**19UCI816- ARTIFICIAL INTELLIGENCE AND ROBOTICS**

**UNIT 2**

**PLANNING**

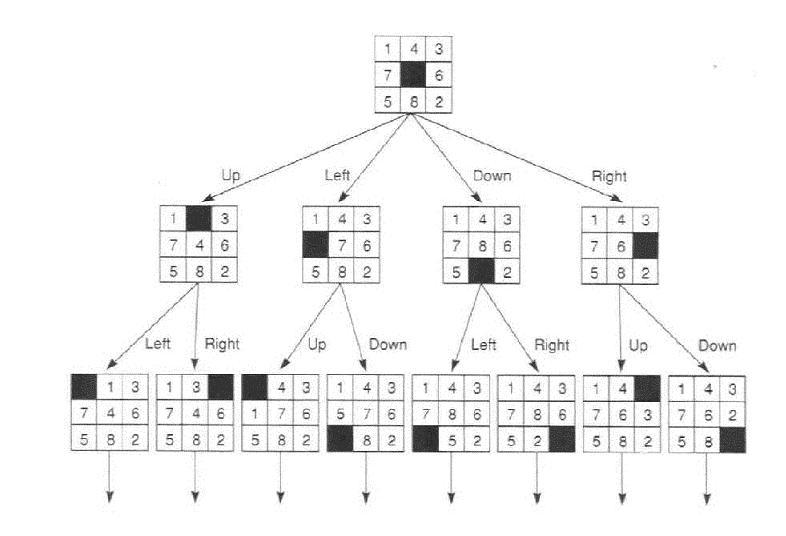
Artificial intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of artificial intelligence!!! But Planning is very important to make any such AI project.

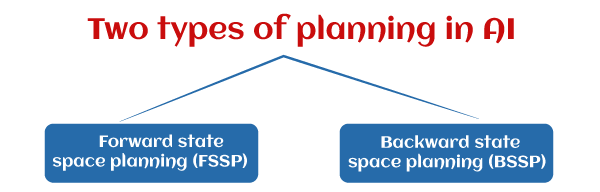
Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.

Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

**Forward State Space Planning (FSSP)** and

**Backward State Space Planning (BSSP)** at the basic level.





### 1. Forward State Space Planning (FSSP)

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

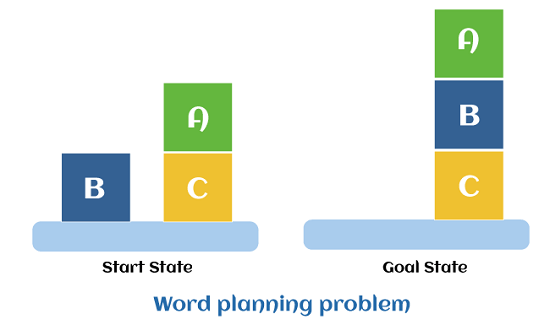
* **Disadvantage**: Large branching factor
* **Advantage**: The algorithm is Sound

### 2. Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

* **Disadvantages**: not sound algorithm (sometimes inconsistency can be found)
* **Advantage**: Small branching factor (much smaller than FSSP)

So for an efficient planning system, we need to combine the features of FSSP and BSSP, which gives rise to target stack planning which will be discussed in the next article.



**Partial-order planning**

**Partial-order planning** is an approach to [automated planning](https://en.wikipedia.org/wiki/Automated_planning) that maintains a partial ordering between actions and only commits ordering between actions when forced to that is, ordering of actions is partial. Also this planning doesn't specify which action will come out first when two actions are processed. By contrast, **total-order planning** maintains a total ordering between all actions at every stage of planning. Given a problem in which some sequence of actions is required in order to achieve a goal, a **partial-order plan** specifies all actions that need to be taken, but specifies an ordering between actions only where necessary.

A **partial-order plan** or **partial plan** is a plan which specifies all actions that need to be taken, but only specifies the order between actions when necessary. It is the result of a partial-order planner. A partial-order plan consists of four components:

* A set of **actions** (also known as **operators**).
* A [**partial order**](https://en.wikipedia.org/wiki/Partial_order) for the actions. It specifies the conditions about the order of some actions.
* A set of **causal links**. It specifies which actions meet which preconditions of other actions. Alternatively, a set of **bindings** between the variables in actions.
* A set of **open preconditions**. It specifies which preconditions are not fulfilled by any action in the partial-order plan.

