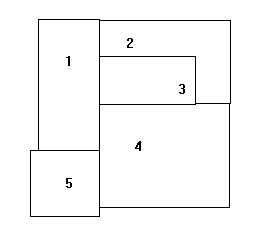
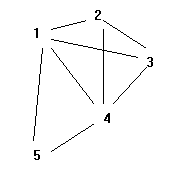
A famous problem in mathematics concerns coloring adjacent planar regions. Like cartographic maps, it is required that, whatever colors are actually used, no two adjacent regions may not have the same color. Two regions are considered adjacent provided they share some boundary line segment. Consider the following map.

  
Fig. 2.1.1

We have given numerical names to the regions. To represent which regions are adjacent, consider also the following graph.

  
Fig. 2.1.2

Here we have erased the original boundaries and have instead drawn an arc between the names of two regions, provided they were adjacent in the original drawing. In fact, the adjacency graph will convey all of the original adjacency information. A Prolog representation for the adjacency information could be represented by the following *unit* clauses, or facts.

adjacent(1,2). adjacent(2,1).

adjacent(1,3). adjacent(3,1).

adjacent(1,4). adjacent(4,1).

adjacent(1,5). adjacent(5,1).

adjacent(2,3). adjacent(3,2).

adjacent(2,4). adjacent(4,2).

adjacent(3,4). adjacent(4,3).

adjacent(4,5). adjacent(5,4).

If these clauses were loaded into Prolog, we could observe the following behavior for some goals.

?- adjacent(2,3).

yes

?- adjacent(5,3).

no

?- adjacent(3,R).

R = 1 ;

R = 2 ;

R = 4 ;

no

One could declare colorings for the regions in Prolog also using unit clauses.

color(1,red,a). color(1,red,b).

color(2,blue,a). color(2,blue,b).

color(3,green,a). color(3,green,b).

color(4,yellow,a). color(4,blue,b).

color(5,blue,a). color(5,green,b).

Here we have encoded 'a' and 'b' colorings. We want to write a Prolog definition of a conflictive coloring, meaning that two adjacent regions have the same color. For example, here is a Prolog clause, or rule to that effect.

conflict(Coloring) :-

adjacent(X,Y),

color(X,Color,Coloring),

color(Y,Color,Coloring).

For example,

?- conflict(a).

no

?- conflict(b).

yes

?- conflict(Which).

Which = b

Here is another version of 'conflict' that has more logical parameters.

conflict(R1,R2,Coloring) :-

adjacent(R1,R2),

color(R1,Color,Coloring),

color(R2,Color,Coloring).

Prolog allows and distinguishes the two definitions of 'conflict'; one has one logical parameter ('conflict/1') and the other has three ('conflict/3'). Now we have

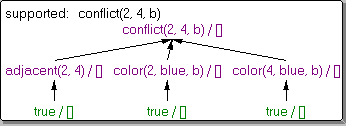
?- conflict(R1,R2,b).

R1 = 2 R2 = 4

?- conflict(R1,R2,b),color(R1,C,b).

R1 = 2 R2 = 4 C = blue

The last goal means that regions 2 and 4 are adjacent and both are blue. Grounded instances like 'conflict(2,4,b)' are said to be consequences of the Prolog program. One way to demonstrate such a consequence is to draw a *program clause tree* having the consequence as the root of the tree, use clauses of the program to branch the tree, and eventually produce a finite tree having all true leaves. For example, the following clause tree can be constructed using fully grounded instances (no variables) of clauses of the program.

  
Fig. 2.1.3

To learn more about the *visual logic* tool used to automatically make digrams like the one in the previous display,

The bottom leftmost branch drawn in the tree corresponds to the unit clause

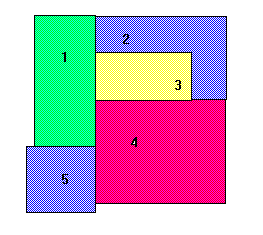
adjacent(2,4).

which is equivalent in Prolog to the clause

adjacent(2,4) :- true.

Now, on the other hand, 'conflict(1,3,b)' is not a consequence of the Prolog program because it is not possible to construct a finite finite clause tree using grounded clauses of P containing all 'true' leaves. Likewise, 'conflict(a)' is not a consequence of the program, as one would expect. We will have more to say about program clause trees in subsequent sections.

We will revisit the coloring problem again in Section 2.9, where we will develop a Prolog program that can compute all possible colorings (given colors to color with). The famous Four Color Conjecture was that no planar map requires more than four different colors. This was proved by Appel and Haken (1976). The solution used a computer program (*not* Prolog) to check on many specific cases of planar maps, in order to rule out possible troublesome cases. The map in in Fig. 2.1.1 does require at least four colors; for example ...

  
Fig. 2.1.4

*Exercise 2.1* If a map has N regions, then estimate how many computations may have to be done in order to determine whether or not the coloring is in conflict. Argue using program clause trees