Planning

- Planning involve reasoning about actions that the agent intends to carry out.
  - Planning is the reasoning side of actions.
  - Reasoning involves the representation of the world that the agent has and the representation of its own actions.
  - Agent plans for actions and executes them to achieve the goal.
- The planning problem can be described at varying levels of detail.
  - Simplest planning to complex planning

Simplest planning problems

- In the simplest planning problems
  - The domain is static
  - The agent has complete information of domain (perception is perfect)
    - Agent can take all facts into account while planning
  - Actions are instantaneous
    - There is no notion of time, but only of sequencing of actions.
  - Actions effects are deterministic
    - Agent is the only one changing the world.

Complex planning

- Each of these constraints can be relaxed to define richer planning domains.
- In these problems planning is a computationally harder task.

- Planning requires explicit representation of actions in a domain.
- This is little different from our view of search problems that assume a move generation function.
- Representation actions explicitly allows us to explore richer domains for example when actions have durations.
- So it requires other approaches to planning different from graph search.

Classical planning

- Reasoning about actions involves simulation of changes in the mental model (representation of the world)
- Agent peers into the future to examine the different possibilities and select the desirable course of action.
  - This is called projection into future.
  - The planner searches through the possible combination of actions to find the plan that will work.

Alternative Approach: Memory based planning

- To exploit agent experience, looking into past.
- If the agent has memory, it can reuse
  - a plan that worked earlier to find a solution
  - Or problem solving experience to find a solution
- It involves representation and reasoning over the events in the past.
- The idea is
  - If a problem is familiar, then a known solution is used from memory
  - Is a problem is new, earlier project based methods have to be used.
- This is more intelligent agent: learning from the past.

Planning Domain Description Language (PDDL)

- A standard notation for describing the world and actions
- Allows to express world in terms of the language and actions in terms of schemas
- The language is based on First Order Logic Notation
- It also allows to state a specific planning problem by describing the initial state and the goals using the constructs define in the schema
- The planning algorithms can be written in a domain independent form.
- The domain and problem descriptions will become input to the domain independent planning algorithms.

- The complexity of planning domain will depend on the expressiveness of the PDDL and the constructs defined.
- Starting from simplest planning domain and problems, a series of more expressive PDDL languages have been defined.
- In each new languages, the world and actions can be described in richer fashion, taking more and more aspects from the real world

STRIPS domains

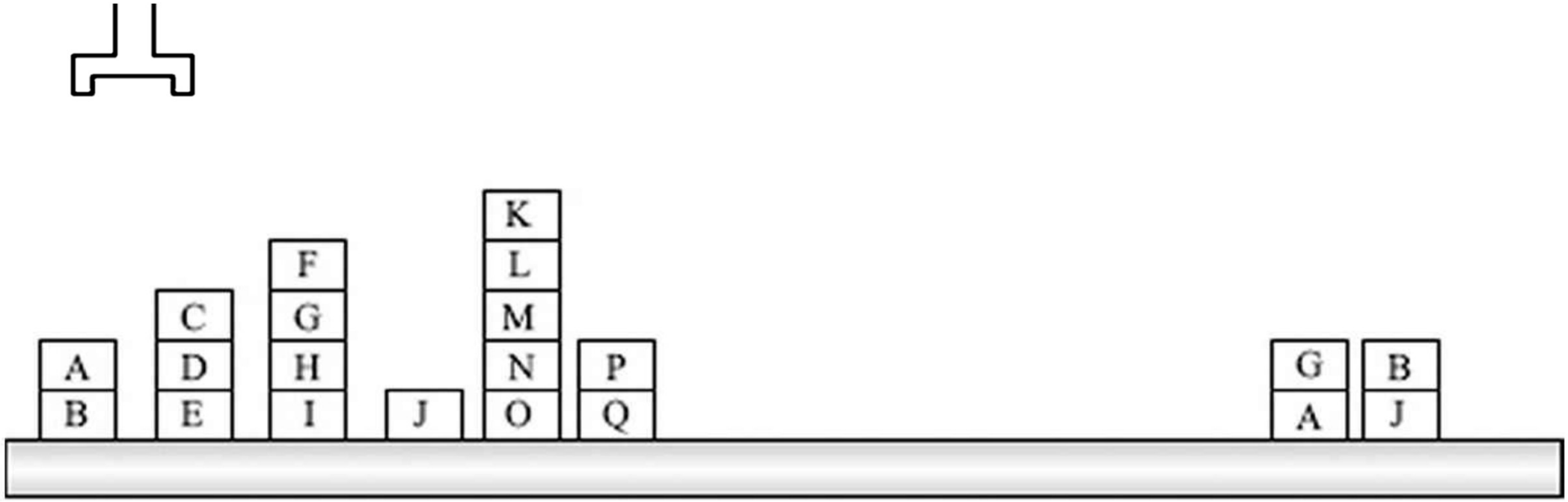
- Simplest of planning domains
  - Pose problems that are computationally hard to solve
- They are basically viewed as search problems in the classical approaches.

