Prolog

The programming language Prolog (short for programming in logic) was developed in France in the 1970s by A. Colmerauer and P. Roussel to help programmers working in the field of artificial intelligence.

A simple Prolog program consists of a set of statements describing some situation together with questions about the situation. Built into the language are search and inference techniques needed to answer the questions by deriving the answers from the given statements. This frees the programmer from the necessity of having to write separate programs to answer each type of question.

Prolog programs include a set of declarations consisting of two types of statements,

Prolog facts and *Prolog rules*.

Prolog facts define predicates by specifying the elements that satisfy these predicates. Prolog rules are used to define new predicates using those already defined by Prolog facts.

Example: Consider a Prolog program given facts telling it the instructor of each class and in which classes students are enrolled. The program uses these facts to answer queries concerning the professors who teach particular students. The predicates

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

The Prolog facts in such a program might include:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

The Prolog facts in such a program might include:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

A new predicate teaches(p, s), representing that professor p teaches student s, can be defined using the Prolog rule

teaches(P, S): instructor(P, C), enrolled(S, C)

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

The Prolog facts in such a program might include:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

Note that a comma is used to represent a conjunction of predicates in Prolog. Similarly, a semicolon is used to represent a disjunction of predicates.

A new predicate teaches(p, s), representing that professor p teaches student s, can be defined using the Prolog rule

teaches(P, S): instructor(P, C), enrolled(S, C)

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

Prolog facts:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

Prolog rule

teaches(P, S):

instructor(P, C), enrolled(S, C)

Prolog answers queries using the facts and rules it is given.

For example,

using the facts and rules listed, the query

?enrolled(kevin,math273)

produces the response

yes

because the fact enrolled(kevin, math273) was provided as input.

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

Prolog facts:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

Prolog rule

teaches(P, S):

instructor(P, C), enrolled(S, C)

The query

```
?enrolled(X ,math273)
```

produces the response

```
kevin
kiko
```

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

Prolog facts:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

Prolog rule

teaches(P, S):

instructor(P, C), enrolled(S, C)

?teaches(X,juana)

This query returns

patel

grossman

instructor(p, c): professor p is the instructor of course c

and

enrolled(s, c): student s is enrolled in course c.

Prolog facts:

- instructor(chan, math273)
- instructor(patel, ee222)
- instructor(grossman, cs301)
- enrolled(kevin,math273)
- enrolled(juana,ee222)
- enrolled(juana,cs301)
- enrolled(kiko,math273)
- enrolled(kiko,cs301)

Prolog rule

teaches(P, S):

instructor(P, C), enrolled(S, C)

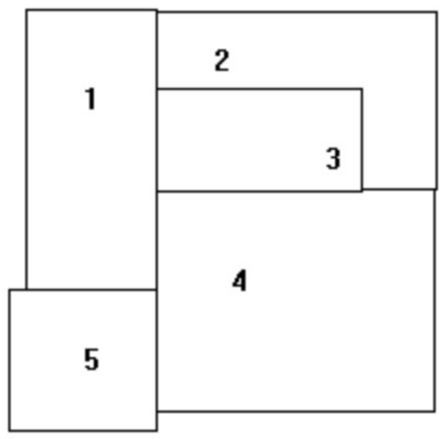
Define a new rule for classfellow(u,v)?

Map Coloring

A famous problem in mathematics concerns coloring adjacent planar regions. It is required that whatever colors are 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.

