REVA UNIVERSITY
Bengaluru, India

Artificial Intelligence

M21DES221 – Academic Year 2024-25 – Sem II Even Semester

MCA , School of CSA.

www.reva.edu.in

Shreetha Bhat
Assistant Professor

Unit III - Planning, Learning and Robotics

Lecture 1

Agenda

- Quick look on Syllabus
- Planning
- The Blocks World Problem
- Goal Stack Planning

Quick Look on Syllabus

Unit 3 – 13 Hrs

Planning, Learning and Robotics

Planning: The Blocks World, Components of a Planning System, Goal Stack Planning, Nonlinear Planning Using Constraint Posting, Hierarchical Planning, Other Planning Techniques.

Learning: Rote Learning, learning by Taking Advice, Learning in Problem-solving, Learning from Examples: Induction, Explanation-based Learning, Discovery, Analogy, Formal Learning Theory. Learning in Neural and Belief Networks' How the Brain Works, Neural Networks, perceptions.

Robotics: Introduction, Robot Hardware, Robotic Perception, Robotic Software Architectures, Application Domains.

Planning

In everyday language, "planning" means figuring out a sequence of actions before performing them.

However, in computer problem-solving, the line between planning and doing often blurs, as computers usually just generate plans without physically acting.

For example, in the 8-puzzle problem, a computer can only plan how to solve it, not move tiles physically.

Planning

- A critical part of Artificial Intelligence which deals with the actions and domains of a particular problem.
- Is considered as the reasoning side of acting.

Example

Reaching a particular destination requires planning

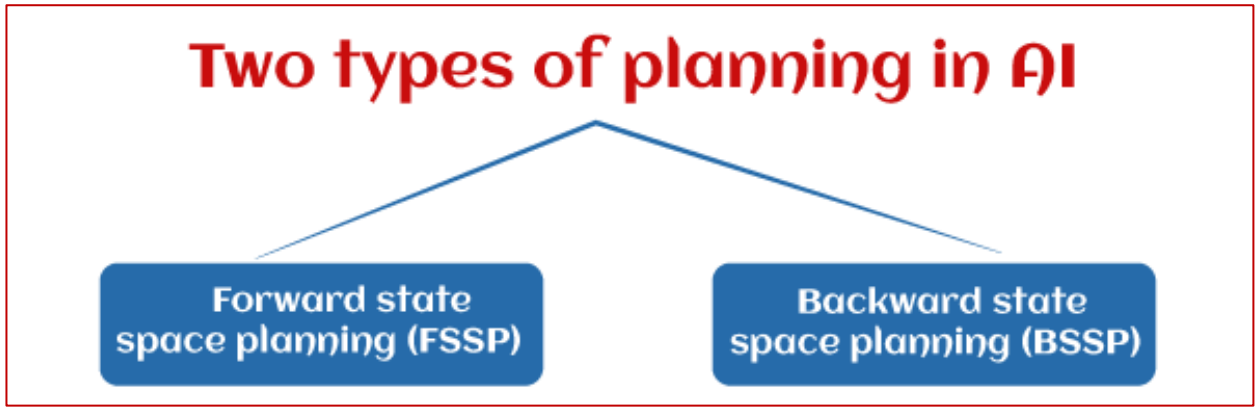
Planning

Any planning system needs 3 things:

1. Domain Description.
2. Action Specification and
3. Goal Description.

Planning

A plan is assumed to be a sequence of actions and each action has its own set of preconditions to be satisfied before performing the action and also some effects which can be positive or negative.



Planning

Forward State Space Planning (FSSP) - Progression

- It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' called a progression.
- It continues until we reach the target position.
- Action should be taken in this matter.

Planning

Forward State Space Planning (FSSP) - Progression

Advantage: The algorithm is Sound

Disadvantage: Large branching factor

Planning

Backward State Space Planning (BSSP) - Regression

- 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.

Planning

Backward State Space Planning (BSSP) - - Regression

Advantage: Small branching factor (much smaller than FSSP)

Disadvantages: not sound algorithm (sometimes inconsistency can be found)

Planning

Components of a Planning System

The plan includes the following important steps:

1. Choose the best rule to apply the next rule based on the best available guess.
2. Apply the chosen rule to calculate the new problem condition.
3. Find out when a solution has been found.
4. Detect dead ends so they can be discarded and direct system effort in more useful directions.
5. Find out when a near-perfect solution is found.

The Blocks World Problem

- There is a flat surface on which some blocks are placed.
- There are a number of square blocks all of same size.
- They can be stacked one upon another.
- There is robot arm that can manipulate the blocks.

The Blocks World Problem

The actions it can perform include:

UNSTACK(A, B) — Pick up block A from its current position on block B. The arm must be empty and block A must have no blocks on top of it.

STACK(A, B) — Place block A on block B. The arm must already be holding A, and the surface of B must be clear.

PICKUP(A) — Pick up block A from the table and hold it. The arm must be empty, and there must be nothing on top of block A.

PUTDOWN(A) — Put block A down on the table. The arm must have been holding block A.

The Blocks World Problem

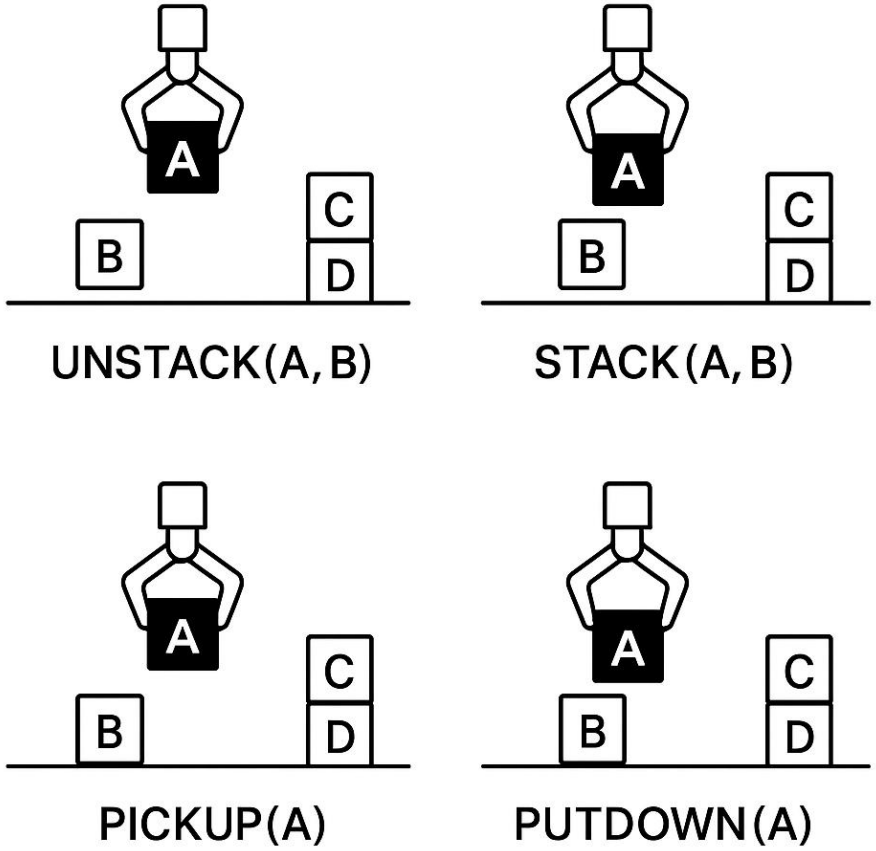
The actions include:

UNSTACK(A, B)

STACK(A, B)

PICKUP(A)

PUTDOWN(A)



The Blocks World Problem

- The robot arm can hold only one block at a time.
- Also, since all blocks are of same size each block can have at most one other block directly on top of it.
- In order to specify both the conditions under which an operation may be performed and the results of performing it , we need to use the following predicates:

The Blocks World Problem

Representing The Configurations As A List Of "Predicates"

1. ON(A,B) : Block A is on Block B.
2. ONTABLE(A) : Block A is on the table.
3. CLEAR(A) : There is Nothing on top of Block A
4. HOLDING (A): The arm is holding Block A.
5. ARMEMPTY : The arm is holding nothing

The Blocks World Problem

- Various logical statements are true in this block world.