The STRIPS Domain

- Search algorithms can also be used for planning
- World is described in states and actions are described as operators
- Operators are applied to a state and transforms into another state
- The operators are close to the idea of production rules being applicable when the associated preconditions hold.
- The first program to use this formalism was STRIPS

The STRIPS Domain

- The program was designed to plan in the domain of block world.
- In this domain a set of labelled blocks are in some configuration to start with.
- A block can be on another block, or a block can be on the table and the table has unlimited capacity.
- The objective of the planner is to rearrange the blocks into a desired (goal) configuration
- The output of the planner is the set of moves that brings about this desired rearrangement



The given state

The goal = on(G, A) ∧ on(B, J)

- The representation borrows from logic, specifically first order logic
- Unlike in classical logic, predicates in planning such as on(C,D) can be true at one time instance and false at another.
  - Such predicates are also known as fluents
- Classical reasoning with first order logic does not allow for assertions to be withdrawn.
- One way to handle the problem is to introduce time as a parameter and have every predicate timestamped.
- However this creates an enormous book-keeping problem of carrying forward facts as time moves on.

- How do we know that if a fact like ontable(B, t1) is true the ontable(B, t2) will be true as well?
- This is well known frame problem
- This problem can be solved by adding the so called “frame axioms” which explicitly specify that all conditions not affected by actions are not changed while executing that action.
- Thus the frame axioms need to be applied at each stage to keep track of what is true as time marches on

- The ingenious solution introduced in STRIPS was to make all changes explicit
  - The assertions that became true as a consequence of the operator being applied will be added,
  - The assertions that became false as a consequence of the operator being applied will be deleted,
  - Whatever was not changed remained the same.

- STRIP operators follow this strategy and are made of the following three components
  - P: Precondition List
    - The assertions needed to be true for the operator to be applicable
  - A: Add List
    - The assertions that became true as a consequence of the operator being applied will be added,
  - D: Delete List
    - The assertions that became false as a consequence of the operator being applied will be deleted,

The World in the block world domain

- On(X, Y): Block X is on block Y
- Ontable(X): block X is on the table
- Clear(X): no block is on block X
- Holding(X): the robot arm is holding X
- Armempty: the robot arm is not holding anything

Operators for the blocks world domain

- PICKUP(X)
  - P: Ontable(X) ^ Clear(X) ^ Armempty
  - A: Holding(X)
  - D: Ontable(X) ^ Armempty
- PUTDOWN(X)
  - P: Holding(X)
  - A: Ontable(X) ^ Armempty
  - D: Holding(X)

Operators for the blocks world domain

- UNSTACK(X,Y)
  - P: ON(X, Y) ^ clear (X) ^ armempty
  - A: holding(X) ^ clear(Y)
  - D: on(X,Y) ^ armempty
- STACK(X,Y)
  - P: holding(X) ^ clear(Y)
  - A: ON(X, Y) ^ clear (X) ^ armempty
  - D: : holding(X) ^ clear(Y)

- Since goal state description may not describe the final state completely
  - There can be many states in which the given goal conditions will be true
- This also implies that backward reasoning will have to deal with incomplete state descriptions.