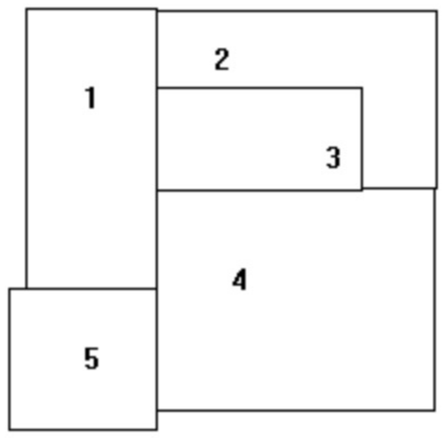
Map Coloring

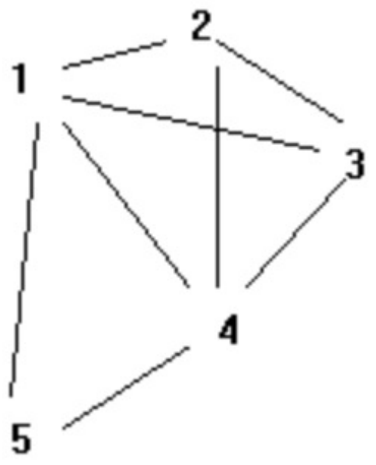
A famous problem in mathematics concerns coloring adjacent planar regions. It is required that whatever colors are used, no two adjacent regions may not have the same color. Two regions are considered adjacent provided they share some boundary line segment. Consider the following map.

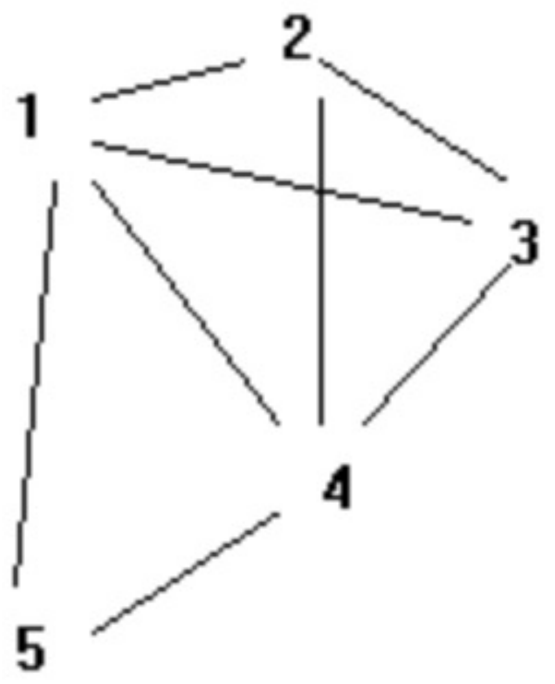


Map Coloring

A famous problem in mathematics concerns coloring adjacent planar regions. It is required that whatever colors are 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.





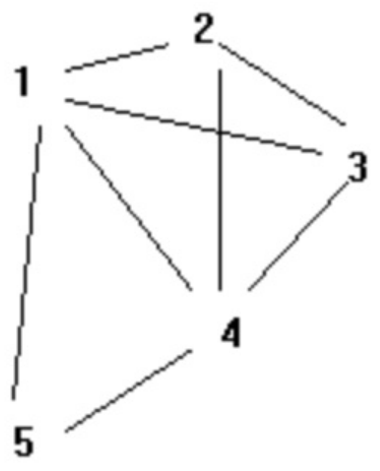


A Prolog representation for the adjacency information could be represented by the following *unit* clauses, or facts.

```
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).
```

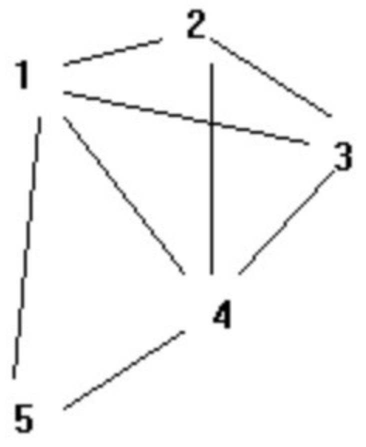
If these clauses were loaded into Prolog, we could observe the following behavior for some goals.

```
?- adjacent(2,3).
yes
?- adjacent(5,3).
no
?- adjacent(3,R).
R = 1 ;
R = 2 ;
R = 4 ;
no
```



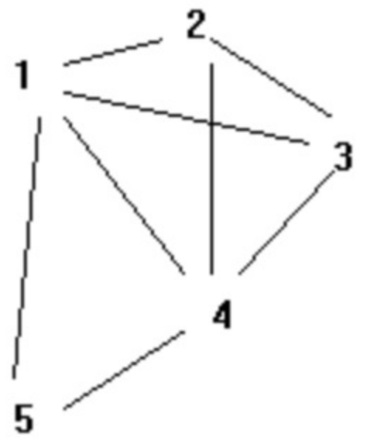
One could declare colorings for the regions in Prolog also using unit clauses.

```
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).
```



