A Brief Description of the Way Forward and Backward Chaining Works

In forward chaining, the inference mechanism starts by evaluating the first rule in the knowledge base. If the antecedent of that first rule is true, then the consequent of the rule is used to search for a conditional with an antecedent identical to the previous consequent. This forward chaining continues until the system is unable to match a consequent with an antecedent. Because the system reasons from the information or data provided, this form of processing is said to be **data driven.** The two rules below will serve as a demonstration of this process:

```
IF A IS true, THEN B IS true
IF B IS true, THEN C IS true
```

a. The following steps define how forward chaining could be applied to the rules above:

(1) If "A is true" is known, the inference mechanism will prove "B is true" by modens ponens[8].

(2) The system then searches forward for a rule that has an antecedent that matches the consequent "B is true". A match is found in the second rule.

(3) Again the law for modens ponens is used to prove that "C is true". Since no further rules can be found with antecedents that match consequents, the system will offer "C is true" as its conclusion.

In backward chaining, the inference mechanism starts with a goal and seeks to find a rule with that goal as its consequent. It then verifies whether or not that rule can be derived from another rule by finding another rule whose consequent matches its antecedent. This process of backward chaining continues until a rule is found that has an independent antecedent. Thus, backward chaining is actually **goal driven** in its problem solving strategy.
The example below demonstrates the implementation of this concept:

Goal statement

[8] "A rule of inference that states: IF A implies B and A is known to be true, then B is true.

```
     D IS true

 Production rules

     RULE 1
     IF    A IS true
     THEN  B IS true

     RULE 2
     IF    B IS true
     THEN  C IS true

     RULE 3
     IF    C IS true
     THEN  D IS true
```

b.The first statement in the example above, "D IS true," is the goal for this knowledge base. The following steps explain how LEVEL5's inference mechanism backward chains to prove this goal:

(1) The system begins by searching for a rule with the goal "D IS true" as its consequent. Since Rule 3 satisfies this condition, the program backward chains to check if the antecedent "C IS true"
can be derived from another rule.

(2) It is discovered that Rule 2 does, in fact, have a consequent that matches the antecedent of Rule 3. The program will now test to see if the antecedent of Rule 2, "B IS true", can be derived from another conditional.

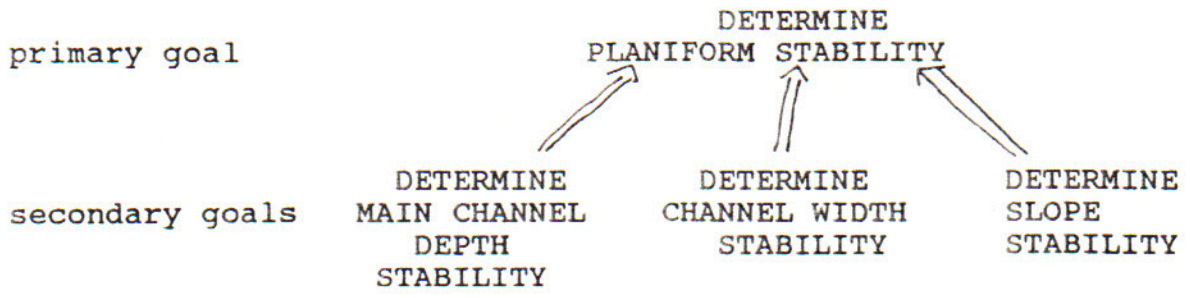
(3) Rule 1 has a consequent that matches the hypothesis in Rule 2.
LEVEL5 searches once more for other supporting rules. Since none can be found, the program asks the user:

```
Is it true that:
A IS true
```

If the user answers yes, the inference mechanism is able to reach the conclusion that "D IS true" based on the law of hypothetical syllogism[9].