In order to keep the possible orders of the actions as open as possible, the set of order conditions and causal links must be as small as possible.

A plan is a solution if the set of open preconditions is empty.

A **linearization** of a partial order plan is a total order plan derived from the particular partial order plan; in other words, both order plans consist of the same actions, with the order in the linearization being a [linear extension](https://en.wikipedia.org/wiki/Linear_extension) of the partial order in the original partial order plan.

### Example

For example, a plan for baking a cake might start:

* go to the store
* get eggs; get flour; get milk
* pay for all goods
* go to the kitchen

This is a partial plan because the order for finding eggs, flour and milk is not specified, the agent can wander around the store [reactively](https://en.wikipedia.org/wiki/Reactive_planning) accumulating all the items on its shopping list until the list is complete.

## Partial-order planner

A **partial-order planner** is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) or [program](https://en.wikipedia.org/wiki/Computer_program) which will construct a partial-order plan and search for a solution. The input is the problem description, consisting of descriptions of the **initial state**, the **goal** and possible **actions**.

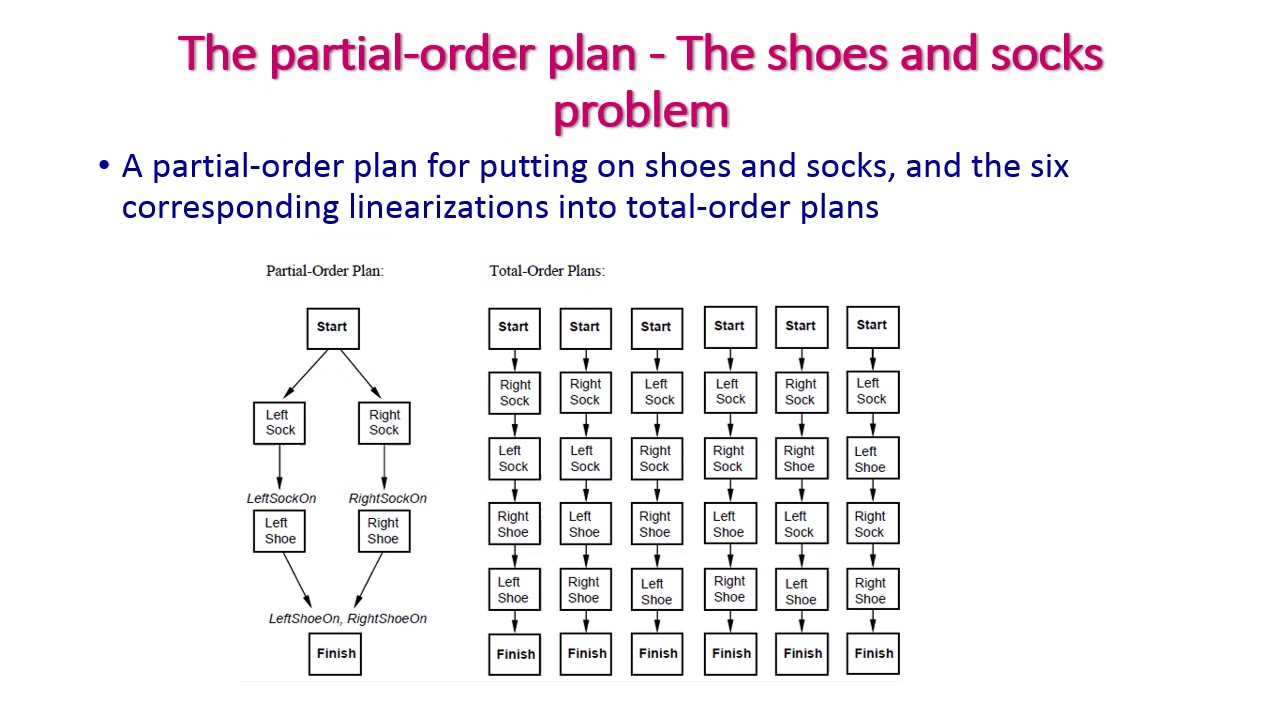
### Threats

As seen in the algorithm presented above, partial-order planning can encounter certain threats, meaning [orderings](https://en.wikipedia.org/w/index.php?title=Orderings&action=edit&redlink=1) that threaten to break connected actions, thus potentially destroying the entire plan. There are two ways to resolve threats:

* [Promotion](https://en.wikipedia.org/wiki/Promotion_(rank))
* [Demotion](https://en.wikipedia.org/wiki/Demotion)

[Promotion](https://en.wikipedia.org/wiki/Promotion_(rank)) orders the possible threat after the connection it threatens. [Demotion](https://en.wikipedia.org/wiki/Demotion) orders the possible threat before the connection it threatens.