Example:

$$[\exists x : \text{HOLDING}(x)] \rightarrow \neg\text{ARMEMPTY}$$
$$\forall x : \text{ONTABLE}(x) \rightarrow \neg\exists y : ON(x,y)$$
$$\forall x : [\neg\exists y : ON(y, x)] \rightarrow \text{CLEAR}(x)$$

1. The first statement: If the arm is holding anything , then it is not empty.
2. The Second Statement: If a block is on the table then it is not also on another table then it is also on another block.
3. The Third Statement: Any Block with no Blocks on it is clear.

Goal Stack Planning

- Goal Stack Planning is one of the earliest methods for solving compound goals that may interact.
- This was the approach used by STRIPS.
- In this method, the problem solver makes use of a single stack that contains both goals and operators that have been proposed to satisfy those goals.
- The problem solver also relies on a database that describes the current situation and a set of operators described as PRECONDITION , ADD and DELETE lists.

Goal Stack Planning

Example

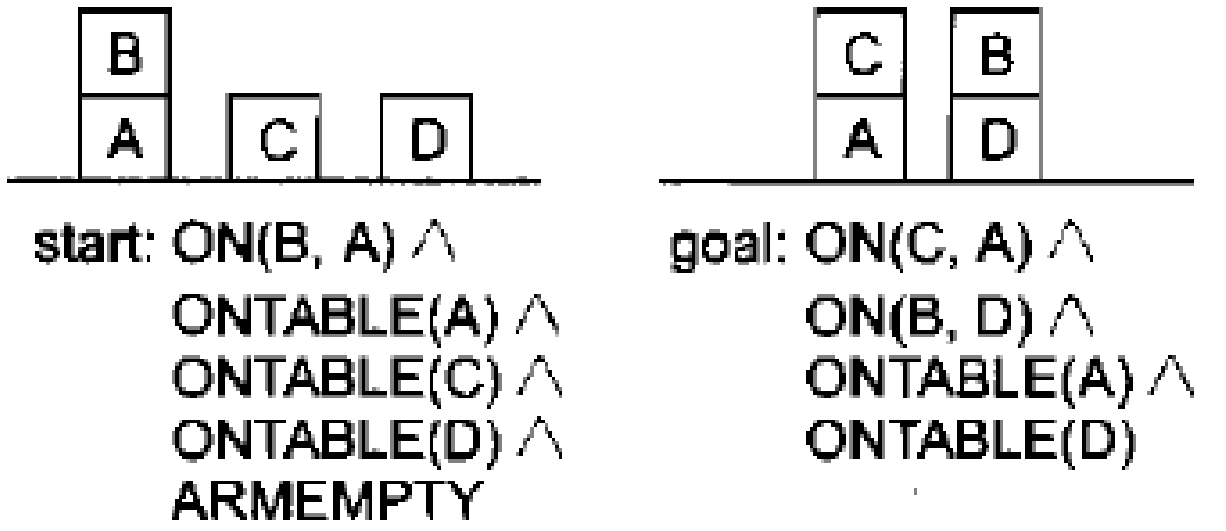


Fig. 13.4 *A Very Simple Blocks World Problem*

The goal stack is simply

ON(C, A) ∧ ON(B, D) ∧ ONTABLE(A) ∧ ONTABLE(D)

Goal Stack Planning

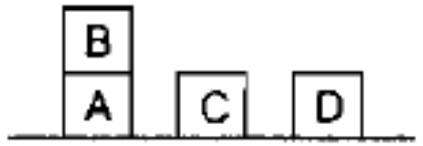
Example

- We want to separate this problem in to four sub problems, one for each component of the original goal.
- Two of the sub problems, ONTABLE(A) and ONTABLE(D) are already true in the initial state. We work on the remaining two.
- Depending on order in which we want to tackle the subproblems, there are two goal stacks that could be created as our first step, where each line represents one goal on the stack and OTAD is an abbreviation for ONTABLE(A) and ONTABLE(D) :

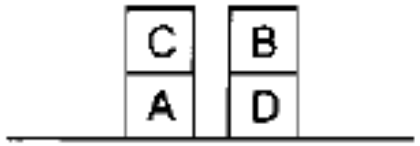
ON(C, A)	ON(B, D)
ON(B, D)	ON(C, A)
ON(C, A) ∧ ON(B, D) ∧ OTAD	ON(C, A) ∧ ON(B,D) ∧ OTAD
[1]	[2]

Goal Stack Planning

Example



start: ON(B, A) ∧
ONTABLE(A) ∧
ONTABLE(C) ∧
ONTABLE(D) ∧
ARMEMPTY



goal: ON(C, A) ∧
ON(B, D) ∧
ONTABLE(A) ∧
ONTABLE(D)

Fig. 13.4 *A Very Simple Blocks World Problem*

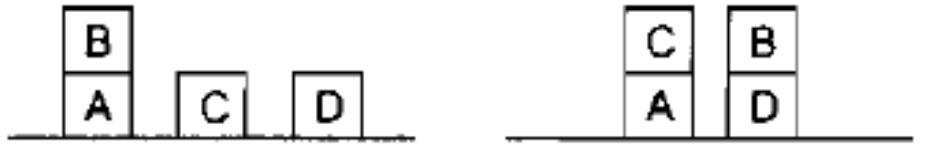
ON(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ OTAD
[1]

STACK(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ OTAD

Precondition: CLEAR(A) ∧ HOLDING(C)

Goal Stack Planning

Example



start: ON(B, A) ∧
ONTABLE(A) ∧
ONTABLE(C) ∧
ONTABLE(D) ∧
ARMEMPTY

goal: ON(C, A) ∧
ON(B, D) ∧
ONTABLE(A) ∧
ONTABLE(D)

Fig. 13.4 *A Very Simple Blocks World Problem*

CLEAR(A)
HOLDING(C)
CLEAR(A) ∧ HOLDING(C)
STACK(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ OTAD

Goal Stack Planning

Example

Next we check to see if CLEAR(A) is true. It is not. The only operator that could make it true is UNSTACK (B, A). So we will attempt to apply it. This produces the goal stack

ON(B, A)
CLEAR(B)
ARMEMPTY
ON(B, A) ∧ CLEAR(B) ∧ ARMEMPTY
UNSTACK(B, A)
HOLDING(C)
CLEAR(A) ∧ HOLDING(C)
STACK(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ OTAD

This time, when we compare the top element of the goal stack, ON(B, A), to the world model, we see that it is satisfied. So we pop it off and consider the next goal, CLEAR(B). It, too, is already true in the world

Goal Stack Planning

Example

Now the top element of the stack is the operator UNSTACK(B, A). We are now guaranteed that its preconditions are satisfied, so it can be applied to produce a new world model from which the rest of the problem-solving process can continue. This is done using the ADD and DELETE lists specified for UNSTACK.

The goal stack now is

HOLDING(C)
CLEAR(A) ∧ HOLDING(C)
STACK(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ OTAD

Goal Stack Planning

Example

At each succeeding step of the problem-solving process, the top goal on the stack will be pursued. When a sequence of operators that satisfies it is found, that sequence is applied to the state description, yielding a new description. Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal. This process continues until the goal stack is empty. Then, as one last check, the original goal is compared to the final state derived from the application of the chosen operators. If any components of the goal are not satisfied in that state (which they might not be if they were achieved at one point and then undone later), then those unsolved parts of the goal are reinserted onto the stack and the process resumed.

Goal Stack Planning

Example

1. UNSTACK(B.A)
2. STACK(B.D)
3. PICKUP(C)
4. STACK(C, A)

Goal Stack Planning

Block Worlds Problem

- The effect of these operations is represented using two lists **ADD** and **DELETE.**
- DELETE List contains the predicates which will cease to be true once the operation is performed.
- ADD List on the other hand contains the predicates which will become true once the operation is performed.

Goal Stack Planning

Block Worlds Problem

The Precondition, Add and Delete List for each operation is rather intuitive and have been listed below.

