Plan Variants and A Plan Refinement Algorithm

Dennis F. Kibler
Paul H. Morris

Technical Report 178

October 1981

Department of Information and Computer Science
University of California, Irvine

This research was performed under
contract N00123-81-C-1165 from the
Naval Ocean Systems Center.

ABSTRACT

The relational production model of problem solving, due to Vere, is used to analyze plan inefficiency. The concept of plan variant is introduced and precisely related to the triangle tables of Nilsson. The concept is used to extend certain classes of inefficiencies. A method is described which "tightens up" plans containing these inefficiencies, often producing optimal ones.

1. Introduction

A number of researchers have proposed techniques for automatic problem solving [3, 5, 7]. In general, these methods are not guaranteed to produce optimal solutions, i.e. plans with the fewest number of operations. In this paper we describe a process of plan improvement which in many cases of interest will convert a non-optimal plan to an optimal one.

Plan refinement is expected to have a number of uses. First it allows more latitude to plan generation programs: they need not be overly concerned with plan efficiency when that can be improved later. Second, it could be applied to user-supplied plans in a man-machine interactive system. Finally, it could be of value to learning systems by improving plans before generalization. There is little point in generalizing an inefficient plan when an efficient one can be used instead. More importantly, by filtering out inessential steps, plan refinement emphasizes what is truly significant. A side benefit of the analysis is that it exposes the non-essential ordering in a linear plan, supplying information that could be used to schedule operations in parallel.

To place our results in a general setting we use the relational production model of problem domains introduced by Vere [6]. Although Vere is primarily concerned with studies of generalization and inductive inference, the model itself is applicable to many areas of investigation. However, we find it convenient to make a slight modification: we use multisets in place of sets in the represention of states and operators. This aids the analysis and allows a simpler description of some operators.

2. Basic Definitions

As in Vere [6], a <u>literal</u> is an ordered list of terms $(t_1\ t_2\ \ldots\ t_r)$ where $r \geq 1$ and each term t_i is either a constant or a variable. We will regard t_1 as a relation name and write the literal in the form $t_1(t_2,\ldots,t_r)$. A <u>ground literal</u> is one in which all the terms are constants.

We represent a state of the problem domain by a multiset (or bag) of ground

literals. The operators are represented by relational productions, which have the form $\alpha \rightarrow \beta$ where α and β are multisets of literals. The literals in α are called preconditions while those in β are postconditions. An operation is an operator plus a substitution that instantiates its pre- and post-conditions to ground literals. An operation is applicable to a state if its preconditions are contained in that state (a multiset S is contained in a multiset T if for all x the number of occurrences of x in S is less than or equal to the number of occurrences of x in T). If the operation is applicable, it transforms the state into a new state obtained by subtracting the preconditions and adding the postconditions. We will say an operator is applicable to a state if it can be paired with some substitution to produce an applicable operation. A problem domain may now be defined as a pair consisting of an initial state and a finite set of operators. We say a state is reachable if it can be obtained from the initial state by a finite sequence of (applications of) operations.

In these definitions we differ from Vere by the use of multisets rather than sets. The difference is reduced (but not eliminated) by the following restriction. Let us say a multiset is setlike if it contains no duplicates. We define a unitary problem domain to be one where every reachable state is setlike. In the remainder of the paper, we will confine our attention to unitary problem domains. It may seem that we have renounced sets, only to reintroduce them. However, the requirement that the problem domain be unitary imposes a strong condition on operators which is used in the subsequent analysis and is conveniently stated in terms of multisets. Incidentally, the use of multisets appears to avoid the difficulty with composition of operators noted in [6].

EXAMPLE: Four Blocks World.

- description: The objects in the world are a table and four equally sized cubic blocks labelled a, b, c and d. The table may support any number of blocks, while each block may support at most one other block. We say a block is clear if it is not supporting another block.

- initial state: on(a,b), ontable(b), ontable(c), ontable(d), clear(a), clear(c), clear(d).

- operators:

  1. clear(X),ontable(X),clear(Y) → clear(X),on(X,Y).

  2. clear(X),on(X,Y),clear(Z) → clear(X),on(X,Z),clear(Y).

  3. clear(X),on(X,Y) → clear(X),ontable(X),clear(Y).

The above is an example of a unitary problem domain. To see this, let us call a state S in the four blocks world normal if it has the following three properties:

1. S is setlike.

2. on(X,Y) and ontable(X) are mutually exclusive in S (i.e. there is no substitution for X and Y such that both literals are simultaneously contained in S).

