REPORT

Multi-agent systems

STRIPS planner

Supervised by:

DR. ZARGAYOUNA MAHDI

Elaborated by:

Aissaoui Ilhem
Diab Bilal

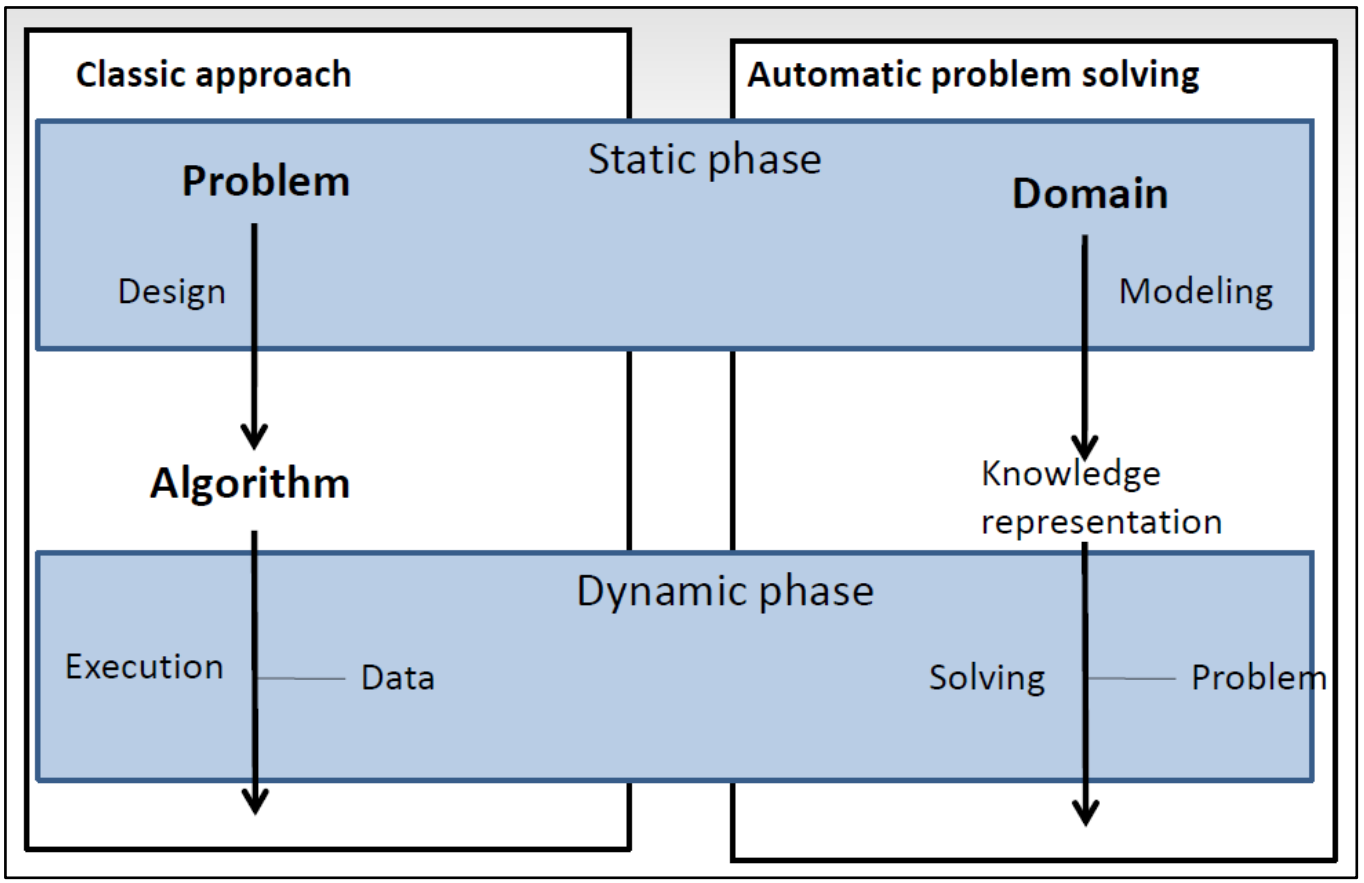
Master 2 SIA
2021/2022

TABLE OF CONTENT

Introduction:

The purpose of creating computers is the automation of problem solving, indeed algorithms are created to automate certain tasks that lead to results. However, like everything else, classical computing based on the algorithmics approaches to problem solving has reached its limits, and this is where the Artificial Intelligence begins.

