April 1982

Report. No. STAN-CS-82-903

Also numbered:
AIM-346

Coloring Maps and the Kowalski Doctrine

by

John McCarthy

Department of Computer Science

Stanford University
Stanford, CA 94305

COLORING MAPS AND THE KOWALSKI DOCTRINE

by John McCarthy

Abstract: It is attractive to regard an algorithm as composed of the logic determining what the results are and the control determining how the result is obtained. Logic programmers like to regard programming as controlled deduction, and there have been several proposals for controlling the deduction expressed by a Prolog program and not always using Prolog's normal backtracking algorithm. The present note discusses a map coloring program proposed by Pereira and Porto and two coloring algorithms that can be regarded as control applied to its logic. However, the control mechanisms required go far beyond those that have been contemplated in the Prolog literature.

COLORING MAPS AND THE KOWALSKI DOCTRINE'

John McCarthy, Stanford University

Kowalski (1979) enunciated the doctrine expressed by the formula

1. program = logic + control

The formula isn't precise, and it won't be precise until someone proposes a precise and generally accepted notion of how control is to be added to an expression of the logic of **a** program. Nevertheless, the idea is attractive, and I believe it can be made to work for some interesting class of programs. It is analogous to my comparison of epistemology and heuristics or Chomsky's competence and performance.

Pereira and Porto (1980) give a logic program for coloring planar maps with four colors and discuss how "selective backtracking" can reduce the search involved in coloring a map from that done by a straightforward PROLOG execution of the same program.

The discussion by Pereira and Porto treats coloring maps purely as an example of logic programming, and the improvements they discuss apply to all logic program systems. We shall consider two mathematical ideas about map coloring that go back to Kempe (1879), the paper containing the original false proof of the four color theorem. While Kempe's proof was false, the ideas are good and were used in almost all subsequent work including the recent successful proof.

The question is whether an algorithm involving these ideas can be regarded as a form of control adjoined to the basic logic program or whether they necessarily involve a new program. If they are to be regarded as control structures, it is not yet clear how they are best expressed. Of course, it is not hard to write a completely new program in PROLOG or any other language expressing the algorithms, and this has been done. The interpreted programs color a map of the United States. However, it is also interesting to try to regard the algorithms as control attached to the Pereira-Porto logic.

2. The Pereira-Porto logic program

There are two parts. The first expresses that the adjacent countries must have different colors by listing the pairs of colors that may be adjacent. We have

'This research was supported by the National Science Foundation under research grant no. MCS81-04877.

3. Reducing the map

Kempe (or perhaps someone still earlier) noticed that countries with three or fewer neighbors present no problem. No matter how the rest of the map is colored, there is always a **color** available for such a country. This suggests "reducing the map" by removing such countries and doing our trial-and-error coloring on the reduced map, confident that once the reduced map is colored, the coloring can be extended to the omitted countries.

The idea is even more powerful, because eliminating countries with three or fewer neighbors may remove enough neighbors from some other countries so that they have three or fewer neighbors and can themselves be removed. Therefore, the reduction process should be continued until a completely reduced map is obtained in which all countries have at least four neighbors. The maps of the states of the U.S. and the countries of Europe, Asia, Africa and South America all reduce to null maps when countries with three or fewer neighbors are successively eliminated

Likewise the map of Figure 1 reduces to the empty map. Thus we may remove country 4 with two neighbors and country 5 with three neigbors. This leaves all the remaining countries with three or fewer neighbors, so the second cycle of reduction leaves the null map, reduced map. Therefore, when we colored in the reverse order 1, 2, 3, 6, 4, 5, each country is colored without changing the color of any previously colored country.

If the programmer performs this reduction before he writes the goal statement, he will write

```
goal(R1,R2,R3,R4,R5,R6) ← next(R1,R2),next(R1,R3),
   next(R1,R6),next(R2,R3),next(R2,R6),next(R3,R6),
   next(R2,R4),next(R3,R4),next(R1,R5),next(R2,R5),
   next(R5,R6).
```

This PROLOG program will run with only the most local backtracking. Namely, after *R1* **has** been chosen arbitrarily, several values will have to be tried for each of the variables *Rl, R2, R3, R6, R4,* and *R5,* but once a value has been found that is compatible with the previously determined variables, it won't have to be changed again.

The new PROLOG program is logically equivalent to the previous program because it is just **a** rearrangement of the literals of a conjunction. However, the programmer has done the control. The interesting question is whether the reduction can be expressed in some **way that** can be regarded as adding control to the original logic, i.e., without changing the original logic.

a. *next(yellow,blue).*
next(yellow,red).
next(yellow,green).
next(blue,yellow).

etc. for all pairs of distinct colors.