Partial-order planning algorithms are known for being both sound and complete, with sound being defined as the total ordering of the algorithm, and complete being defined as the capability to find a solution, given that a solution does in fact exist.



**Hierarchical task network**

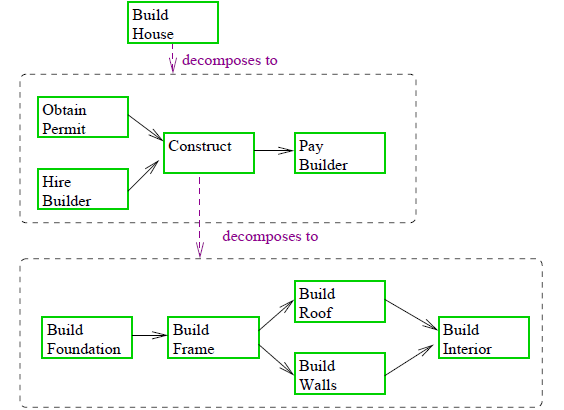
In [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence), **hierarchical task network (HTN) planning** is an approach to [automated planning](https://en.wikipedia.org/wiki/Automated_planning) in which the dependency among actions can be given in the form of hierarchically structured networks.

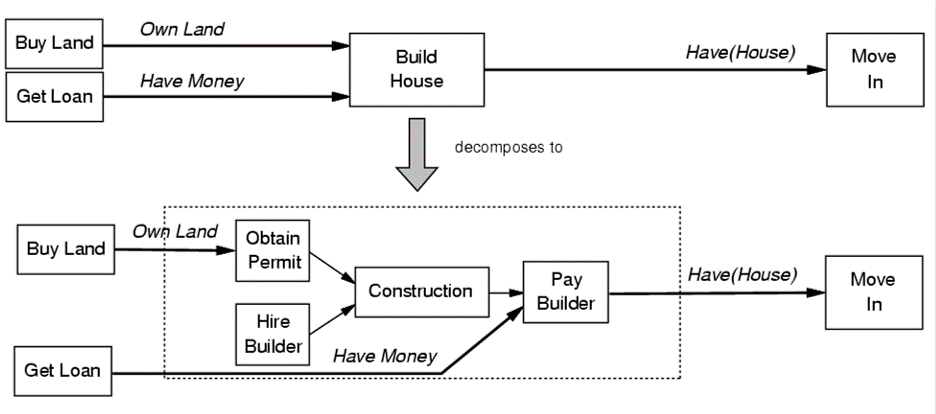
Planning problems are specified in the hierarchical task network approach by providing a set of tasks, which can be:

1. primitive (initial state) tasks, which roughly correspond to the actions of [STRIPS](https://en.wikipedia.org/wiki/Stanford_Research_Institute_Problem_Solver);
2. compound tasks (intermediate state), which can be seen as composed of a set of simpler tasks;
3. Goal tasks (goal state), which roughly corresponds to the goals of [STRIPS](https://en.wikipedia.org/wiki/Stanford_Research_Institute_Problem_Solver), but are more general.

A solution to an HTN problem is then an executable sequence of primitive tasks that can be obtained from the initial task network by decomposing compound tasks into their set of simpler tasks, and by inserting ordering constraints.

A primitive task is an action that can be executed directly given the state in which it is executed supports its precondition. A compound task is a complex task composed of a partially ordered set of further tasks, which can either be primitive or abstract. A goal task is a task of satisfying a condition. The difference between primitive and other tasks is that the primitive actions can be directly executed.





**HTN PLANNING**

* Capture hierarchical structure of planning algorithm
* Planning domain contains no primitive actions and schemas for reducing them
* Reduction schemas:
* Given by the schemas
* Express preferred ways to accomplish to a task

**Planning Graphs**

♦ Planning graphs are an efficient way to create a representation of a planning problem that can be used to

⁄Achieve better heuristic estimates

⁄Directly construct plans

♦ Planning graphs only work for propositional problems.

Planning graphs consists of a sequential of levels that correspond to time steps in the plan.

⁄ Level 0 is the initial state.