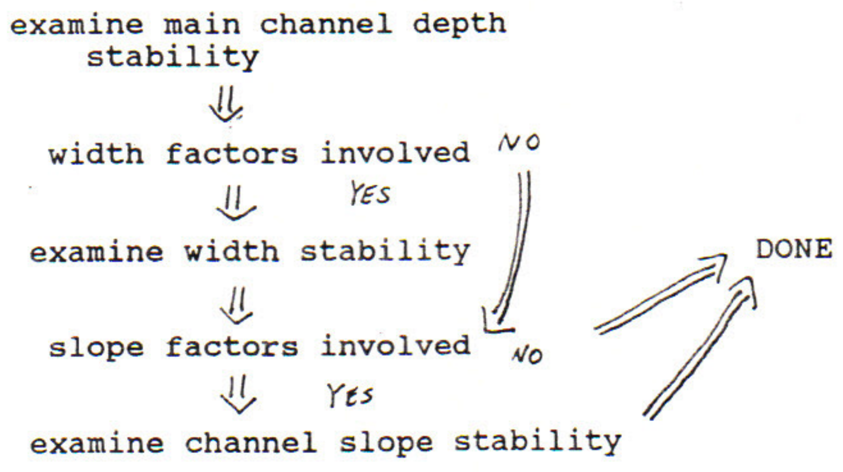
[9] A rule of inference that states: IF A, then B. If B, then C. Therefore, If A, then C.

Suppose we try and create a fully backward chaining problem solving strategy to implement our stream bed flow expert system rule base. We need one primary goal which all the other rules work backward toward solving; a series of secondary goals each of which has information needed by the primary goal; and a whole series of secondary factors which contribute to the information required to satisfy the secondary goals.


primary goal
DETERMINE PLANIFORM STABILITY
secondary goals
DETERMINE MAIN CHANNEL DEPTH STABILITY
DETERMINE CHANNEL WIDTH STABILITY
DETERMINE SLOPE STABILITY

secondary factors: channel depth, channel slopes, channel width, fines present on the bar surface, type of bar, etc.[10]

In this simplification of our river channel flow system the flow chart for the problem solving strategy would be:



[10] The secondary factors are connected to the secondary goals by sets of control rules which are explained below.

In order to implement this problem solving strategy in backward chaining we would need what is called control rules. For example,

we would write

```
RULE 1
IF  Channel Depth and Slope Stability known
AND  Channel Width Stability known
THEN Planiform Stability known  {conclusion 1}

RULE 2
IF Channel Depth Stability known
AND Channel Slope Stability known
THEN Channel Depth and Slope Stability known  {conclusion 2}

RULE 3
IF Channel Width IS increasing,decreasing,stable
THEN Channel Width Stability known  {conclusion 3}

RULE 4
IF Channel Depth IS increasing,decreasing,stable
THEN Channel Depth Stability known  {conclusion 4}

RULE 5
IF Channel Slope IS increasing,decreasing,stable
THEN Channel Slope Stability known   {conclusion 5}
```

The goal of the above abbreviated backward chaining system is to arrive at the conclusion "Planiform Stability is known". Since Rule 1 has that conclusion as its consequent, the inferencing mechanism tries to satisfy the two antecedents of Rule 1: "Channel Depth and Slope Stability known" and "Channel Width Stability known". The consequent of Rule 3 matches the latter so the inferencing mechanism tries to satisfy the antecedent of Rule 3: "Width is increasing, decreasing, stable".We would then have a rule dependency map for the full rule set as below:

$$\begin{tikzcd}
\text{RULE 4} \arrow[dr, Rightarrow] & & \\
& \text{RULE 2} \arrow[dr, Rightarrow] & \\
\text{RULE 5} \arrow[ur, Rightarrow] & & \text{RULE 1} \\
& \text{RULE 3} \arrow[ur, Rightarrow] &
\end{tikzcd}$$

or a similar graph if we plotted the relationships by rule conclusions instead of rule numbers.

$$\begin{tikzcd}
\text{CONC 4} \arrow[dr, Rightarrow] & & \\
& \text{CONC 2} \arrow[dr, Rightarrow] & \\
\text{CONC 5} \arrow[ur, Rightarrow] & & \text{CONC 1} \\
& \text{CONC 3} \arrow[ur, Rightarrow] &
\end{tikzcd}$$

