LEARNING SIMPLE ARITHMETIC

Pavel Brazdil
Department of Artificial Intelligence
University of Edinburgh
Scotland

Sussman (1973) has shown that strategies could be reordered according to experience. In his examples no conflicts were experienced after reordering. We shall illustrate that conflicts do arise and present a way of trying to overcome them. The program written (ELS) accepts a set of partially ordered Horn clauses as input and a sequence of problems to solve. The search tree obtained is examined in order to generate a priority ordering for the clauses used. If conflicts are detected, ELS tries to determine the conditions under which the clauses involved should be applied. For example, suppose the problem 3+2=X is to be solved and we are given the clauses:

assoc:	X1+(X2+X3)=X4 ← (X1+X2)+X3=X4.
subs:	X1+X2=X3 ← X1=XO, XO+X2=X3.
subz:	X1+X2=X3 ← X2=XO, X1+XO=X3.
sue:	O+1=1 ←. 1+1=2 ←. ...etc,
pred:	1=1+O ←. 2=1+1 ←. ...etc.

Generally, the problems are solved in a breadth first manner, but priority orderings between individual clauses are respected. First all clauses that match the given subgoal are chosen. In our case both the clauses 'subs' and 'subz' match the initial goal 3+2-X, as this goal matches X1+X2=X3. Both clauses are actually tried in the search, since initially there is no priority ordering specified between them. If it had been specified that 'subz<subs', only the clause 'subz' would have been tried.

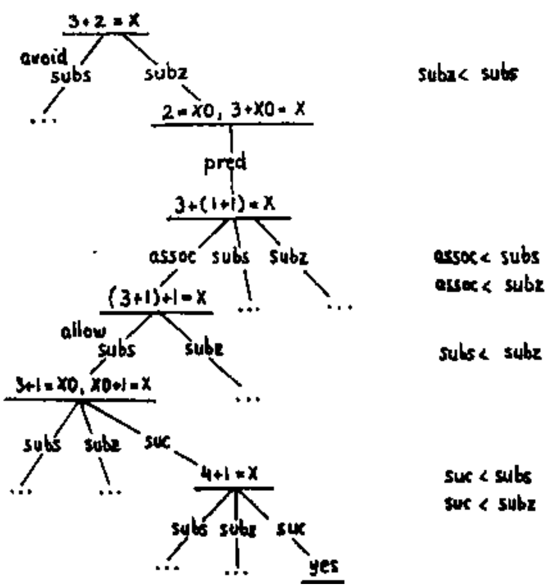

Figure 1. The search tree for 3+2=X. The new priority orderings obtained are shown on the right.

Figure 1 shows a part of the search tree produced by ELS when solving the goal 3+2=X. The search tree contains a number of choice points. Every time a choice is made ELS makes a note of which clause was given preference. After a successful solution has been reached (marked 'yes'), the preferences noted on the solution path are integrated with the existing priority orderings and then used to guide future searches.

The integration is not just a matter of adding the new orderings to the old ones as conflicting requirements can occur. In our example we have one such conflict. First we require 'subz<subs' and then 'subs<subz'. To resolve the conflict, ELS looks back into the search tree. It assumes that one of the variable assignments in the clause 'subs' (or 'subz') is critical. In order to make a good guess which is the critical assignment, it compares the application which lies on the solution path (marked 'allow') with an application which lies just off the solution path (marked 'avoid'). A new clause is generated on the basis of this comparison. Care is taken so that it would mismatch the application marked 'avoid', but further unnecessary specializations are avoided. In our case ELS generates the clause 'subs.l'. Thus:

allow subs:	(3+1)+1=X ← ...
avoid subs:	3+2=X ← ...
new subs.l:	(Y1+Y2)+X2=X3 ← ...

The new clause 'subs.l' is added to the original set of clauses together with the priority ordering 'subs.Ksubs' and 'subs.Ksubz', which replaces the priority ordering 'subs<subz'. The problem 3+2=X can now be solved without search. The knowledge thus acquired helps the system to solve a number of similar problems, such as 1+4=X and (2+l)+5=X, which are both solved without search. The work described here is currently being extended to the domain of simple equation solving.

Our work is also related to Waterman (1975). The key point in our approach is analysis of the past search for the purpose of conflict resolution. Initially only priority orderings are used to describe a solution path, but if a conflict is detected, the old search tree is revisited in order to pull out more information about the clauses involved in the conflict.

References

[1] Sussman, G.J. (1973) A Computational Model of Skill Acquisition, MIT Tech. Report AI TR-297, MIT, Cambridge.

[23 Waterman, D.A. (1975) Adaptive Production Systems, Proceedings of 4th IJCAI, Tbilisi, USSR, 1975.