⁄ Each level consists of a set of literals and a set of actions that represent what might be possible at that step in the plan

⁄ Might be is the key to efficiency

⁄Records only a restricted subset of possible negative interactions among actions.

Each level consists of

♦ Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.

♦ Actions = all those actions that could have their preconditions satisfied at that time step, depending on which of the literals actually hold.

**PG Example**

Init(Have(Cake))

Goal(Have(Cake) ∧ Eaten(Cake))

Action(Eat(Cake),

PRECOND: Have(Cake)

EFFECT: ¬Have(Cake) ∧ Eaten(Cake))

Action(Bake(Cake),

PRECOND: ¬ Have(Cake)

EFFECT: Have(Cake))

**Planning with Propositional logic**

Propositional logic, also known as sentential logic and statement logic, is the branch of logic that studies ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statements or sentences, as well as the logical relationships and properties that are derived from these methods of combining or altering statements. In propositional logic, the simplest statements are considered as indivisible units, and hence, propositional logic does not study those logical properties and relations that depend upon parts of statements that are not they statements on their own, such as the subject and predicate of a statement. The most thoroughly researched branch of propositional logic is classical truth-functional propositional logic, which studies logical operators and connectives that are used to produce complex statements whose truth-value depends entirely on the truth-values of the simpler statements making them up, and in which it is assumed that every statement is either true or false and not both. However, there are other forms of propositional logic in which other truth-values are considered, or in which there is consideration of connectives that are used to produce statements whose truth-values depend not simply on the truth-values of the parts, but additional things such as their necessity, possibility or relatedness to one another.