In AI, it is no longer a question of programming a solution (as in the 3rd and 4th generation languages respectively L3G and L4G) but of giving constraints, actions and rules and letting the program build reasoning and deductions, this type of program is commonly called Cognitive Agent. It interacts in an architecture called: BDI architecture.



The picture above shows the difference with the algorithmic approach, in the automatic approach, the reasoner only needs a knowledge base, a knowledge representation of the world, a set of actions and the goal to achieve.

In this project we are going to implement a planner (reasoner) that from the modeling of the environment, an initial and final state and a number of rules gives us the sequence of actions leading from the initial world to the world satisfying the goal. It can be used by intelligent agents and autonomous robots. We are going to use the STRIPS language to describe the environment, and details how the planner search the solution by giving some searches strategies, and the modeling of the space of solutions.

Our program will receive a file containing all our environment with all the actions and states described with STRIPS, in output our program will provide us the sequence of actions that lead from the initial state to the final state

Cognitive agent:

The cognitive agent is a certain type of program which is intelligent. It is aware of its environment and has a broad vision of the world, it can interact with other agents in a collaborative way to solve problems and also interacts with the environment, it is a Social Model of organization (planning, commitment).
It has:

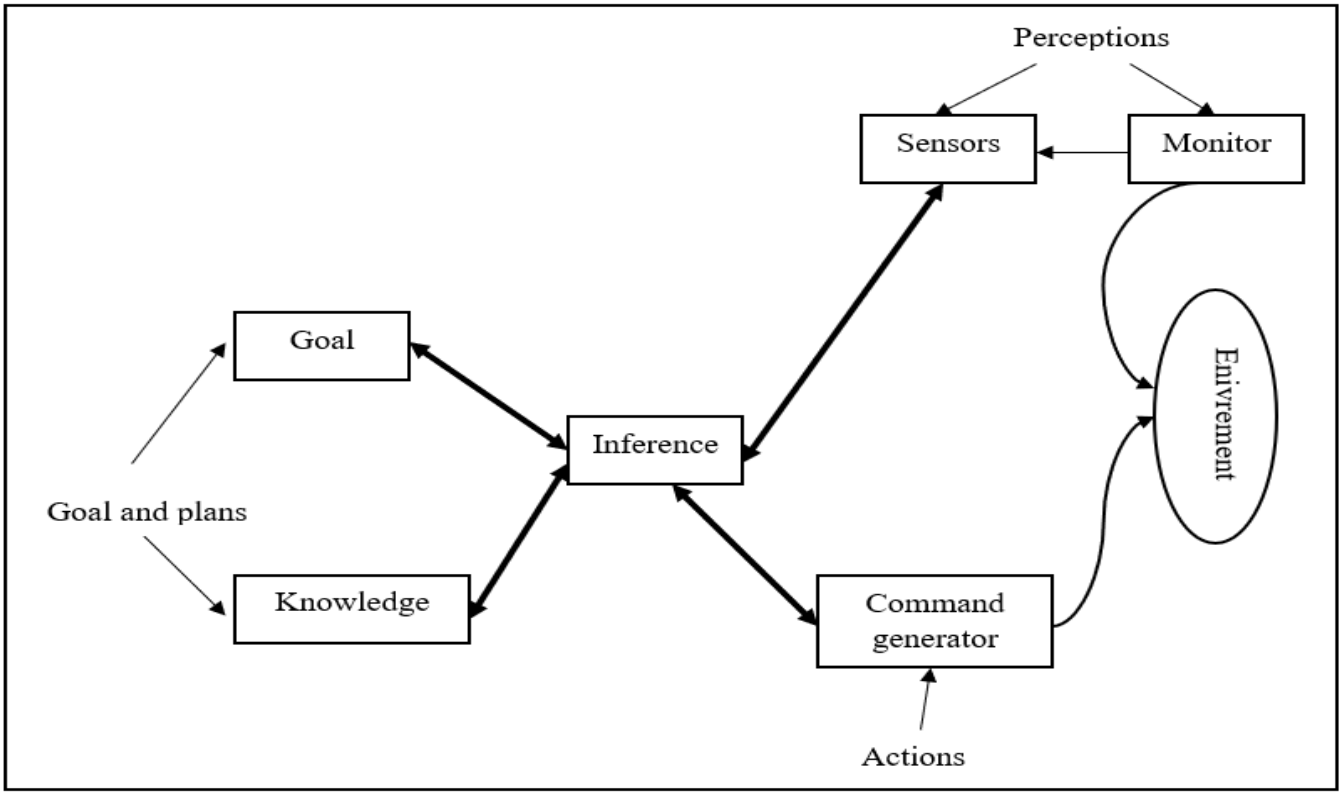
- Set of states which include the initial and final state
- Set of percepts which allow to analyze and take decision about the best action
- Set of actions