3. on(X,Y) and clear(Y) are mutually exclusive in S.

Clearly, the initial state is normal. Let us verify that the first operator preserves normality. The only literal added is on(X,Y). This cannot cause a duplication since ontable(X) is a precondition, and that excludes on(X,Y). The exclusion properties are also maintained since the literals excluded by on(X,Y) are deleted by the operator. Similar arguments show that the second and third operators preserve normality. Thus, all reachable states are normal, and hence setlike.

Note that the second operator does not need an $X \neq Z$ condition. Because we are using multiset containment of preconditions, clear(X) and clear(Z) must be matched to different occurrences of ground literals. Since the states are setlike, this ensures that X and Z cannot be instantiated to the same thing.

3. Plan Variants

In this section we introduce a theoretical construct of importance to our study of plan refinement. We require some preliminary definitions. Given a specific problem domain, a plan is a list of operations $(A_1 \ldots A_n)$ such that each A_i is applicable to the state obtained by starting with the initial state and applying the preceeding operations in order. Plans are generally in the service of goals. A goal is a multiset of literals. It may be thought of as a kind of operator with preconditions but no postconditions. A plan achieves a goal if the goal is applicable to the state obtained by starting with the initial state and applying all the plan operations in order. Now suppose A and B are operations in a plan, with A immediately preceeding B. We say A transposes with B if the list obtained by interchanging the positions of A and B is also a plan; we call such an interchange a transposition. Notice that a transposition leaves the effect of a plan unaltered, since the net effect of A followed by B is the same as B followed by A when both are applicable. We say a plan P is a temporal variant of a plan Q if it is obtained from it by a finite sequence of transpositions. Note that a plan is considered a temporal variant of itself; a temporal variant is called proper if it differs from the original.

For unitary problem domains there is an easily computed and plan-independent criterion for transposability. Let us say an operation A supplies a ground literal g if g is a postcondition but not a precondition of A (in STRIPS terminology [3], g is an "add" condition). It consumes g if g is a precondition but not a postcondition (in STRIPS terminology, a "delete" condition). Then adjacent operations A and B in a plan transpose if and only if A does not supply preconditions of B and B does not consume preconditions of A.

There is a precise relationship between temporal variants and an augmented version of the triangle tables of Nilsson [3], in unitary domains. Recall that in the triangle table, if A and B are operations in a plan with A

preceeding B, then the cell below A and to the left of B contains the literals (if any) that are supplied by A and survive through intervening operations to become preconditions of B. We may extend the triangle table to a "square table" where the cell above B and to the right of A contains any literals that are postconditions of A and survive to be consumed by B. We will write $A \propto B$ if at least one of the two cells is non-empty. Since $A \propto B$ only if A preceeds B in the plan, the relation $\propto$ does not have cycles. Let $\propto^*$ be the reflexive transitive closure of $\propto$. Clearly $\propto^*$ is a partial ordering. We have the following lemmas.

Lemma 1: If A immediately preceeds B, then $A \propto^* B \Leftrightarrow$ A does not transpose with B.

Proof: $\Rightarrow$: Suppose $A \propto^* B$. Then $A \propto B$. If A supplies preconditions of B, we are done. Otherwise, we must have B consumes postconditions of A. These must also be preconditions of A, since otherwise A would be supplying them. Therefore, A does not transpose with B.

$\Leftarrow$: Suppose A does not transpose with B. Then either A supplies preconditions of B, or B consumes preconditions of A. In the first case we are done. In the second case, the preconditions of A that B consumes must also be postconditions of A, since otherwise B's preconditions would not be satisfied. Thus $A \propto B$ and so $A \propto^* B$. ■

Lemma 2: If $A \propto B$ in a plan Q, then $A \propto B$ in the plan Q1 obtained from Q by a transposition.

Proof: The transposition cannot be of A with B since $A \propto B$. The cases remaining are

1. The transposition does not move an operation into the segment from A to B.

2. The transposition moves a single operation into the segment from A to B.

By definition, the operations initially between A and B do not consume the literals on which $A \propto B$ depends. They also do not supply them, since otherwise a reachable state would contain a duplicate. Thus case 1 does not affect $A \propto B$. In case 2, the operation moved in also does not supply or consume the literals on which $A \propto B$ depends: if it supplied such a literal, then a reachable state would contain a duplicate, contrary to our assumption of a unitary domain. If it consumed such a literal, then a precondition of B would be destroyed, disrupting the plan, i.e. the purported transposition would be invalid. ■

Lemma 3: If $A \propto^* B$ in a plan Q, then $A \propto^* B$ in the plan Q1 obtained from Q by a transposition.