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

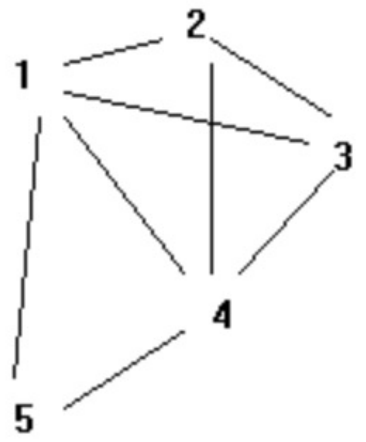
```
color(1,red,a).     color(1,red,b).
color(2,blue,a).    color(2,blue,b).
color(3,green,a).   color(3,green,b).
color(4,yellow,a).  color(4,blue,b).
color(5,blue,a).    color(5,green,b).
```



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.

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).
```

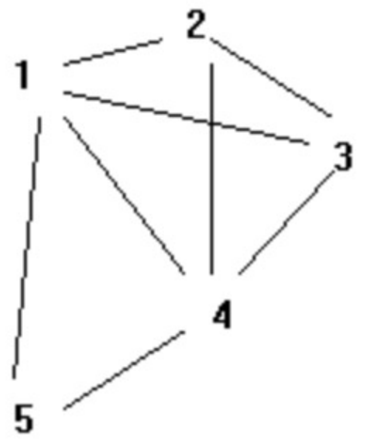


Here we have encoded 'a' and 'b' colorings. We want to write a Prolog definition of a conflictive coloring, meaning that two adjacent regions have the same color. For example, here is a Prolog clause, or rule to that effect.

```
conflict(Coloring) :-
   adjacent(X,Y),
   color(X,Color,Coloring),
   color(Y,Color,Coloring).
```

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).
```



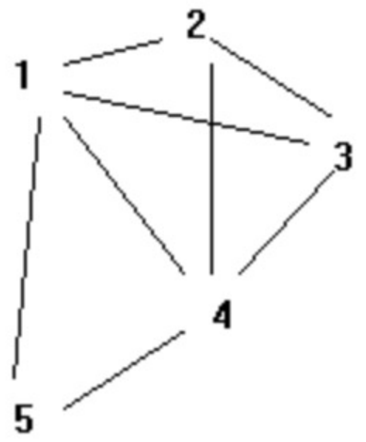
Here we have encoded 'a' and 'b' colorings. We want to write a Prolog definition of a conflictive coloring, meaning that two adjacent regions have the same color. For example, here is a Prolog clause, or rule to that effect.

```
conflict(Coloring) :-
   adjacent(X,Y),
   color(X,Color,Coloring),
   color(Y,Color,Coloring).
```

```
?- conflict(a).
no
?- conflict(b).
yes
?- conflict(Which).
Which = b
```

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).
```

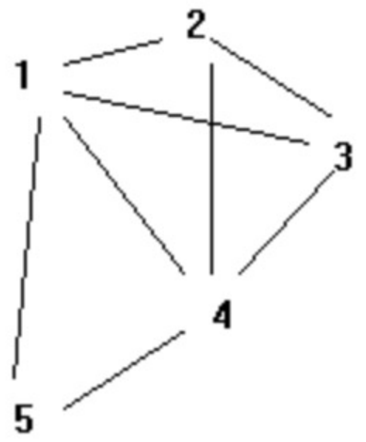


Here is another version of 'conflict' that has more logical parameters.

```
conflict(R1,R2,Coloring) :-
   adjacent(R1,R2),
   color(R1,Color,Coloring),
   color(R2,Color,Coloring).
```

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 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).
```

```
?- conflict(R1,R2,b).
R1 = 2   R2 = 4
?- conflict(R1,R2,b),color(R1,C,b).
R1 = 2   R2 = 4   C = blue
```