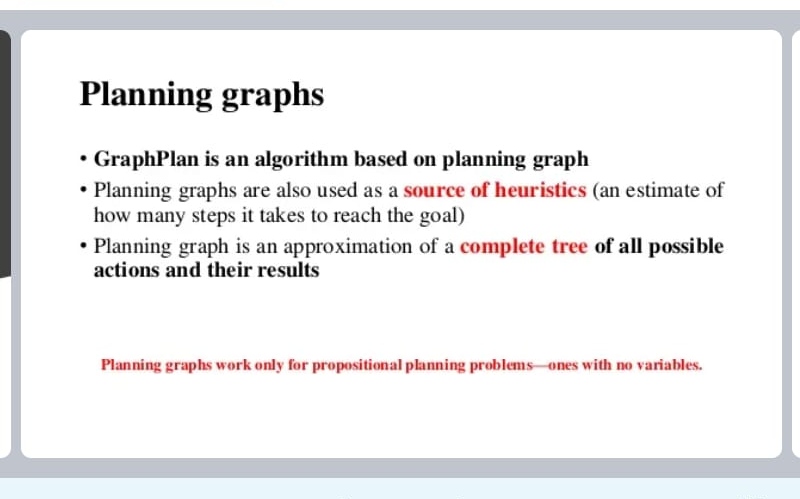


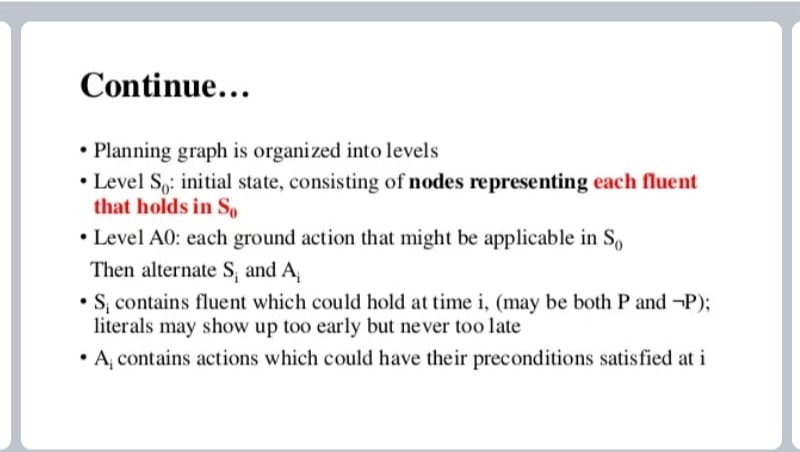


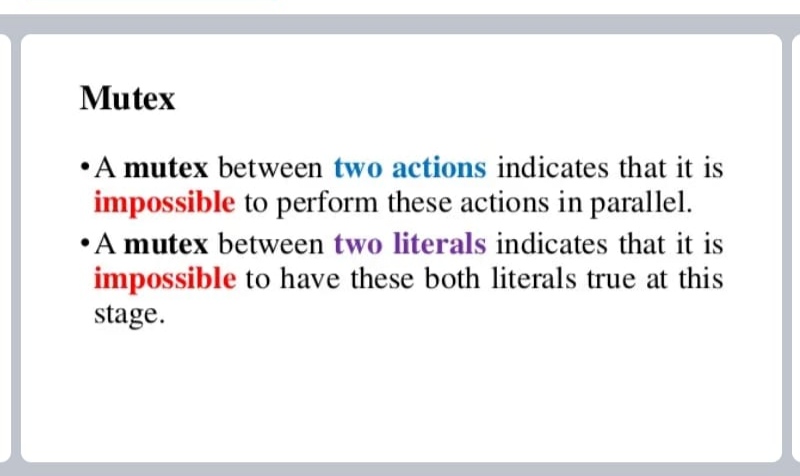


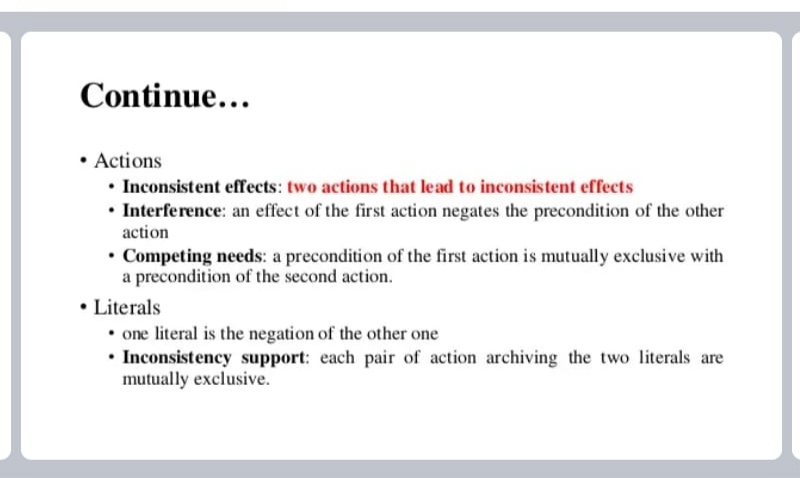


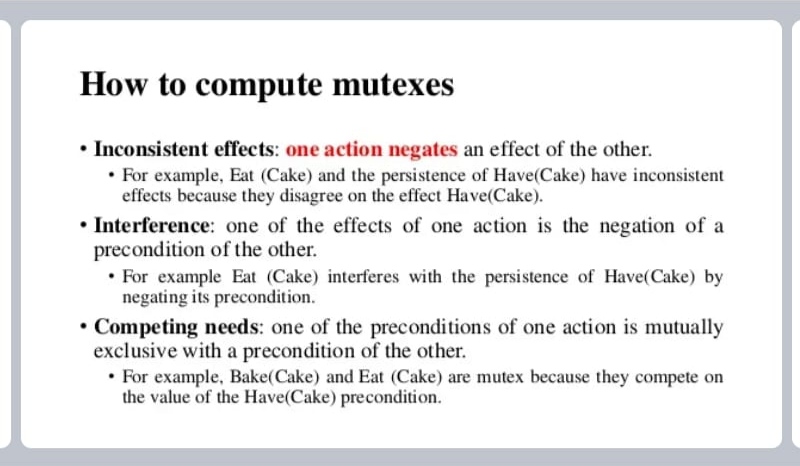