Proof: Immediate.

Lemma 4: If $A \propto^* B$ is false, then it remains false after a transposition.

Proof: If a plan Q1 is obtained by transposing C with D in a plan Q, then interchanging D with C in Q1 is also a valid transposition. Thus, the result follows from the previous lemma. ■

Lemma 5: If $A \propto^* B$ is false, then it is false in every temporal variant.

Proof: Immediate.

The following theorem relates square tables to temporal variants.

Theorem 6: If A and B are operations in a plan Q, then $A \propto^* B \Leftrightarrow$ there is no temporal variant of Q in which A follows B

Proof: $\Rightarrow$: It suffices to show that if $A \propto B$ then there is no temporal variant in which A follows B. Suppose there is. Since a temporal variant is obtained by a sequence of interchanges of adjacent elements, A must "pass" B at some stage, i.e. A must transpose with B. But this conflicts with $A \propto B$.

$\Leftarrow$: Suppose $A \propto^* B$ is false. We will show that there is then a temporal variant such that A follows B and, moreover, the positions of the operations originally to the left of A and to the right of B are undisturbed. The proof is by induction on the distance between A and B. Suppose first the distance is 1, i.e. A is adjacent to B. In this case we may simply transpose A with B to achieve the desired result. Suppose the distance is n, where $n > 1$. Let C be an operation intermediate to A and B. Then either $A \propto^* C$ is false, or $C \propto^* B$ is false. The proof is similar in both cases. We will consider the case where $A \propto^* C$ is false. Then the inductive hypothesis can be applied to A and C to produce a temporal variant where A follows C, and where the operations originally to the left of A and the right of C are undisturbed. It follows that A is moved closer to B. Then lemma 5 allows us to apply the inductive hypothesis to A and B to give the desired result. ■

The relation $\propto^*$ describes the ordering constraints within a plan that arise from interactions among the operations. Plan graphs [2] were an earlier attempt to characterize these. Plan graphs, however, (as defined in [2]) provided fewer opportunities for parallelism, and were less directly related to triangle tables. The desire to obtain a precise relationship with triangle tables suggested a formal approach, and led to the present work.

4. Inefficiencies

We now turn our attention to removable plan inefficiencies. First we will define certain obvious kinds, called <u>blatant</u> inefficiencies. The concept of temporal variant will then be used to extend each definition to a wider class.

We distinguish the following types of blatant inefficiency:

1. A state repeats during the execution of the plan.
2. A consecutive subsequence of operations in the plan is equivalent to a single operation.

Notice that these are independent of the goal. Thus, it is possible to exclude them dynamically from search paths in a planning system. In [1] a system is described that induces metarules designed to avoid inefficiencies of these types. However, the system does not deal with the other forms of inefficiencies discussed below.

Methods exist to detect and remove direct occurrences of these inefficiencies. Type 2 is somewhat expensive to treat, requiring $O(mn^2)$ for the obvious algorithm, where n is the length of the plan and m is the number of operators.

At this point we introduce "guilt by association." Intuitively, if a plan is inefficient then so are its temporal variants. This leads us to define: a plan has a <u>disguised</u> inefficiency of type i if it is a proper temporal variant of a plan with a blatant inefficiency of type i.

A third type of inefficiency involves operations that do not contribute either to the goal or subsequent operations. We say an operation is <u>redundant</u> if it fails to supply literals which are either preconditions of subsequent operations or are part of the goal. Redundant operations may be summarily dismissed from a plan. Notice that when one redundant operation is removed, an earlier operation that was supplying it may then become redundant in turn. The concept of temporal variant is not useful in extending this class.

