (3) putting the new subtasks, if any, on the agenda. A task is represented as the name of an operator concatenated to a list of arguments. Some operators are primitive, in that executing them causes a state change in the problem state. The other operators, called macro-operators, do not make state-changes, but instead create new subtasks.

The key step in the execution cycle is the first one, where a task is selected from the agenda. The half-order theory includes no information on how to do this selection. It is the job of Cirrus to induce it. The induced information is called a *scheduling strategy*.

Hand analysis of a number of protocols has shown that selection of a task from the agenda is partially a function of the problem state (VanLehn & Ball, 1987). Thus, the first version of Cirrus, Cirrus-I, represented a scheduling strategy as a classifier. Given a problem state, the classifier outputs an operator name. This is the operator that should be chosen from the agenda. The induction problem is thus a standard one: given a set of input-output pairs, each consisting of a state and an operator, induce a function that reproduces this I/O behavior. The function should be defined over all its range, so that it makes predictions about the future. Of course, the induction algorithm must be biased, with the usual tradeoffs of parsimony and accuracy in the induced classifier. Cirrus-I used a modified version of ID3 as its induction algorithm (Quinlan, 1983). The classifiers it induced were n-ary decision trees whose leaves bore the name of the operator to be output by the classifier. For the protocols that were used to pilot-test Cirrus, a typical classification problem involved an I/O table of about 200 pairs (VanLehn & Garlick, 1987). Problem states (the input halves of the pairs) were represented by attribute value vectors with about 50 attributes. Operators (the output halves of the pairs) numbered about 12.

This approach proved to have several fatal flaws, which are discussed below. A number of

other classification algorithms were tried in place of ID3, with poor results.

Eventually, we discovered that actually making the induction problem harder made it solvable. Instead of inducing a function that maps problem states into a small finite set of operator names, the new version of Cirrus, Cirrus-II, induces a partial order that can be used to sort the agenda. More specifically, it induces a set of *constraints*, where a constraint A>B means that task A is preferred over task B whenever both are on the agenda. There are two kinds of constraints, conditional and unconditional. A conditional constraint depends on the problem state and an unconditional constraint does not. Thus, an unconditional constraint, A>B, means that A is *always* preferred to B. A conditional constraint, A>B when X, means that A is preferred to B only when condition X is true of the problem state. Conditional constraints are needed because some subjects' agenda choices depend on the problem state, as mentioned earlier. Thus, a scheduling strategy consists of a set of constraints, some of which may be conditional. It is the job of Cirrus-II to induce such a scheduling strategy given a protocol. This induction problem appears more difficult (which is why we did such a thorough exploration of the classification approach before taking this approach), but it turns out to run faster and produce better results than the classification approach.

Given this representation of a scheduling strategy, the model interpreter chooses an item from the agenda by first testing the conditions on the conditional constraints in order to assemble a set of applicable constraints. It then walks down the agenda, selecting the task or tasks that are maximal according to the applicable constraints. Since the set of applicable constraints changes slowly, a version of the RETE algorithm (Forgy, 1982) is used to speed up agenda selection.[1] The scheduling strategy induced by Cirrus-II is accurate just to the extent that (1) it always finds

[1] This technique may be applicable to other agenda-based problem solvers, such as blackboard architectures.

that just one task is maximal according to the constraints, and (2) that task is in fact the one that the subject chose.

The solution used in Cirrus-II may have wide applicability, because the problem faced by Cirrus seems a rather common one, after one strips away all the application-specific details. The problem is to induce a function of two arguments, call them X and Y, such that the output of the function is always a member of the set Y. This is like a function whose range is given to it as an argument. Many application problems might correspond to this one, or could be made to correspond to this one fairly easily. For instance, suppose that the problem is to induce a one-argument classifier, F(X), and there exists some reliable, fast approximation to F, call it G(X), which produces a set of classes such that F(X) is guaranteed to be in the output set. That is, G can only eliminate some of the possible outputs, rather than determine the output of F precisely. Given this fast, reliable, but crude approximation to F, one can convert the original problem into the problem of inducting a function H of two arguments, where the second argument provides a set from which the output comes. Then F(X) = H(X,G(X)). Thus, traditional classification techniques can be used to induce G, and the techniques employed by Cirrus can be used to induce H.

