April 1970

SOME REMARKS ON RESOLUTION STRATEGIES

by

Robert E. Kling

Artificial Intelligence Group
Technical Note 28

SRI Project 8259

The research reported here was supported by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS12-2221.
SOME REMARKS ON RESOLUTION STRATEGIES

The following comments are excerpted from a letter I wrote discussing some problem-dependent aspects of resolution-logic theorem proving. The underlying focus of these remarks is the nature of the information that a user needs to specify for a problem-oriented strategy to be employed by a multi-strategy system. However, I was more concerned with summarizing some of my experiences and with articulating certain questions than with reaching particular conclusions about the necessary ingredients for such a language, let alone deriving a preliminary language design. Nevertheless, some of these comments seem to have been interesting to other readers and may benefit from a wider circulation.

I'd like to classify the various strategies I know, some of which we currently use on QA 3.5 under three headings, and then describe a few additional details of strategic nature which aren't subsumed in this classification.

Attention (Ordering) Strategies:

(Which clause pair shall we consider next?)

(1) Unit Preference (Wos) - QA3 - A classical strategy.

(2) Preference Set (Kling) - QA3 - Partition memory, allowing some axioms likely to be used in a given proof or "preferred status." Draw in other axioms only if the preferred axioms don't lead to proof (by a preset level, number of search nodes, or other criterion). This is independent of unit preference.

(3) Splitting (Slagle) - Similar to backwards chaining. It entails developing an AND/OR subgoal tree in which goals are satisfied by resolution deductions through the tree to □. (Might be compatible with modified unit preference at each level of tree search, but it's really motivated by radically different considerations.)

Selection Strategies:

(Given a set of clauses, shall they be resolved?)

(1) T-Support (Wos) - QA3 - Classical.

(2) Hillclimbing (Green) - QA3 - Need a metric for the space.
(3) Predicate Filter (Kling) - QA3 - Accept a clause iff all its predicates are on a "filter list." (A crude way to trim the database by "subject area" and relations likely to enter into a given proof—very useful with analogies.)

(4) Indicator Test (Kling) - QA3 - A helpful strategy for neglecting irrelevant but pregnant clauses in second-order domains like algebra. Some predicates—e.g., group[g;*] or map[f;x;y]—have "indicators"—e.g. the group operator *, the map function f, etc. Some clauses contain these "second-order predicates" and fully first-order predicates—e.g.,

\[ \text{subgroup[h;g;*]} \lor \text{subset[h;g]} \]

These clauses will not be resolved on the set predicates (in, subset, factorset, intersection, etc.) until all the indicators in the clause are fully instantiated.

(5) Ancestry Filter (Luckham) - QA3 - See his paper from Stanford AI Project.

(6) Merging (Andrews) - QA3 - See his paper in JACM.

(7) Length Plus Level Bound - QA3 - Set in advance.

(8) Term Depth - QA3 - Set in advance.

Deletion Strategies:

(Given a resolvent, should it be added to the active search tree?)

(1) Doublearestest (Kling) - Do not accept a resolvent unless it resolves with at least one node in the developed search space. (Actually more elaborate, but an incomplete strategy. Resolvents that fail this criterion must be saved and tried again, or used themselves if no double resolvents exist.)

(2) Subsumption (Robinson) - QA3.

(3) Forbidden States (Kling) - QA3 - Some values of state variables are "forbidden" on semantic grounds, and clauses containing states with these values are edited out.

(4) Answer-Units Only (Green) - QA3 - Keep clauses which only have an answer-clause which is a unit.

Other:

(1) Predicate Evaluation (Green) - QA3 - Associate a LISP form with a predicate so that the literal may be evaluated to T, NIL, or even return a different literal.
Function Evaluation (Green) - QA3 - Associate certain functions with LISP forms, for evaluation.

I want to provide a simple example about the strategic use of axioms and how we simplify some of our searches by clever tricks. Since the example I develop uses predicate and function evaluations, I'll develop an example of predicate evaluation first.

Predicate evaluation has turned out to be an effective way to embed semantic information in a comparatively efficient way. Cordell's paper "Application of Theorem-Proving Techniques to Problem Solving" describes our state-transformation approach to problem solving. Consider a simple robot situation in which we want the robot to reach a box which is situated on a platform. To reach the platform top, the robot must roll up a wedge. (This is a loose analogue of the monkey-bananas problem.) A simple axiomatization may include the following axioms:

A1. \( \forall (r \ w \ p \ & \ s) \ on[r;l;s] \land on[w;l;s] \land on[p;l;s] \land at[w;p;s] \land at[r;p;s] \rightarrow on[r;p;rollup[r;w;s]] \)

A2. on[robot;floor;initial-state]

A3. on[platform1;floor;initial-state]

A4. on[wedge1;floor;initial-state]

A5. on[box1;platform1;initial-state]

Axiom A1 states that if the robot, wedge, and platform are all on level \( l \) and together in state \( s \), then the robot can get onto the platform by rolling up the wedge. Now other action axioms are needed to develop a state \( s \), such that at[robot;platform1;s] and at[wedge1;platform1] are true. But A3 or A4 could clash with a partially instantiated derivative of A1 to yield

\[ on[robot;wedge1;rollup[r;w;s]] \]

or

\[ on[robot;p;rollup[r;platform1;s]] \]

etc.