**PLANNING GRAPH**

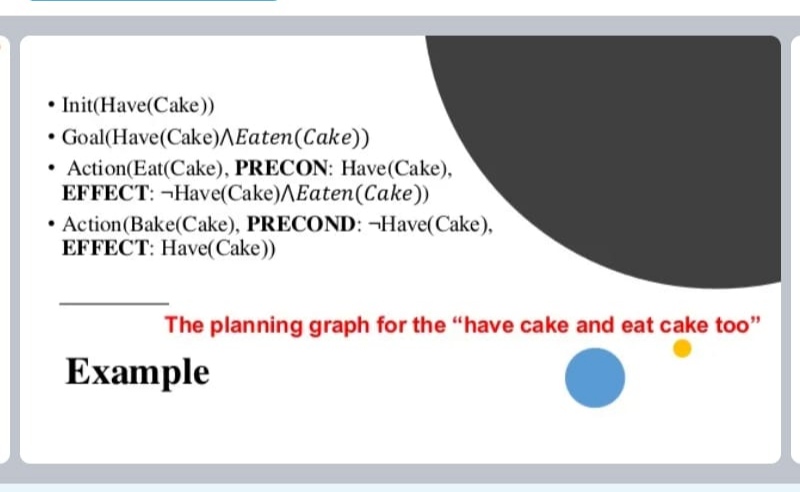
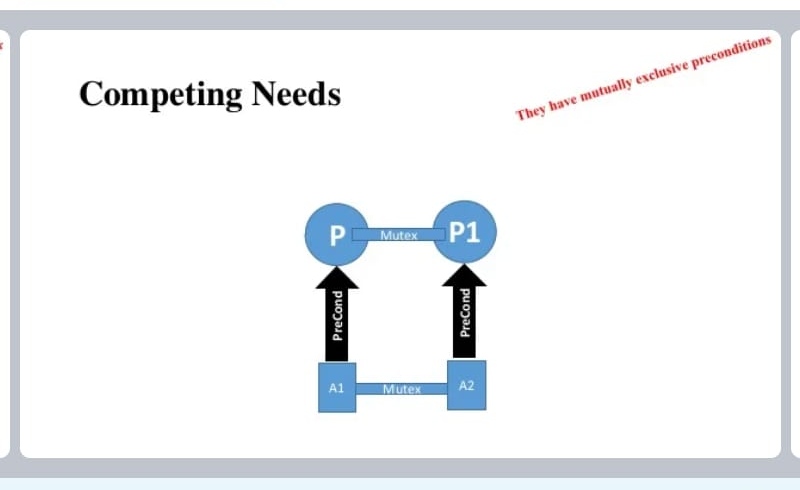
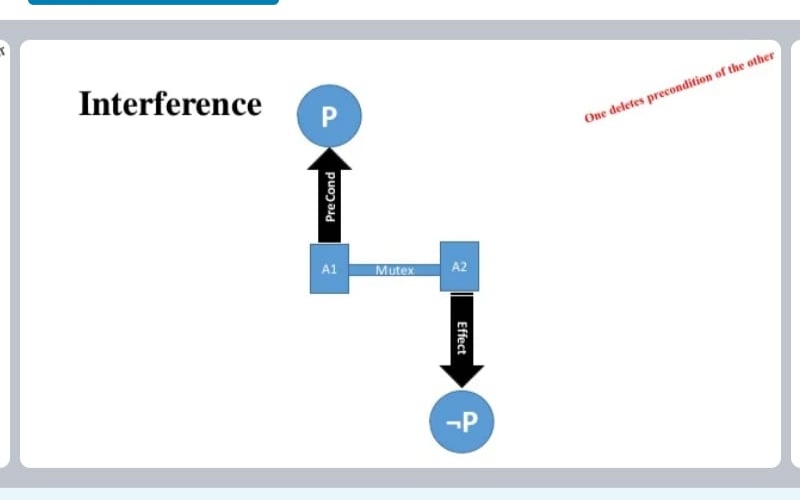
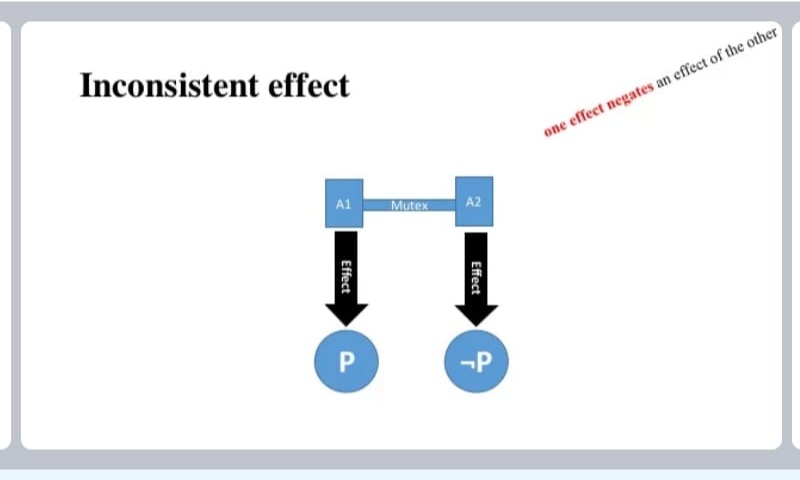












**PLANNING GRAPH**

