# Prolog Practical 9: A Simple Version of STRIPS

AIPP 2004
Tim Smith

*Please use this exercise specification instead of that in the course notes.

First, read the beginning of chapter 16 in the course notes ('Planning in Prolog') before attempting this exercise.

## Introduction

Download the file simstrips.pl from the practical section of the course website:
http://www.inf.ed.ac.uk/teaching/courses/aipp/#Practicals_

This file contains a simple version of a STRIPS-type linear planner coded to solve the monkey and bananas problem. This is the program discussed in the planning chapter of the course notes.

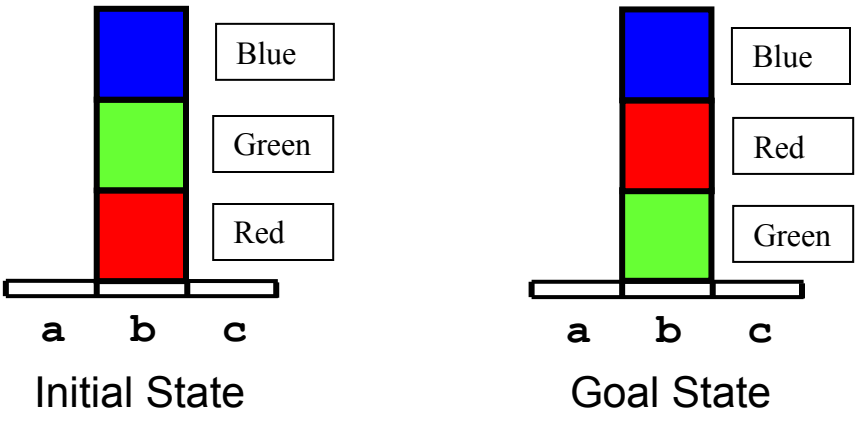
Have a look at the file and make sure you understand the structure of the program. Think back to how the program is supposed to work (operators with preconditions, add and delete lists applied to some world state); try and match this to the program. Run the program by calling the top level goal: `test(P)`.

The test/1 predicate has a subgoal solve/3:

1. The first argument of solve/3 represents the **initial state** of the world;
2. the second represents the **goal state** to be achieved;
3. and the third will become instantiated to the **plan** that, when applied to the initial state, will achieve the goal state.

## Making a Blocksworld Planner

You will now modify this planner to solve the Blocksworld problem depicted below:



The problem consists of:

- three locations: a, b, and c;
- three coloured blocks: red, green, and blue.

A plan needs to be devised that can move the blocks one at a time either on to the table or on to another block so that the eventual configuration matches that of the Goal State. The Monkey and Bananas planner can be modified to solve this Blocksworld problem.

Complete each section of this practical in order, writing the appropriate answers in the spaces provided before adding the code to your program. Show your answers to the demonstrator once you have a working planner.

1. Choose a representation for the blocksworld and use it to represent the initial state and goal states as depicted. The closer you stick to the representation used in the Monkey and Bananas problem the easier it will be to write the operators. Remember:
    a. you only need to represent as much of the problem domain as is necessary to solve the problem. Keep it simple;
    b. your state representations shouldn't include tests such as \+ X or X \= =Y) as they are not called, they are only matched to other lists of states.

Initial State:

_____
_____
_____
_____

Goal State:

_____
_____
_____
_____

2. Compare the initial state and goal states. What operators do we need to move the blocks so that their configuration matches the goal state? You should have as few operators as possible. Unlike the blocksworld MEA planner presented in the lectures, your planner cannot perform any 'tests' about world states (the can/1 predicate from lecture 16). All properties necessary for an operator to be applied to the world must be represented in the operators preconditions.

    Write the new operators in the space below, listing their preconditions, delete lists, and add lists.

[Operator] [Preconditions] [Delete list] [Add list]

3. Write your operators, initial state, and goal states into the program (replacing the original M&B examples). Try running the planner. Does it generate a solution?
4. If it doesn't generate a solution try adding "write(Op), nl," in the second to last position of the main solve/4 clause so that you can see the paths it is trying. What is it doing wrong? Are the moves it is trying to perform valid?
5. If your operators and states are correct the problem should be caused by the way solve/4 chooses operators to implement. Because it can make no explicit tests of object properties it has to rely on operator preconditions. There are two ways you could solve this problem either add extra state descriptions in the preconditions or use one of the utility predicates already provided to ensure that the operator preconditions can only match a state in the world once. Perform both types of modification and write the extra bits of your code in the spaces below:

Adding extra preconditions:

Using a predicate to check that preconditions only match a world state once (remember you can use most predicates in multiple ways. For example, to strip lists apart, create new lists, or check list contents) :

Now test both versions of your code to see if the planner now works.