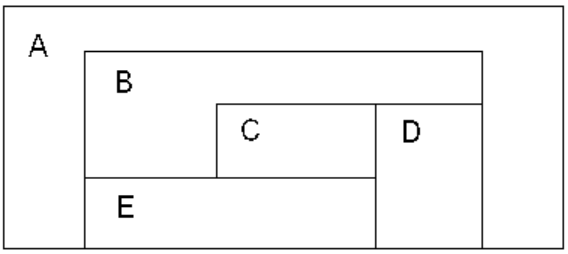
Problem Sheet 07

01. The 4-Color Problem

Find an "admissible" coloring of a map such that all adjacent countries have different colors. It had been conjectured for a long time that such a coloring is possible with only 4 colors. This conjecture was formulated precisely by F. Guthrie already in 1852, but proved by K. Appel und W. Haken only in 1976 using a computer program.

Write a Prolog program that finds all admissible colorings of a given map using the colors **red, blue, yellow and green.**



02. Sri Lanka has **nine provinces**: Northern, North Central, North Western, Eastern, Central, Sabaragamuwa, Western, Uva, and Southern. The adjacency relationships are as follows:

- Northern → North Western, North Central, Eastern
- North Central → Northern, North Western, Eastern, Central
- Eastern → North Central, Northern, Uva, Southern, Central
- North Western → Northern, North Central, Central, Sabaragamuwa, Western
- Central → North Central, North Western, Eastern, Uva, Sabaragamuwa
- Sabaragamuwa → Central, North Western, Western, Southern, Uva
- Western → North Western, Sabaragamuwa, Southern
- Uva → Eastern, Southern, Sabaragamuwa, Central
- Southern → Western, Sabaragamuwa, Uva, Eastern

Task:

1. Use **only four colours**: **green, blue, yellow, purple**.
2. The provinces **Northern, Western, and Uva must be coloured green**.
3. Assign colours to the remaining provinces so that **no two adjacent provinces share the same colour**.
4. Represent this problem in **Prolog** using:
    - Facts for adjacency, Facts for available colours
    - Rules to enforce that adjacent provinces have different colours
5. Write a **Prolog predicate** to produce a valid colouring for the entire map.

03. Imagine a monkey in a room, eyeing a bunch of bananas hanging from the middle of the ceiling. The monkey wants to get those bananas, but it needs to figure out the exact sequence of moves to reach them efficiently. Shall we help the monkey to grasp the bananas?

This seemingly simple scenario is a classic puzzle in the field of artificial intelligence that illustrates the power of logical problem-solving.

Background of the Problem

There is a monkey in a room with a bunch of bananas hanging from the middle of the ceiling. The room contains a box. The monkey's task is to figure out how to use the box to reach the bananas.

However, the monkey is allowed to perform specific actions to get the bananas.

- Monkeys can **walk** on the floor
- Monkey can **push** the box
- Monkey can **climb** on the box
- Monkey can **grasp** the bananas

But the monkey cannot jump or do any other action. For example, if the monkey is on the box which is at the door, a monkey cannot get the banana since it is not allowed to jump.

To grasp the bananas:

- The monkey must push the box to the middle of the room.
- Then, the monkey must climb onto the box.
- While standing on the box, the monkey can grasp the bananas.

The challenge is to determine the correct sequence of actions that will enable the monkey to achieve the goal.

Solution:

Approach to Solve the Problem

Normally, any real-world problem can be modeled as a set of states and actions like this:

We can model the problem in terms of states and actions. Normally, first, we model actions and then states.

Let's model the actions by using the predicate:

```
move(State1, Action, State2).
```

Also, we can represent the state of the monkey-like this. It includes the monkey's horizontal position, vertical position, and box position, and has/has not grasp the banana.

```
state(HPos, VPos, BPos, Has/Hasnot).
```

So, if the monkey performs an action, we can represent it in the below format.

```
move(state(HPos1, VPos1, BPos1, Has/Hasnot1),Action,state(HPos2,
VPos2, BPos2, Has/Hasnot2)).
```

In order to solve the problem, first, we need to identify the initial state. It is very important to see whether the monkey can reach the bananas.

All possible horizontal positions for the monkey:

- middle
- at door
- at window

All possible vertical positions for the monkey:

- on floor
- on box

All box positions:

- middle
- at door
- at window

Modeling Monkey Banana Problem using Prolog

Let's write a predicate **move** for the four actions: Grasp, Climb, Push, and Walk.

Grasp

When the box is at the middle, and the monkey is on the box, and the monkey does not have the banana (has not state), then by doing the grasp action, the state will change from **has not** state to **has** state.

```
move(state(middle, onbox, middle, hasnot),grasp,state(middle,
onbox, middle, has)).
```

Climb

When the monkey and the box are at the same horizontal position, by performing the climb action, a monkey can move on to the box.

```
move(state(P, onfloor, P, H),climb,state(P, onbox, P, H)).
```

Push

When the box and the monkey are in the same vertical position, a monkey can push the box.

```
move(state(P1, onfloor, P1, H),push(P1, P2),state(P2, onfloor, P2, H)).
```

Walk

Monkey can move one position to another position, by performing the walk action.

```
move(state(P1, onfloor, B, H),walk(P1, P2),state(P2, onfloor, B, H)).
```

Also, we need to define rules to reach the goal which is the monkey grasp the banana.

```
R1: canget(state(_, _, _, has)).

R2: canget(State1) :- move(State1, _, State2), canget(State2).
```

Here, we use another predicate `canget()`, which will perform the move predicate from state1 to state2 using previously defined four actions. Then, it performs canget() on state2. When the 'hasnot' state becomes 'has' state, we will stop the execution.

Okay… we have modeled the problem in Prolog.

Now, let's see an example.

Example

```
? – canget(state (atdoor, onfloor, atwindow, hasnot)).
```

In this example, the monkey is at the door, the box is at the window, and the monkey does not have the banana. So, our goal state is:

```
state (middle, onbox, middle, has)
```

To print the sequence of actions to grasp the banana from a given state, we can slightly modify the code like this.

```
move(state(middle, onbox, middle, hasnot), grasp, state(middle, onbox, middle, has), [grasp]).
move(state(P, onfloor, P, H), climb, state(P, onbox, P, H), [climb]).
move(state(P1, onfloor, P1, H), push(P1, P2), state(P2, onfloor, P2, H), [push(P1,P2)]).
move(state(P1, onfloor, B, H), walk(P1, P2), state(P2, onfloor, B, H), [walk(P1,P2)]).
```

```
canget(state(_, _, _, has), []).

canget(State1, Sequence) :-
 move(State1, Move, State2, Action),
 canget(State2, SubGoal),
 append(Action, SubGoal, Sequence).
```

We can get the output like this.

```
?- canget(state(atdoor,onfloor,atwindow,hasnot), Goal).
Goal = [walk(atdoor, atwindow), push(atwindow, middle), climb, grasp] .

?-
```