Belief–desire–intention software model (BDI)

BDI a software model developed for programming intelligent agents. It's a knowledge representation model characterized by the implementation of an agent which has:

- Beliefs (about itself, the others agents and the environment)
- Desires (goals)
- Intentions (plans)
- The reasoner which from all the information and knowledges will infer.

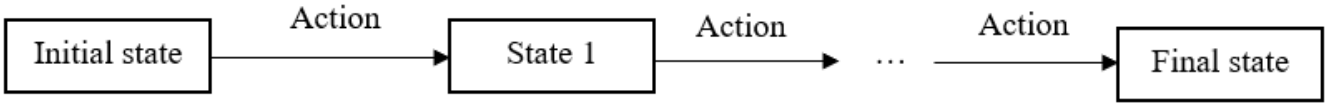
The figure bellow represents the BDI architecture:



The agent has perceptions, which are the tools that allows it to be aware of and act with its environment and to feed its knowledge base. The perceptions above are represented by the sensors (allows to see the environment) and the monitor (allows to act on the environment).

Depending on the goal, the basic knowledge and the knowledge fed by the perceptions, the intelligent program (the reasoner) will make inference and deduction and perform an action using to the perceptions (the monitor specially) to act on the environment.

The reasoner is the center of this architecture, it has a view on everything and we can summarize the resolving like that:



Strips language:

The Sandford Research Institute Problem Solver (STRIPS) is an automatic planner used to express automated planning problem instances. It works by executing a domain and problem to find a goal. A Strips program for one problem is composed by:

- A set of essentials formulas: they are the base which describe the world
- A set of actions (a la trips), these rules are a triplet which is composed by:

  PRE: it's essentials formulas which must be valid in the current state the ensure the success of the action.

  DEL: the formula will be removed from the state of the world after the executing of the action

  ADD: the fact that will become true once the action performed.

  Every action is a conjunction of essentials formulas.