The remaining PROLOG statement is distinct for each map. For the map of Figure 1, which they use as an example,

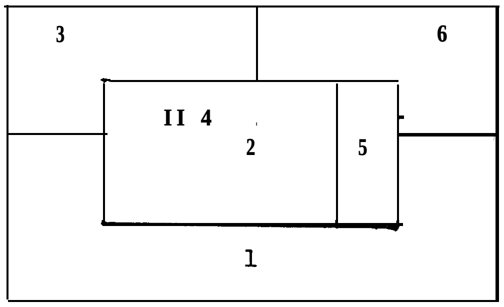

Figure 1

it is

b. *goal(R1,R2,R3,R4,R5,R6) ← next(R1,R2),next(R1;R3),next(R1,R5),*
next(R1,R6),next(R2,R3),next(R2,R4,next(R2,R5),next(R2,R6),
next(R3,R4),next(R3R6),next(R5,R6).

where each literal expresses the fact that a particular pair of adjacent regions must be compatibly colored.

Pereira and Porto give a trace of the execution of the program by standard depth first PROLOG. They point out that when an attempt to satisfy a literal fails, because the two adjacent regions mentioned have been assigned the same color, standard PROLOG will take back the most recent assignment of a color even if the region most recently colored was neither of those involved in the incompatibility. Their intelligent backtracking will change the color of one of the regions giving the Incompatibility.

An outsider to logic programming may react unsympathetically and comment that this is just one more example of a logic programming system, with its standard way of doing searches, tripping over its own feet. However, we should also recall that brief and easy statement of the PROLOG program for the coloring and not give up this virtue without a fight.

Nevertheless, "intelligent backtracking" doesn't make (1) and (2) into a good coloring program. Indeed we shall argue that it doesn't even do full justice to the logic of the program. To see this we need two ideas of Kempe.

4. Kempe transformations

Another idea of Kempe's can be used to strengthen the reduction process, but regarding it as mere control added to the original logic program seems even harder.

The strengthened reduction procedure also removes countries with four neighbors so that the reduced map contains only countries with five or more neighbors. The validity of this reduction depends on the following Kempe proof that if we have colored a partial map and want to add a country with four neighbors, we can always revise the coloring of the partial map to permit coloring the four neighbor country.

If fewer than four colors have been used to color the neighbors, there is no problem, so suppose that the four neighbors have been colored with four different colors as shown in Figure 2.

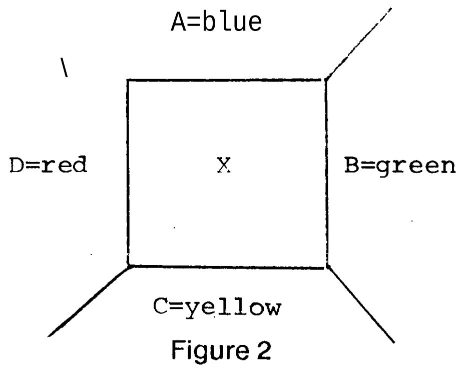

Figure 2

Consider the set S of all countries that can be reached from the blue country A on top of Figure 2 by a path going through only blue and yellow countries. S is called the blue-yellow Kempe component of country A. There are two cases. Either it contains country C or not. If not, we recolor the partial map by reversing blue and yellow on all countries in S. This still leaves the partial map properly colored. Since S does not contain C, C remains yellow while A has become yellow. This makes blue available to extend the coloring to X.

In the other case, S contains C, i.e, there is a chain of adjacent countries from *A* to C each of which is colored blue or yellow. Then there cannot be a red-green chain from *B* to *D* (by the topology of the plane or sphere), so that a red-green Kempe transformation applied to the red-green Kempe component of *D* will make *D* green, leaving red available to color X.

The fact that a blue-yellow path from A to C blocks a red-green path from *B* to *D* is where we have used the fact that the map is on a plane or sphere.

This justifies eliminating countries with four neighbors in the reduction process. If we have colored a partial map and want to add a country with four neighbors, we can do so, but we may have to modify the previous coloring by means of a Kempe transformation.

Our improved coloring algorithm then reduces the map by repeatedly dropping countries with four or fewer neighbors, colors the reduced map exhaustively, and then **colors** the dropped countries in the reverse order using Kempe transformations when necessary.