OPERATORS	PRECONDITION	DELETE	ADD
STACK(X,Y)	CLEAR(Y)∧ HOLDING(X)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(X,Y)
UNSTACK(X,Y)	ARMEMPTY∧ ON(X,Y)∧ CLEAR(X)	ARMEMPTY∧ ON(X,Y)	HOLDING(X) ∧CLEAR(Y)
PICKUP(X)	CLEAR(X)∧ ONTABLE(X)∧ ARMEMPTY	ONTABLE(X)∧ ARMEMPTY	HOLDING(X)
PUTDOWN(X)	HOLDING(X)	HOLDING(X)	ONTABLE(X)∧ ARMEMPTY

Goal Stack Planning

Steps

- Define the Final Goal
- Break Down into Subgoals
- Push Goals onto the Stack
- Solve Each Subgoal
- Pop Completed Goals
- Repeat Until Goal is Met

Unit III - Planning, Learning and Robotics

Lecture 2

Recap

- Quick look on Syllabus
- Planning
- The Blocks World Problem
- Goal Stack Planning

Agenda

- **Nonlinear Planning Using Constraint Posting**

Nonlinear Planning Using Constraint Posting

- Most problems require an intertwined plan in which multiple subproblems are worked on simultaneously.
- Such a plan is called a nonlinear plan because it is not composed of a linear sequence of complete subplans.
- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators , partial orderings between operators and binding of variables within operators.

Nonlinear Planning Using Constraint Posting

- The goal stack planning method attacks problems by solving the goals one at a time in order.
- A plan generated by this method contains a sequence of operators.
- These operators are used to attain the first goal followed by a complete sequence for the second goal etc.
- But difficult problems cause goal interactions.
- The operators used to solve one subproblem may interfere with the solution to a previous subproblem.

Nonlinear Planning Using Constraint Posting

- Most of the problems require an intertwined plan in which multiple subproblems are worked on simultaneously.
- Such a plan is called a nonlinear plan because it is not composed of a linear sequence of complete subplans.
- In simple words, a plan that consists of sub-problems, which are solved simultaneously is said to be a non-linear plan.

Nonlinear Planning Using Constraint Posting

- In case of the goal stack planning it poses some problems.
- Achieving a goal could possibly undo any of the already achieved goals and its called as Sussman`s anomaly.
- In linear planning, just one goal is taken at a time and solved completely before the next one is taken.

Example: Car for Servicing, Wearing of shoes

Nonlinear Planning Using Constraint Posting

Constraint posting builds up a plan by:

- suggesting operators,
- trying to order them, and
- produce bindings between variables in the operators and actual blocks.

Nonlinear Planning Using Constraint Posting

- At any given time in problem solving process, we may have a set of useful operators.
- But perhaps no clear idea of how those operators should be ordered with respect to each others.
- Here a solution is partially ordered to generate the actual plan, we convert the partial order into any number of total order.

Nonlinear Planning Using Constraint Posting

- Let us now examine several operations for nonlinear planning in a constraint posting environment by incrementally generating a nonlinear plan to solve the Sussman anomaly problem.

Nonlinear Planning Using Constraint Posting

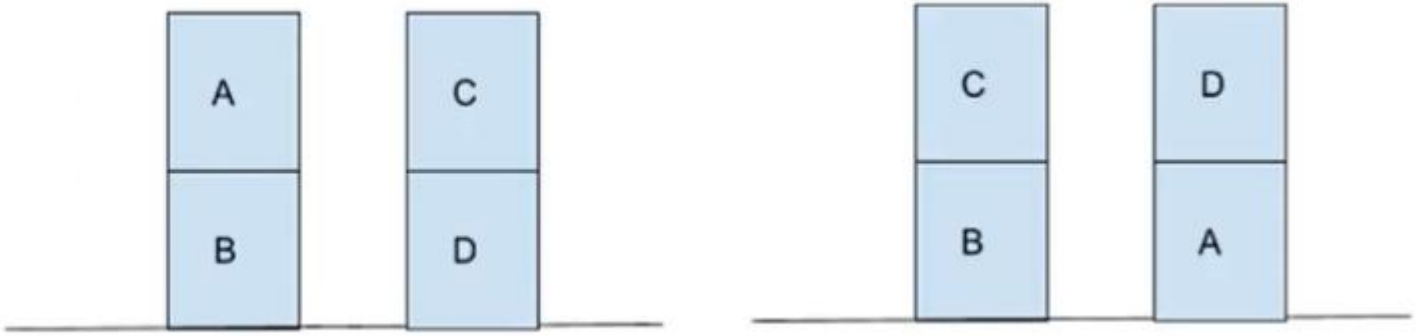
- The initial plan consists of no steps and by studying the goal state ideas for the possible steps are generated.
- There is *no order or detail* at this stage.
- Gradually more detail is introduced and constraints about the order of subsets of the steps are introduced.
- This will continue until a *completely ordered* sequence is created.

Nonlinear Planning Using Constraint Posting

- Step Addition : Creating new steps for a plan
- Promotion : Restrict one step to come before another in a final plan.
- DeClobbering : Placing one step s2 between two old steps s1 and s3 such that s2 makes it possible for s3 to be apply which was previously prevented by s1
- Simple Establishment : Assigning a value to a variable.
- Separation : Preventing the assignment of certain values to a variable.

Nonlinear Planning Using Constraint Posting

Example



Steps Used:

1. Step Addition
2. Promotion

ON(A,B) /\ ON(C,D)

ON (A,B) /\ ON (C,D) /\ ONTABLE(B) /\ ONTABLE(D)

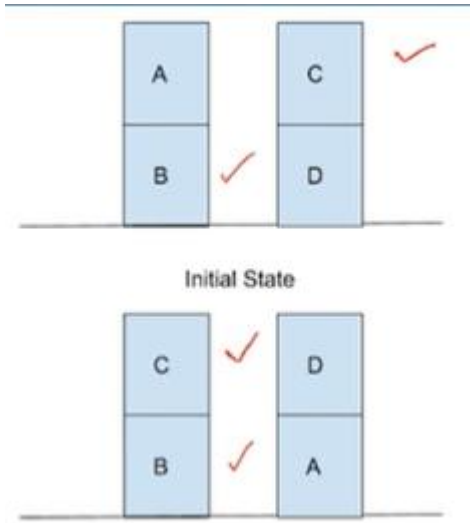
ON (C,B) /\ ON (D,A) /\ ONTABLE(B) /\ CLEAR(C) /\ CLEAR(D) /\ ONTABLE(A)

Predicates

https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example



Let's take the initial state as our current state and start the plan creation with the goal state.

ON (C,B) ∧ ON (D,A) ∧ ONTABLE(B) ∧
CLEAR(C) ∧ CLEAR(D) ∧ ONTABLE(A)

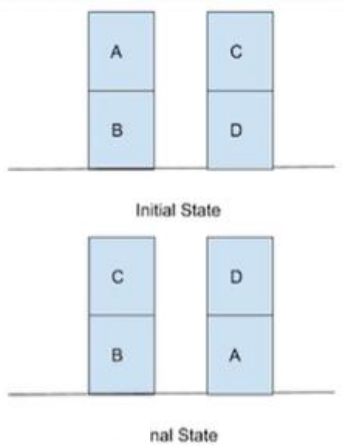
- If we observe above we could understand that certain subgoals are already true and it is highlighted in Red.
- Therefore, no need to consider that part in our process and consider the unachieved goals only.

ON (C,B) ∧ ON (D,A) ∧ CLEAR(D) ∧ ONTABLE(A)

https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example



These three subgoals are not true in our current situation, so use step addition to achieve these subgoals.

ON (C,B) ∧ ON (D,A) ∧ CLEAR(D) ∧ ONTABLE(A)

HOLDING(C)
CLEAR(B)

STACK(C,B)

ARMEMPTY
ON(C,B)
~HOLDING(C)
~CLEAR(B)

HOLDING(D)
CLEAR(A)

STACK(D,A)

ARMEMPTY
ON(D,A)
~HOLDING(D)
~CLEAR(A)

ON (x,D)
ARMEMPTY
CLEAR(x)

UNSTACK(x,D)

HOLDING(x)
CLEAR(D)
~ARMEMPTY
~ON(x,D)

HOLDING (A)

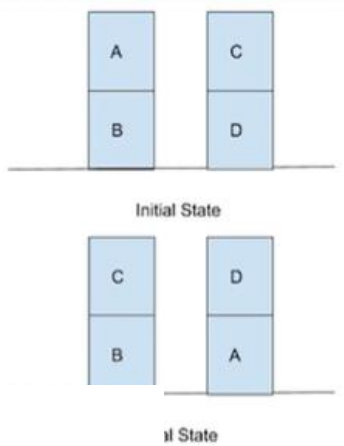
PUTDOWN(A)