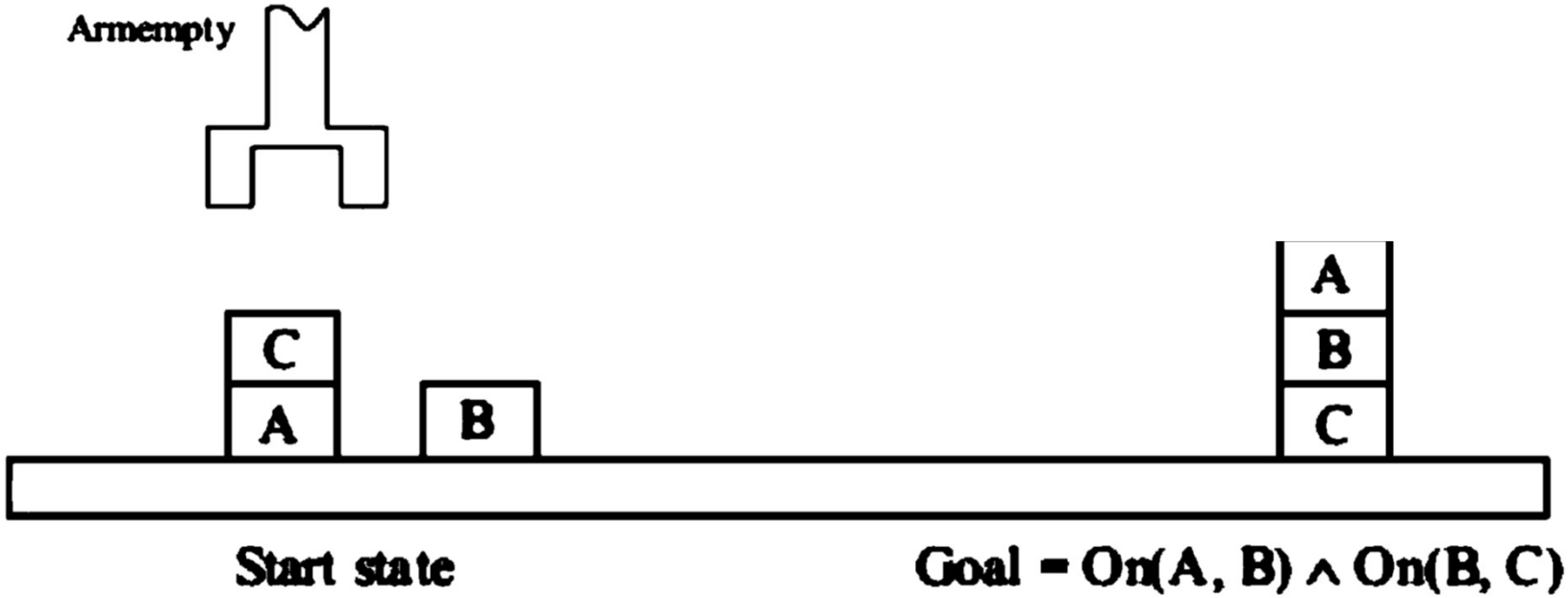
Planning Algorithms in the STRIP domain

- The search algorithms can easily be adapted to the task of planning
  - Known as forward reasoning/search
- We can start the search from the goal description
  - Known as backword reasoning/search
- We can combine the good features of both, forward and backward search
  - Known as goal stack planning
- We can seek to fill in the actions constituting the plan in a nonlinear order, filling in actions as and when we spot their requirements
  - Plan space planning and Partial order planning
- Finally, hierarchical planning

Forward state space planning

- Given a state, and given a set of operations, the actions can be determined that can be applied in the state to generate successor states.
- This corresponds to implement MovGen function

Planning problem from domain of blocks world



Same problem described in PDDL

```
(define (problem blocksProblem1)
        (:domain Blocks)
        (:objects
                A - block
                B - block
                C - block )
```

```
 (:init
        (onTable A)
        (onTable B)
        (on C A)
        (clear B)
        (clear C)
        (armempty))
 (:goal (and (on A B) (on B C)))
)
```