Given this conceptualization of the problem, it is simple to see the advantage of the Cirrus-II approach over others. The Cirrus-I approach is to simplify the problem by ignoring Y. A brute-force remediation of its ills would attempt to induce a classifier of both arguments; this approach was considered and rejected for reasons discussed below. The Cirrus-II approach simplifies the problem by decomposing the target function F(X,Y) into Sort(S(X),Y), where S(X) is a function that produces information used by a fixed, well-defined sorting function that selects a member of Y for output. Our experience is that this approach produces more accurate fits to the data

and more easily understood models.

This paper first describes the representations of Cirrus in more detail and some specifics about the protocols that we have been using to pilot-test the system. It then presents the problems that Cirrus-I had, and the solution used by Cirrus-II. Results from our computational experiments are presented next, followed by plans for further extensions and tests of the system.

2. Procedure Representation

Cirrus assumes that the procedural knowledge used by the students is hierarchical and conditional in nature. See Table 1 for the description of a subtraction procedure in this format. The rule-like entities stand for macro-operators, which reduce a task (left side of arrow) to subtasks (on the right side of arrow). Conditionality is notated to the right of the subtasks.

The appropriateness and adequacy of various control regimes are discussed in detail in (VanLehn & Garlick, 1987) and (VanLehn & Ball, 1987); an agenda-based control strategy was chosen as the strategy that is parsimonious (in that it makes the least number of assumptions) and yet is still adequate to account for the data. As an illustration of this control regime, Figure 1 shows an execution trace of the subtraction procedure of Table 1.

In order to be able to use an agenda-based control strategy with the subtraction task-subtask hierarchy knowledge, we need to have information about the correct operator to choose from an agenda, given the state of the agenda and the external problem state. This is the learning problem which is addressed by Cirrus.

3. Cirrus' Learning Task

Essentially, what Cirrus needs to learn are concepts that specify which task to choose from an agenda, given a particular external problem state, a particular agenda state, and a particular history of previously applied operators. This state information, which we call the total state, is represented as a set of attribute-value pairs.

For example, consider the subtraction problem, worked by Paul, in Figure 1, which consists of the execution trace of the problem, the action-sequence protocol, and the agenda trace. The first phase of Cirrus involves plan recognition; it does this by treating the subtraction knowledge in Table 1 as a context-free grammar, and parses the protocol as if it were a string which had been produced by this grammar. The output of this stage is a parse tree (see Figure 2) which is also a task-subtask representation of the problem solution; the leaves consist of the primitive operator applications present in the original protocol, and the interior nodes represent the application of macro-operators.

The task-subtask tree is then traversed in an order which obeys certain conventions discussed in (VanLehn & Garlick, 1987), that incorporate the by-now standard hypothesis that people ordinarily expand goal trees depth-first rather than breadth-first (Newell & Simon, 1972). The output of this phase is a sequence of pairs, each pair consisting of the task which was chosen from the agenda (the name of the tree node which was traversed), and the attribute-value representation of the total state at that instant.

Cirrus now has all the information necessary to begin inducing scheduling concepts. It is at this point that the major difference between Cirrus-I and Cirrus-II occurs. In order to facilitate the explanation of this, we will initially discuss the solution used by Cirrus-I, and discuss its functionality and limitations. Then, we will go on to discuss a solution which turns

an unmanageable machine learning problem into one that can be attacked fairly easily and was implemented as Cirrus-II.

3.1. Flaws in Cirrus-I

The scheduling strategy induced by Cirrus-I was represented as a decision tree. The internal nodes of the tree consist of attribute tests on the total state. Each leaf of the tree contains an operator's name which indicates the task to be chosen from the agenda.