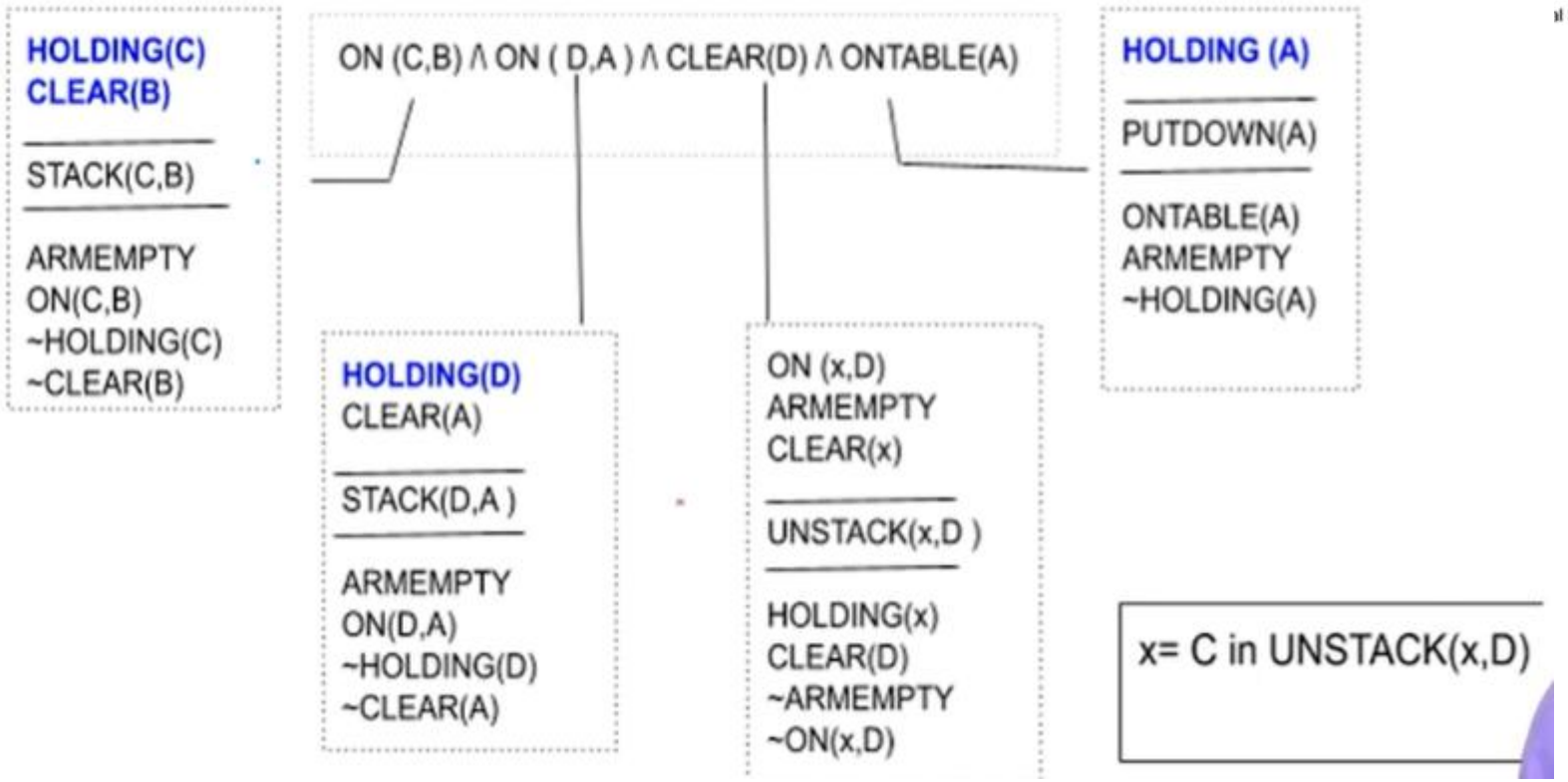
ONTABLE(A)
ARMEMPTY
~HOLDING(A)

Nonlinear Planning Using Constraint Posting

Example

Now check the precondition of all the actions are satisfied or not.
The unsatisfied preconditions are highlighted below.