EXAMPLES: We will denote the operators in the Four Blocks World (described

above) by move(X,table,Y), move(X,Y,Z) and move(X,Y,table), respectively. We assume the same initial state. Imagine the goal is on(c,d) and on(d,b). Then the plan

move(c,table,d), move(c,d,table), move(a,b,table), move(d,table,b), move(c,table,d)

has a blatant inefficiency of type 1 (the initial state repeats after the second operation), while its temporal variant

move(c,table,d), move(a,b,table), move(c,d,table), move(d,table,b), move(c,table,d)

has a disguised inefficiency of type 1, but no blatant inefficiency. The plan

move(d,table,c), move(a,b,table), move(d,c,b), move(c,table,d)

has a disguised inefficiency of type 2 (there is a temporal variant where the first and third operations become adjacent and reduce to move(d,table,b)). Finally, the plan

move(a,b,table), move(b,table,a), move(d,table,b), move(c,table,d)

has an inefficiency of type 3 (the second operation is redundant).

5. Algorithm

We are now ready to outline the plan improvement algorithm:

> First, inefficiencies of type 3 are repeatedly removed until none remain. Next, the top level procedure removes inefficiencies of types 1 and 2, one at a time, until there are none left. The sub-procedure for removing a single inefficiency of type 1 or 2 generates all temporal variants until it finds one with a blatant inefficiency. If such is found, it is removed and the top level procedure continues with this new plan. If none is found, the subordinate procedure fails and the current plan is returned as the output.

A program implementing the above algorithm has been written in PROLOG [4], a language that provides facilities for non-determinism. Temporal variants are generated by a non-deterministic algorithm which resembles an insertion sort in which transposability is used to decide where an element can be inserted.

6. Limitations

It should be pointed out that the foregoing results do not exclude the possibility that a plan could be a permutation of another plan, and equivalent to it, without being a temporal variant. For example, if the Four Blocks World is augmented by imagining a hand that picks up the blocks and repositions them, we might represent the operators as follows:

1. clear(X),ontable(X) → holding(X)
2. clear(X),on(X,Y) → holding(X),clear(Y)
3. holding(X) → clear(X),ontable(X)
4. holding(X),clear(Y) → clear(X),on(X,Y)

Any one of our previous sample plans could now be expressed by converting every move operation to a pair of "pickup" and "putdown" operations. It is not hard to see that the transformed plan has no proper temporal variants! This is because a pickup operation will never transpose with a putdown operation (or vice versa). In the new representation, what were formerly temporal variants continue to be equivalent plans, but they now fall outside the scope of our algorithm. Thus, facilitating transpositions can be an important issue in representing problem domains. Another way around this would be to transpose consecutive subsequences, rather than single operations. This is an area for future study.

There is another reason why a plan output by our system may be less than optimal. In certain domains, it may be possible to replace a consecutive subsequence of operations by a shorter subsequence, even though no subsequence can be replaced by a single operation (this appears not to be the case for the blocks world). Of course, our algorithm could be extended to deal with this case, but the computational cost would rapidly become infeasible. It may even be the case in some domains that an improved plan can be obtained by first replacing a subsequence by a longer subsequence, thereby facilitating other substitutions which eventually lead to a shorter plan. We do not intend to consider such combinatorially explosive schemes.

7. Conclusions

Notwithstanding the limitations mentioned, we have proposed an algorithm for plan improvement which is relatively inexpensive, and which leads to considerable improvement in practical situations. We have indicated the applications of such a system. The analysis of plan structure contained herein should also be of general importance in the study of plans. The advantages of precise analysis in confirming relationships and indicating pitfalls suggest that a judicious degree of formalization can be helpful for an understanding of Artificial Intelligence systems.

REFERENCES

[1]
Anzai, Yuichiro.
How One Learns Strategies: Processes and Representation of Strategy Acquisition.
In Proceedings of AISB/GI Conference on Artificial Intelligence, pages 1-14. Hamburg, 18-20 July, 1978.

[2]
Kibler, D. and P. Morris.
Dont be Stupid.
In Seventh International Joint Conference on Artificial Intelligence, pages 345-347. Univ. of British Columbia, Vancouver, B.C., Canada, August, 1981.

[3]
Nilsson, N.J.
Principles of Artificial Intelligence.
Tioga, 1980.

[4]
Pereira, L.M., F.C.N. Pereira, and D.H.D. Warren.
Users Guide to DECsystem-10 Prolog.
Technical Report, Dept. of Artificial Intelligence, Univ. of Edinburgh, September, 1978.
version 1.32.

[5]
Sacerdoti, E.D.
A Structure for Plans and Behavior.
Elsevier, 1977.

[6]
Vere, S.A.
Relational Production Systems.
Artificial Intelligence8:47-68,1977.