- Lets effect(a) denotes all the effects of action a.
- Lets $effect^{+}(a)$ denotes the set of positive effects of action a
- This is the same as the add list in the STRIPS notation
- Corresponding to the delete list, we have $effect^{-}(a)$
- Then given as state S in which the action a is applicable, the next state S' after the action a is applied is given by

$$S' \leftarrow \{S - effects^{-}(a)\} \cup effects^{+}(a)$$

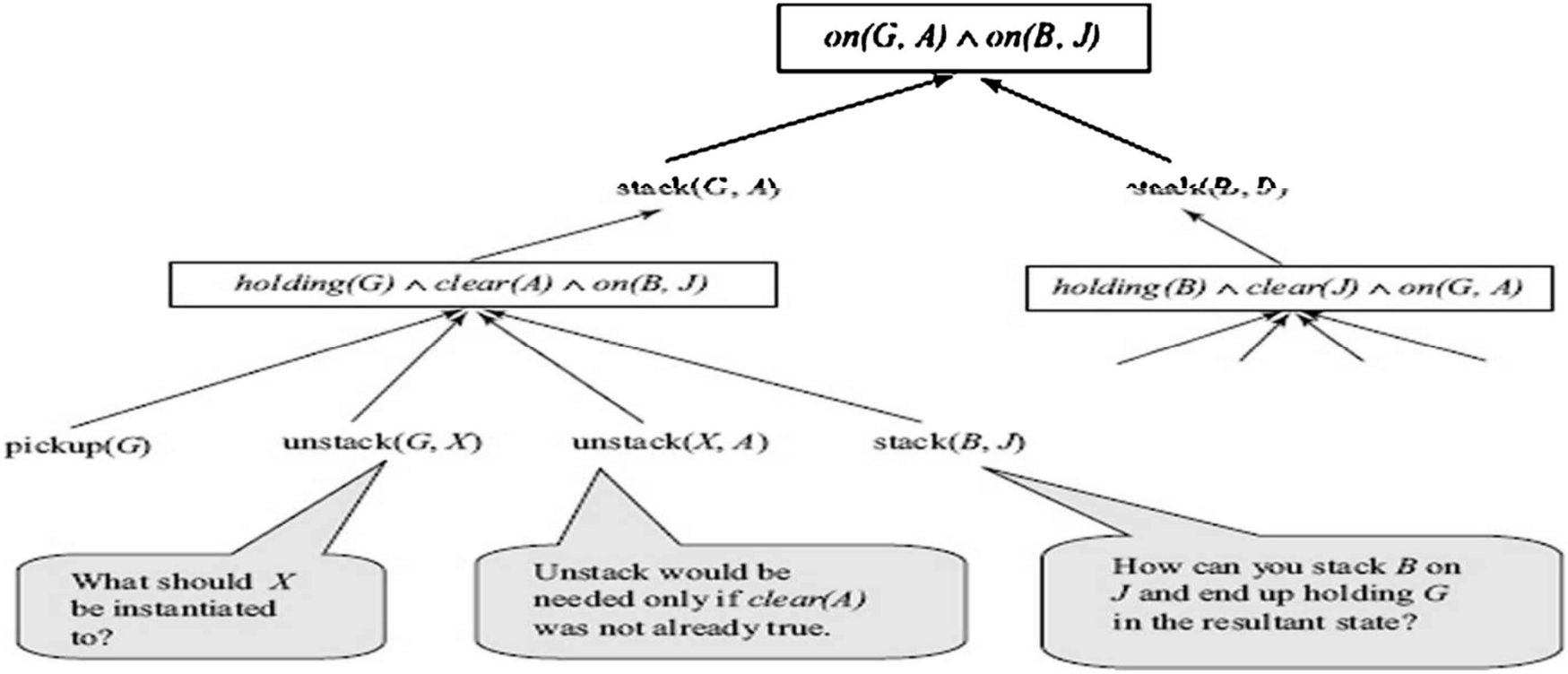
- We say that the state has progressed through the action a.
- Let us define a function Progress(A, S) that returns the successor state when action A is applied to S
- Given the progress(A,S) function, the search algorithms can be used.
- However they will suffer with same drawbacks
  - The search space generated will be huge
  - The search algorithm will have no sense of direction

Backward state space planning

- FSSP is not knowing the goal and simply explores the set of all future states possible in some predetermined order.
- Backward Goal based reasoning is fundamental to intelligent behaviour
- It is also the foundation of reasoning with logic
- For backward planning we first need to define the regression- opposite of progression
  - It allows us to move back from set of goal clauses to a set of subgoal clauses