One way of eliminating semantically senseless clauses like these is to use predicate evaluation for embedding types. For example:

A1'. \( \forall (r \ w \ p \ & \ s) \ on[r;l;s] \land on[w;l;s] \land on[p;l;s] \land at[w;p;s] \land at[r;p;s] \land wedge[w] \land platform[p] \rightarrow on[r;p;rollup[r;w;s]] \)
If wedge[x] and platform[x] are associated with evaluable LISP forms that are T or NIL for appropriate (ground) arguments, then semantically senseless clauses will evaluate to T and be deleted. Semantically sensible clauses will be acceptable and the additional type literals will "disappear" after evaluation (to NIL).

Function evaluation is also helpful in embedding some semantics into our axiomatizations. We have a function \( p[x] \) which will unify (thus resolve) or subsume with any permutation of the argument list \( x \). Thus in geometry, we will say \( \text{triangle}[p(A B C)] \) instead of \( \text{triangle}[A;B;C] \) and

\[
\forall (x y z) \ \text{triangle}[x;y;z] \rightarrow \text{triangle}[z;x;y] \wedge \text{triangle}[y;z;x] 
\]

Consider the following theorem: The intersection of two Abelian groups is Abelian--i.e.

\[
\text{int}[C;A;B] \wedge \text{abelian}[^1][A] \wedge \text{abelian}[^1][B] \rightarrow \text{abelian}[^1][C]
\]

In proving this theorem, if one knows that the intersection of two groups is a group, one should easily derive \( \text{group}[^1][C] \). So in the course of proving this theorem the following clauses may appear in the search space:

A1. \( \neg \text{group}[^*][z] \vee \neg \text{int}[z;x;y] \vee \neg \text{group}[^*][x] \vee \neg \text{group}[^*][y] \)

C1. \( \text{group}[^1][A] \)

C2. \( \text{group}[^2][B] \)

Four resolvents may be derived:

R1. \( \text{group}[^1][z] \vee \neg \text{int}[z;A;A] \)

R2. \( \text{group}[^1][z] \vee \neg \text{int}[z;B;B] \)

R3. \( \text{group}[^1][z] \vee \neg \text{int}[z;A;B] \)

R4. \( \text{group}[^1][z] \vee \neg \text{int}[z;B;A] \)

We really want either R3 or R4, and R1 and R2 are genuinely spurious.

If we associate an evaluable form \( \text{int}[x;y;z] \) with intersection \( [x;y;z] \), then we can easily throw away a clause that contains a term of the form \( \text{int}[z;x;x] \). By using the function \( p[x] \), described above, in a new axiomatization, R3 and R4 may be compressed into a single clause \( z \) which would be like

R5. \( \text{group}[^1][z] \vee \neg \text{int}[z;p((A B))] \)

This joint use of evaluation procedures is merely a clever tactical device. The unsolved problems that we face here include various ways of specifying the use of a particular axiom or lemma. Often a unit lemma is used once, to resolve with a particular axiom, and is then forgotten. Often an axiom, such as the preceding relationship between a pair of groups and their intersection, is quickly resolved to some simple form--e.g. R5. Or R5 is resolved
with the premise \( \text{int}[C;p[(A \lor B)]] \) and is forgotten. Rarely are intermediate results like \( R5 \), or even \( \text{group}[z;*1] \lor \neg \text{int}[z;p[(A \lor B)]] \lor \neg \text{group}[y;*1] \) used on more than one line of reasoning. But we aren't sure how to specify the use of an axiom (or sequence of axioms) to focus on their key results (maximal resolvents) in a way we can specify as a problem-dependent user-supplied strategy.

Another problem-dependent strategy arises in our "three box" problem. Initially we have three boxes dispersed over a room, and the position of each box is noted—e.g., \( \text{position}[A;p;\text{initial-state}] \) (Box A is initially at \( p_1 \)). We then ask QA3 to find a sequence of actions that will bring the boxes together. The problem statement is

\[ \exists(s_f, p) \text{ position}[A;p;s_f] \land \text{position}[B;p;s_f] \land \text{position}[C;p;s_f] \]

A wise problem solver would fix the initial positions of A, B, or C as the target position and move the other boxes. QA3, working with breadth-first search, sets up each of A, B, and C as a target position and problem solves in parallel. What kind of executive do we need to automatically restate the problem as

\[ \exists(s_f) \text{ position}[A;p_1;s_f] \land \text{position}[B;p_1;s_f] \land \text{position}[C;p_1;s_f] \]

or to shift the style of search on the initial problem?