[7]
Warren, D.H.D.
WARPLAN: A System for Generating Plans.
Memo 76, Dept. of Computational Logic, School of Artificial Intelligence, University of Edinburgh, June, 1974.

References

[ALS79] Anderson, T., P.A. Lee, and S.K. Shrivastava. System Fault Tolerance, in T. Anderson and B. Randell (eds.) Computing Systems Reliability, Cambridge University Press, 1979.

[AVI75] Avizienas, A. "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing," Proceedings of the Int. Conference on Reliable Software, SIGPLAN Notices, vol. 10, no. 6, 1975.

[CAR79] Carter, W.C. "Hardware Fault Tolerance," in T. Anderson and B. Randell (eds.) Computing Systems Reliability, Cambridge University Press, 1979.

[CA78] Chen, L. and A. Avizienis. "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," Eighth Int. Conf. on Fault-Tolerant Computing, Toulouse, France, June 1978.

[CHE80] Cheung, R. C. "A User-Oriented Software Reliability Model," IEEE Trans. on Software Engineering, vol. SE-6, no. 2, March 1980.

[ELM72} Elmendorf, W.R. "Fault-Tolerant Programming," Digest of 1972 Int. Symp. on Fault-Tolerant Computing, pp. 79-83.

[GH75] Gannon, J.D. and J.J. Horning. "Language Design for Programming Reliability," IEEE Trans. on Software Engineering, vol. SE-1, no. 2, June 1975, pp. 179-191.

[GS78] Gannon, T.F. and S.D. Shapiro. "An Optimal Approach to Fault Tolerant Systems Design," IEEE Trans. on Software Engineering, vol. SE-4, no. 5, Sept. 1978, pp. 390-409.

[GY76] Gerhart, S.L. and L. Yelowitz. "Observations on the Fallibility in Applications of Modern Programming Methodologies," IEEE Trans. on Software Engineering, vol. SE-2, 1976, pp. 195-207.

[GOO75] Goodenough, J.B. "Exception Handling: Issues and a Proposed Notation," CACM, vol. 18, no. 12, Dec. 1975, pp. 683-696.

[HEC76] Hecht, H. "Fault-Tolerant Software for Real-

Time Applications," Computing Surveys, vol. 8, no. 4m Dec. 1976.

[HEC79] Hecht, H. "Fault-Tolerant Software," IEEE Trans. on Reliability, vol. R-28, no. 3, Aug. 1979.

[KY75] Kane, J.R. and S.S. Yau. "Concurrent Software Fault Detection," IEEE Trans. on Software Engineering, vol. SE-1, no. 1, March 1975.

[LIP79] Lipow, M. "Prediction of Software Errors," Journal of Systems and Software, 1, 1979 pp. 71-75.

[LS79] Liskov, B.H. and A. Snyder. "Exception Handling in Clu," IEEE Trans. on Software Engineering, vol. SE-6, no. 6, Nov. 1979, pp. 546-558.

[LIT80] Littlewood, B. "Theories of Software Reliability: How Good Are They and How Can They be Improved," IEEE Trans. on Software Engineering, vol. SE-6, no. 5, Sept. 1980.

[MSB75] Melliar-Smith, P.M. and B. Randell. "Software Reliability: The Role of Programmed Exception Handling," Proc. Int. Conf. on Reliable Software, SIGPLAN Notices, vol. 10, no. 6, June 1975.

[MUS75] Musa, J.D. "A Theory of Software Reliability," IEEE Trans. on Software Engineering, vol. SE-1, no. 3, Sept. 1975.

[PAR77] Parnas, D.L. "The Use of Precise Specifications in the Development of Software," IFIP-77, North Holland, 1977.

[Ram] Ramamoorthy, C.V., F.B. Bastani, J.M. Favaro, Y.R. Mok, C.W. Nam, and K. Suzuki. "A Systematic Approach to the Development and Validation of Critical Software for Nuclear Power Plants."

[RAN75] Randell, B. "System Structure for Software Fault Tolerance," Proc. Int. Conf. on Reliable Software, SIGPLAN Notices, vol. 10, no. 6, June 1975.

[RAN79] Randell, B. "System Reliability and Structuring," in T. Anderson and B. Randell (eds.), Computing Systems Reliability, Cambridge University Press, 1979

[RAU] Rault, J.C. "An Approach Towards Reliable Software,"

[SW78] Schick, G.J. and R.W. Wolverton. "An Analysis of