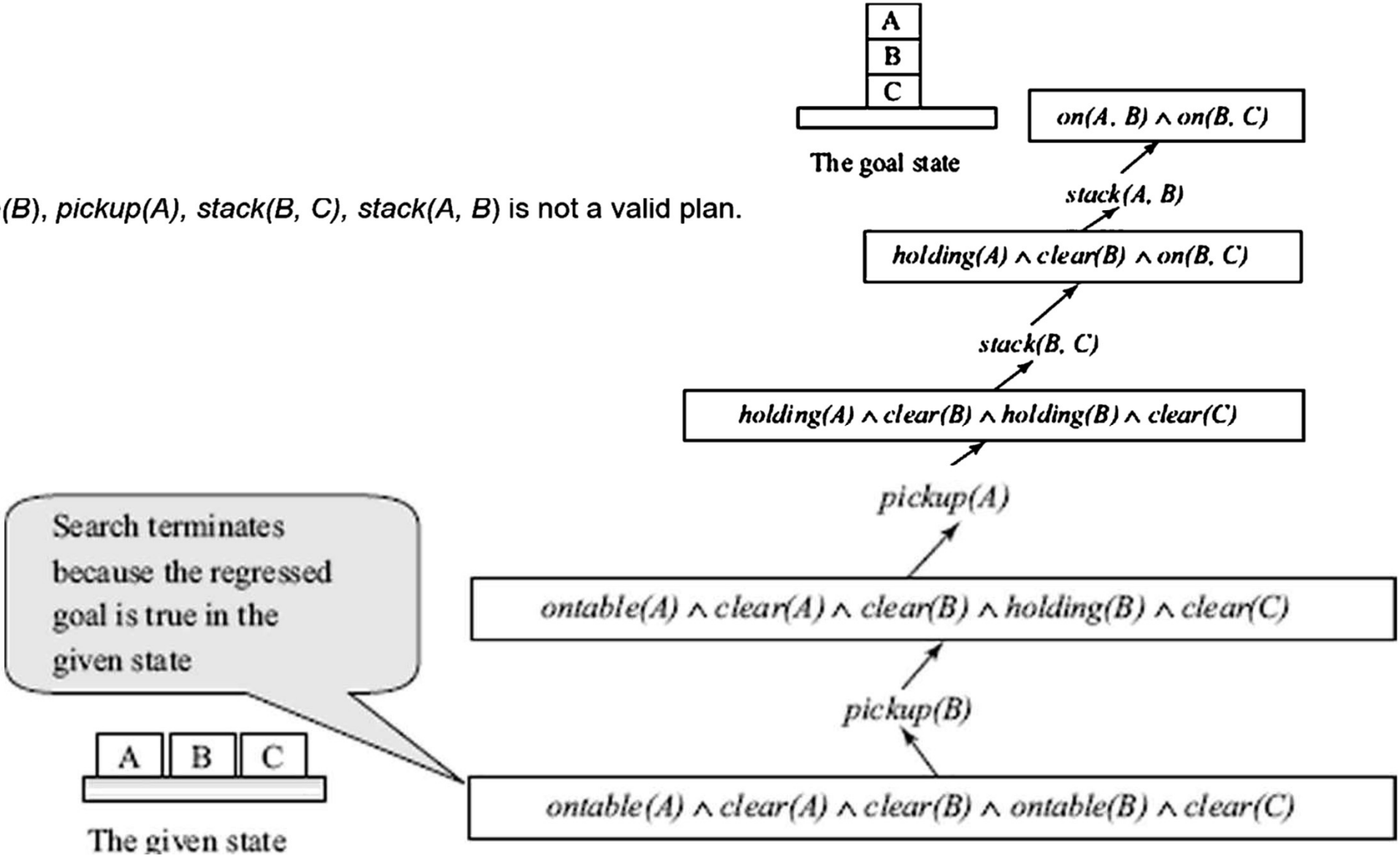
- The regressed goal represents the minimum set of facts that must be true in a state in which the action can be applied and which would result in a goal state.
- Form the set of goal facts , remove the effect of the action and add the preconditions of the action
- Lets define a function regress(A,G) that returns the regressed goal over action A when applied to goal G
- The regresses goal represents the minimum set of facts that must be true in a state in which action can be applied and which would result an goal state

- However the regressor process is not sound



- The relevance of action to a goal simply means that it looks like that the action could achieve some part of the goal.
- But it does not mean that the action may be applicable in the preceding state.

The plan *pickup(B)*, *pickup(A)*, *stack(B, C)*, *stack(A, B)* is not a valid plan.



- One can generate plan and check whether it is a valid plan or not