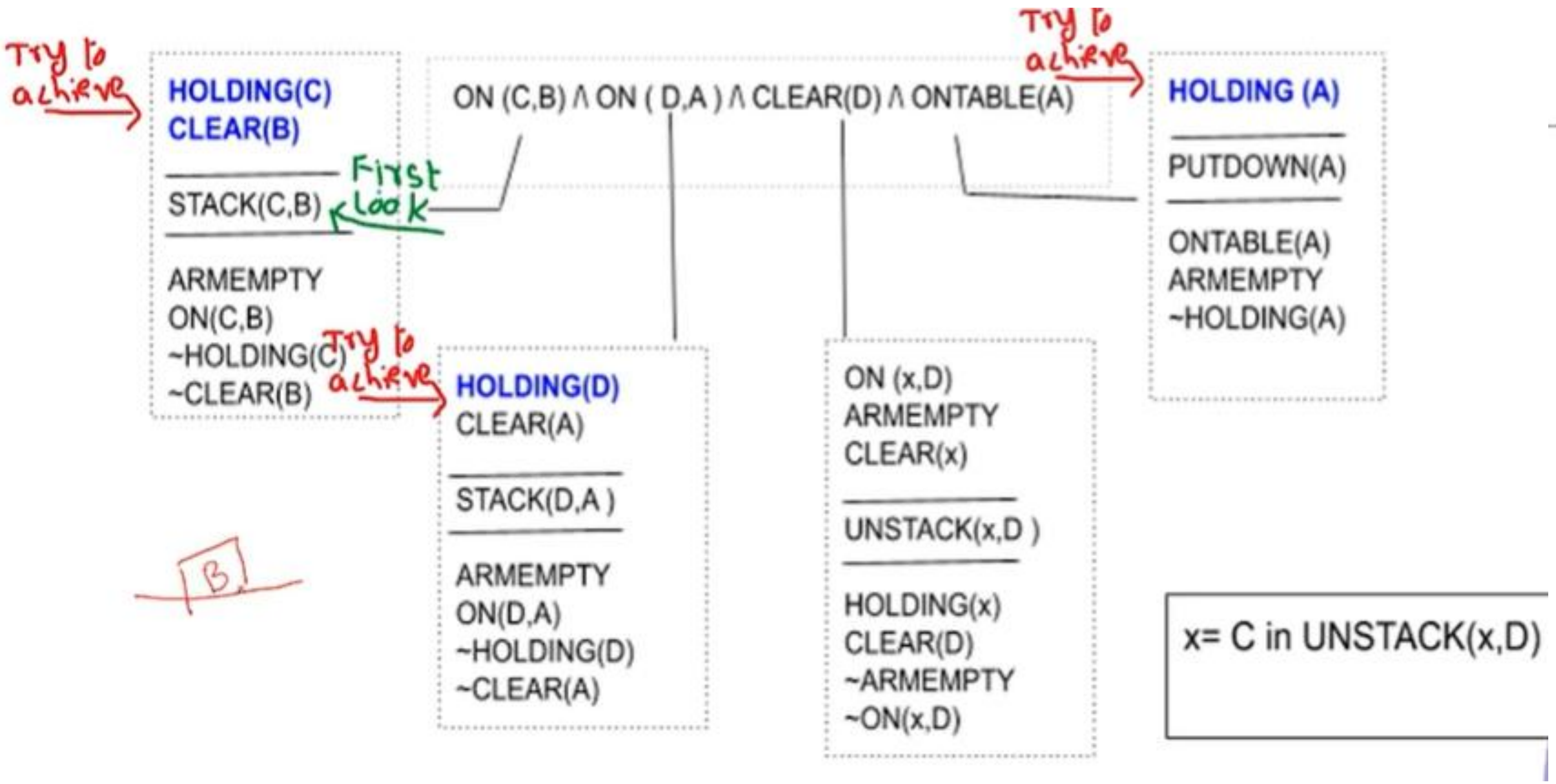


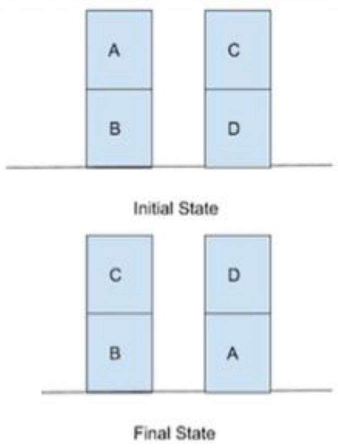


https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example

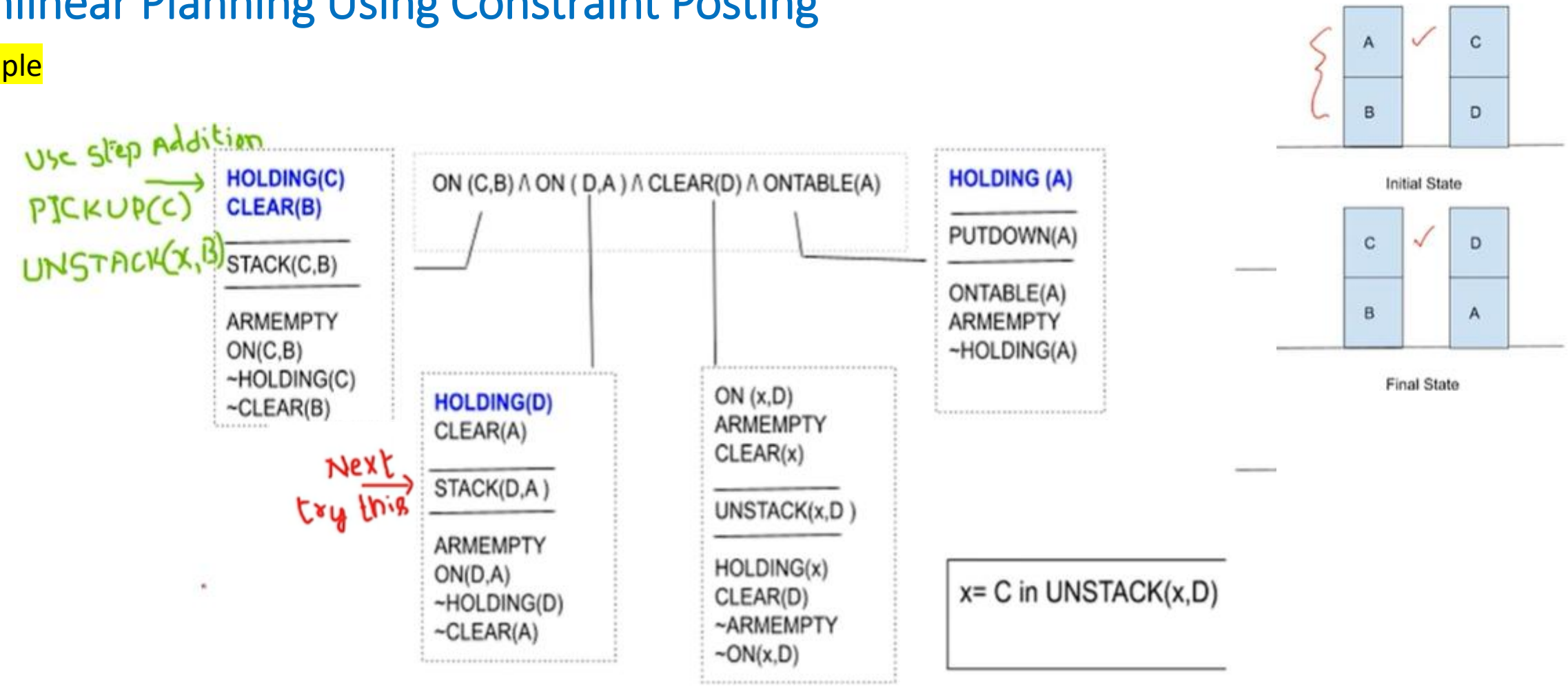




https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

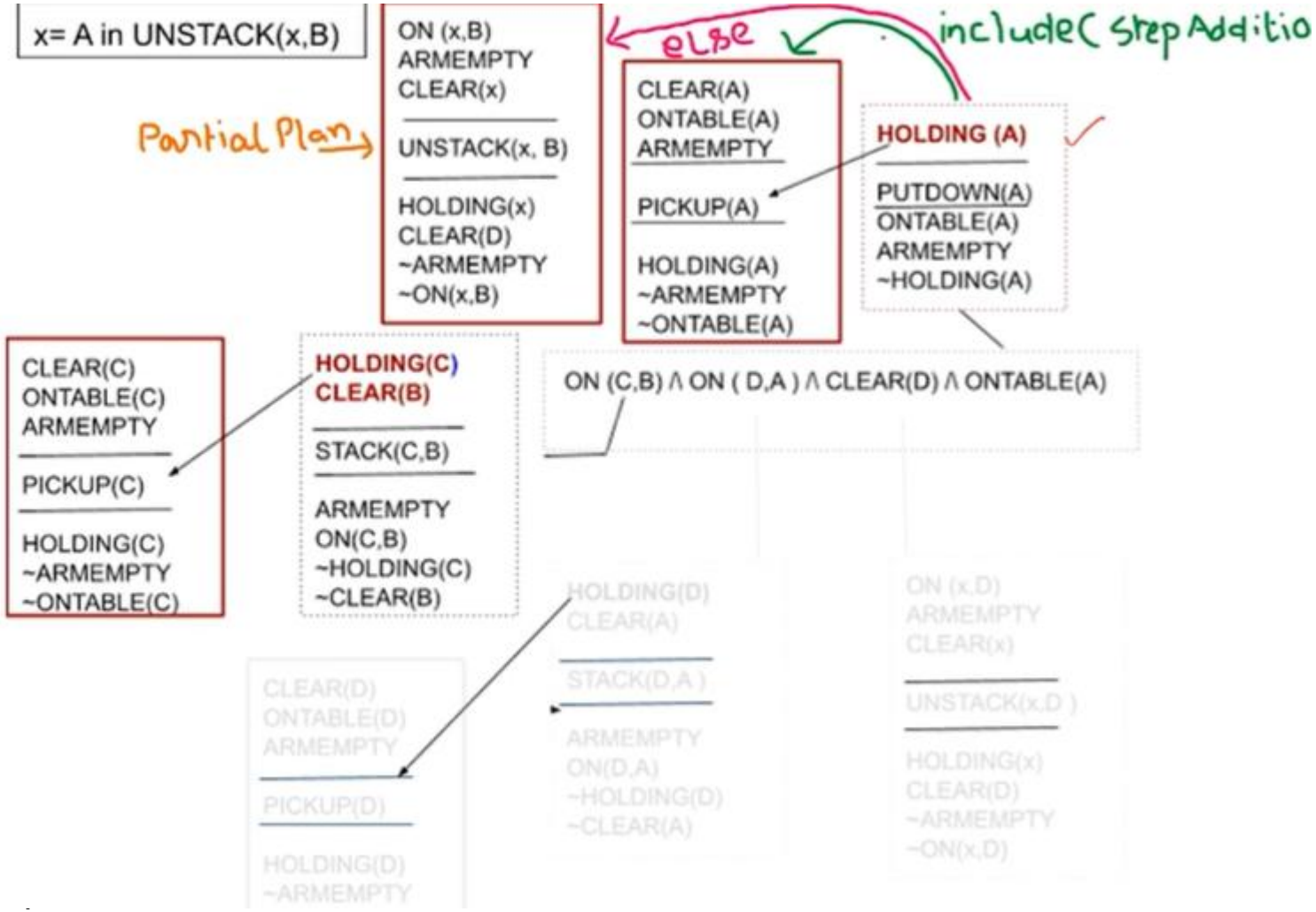
Example

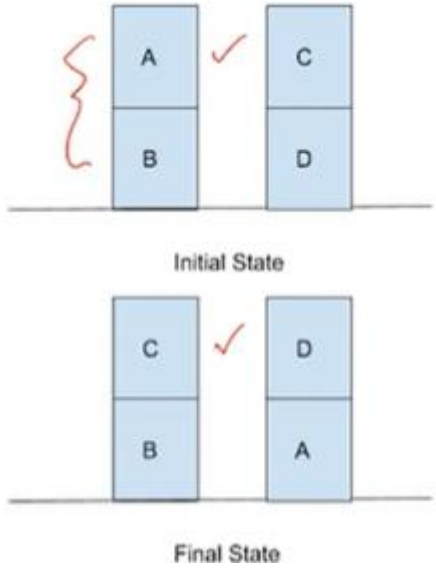


https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example

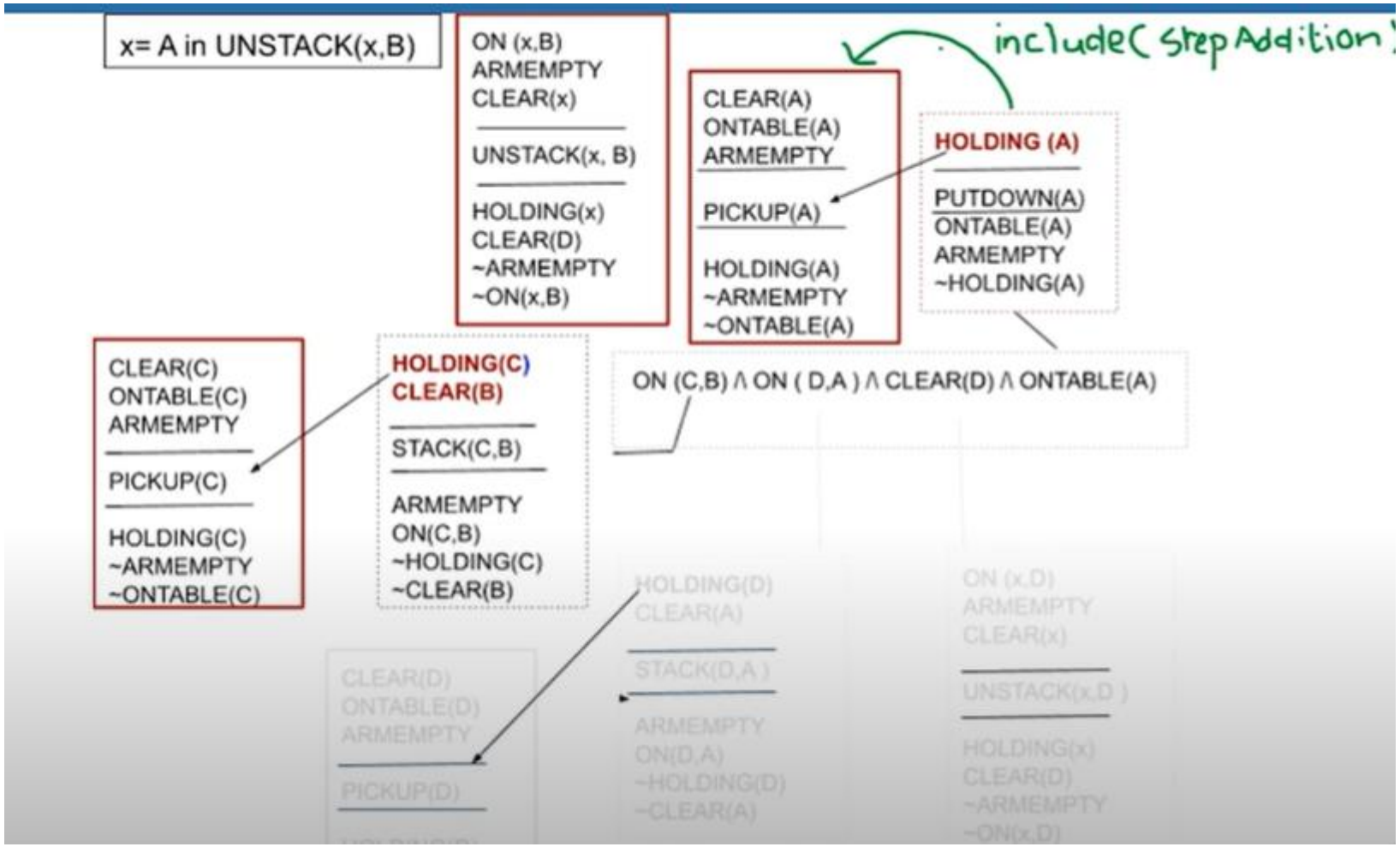




https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

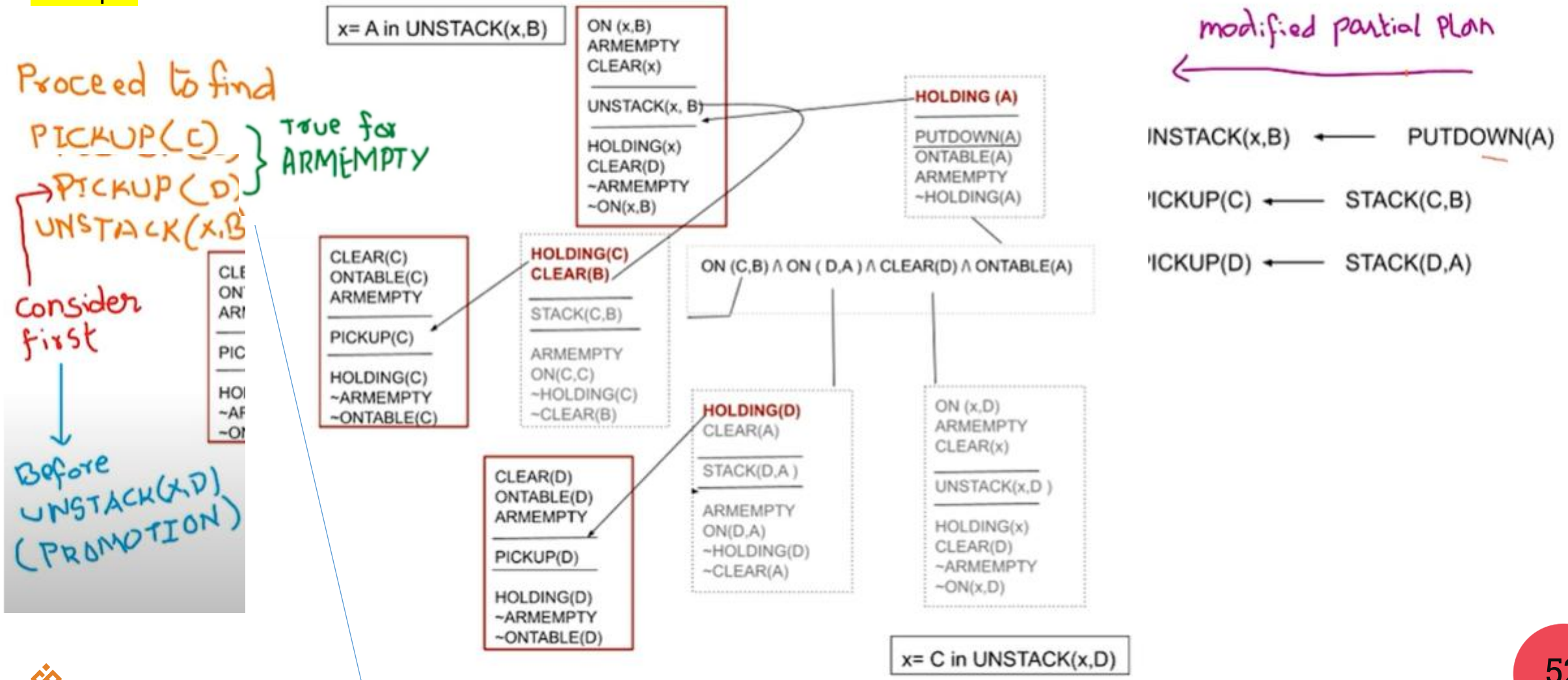
Example



https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example

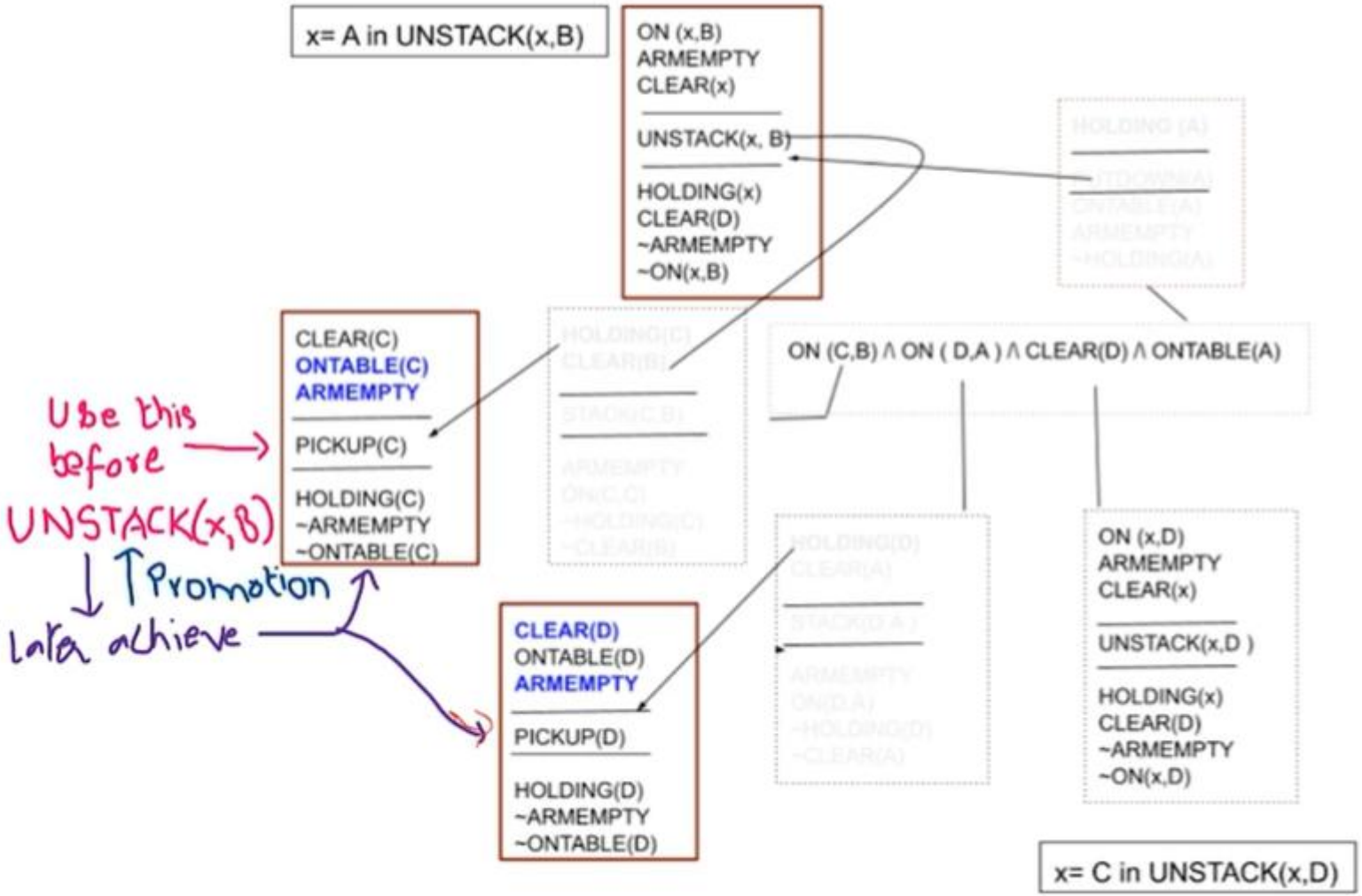


Promotion

https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

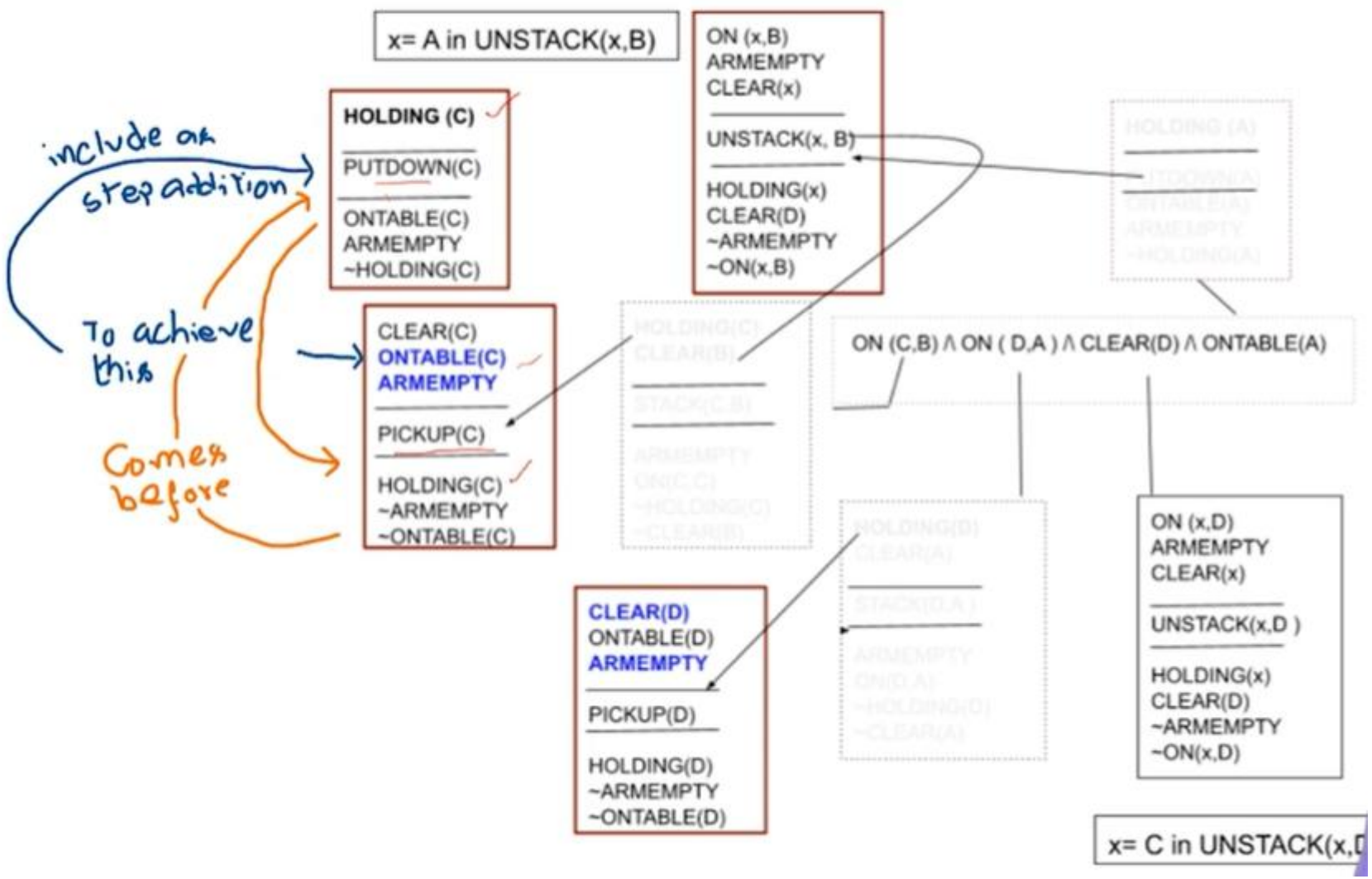
Example



https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

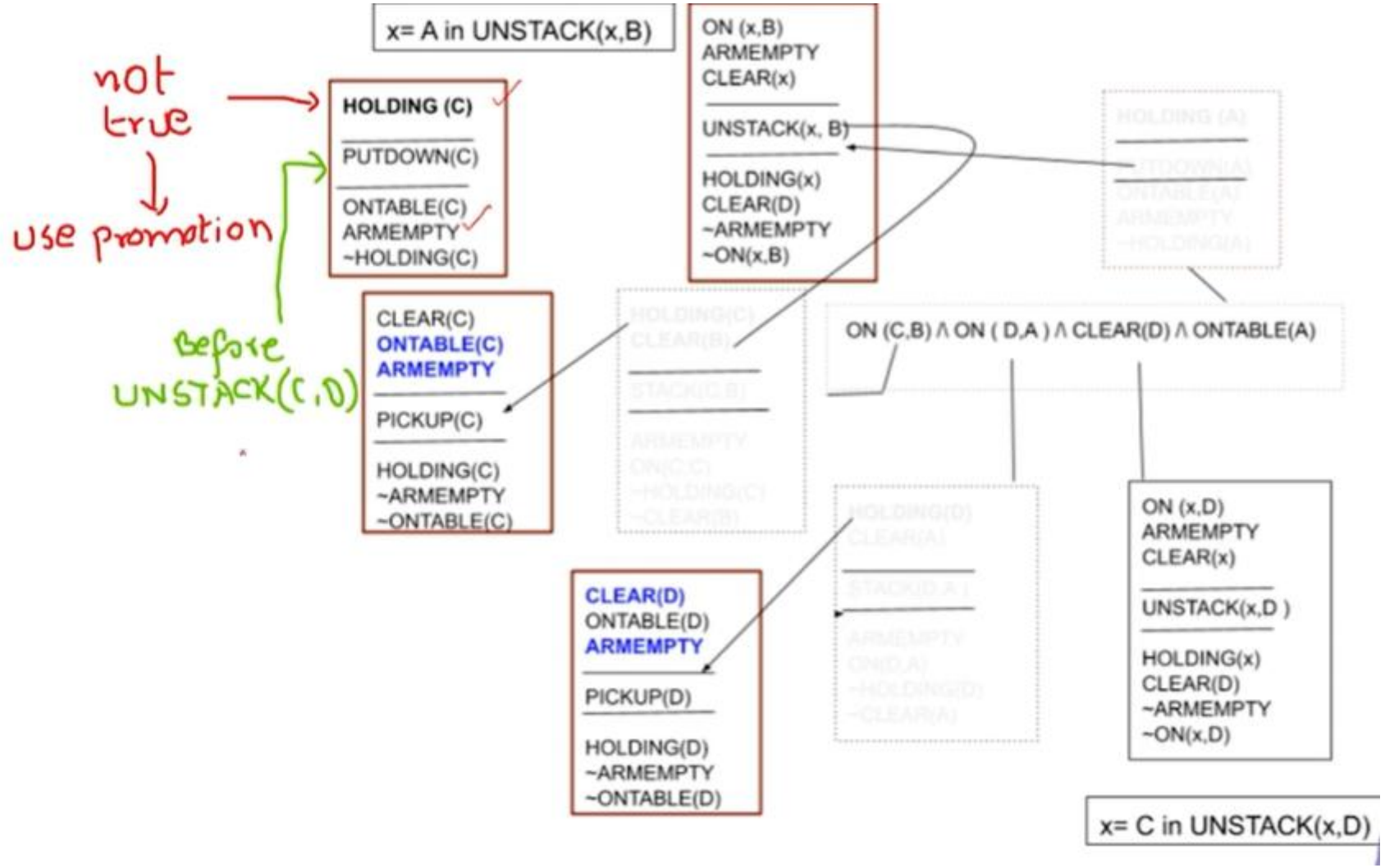
Example



https://www.youtube.com/watch?v=8_g6lq01jpk

Nonlinear Planning Using Constraint Posting

Example



https://www.youtube.com/watch?v=8_g6lq01jpk

Unit III - Planning, Learning and Robotics

Lecture 3

Recap

- **Nonlinear Planning Using Constraint Posting**

Agenda

- **Hierarchical Planning**

Hierarchical Planning

- To solve hard problems , a problem solver may have to generate long plans.
- To do that, it is important to be able to eliminate some of the details of the problem until a solution that addresses the main issue is found.
- Early attempts to do this involved the use of macro operators in which larger operators were built from smaller ones.
- But in this approach, no details were eliminated from the actual descriptions of the operators.

Hierarchical Planning

The ABSTRIPS system NLP tool was developed for automatic text summarization.

- A better approach was developed in the ABSTRIPS system, which planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction were ignored.

```
(OPERATOR
      (PRECONDITIONS
          (and(...)
              (forall (w ...)...)
              (not
                  (exists ...)
                  (or.....)))
      (POSTCONDITIONS
          (ADD (...))
          (DELETE (...))
          (if (and (...)(...))
              (ADD (...)(...))