The difficulty with this problem solving strategy is that when we add rules to get the information that is needed to form the conclusions in the higher level rules,the hypotheses for those rules may contain variables which fire other rules that alter previous conclusions. For example, suppose we have a rule that says :

```
RULE 6
If A diagonal bar is present
THEN Channel Width IS increasing
```

then we cannot determine what RULE 3 (which involves the Channel Slope in its conclusion) will say until RULE 6 (which involves Channel Width) is evaluated. If the program happens not to consider the rules in the correct order (as it doesn't in this example) then the conclusion reached will not be valid. Thus we see that in the most general situation conclusions reached upon considering beginning rules may have to be reevaluated in light of later conclusions that the rule-based system reaches.

How to Create Forward Chaining in a Backward Chaining System

Although PROLOG goal driven systems are not designed primarily for data driven problems (see Brakto (1986) and Amble (1987) for an explanation of how these systems work) , it is possible to simulate forward chaining in a backward chaining search using global variables[11] and recalling the goal (cycling) after each success. Newer versions (2.5 and later) of the LEVEL5 software, written for IBM compatible personal computers, have significant object oriented features which make forward chaining in this type of system easier. However, LEVEL5 version 1.1 which is available for the MacIntosh and used for this report can also be used for these type of problems.

In cycling the program there must be **only one** top-level goal (which is called "forward chain" in the manual[12]). A global variable (called "step") then allows you to consider different groups of rules on different passes through the rule base. The programmer must then organize the groups of rules so that every group is consider in a fixed sequence of different "steps".

The hierarchy of goal levels shown in the manual where they are listed in outline form such as 1., 1.1, 2., 2.1, only work when you need to make a single pass through the set of rules. First goal selection determines which upper level rule you want the compiler to unify variables[13] on. This works just like when a PROLOG compiler asks you which goal to solve for in its predicate rule base. It does not prioritize the goals and search for all possible solutions. But, the search levels are created by the developer placing what is sometimes called a "salience factor"[14] (or operator precedence factor) on the rule when it is placed on the goal stack[15]. When a new hypothesis is placed on the search

---

[11] "A value established for use when no procedure or binding (a place in memory reserved for a value associated with a symbol) primitive supplies a value." Winston and Horn, op.cit.

[12] LEVEL5 for the Apple Macintosh, User's Guide, by Information Builder's, Inc. 1250 Broadway, New York,N.Y. 10001.

[13] "The process of comparing two pattern expressions to see if they can be made identical by a consistent set of substitutions." Winston and Horn, op.cit.

[14] " A priority number given to a rule. When multiple rules are ready to be satisfied or as is sometimes said, for "firing", they are fired in order of priority. The default salience is zero. Rules with the same salience are fired according to the current conflict resolution strategy." NASA op.cit.

[15] A list of all the goals that the inference mechanism is backward chaining in order to satisfy. The goal at the top of the list is the goal which the compiler is currently searching the

stack[16] (also called an agenda) salience numbers are checked to make the insertion. The search then keeps going until it bottoms out on a lower level goal. Problems suited for this type of program organization are for instance, diagnostic rule bases or classification problems with only one end conclusion. Groups of definitions of objects fall into this category.

One way to order the rule base is to list which rules have variables in their hypotheses, define a partial order on the rules by letting the rules be nodes in a graph and connect two nodes (rules) with an edge if the conclusion of one is used in the hypothesis of another [17], and then topologically sort the rule base according to this partial order:

ALGORITHM TO TOPOLOGICALLY SORT RULES IN AN EXPERT SYSTEM

0) For the whole set of nodes of conclusions in the rules:

1) If every conclusion node has a predecessor, then stop. The rule based system has a cycle and is infeasible (that is, a partial order cannot be defined on it).

2) pick a node V which has no predecessor

3) place V on a list of ordered nodes
 a) if a terminal conclusion node is reached, print out the list of rules used on the way to reach that conclusion.