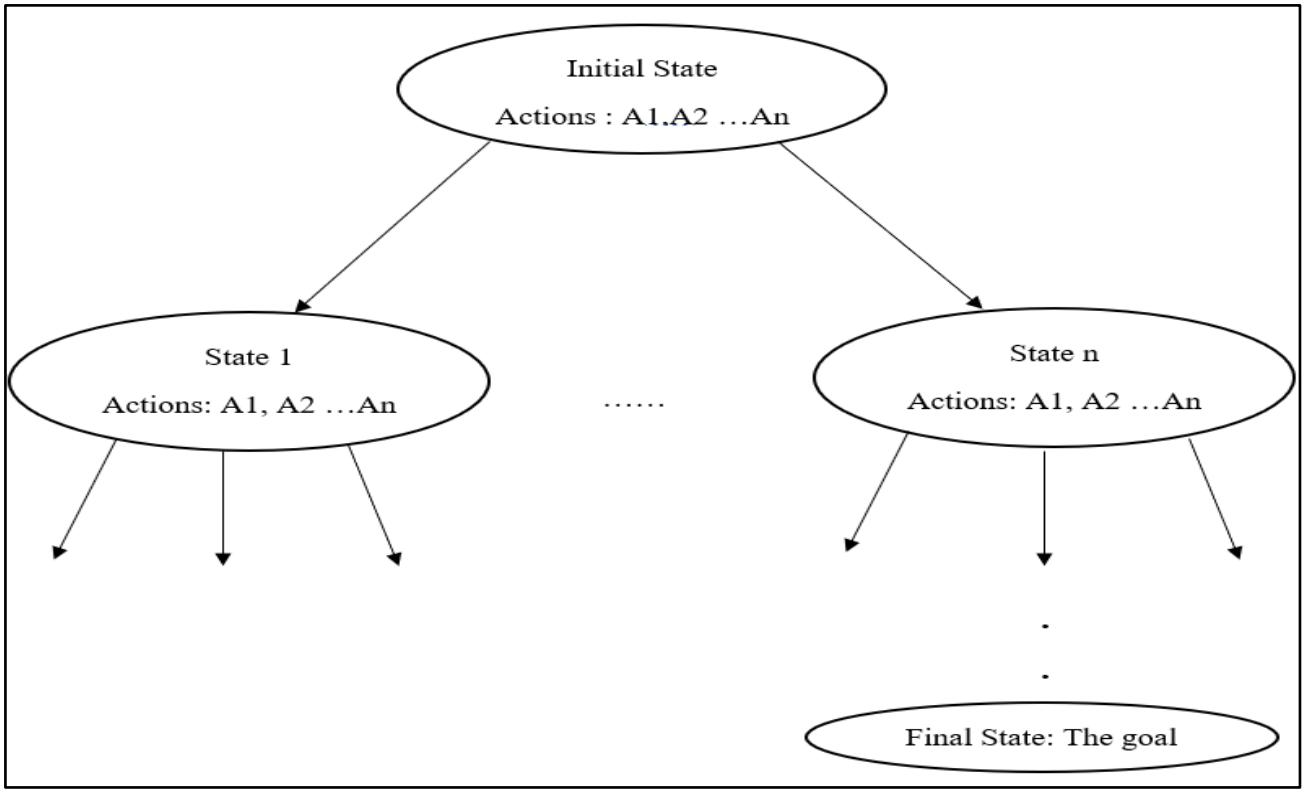
- A set of semantic interpretation formula to recognize a valid situation of the world.
- The initial and final state composed of essentials formula.

The planning:

The planning is the inference, taking into account the initial state and the actions allowed, the planner will explore the set of possibility to reach to the final state.

From the initial state, the planner will execute every action to reach different states by exploring several arrangements and from every new state the planner will execute every action allowed in that state.

The problem of finding a sequence of moves that transforms a given initial state into a given goal state can be formulated as a graph search problem. The nodes of the graph are the possible states and the arcs correspond to the actions that transform one state into another. The problem is to find a path in this graph from a given initial state to a given goal state.



To find that path, there are several search strategies including

- Depth-first.
- Breadth-first.

There are other strategies which use a heuristic as A* algorithm, but there are not generic and they are modeled according to each problem.

There are different manners to build the graph which represents the space of solutions:

- We can construct it as the figure above by starting with the initial stat: **Forward chaining**.
- We can also begin with the goal (the final stat) and go up until reach the initial state: **Backward chaining**.

Searching strategies:

1- Depth-first Search:

In this strategy, we start from the root nodes (initial stat) and explore as far as possible along each branch before backtracking.

All the children of a node are stored on a FIFO queue, and every child and its children is evaluated, once the first child of the initial state has been evaluated as well as all its children, we move on to the second children of the initial state.

2- Breadth-first Search:

In this strategy, and from the initial stat, we explore and evaluate all nodes at the present depth, once the first stage is done, we move to the second floor on the nodes at the next depth until we find the solution. it evaluates the current depth.