              (DELETE (...)(...)))))
```

Fig. 13.10 *A Complex Operator*

Hierarchical Planning

The ABSTRIPS approach to problem-solving is as follows:

- First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
- These values reflect the expected difficulty of satisfying the precondition.
- To do this, do exactly what STRIPS did, but simply ignore preconditions of lower-than-peak criticality.
- Once this is done, use the constructed plans as the outline of a complete plan and consider preconditions at the next lowest criticality level.

Hierarchical Planning

- Augment the plan with operators that satisfy those preconditions.
- Again, in choosing operators, ignore all preconditions whose criticality is less than the level now being considered.
- Continue this process of considering less and less critical preconditions until all of the preconditions of the original rules have been considered.
- Because this process explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has been called length-first search.

Hierarchical Planning

Example

trying to solve a problem involving a robot moving around in a house and we are considering the operator PUSH-THROUGH- DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is (in the normal situation) nothing we can do about it if it is not true. But the precondition that the door be open is of lower criticality if we have the operator OPEN-DOOR. In order for a hierarchical planning system to work with STRIPS-like rules, it must be told, in addition to the rules themselves, the appropriate criticality value for each term that may occur in a precondition. Given these values, the basic process can function in very much the same way that nonhierarchical planning does. But effort will not be wasted filling in the details of plans that do not even come close to solving the problem.

Other Planning Techniques

- Triangle Tables [Fikes *et al.*, 1972; Nilsson, 1980]—Provide a way of recording the goals that each operator is expected to satisfy as well as the goals that must be true for it to execute correctly. If something unexpected happens during the execution of a plan, the table provides the information required to patch the plan.