The problem of inducing the decision tree can be considered as a classification problem. Each operator is equivalent to a different concept or class; a given total state can be said to be in the class corresponding to the operator that was actually chosen from the agenda at that time. A modified version of Quinlan's ID3 algorithm (Quinlan, 1986) is used to perform the classification task. A diagram of Paul's decision tree appears in (VanLehn & Garlick, 1987).

The major drawback inherent in representing the scheduling information this way is that it outputs operator names and not tasks (a task is an operator name plus arguments). This is a problem because it is possible for the same operator to appear on the agenda multiple times in different tasks. For example, when a "borrow-across-zero" operation is undertaken, the **Add/10** operator will appear at least twice on the agenda. The scheduling knowledge as outlined above would arrive at a decision to choose **Add/10** from the agenda, but then be stuck because it would not know which **Add/10** to choose.

A brute-force solution is to induce classifiers that treat distinct tasks as distinct classes. This approach does not work well because it creates classes that are too specialized, and thus do not allow the induced classifier to adequately schedule agenda states that it was not trained on. For instance, the classifier may output a task that is not on the current agenda, even though it appeared on the agenda of similar total states during training. We considered four separate

modifications, each one adding extra structure to allow task relationships to be expressed. We also tried replacing ID3 with the noise-handling version of Mitchell's candidate-elimination algorithm (Mitchell, 1978). None of these approaches worked well (see Kowalski & VanLehn, in preparation).

Another drawback of Cirrus-I is that it produced large decision trees, ranging in size from 30 to 50 nodes in number, and extending to depths of 7 or 8. Besides being computationally expensive to produce, large trees resulted in limited readability, making it difficult to recover any sense of the overall strategy that the subject was using. It also meant that noise points which occurred in the protocol data extended their influence throughout various parts of the tree, decreasing understandability by an even greater degree.

3.2. Cirrus-II's Solution

Cirrus-II was designed to overcome these problems. Scheduling information is represented as relative order relationships between pairs of tasks that take column argument relations into account. Ordering relationships are called constraints. Examples of constraints are: "**Diff** > **Add/10** when they are in the same column", "**Decr** > **Diff** when **Decr** is in a column left-adjacent to the column in which **Diff** is located ", etc.

The original pairs produced by the tree-walking phase can now be used to generate concepts of a different type. For example, suppose that the agenda contains the following tasks: **Sub1col**$_2$, **Diff**$_1$, **Add/10**$_1$, **ScratchMark**$_2$, **Decr**$_2$[2]; suppose further that the chosen task was **ScratchMark**$_2$, and that the attribute-value representation of the problem state is **AV**. Instead of forming the pair <**ScratchMark** - **AV**>, a number of pairs are formed. Each pair in the set represents a single constraint which must have held in order for the given task to have been

[2]Columns are numbered from right to left and subscripts indicate arguments (columns) of the tasks

chosen. For example, the following constraints (among others) hold during this particular case: "**ScratchMark > SUB1COL: same-column**" "**ScratchMark > Diff: left-adjacent**". If there are N + 1 tasks on the agenda, and one is chosen, then more than N constraints are produced, because there are multiple ways to characterize relationships between the arguments of two tasks.

At this point, it would seem that we have a classification task again; each constraint that is possible in the representational vocabulary corresponds to a distinct class. However, a given total state will usually be a member of many different classes. So, the classification problem has now become one of classifying a given problem state into a set of non-disjoint classes. This is a non-trivial problem, especially considering the total number of classes (approximately 1000) and the added difficulty of noise being present in the protocols.

We now introduce the concept of complementary constraints. A pair of complementary constraints would be a pair such as "**Decr > Diff: left-adjacent**" and "**Diff > Decr: right-adjacent**". So, if $\mathbf{Diff_1}$ and $\mathbf{Decr_2}$ were on the agenda, the first constraint would vote to have $\mathbf{Decr_2}$ chosen from the agenda, and the second would vote to have $\mathbf{Diff_1}$ chosen. In the instances where the first constraint is applicable, the second constraint is necessarily violated, and vice versa.