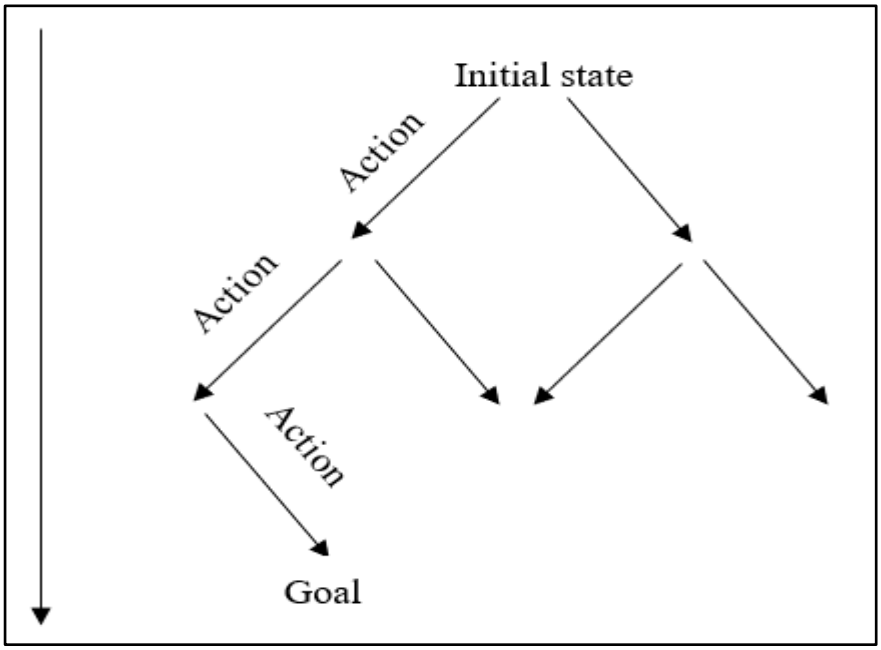
Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

Type of inference:

1- Forward chaining:

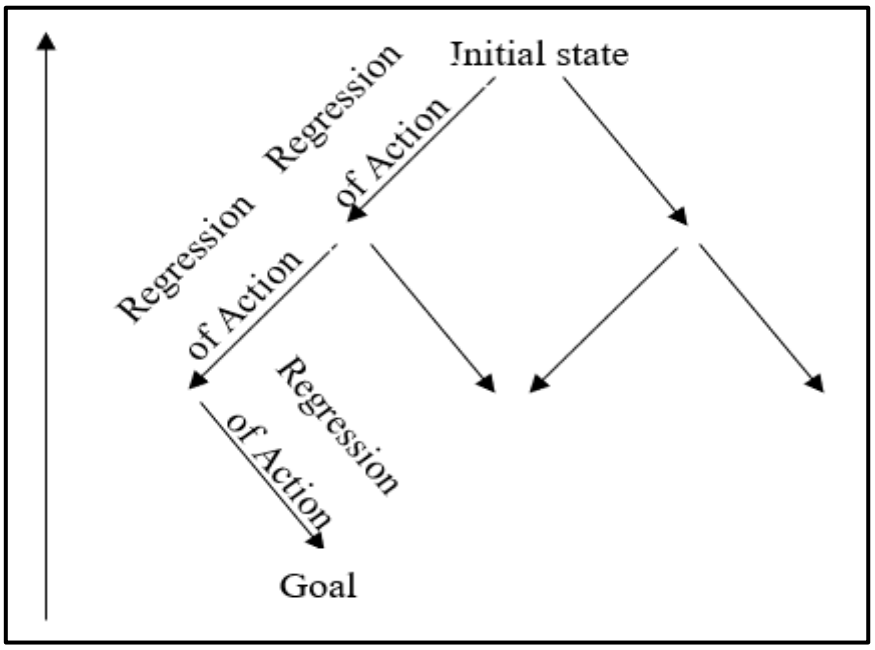
Forward chaining is the logical process of inferring unknown truths from known data and moving forward using determined conditions and rules to find a solution.

We start with the initial stat and developpe the graph according to the actions and rules, the condition to stop is to verify that the goal is satisfied with the state of the current world. The opposite of forward chaining is backward chaining.



2- Backward chaining:

Backward chaining is the invers of forward chaining, we start with the final stat and go up in the graph while regressing the rules, the condition to stop is to verify that state of the current world is equals to the initial state the goal is satisfied with the. The opposite of forward chaining is backward chaining.



Implementation:

In this section, we will detail the implementation. The problem has been described in a text file with the STRIPS language, we used python to parse the file and extract the initial, the final state and the different actions and rules.

We decided to implement Forward chaining with Breadth-first Search.