- Metaplanning [Stefik, 1981a]—A technique for reasoning not just about the problem being solved but also about the planning process itself.
- Macro-operators [Fikes and Nilsson, 1971]—Allow a planner to build new operators that represent commonly used sequences of operators. See Chapter 17 for more details.
- Case-Based Planning [Hammond, 1986]—Re-uses old plans to make new ones. We return to case-based planning in Chapter 19.

Learning

- One of the criticisms of AI is that machines cannot be called intelligent until they are able to learn to do new things and adapt to new situations.
- The ability to adapt to new surroundings and to solve new problems is an important characteristic of intelligent entities.
- Rather than asking in advance whether it is possible for computers to “learn”, it is much more enlightening to try to describe exactly what activities we mean when we say ‘learning” and what mechanisms could be used to enable us to perform those activities.

Learning

Learning includes the following:

- Skill Refinement
- Knowledge Acquisition
  - Rote Learning
  - Learning by taking advice
  - Learning by Problem Solving
  - Learning from examples

Learning

Rote Learning

- When a computer stores a piece of data, it is performing a rudimentary form of learning.
- In case of data caching, we store computed values so that we do not have to recompute them later.
- When computation is more expensive than recall, this strategy can save a significant amount of time.
- Such caching is known as Rote Learning.

Learning

Rote Learning

Rote Learning Capabilities include:

- Organized Storage of Information: For stored values to be more efficient than recomputation, there must be a method to access the appropriate stored value quickly.
- Generalization: The potential number of distinct objects that could be stored may be vast. To keep the storage requirements manageable, some form of generalization is necessary.

Learning

Rote Learning

Advantages:

1. Foundation Building for Basic Knowledge
2. Improves Quick Recall
3. Enhances Concentration and Focus
4. Boosts Confidence in Learners
5. Efficient for Standardized Testing
6. Supports Language Acquisition

Learning

Rote Learning

Disadvantages:

1. Lacks Depth of Understanding
2. Limited Critical Thinking Development