5. Realizing the reduction algorithm by cont rol of the Pereira-Porto logic

From the point of view of logic programming, successively reducing the **map by postponing** countries with three or fewer neighbors is an example of a more general **notion -** that of a postponable variable. A variable in the body of a clause is postponable if, no matter how the other variables are assigned, there is a value for this variable that **causes** all the goals involving that variable to be satisfied. Clearly any postponable variable can **be** postponed to the end. Moreover, just as in the map coloring problem, postponing some variables may remove enough goals involving other variables so that they in turn become postponable.

If there were only one stage of postponement, we could regard postponement as a case of selecting the first goal to be attempted, the postponable variables being rejected for selection. However, this wouldn't prevent the selection of a variable postponable in the second stage. Therefore, the postponement process should be completed before any goals **are** selected for attempt.

The postponability of a variable is expressed by a postponement lemma. For example, the postponability of *R4* is expressed by the formula

$$\forall R2\ R3.3\ R4.\ (next(R2,R4) \wedge next(R3,R4)).$$

Notice that our quick recognition of the postponability of *R4* is based on the symmetry. We say that whatever colors are assigned to *R2* and *R3,* a compatible color can be found for *R4.* We don't have to enumerate the possible assignements to *R2* and *R3*. A program would have to do more work unless it also discovered or was told that coloring problems are invariant when the names of the colors are permuted.

We can imagine several combinations of programmer and computer effort in postponing variables. We already discussed the case in which the programmer himself re-ordered the goals in the clause. The other extreme is that the PROLOG compiler attempt to prove postponability lemmas. Since some cases of postponability may depend on some variables already having values, additional postponements can be accomplished by a suitable interpreter. Since most variables in most programs are not postponable, it seems wasteful to have the interpreter always try for postponement. Therefore, it is also possible for the user to specify that the compiler and/or run-time system look for postponable variables, perhaps by enclosing the clause or part of it within which postponable variables **may be expected within a** macro. Thus the above program might be written

goal(R1,R2,R3,R4,R5,R6) ← postpone(next(R 7, R2),next(R1,R3),...,etc.).

The most powerful way of achieving postponement is for the programmer to use the full power of PROLOG to transform the body. Alain Colmerauer (1981) wrote such a program for rewriting the Pereira-Porto coloring program. If the programmer can arbitrarily **rewrite the** program, he may change the logic as well as the control. However, we can imagine that **a** restricted set of re-arrangement operators are used that is guaranteed to only affect the control.

I was informed by Hervé Gallaire that the system for specifying control described in (Gallaire 1981) could not express the postponement heuristic for the coloring problem, but that a small modification to the system would make it possible.

6. Realizing the Kempe transformation algorithm

Realizing the Kempe transformation algorithm as control of the Pereira-Porto logic presents a more difficult challenge to the designers of control languages for logic programming. Of course, the postponement part of the algorithm is the same as before; the difficulty comes when it is necessary to color a country with four differently colored neighbors.

The first step is to identify opposite neighbors of the four neighbor country. This depends not merely on the fact that the map is planar but on the actual imbedding in the plane. This information has been discarded when the map is represented as a graph. If the graph is described by giving for each country a list of its neighbors, the imbedding information can be including by listing the neighbors in cyclic order - clockwise or counterclockwise. Otherwise, it can be restored in general only with difficulty. Figure 3 shows cases where this **isn't** trivial. Of course, we can modify the algorithm to try every pair of vertices to see if **they are** unconnected by a path of their two colors. The above argument shows that this is **guaranteed** to succeeed but presumably at somewhat greater cost than if the cyclic order is known.

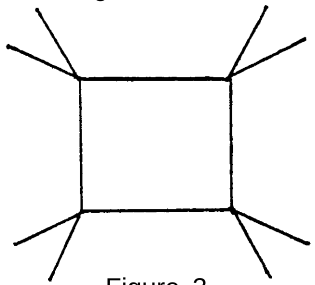

Figure 3

Looking for a changable country is a process of search whereby only certain values are allowed for certain variables and goals that become unsatisfied are re-satisfied by changing only certain variables in certain ways. A good control system for logic programs should permit the expression of such strategies.

7. References

Colmerauer, Alain (1981) personal communication

Kempe, A.B. (1879) “On the geometrical problem of four colors”. Amer. J. Math. 2 (1879), 193- 204.

Gallaire, Hervé (1981) personal communication

Kowalski, Robert (1979) Logic for Problem Solving, North-Holland.

Pereira, Luis Moniz and Antonio Porto (1980) Selective Backtracking for Logic Programs, Departamento de Informatica, Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, 1899, Lisboa, Portugal