Presentation of the domaine:

The SRIPS file has the following skeleton:

```
Initial state: State 1, State 2 …. State n.
Goal state: State 1… State m.
Action:
        Action 1 (parameters):
                Preconditions: State 1, State 2 …. State n.
                Postconditions: State 1, State 2 …. State m.

        Action 2 (parameters):
                Preconditions: State 1, State 2 …. State i.
                Postconditions: State 1, State 2 …. State j.

        Action n (parameters):
                Preconditions:  State 1, State 2 …. State k.
                Postconditions: State 1, State 2 …. State l.
```

We present the general algorithm on which we have based on in our project:

- Create the STRIPS file with an example.
- Import, parse and extract the text file which contains the description on the world and the actions.
- Apply the Breadth-first Search strategy.

We detail Breadth-first Search algorithm:

1- Create an empty FIFO queue that represent the state of each node of the graph where each state is a couple: State of the world and list of actions that lead from the state of the initial world to the current state.

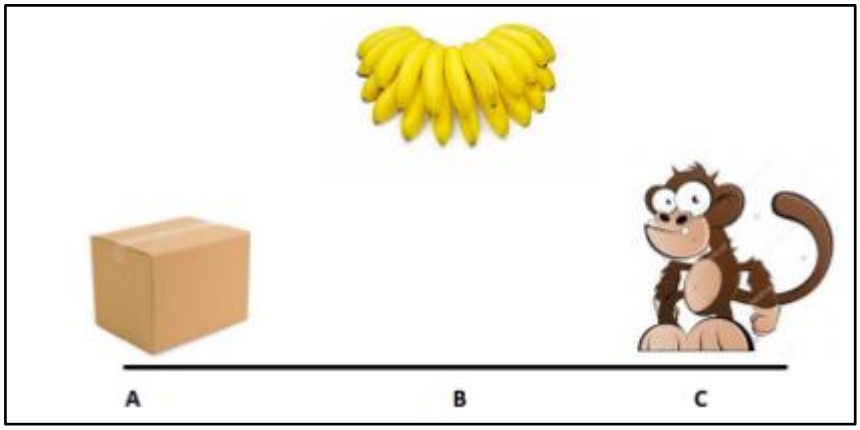
2- Initialize the queue with the initial state and empty list of actions.
3- Add the initial state in the queue.
4- Get the first node from the queue.
5- Mark the current node as explored.
6- Get the possible actions to apply to the current of the node.
7- For each action:
    - Get the state reached after performing the action.
    - Add the new state and the action to the node.

8- If the new node is visited or is in the queue then go to step 4.

9- If the node is the goal, then terminate with a success and return the plan (chain of actions and the final state).
10- Add the current node to the queue.
11- Go back to step 4.

Example 1: The monkey-and-bananas problem

The monkey-and-bananas problem is faced by a monkey in a laboratory with the bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas if he climbs on it. Initially, the monkey is at A, the bananas at B, and the box at C. The monkey and box have height Low, but if the monkey climbs onto the box he will have height High, the same as the bananas. The actions available to the monkey include Go from one place to another, push an object from one place to another, ClimbUp onto or ClimbDown from an object, and Grasp or Ungrasp an object. Grasping results in holding the object if the monkey and object are in the same place at the same height.