Hand analysis (VanLehn & Ball, 1987) indicated that the scheduling strategy of a given student can be represented by a set of constraints such that the majority of the constraints are never violated. In fact, in the case of Paul, about 250 from the possible 1000 constraints were relevant, and of these, only 14 are violated. These 14 occur in 7 complementary pairs: either one or the other of the constraints in a complementary pair will apply in a given situation, but not both. We can now recast the classification problem above as a small series of simple discrimination problems. It is only necessary to consider each relevant complementary pair, and

use an induction algorithm to induce the necessary conditions for the application of one or the other. ID3 is used to perform this function.

Cirrus-II produces multiple decision trees, instead of just one, as produced by Cirrus-I. On our pilot data sets, these decision trees range in size from 3 to 15 nodes, and extend to depths of 4 or 5. As a result of the reduced size, the trees are more readable, and more readily integrated into an overall strategy for a subject. This readability allows the occurence of noise in the protocols to be localized to a small number of decision trees for each student, thus allowing the presence of noise to be made explicit to the experimenter, and not allowing it to cloud the overall results. For those decision trees based on non-noisy data, the trees are smaller (3 to 8 nodes, depth of 2 to 4).

4. Results

Cirrus-II is fully implemented, including the model interpreter. The results obtained have been almost perfect: Cirrus-II correctly models each student at every choice point, except for one ambiguous point in the protocol of one student. Furthermore, the classification trees produced have been very helpful in the formulation of strategies to describe each student's behavior. In comparison, when these same protocols were analyzed by hand (VanLehn and Ball, 1987), up to one-third of the choice points were left undetermined. So, not only does Cirrus-II do the job it was designed to do, it outperforms the human analysts in its ability to fully recreate the students' protocols.

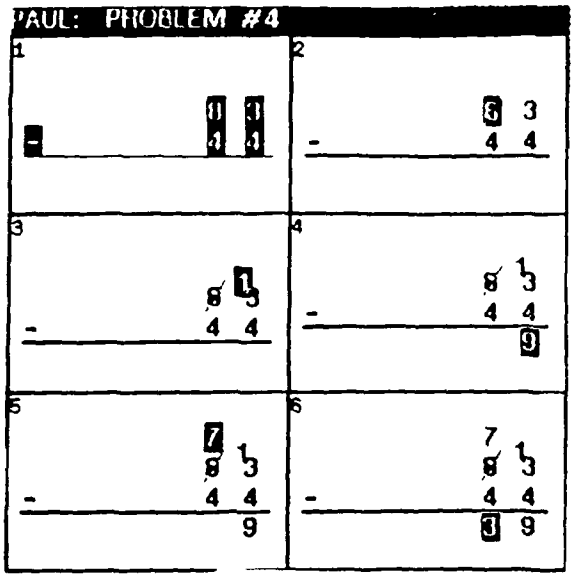
Another experiment is to show both Cirrus and the human analysts only the first halves of each subject's protocol. They produce models, and the models' fits are evaluated against the second halves of each protocol. We have not yet performed this experiment, because the human

analysts are too familiar with our pilot data. We are in the process of obtaining more data (see below).

5. Further Work

There are some areas of Cirrus-II which require further work. These include allowing the theorist greater flexibility in using an attribute-value representation by making it possible to assign weights to attributes, consistent with the importance which the theorist wishes to attach to them. We also have some ideas about integrating the knowledge in the separate decision trees to form coherent overall strategies automatically. Cirrus-II will soon have the ability to induce the macro-operator conditions, and will almost certainly use ID3 for this task.

While it is certainly the case that subjects working subtraction problems cannot be compared directly to experts working in a knowledge-rich problem domain, one of the most exciting areas of further research with Cirrus will involve testing the system's ability to analyze experts performing more complicated tasks. A preliminary study has been done in this area, using action-sequence protocols generated by experts in a somewhat richer mathematical domain. Protocols were collected from expert subjects working with Sketch, an intelligent tutoring system (Trowbridge, Larkin & Scheftic, 1987). Their task was to transform the graphs of basic transcendental functions appropriately to produce a graph which was the correct representation of a given equation. Initial results are very promising.



