Yachay Tech University

Mathematical and Computational Logic

Prolog Lab 8: Map Coloring + Optimization

1. Context and Goal

This lab extends your previous Map Coloring lab. Previously, you modeled a map as a graph and found valid colorings using a fixed number of colors K (e.g., 3 or 4). Now, the goal is to determine the *minimum* number of colors needed to color the map so that no two adjacent regions share the same color.

2. Prerequisites

You should already have:
• A predicate regions_au/1 and edges_au/1 for the Australia map.
• A predicate regions_sa/1 and edges_sa/1 for the South America map (from the previous lab).
• A core predicate that, given Regions, Edges, and K, finds a valid coloring using CLP(FD).

3. Recap: Core Coloring Predicate

A typical structure is:

```
color_map(Regions, Edges, K, Vars) :-
   same_length(Regions, Vars),
   Vars ins 1..K,
   apply_edges(Regions, Vars, Edges),
   labeling([ffc], Vars).
```

where Vars is a list of integer color indices, one per region, and apply_edges/3 enforces ColorA #\= ColorB for every adjacency A-B.

4. Simple Optimization Strategy: Search over K

Instead of trying to encode "K is the maximum of Vars" directly inside CLP(FD), we will use a simple *meta-level search* on K:

1. Pick an upper bound MaxK (e.g., 4 for planar maps, or length(Regions)).
2. Try K = 1, 2, ..., MaxK in order.
3. For each K, call color_map(Regions, Edges, K, Vars).

4. The first K that succeeds is the minimum number of colors.

This approach is easy to understand and reuses your existing model.

5. Designing min_colors/5

We define:

```
min_colors(Regions, Edges, MaxK, MinK, Vars) :-
    between(1, MaxK, K),
    color_map(Regions, Edges, K, Vars),
    MinK = K,
    !.
```

Explanation:

- between(1, MaxK, K) generates K = 1,2,...,MaxK.
- color_map/4 attempts to color the map with K colors.
- The first time color_map/4 succeeds, we bind MinK to K.
- The cut (!) prevents Prolog from searching for larger K values.

6. Integrating with Existing Code

You likely already have something like:

```
regions_au([...]).
edges_au([...]).
regions_sa([...]).
edges_sa([...]).
color_map(Regions, Edges, K, Vars) :- ...
```

Now you can define convenience predicates:

```
min_colors_au(MaxK, MinK, Vars) :-
    regions_au(Rs), edges_au(Es),
    min_colors(Rs, Es, MaxK, MinK, Vars).

min_colors_sa(MaxK, MinK, Vars) :-
    regions_sa(Rs), edges_sa(Es),
    min_colors(Rs, Es, MaxK, MinK, Vars).
```

7. Lab Tasks

Task A – Implement min_colors/5:

1. Add min_colors/5 as above.
2. Add helper predicates min_colors_au/3 and min_colors_sa/3, if needed in your implementation.
3. Test with queries like:

```
?- min_colors_au(4, MinK, Vars).
?- min_colors_sa(6, MinK, Vars).
```

Task B – Pretty Printing:

4. Reuse your pretty_color_by_region/2 from the previous lab, or implement it now.
5. Print Region=ColorName pairs for the minimal coloring:

```
?- min_colors_au(4, MinK, Vars), regions_au(Rs), pretty_color_by_region(Rs, Vars).
```

Task C – Experiments and Reflection:

6. Try different MaxK values and observe behavior.
7. Try different labeling strategies in color_map/4 (e.g., [], [ffc], [min]).
8. Record the minimum colors needed for Australia and South America.

9. Deliverables

• GitHub repository updated