4) delete all edges leading out from V to other nodes in the network

5) Go to step 0).[18]

---

knowledge base of consequents in order to unify variables on.

16 "A list of all rules that are presently ready to be satisfied. It is sorted by salience values and the conflict resolution strategy. The rule at the top of the search stack or agenda is the next rule that will fire.", NASA, op. cit.

17 Recall the definition from mathematics that a partial order is a relation rel(x,y) between objects in a set that satisfies the reflexive (rel(x,x) is always true), and transitive conditions (rel(x,y) and rel(y,z) true implies that rel(x,z) true). If the relation also satisfies the symmetric condition (rel(x,y) true implies rel(y,x) true) then it is called an equivalence relation.

18 A further discussion of the way this algorithm works in the case of any partial order and how to write the pseudo code for a simple version of it is given in the books: Fundamentals of Data Structures by E. Horowitz and S. Sahni, Computer Science

Example:

a] initial network:

$$\begin{tikzcd}
& 2 \arrow[dr] & \\
1 \arrow[ur] \arrow[r] \arrow[dr] & 3 \arrow[r] \arrow[dr] & 5 \\
& 4 \arrow[r] \arrow[ur] & 6
\end{tikzcd}$$

b] node visited - 1

remaining network:

$$\begin{tikzcd}
2 \arrow[dr] & \\
3 \arrow[r] \arrow[dr] & 5 \\
4 \arrow[r] \arrow[ur] & 6
\end{tikzcd}$$

c] node visited - 4

remaining network:

$$\begin{tikzcd}
2 \arrow[dr] & \\
3 \arrow[r] \arrow[dr] & 5 \\
& 6
\end{tikzcd}$$

Press, Rockville, MD, 1982.,and Algorithms + Data Structures = Programs by Niklaus Wirth, Prentice Hall, 1976. A C source code implementation of it along with a further discussion is included in Appendix II.

d] node visited 3

remaining network:

$$\begin{tikzcd} 2 \arrow[dr] & \\ & 5 \\ & 6 \end{tikzcd}$$

e] node visited 6

remaining network:

$$\begin{tikzcd} 2 \arrow[dr] & \\ & 5 \end{tikzcd}$$

at this point a terminal conclusion is reached and the number of levels of rules needed [3] is printed out

f] node visited 2

remaining network:

$$5$$

g] node visited 5

An example of how the above algorithm works is illustrated in figures 1 and 2 above.

First a successor list for each rule conclusion node is created:

```
successor list for vertex  1  [ 2 3 4]
successor list for vertex  2  [ 5]
successor list for vertex  3  [ 5 6]
successor list for vertex  4  [ 6 5]
successor list for vertex  5  []
successor list for vertex  6  []
```

Then the algorithm produces a topological ordering of vertices as shown in the figures and as listed below:

```
   1   4   3   6
terminal conclusion reached
  3 levels of rules required

   2   5
terminal conclusion reached
```

```
3 levels of rules required
```

With the above topological ordering when rule 6 is considered, the information for either rule 3 and 4 is then required. And each of these rules will require the information from rule 1. This path of rules then forms at least two forward chaining "cycles" or "steps" . We do not know beforehand whether rule 3 or rule 4 will provide the way to satisfy rule 6, hence a separate step is required to recycle through the rules to cover all possible cases (see the next section for the details of how this is implemented in the code). To better organize things for future additions to the rule base, it is prudent to add another step for rule1 and thus use three steps for this path of rules. For step 4 in this example we consider rule 5. This rule then requires the information from rule 2 and rule 1 in that order. Rule 1 has already been considered in step 1. With this path all rules have been considered. Therefore two more steps of recycling through the rules are required to consider the whole rule base. These two steps will then insure that all the information necessary to reach any possible conclusion has been entered.