Protocol: $\mathbf{ScratchMark_2\ Add/10_1\ Diff_1\ Decr_2\ Diff_2}$

Initial agenda: $\{\mathbf{Sub_1}\}$
$\{\mathbf{Sub_2\ Sub1col_1}\}$
$\{\mathbf{Sub_2\ Regroup_1\ Diff_1}\}$
$\{\mathbf{Sub_2\ From_2\ Add/10_1\ Diff_1}\}$
$\{\mathbf{Sub_2\ Scratchmark_2\ Decr_2\ Add/10_1\ Diff_1}\}$
$\{\mathbf{Sub_2\ Decr_2\ Add/10_1\ Diff_1}\}$
$\{\mathbf{Sub_2\ Decr_2\ Diff_1}\}$
$\{\mathbf{Sub_2\ Decr_2}\}$
$\{\mathbf{Sub_2}\}$
$\{\mathbf{Sub1col_2}\}$
$\{\mathbf{Diff_2}\}$
Final agenda: $\{\}$

Figure 1: Paul's problem #4, his protocol, and the agenda trace

Macro-operators

1	$\mathbf{Sub}_i \rightarrow \mathbf{Sub1col}_i\ \mathbf{Sub}_{i-1}$	Column $i+1$ is not blank
2	$\mathbf{Sub}_i \rightarrow \mathbf{Sub1col}_i$	Column $i+1$ is blank
3	$\mathbf{Sub1col}_i \rightarrow \mathbf{Show/Top}_i$	Bottom of column i is blank
4	$\mathbf{Sub1col}_i \rightarrow \mathbf{Diff}_i$	Top of column $i \geq$ bottom of column i
5	$\mathbf{Sub1col}_i \rightarrow \mathbf{Regroup}_i\ \mathbf{Diff}_i$	Top of column $i <$ bottom of column i
6	$\mathbf{Regroup}_i \rightarrow \mathbf{From}_{i+1}\ \mathbf{Add/10}_i$	True
7	$\mathbf{From}_i \rightarrow \mathbf{ScratchMark}_i\ \mathbf{Decr}_i$	Top of column $i \neq 0$
8	$\mathbf{From}_i \rightarrow \mathbf{Regroup}_i\ \mathbf{From}_i$	Top of column $i = 0$

Primitive Operators

$\mathbf{ShowTop}_i$	Copy top of column i into the answer slot of column i
$\mathbf{Diff}_i$	Take the diffference in column i and write it in the answer slot
$\mathbf{Add/10}_i$	Add 10 to the top of column i
$\mathbf{ScratchMark}_i$	Put a slash through the top digit of column i
$\mathbf{Decr}_i$	Decrement the top digit of column i

Table 1: Operators for subtraction (VanLehn & Garlick, 1987)