Definition of problem

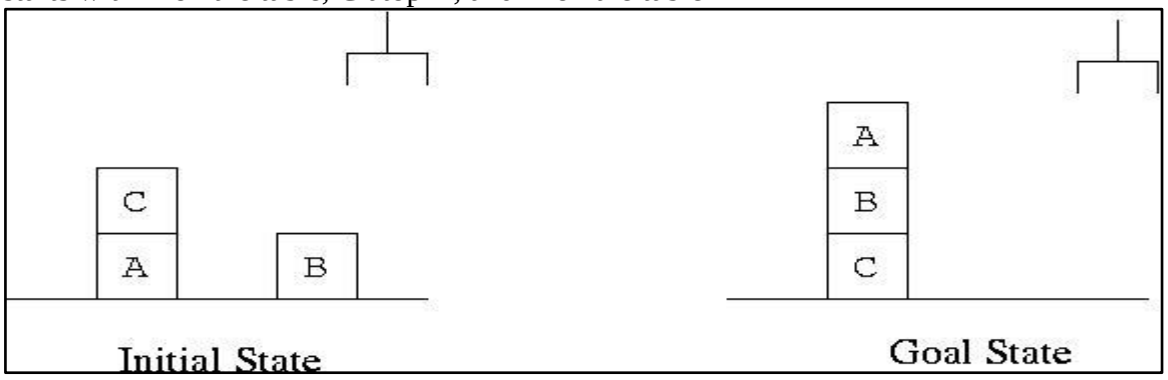
```
Initial state: At(A), Level(low), BoxAt(C), BananasAt(B)
Goal state:    Have(bananas)

Actions:
               // move from X to Y
               _Move(X, Y)_
               Preconditions:  At(X), Level(low)
               Postconditions: not At(X), At(Y)

               // climb up on the box
               _ClimbUp(Location)_
               Preconditions:  At(Location), BoxAt(Location), Level(low)
               Postconditions: Level(high), not Level(low)

               // climb down from the box
               _ClimbDown(Location)_
               Preconditions:  At(Location), BoxAt(Location), Level(high)
               Postconditions: Level(low), not Level(high)

               // move monkey and box from X to Y
               _MoveBox(X, Y)_
               Preconditions:  At(X), BoxAt(X), Level(low)
               Postconditions: BoxAt(Y), not BoxAt(X), At(Y), not At(X)

               // take the bananas
               _TakeBananas(Location)_
               Preconditions:  At(Location), BananasAt(Location), Level(high)
               Postconditions: Have(bananas)
```

Solution using Breadth First Search (forward chaining)

```
_Move(A, C)
_MoveBox(C, B)
_ClimbUp(B)
_TakeBananas(B)
Final state:
[('BananasAt', ['B']), ('BoxAt', ['B']), ('At', ['B']), ('Level', ['high']), ('Have', ['bananas'])]
--- 0.08409643173217773 seconds ---
```

Example 2: The world of Cubes problem (Sussman anomaly)

The Sussman anomaly is a problem in artificial intelligence, first described by Gerald Sussman, that illustrates a weakness of non-interleaved planning algorithms, which were prominent in the early 1970s. In the problem, three blocks (labeled A, B, and C) rest on a table. The agent must stack the blocks such that A is atop B, which in turn is atop C. However, it may only move one block at a time. The problem starts with B on the table, C atop A, and A on the table



Definition of problem

```
Initial state: ONTABLE(A), ONTABLE(B), ON(C,A), CLEAR(B), CLEAR(C), HANDEMPTY(R)

Goal state:    ON(B, C), ON(A, B)

Actions:
               // Move a disk from source to dest
               PickUp(X)
               Preconditions: ONTABLE(X), CLEAR(X), HANDEMPTY(R)
               Postconditions: HOLDING(X),  not ONTABLE(X), not HANDEMPTY(R)

               // Move a disk from source to dest
               PutDown(X)
               Preconditions:  HOLDING(X)
               Postconditions: CLEAR(X), ONTABLE(X), not HOLDING(X), HANDEMPTY(R)

               // Move a disk from source to dest
               Stack(X, Y)
               Preconditions:  HOLDING(X), CLEAR(Y)
               Postconditions: HANDEMPTY(R), ON(X,Y), CLEAR(X), not HOLDING(X), not CLEAR(Y)

               // unstack f
               UnStack(X, Y)
               Preconditions:  HANDEMPTY(R), ON(X,Y), CLEAR(X)
               Postconditions: HOLDING(X), CLEAR(Y), not ON(X,Y), not HANDEMPTY(R)
```

Solution using Breadth First Search (forward chaining)

```
UnStack(C, A)
PutDown(C)
PickUp(B)
Stack(B, C)
PickUp(A)
Stack(A, B)
Final state:
[('CLEAR', ['A']), ('CLEAR', ['C']), ('ONTABLE', ['C']), ('ON', ['B', 'C']), (
'CLEAR', ['B']), ('HANDEMPTY', ['R']), ('ON', ['A', 'B']), ('CLEAR', ['A'])]
--- 0.08469414710998535 seconds ---
```