If, in entering the input information, we change the order in which the nodes coming out of a given vertex are ordered the program gives a different output. This is a result of the fact that for a given set of order relations there may be many different ways of defining a partial order on them. Consider what happens in the above algorithm if we change the input order:

```
successor list for vertex  1  [ 4 3 2]
successor list for vertex  2  [ 5]
successor list for vertex  3  [ 6 5]
successor list for vertex  4  [ 6 5]
successor list for vertex  5  []
successor list for vertex  6  []
```

the algorithm will then produce the following topological ordering of vertices:

```
   1   2   3   4   5
terminal conclusion reached
  3 levels of rules required

   6
terminal conclusion reached
  3 levels of rules required
```

However, upon examining this output, it can be seen that the number of paths, "cycles", or "steps", required to enter all the information into the classification system is the same in the two cases. Also, the maximum depth or number of levels of backward reasoning for all cycles will be the same for both cases. The next section will explain the coding procedures for implementing these forward chaining cycles or steps which are determined by the topological order.

Procedures to Use in Creating the Knowledge Base in a Classification Type Expert System

In order to create a knowledge base there is an organized procedure that one can follow:

a. Establish the facts by:

(1) collecting all the relevant facts and information

(2) divide the objects in the facts into different categories

(3) outline or catalog the complete set of facts according to these categories.

(4) write down all the rules (involving forward chaining) relating these categories.

(5) write down a decision tree for what the expert system is trying to analyze such as that shown in the previous paragraph.

(6) determine the one goal which the expert system is trying to satisfy.

(7) write down all the backward chaining rules which help to satisfy that one goal.

(8) relate the forward chaining and backward chaining rules in order to have the expert system perform its task in one program run.

i) write down a complete table of all the variables and conclusions involved in the knowledge base.

ii) topologically sort the rules based on the order the conclusion occur in.

iii) plot all the paths in the knowledge base in which variables can be instantiated and conclusions reached (this is done in paragraph 20 in this report).

iv) use global variables to group the instantiation of the variables and predecessor conclusions involved in the hypotheses.

For an example that explains the forward chaining procedure consider the following abbreviated example using the river

(5)

If we add a rule or topic for obtaining the initial information we need to start the forward chaining, and a final rule or topic to generate the report and write it to a file we can now write down an outline or decision tree for what the expert system is trying to analyze.

1. Introduction		
1.1 River's name entered	RULE 1	
2. Draw conclusions		
2.1 Bar composition determined	RULE 2	RULE 3
2.2 Channel depth determined	RULE 4	
3.		
3.1 Report filed	RULE 5	

(6) In this case the one goal that the expert system is trying to satisfy is to generate the final report (which by the way should contain all the conclusions the forward chaining has generated)

If we call this one goal "forward chain"

We can now write down a flow diagram for the program to reach all the conclusions we want:

step 1

river's name

step 2

bar composition

step 3

channel depth

step 4

file a report containing all the conclusions reached

(7) We can write this rule to finish the forward chaining as:

```
RULE 5
IF previous steps complete
AND FILE results file footer
THEN stop
```

The problem at this point is that there hasn't been a condition channel flow information[19]:

(1). We have various attributes that describe the river channel flow geometry: among these are bar composition framework gravel, censored gravel, filled gravel, or matrix gravel, channel depth(a numerical value). Once we know these attributes we have a list of rules relating them from which we can draw inferences.

(2) We define a data structure:

```
ATTRIBUTE  The  bar composition
AND  channel depth
```

We will also need a string variable which hold's the river's name to write on the final report:

```
STRING   The river's name
```

(3) We can now organize this information in the following outline

1. River's name entered
2. Bar composition determined
3. Channel depth determined

(4) we can now write down the forward chaining rules involving these attributes and variables:

```
RULE 1
IF The river's name <>""
AND FILE the results
THEN River's name entered

RULE 2
IF Fines are present on the bar surface
THEN The bar composition IS matrix gravel
AND Bar composition determined

RULE 3
IF NOT Fines are present on the bar surface
AND Grains on the bar surface are interlocked with
voids
THEN The bar composition IS Framework Gravel
AND Bar composition determined

RULE 4
IF Channel Depth > 0
THEN Channel Depth determined
```

[19] The steps in the example are ordered according to the letters in the above procedures.