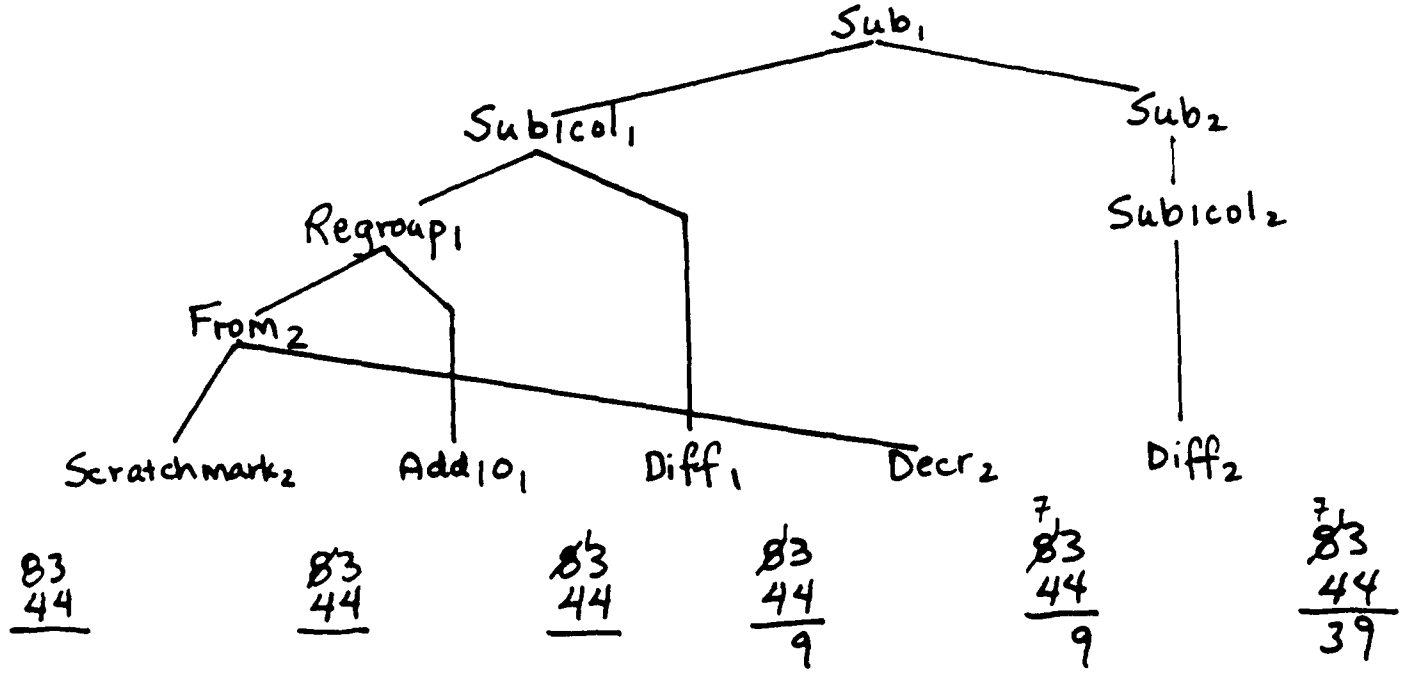


Figure 2: The parse tree for Paul's problem #4 (VanLehn & Garlick, 1987)

REFERENCES

Chi, M.T.H., Glaser, R. & Rees, E. (1982) Expertise in problem solving. In R.J. Sternberg (Ed.) *Advances in the Psychology of Human Intelligence (Vol. 1)*. Hillsdale, NJ: Erlbaum.

Forgy, C.L. (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*. 19, pp. 17 - 37.

Kowalski, B. & VanLehn, K. (in preparation) Cirrus-II: Induction of Partial Orders. to appear as Technical Report, Departments of Computer Science and Psychology, Carnegie-Mellon University, Pittsburgh, PA.

Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A. & Lederberg, J. (1980) *Applications of Artificial Intelligence in Organic Chemistry: The DENDRAL project*. New York: McGraw-Hill.

Mitchell, T.M. (1978) *Version Spaces: An Approach to Concept Learning*. Ph.D. Dissertation: Department of Electrical Engineering, Stanford University, Stanford CA.

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

Quinlan, J.R. (1983) Learning efficient classification procedures and their application to endgames. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell, (Eds.), *Machine Learning: An artificial intelligence approach*. Palo Alto: Tioga Publishing Company.

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, Volume II*. Los Altos, CA: Morgan-Kaufmann.

Trowbridge, D., Larkin, J. & Scheftic, C. (1987) Computer-based tutor on graphing equations. In *Proceedings of National Education Computer Conference*. Eugene, OR: ICCE.

VanLehn, K. (in press) Problem solving and cognitive skill acquisition. In M.I. Posner (Ed.) *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.

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

VanLehn, K. & Garlick, S. (1987) Cirrus: An automated protocol analysis tool. In Langley, P. (Ed.) *Proceedings of the 4th International Workshop on Machine Learning*. Los Altos, CA: Morgan-Kaufmann.