3. Hinders Creativity
4. Difficult to Apply Knowledge in New Contexts
5. Can Lead to Boredom and Disinterest
6. Forgets Information Quickly After Memorization
7. Discourages Problem-Solving Skills

Learning

Learning by Taking Advice

- The programmer is a sort of teacher and the computer is a sort of student.
- After being programmed, the computer will be able to do something it previously could not.
- Executing the program may not be such a simple matter.
- Suppose the program is written in a high-level language like LISP, some interpreter or compiler must intervene to change the teacher's instructions into code that the machine can execute directly.

Learning

Learning by Taking Advice

- People process advice in an analogous way.
- In chess, the advice “fight for control of the center of the board” is useless unless the player can translate the advice into concrete moves and plans.
- A computer program might make use of the advice by adjusting its static evaluation function to include a factor based on the number of center squares attacked by its own pieces.

Learning

Learning by Taking Advice

- Mostow describes a program called FOO, which accepts advice for playing hearts, a card game.
- A human user first translates the advice from English into a representation that FOO can understand.

Example:

"Avoid taking points " becomes: (avoid (take-points me) (trick))

Learning

Learning by Taking Advice

- FOO must operationalize this advice by turning it into an expression that contains concepts and actions FOO can use when playing the game of hearts.
- One strategy FOO can follow is to UNFOLD an expression by replacing some term with its definition.
- By UNFOLDing the definition of avoid, FOO comes up with:

(achieve (not (during (trick) (take-points me)))

- FOO considers the advice to apply to the player called “me”. Next FOO UNFOLDs the definition of trick.

Learning

Learning by Taking Advice

```
(achieve (not (during
                (scenario
                   (each pl (players) (play-card pl))
                   (take-trick (trick-winner)))
                (take-points me))))
```

In other words, the player should avoid taking points during the scenario consisting of (1) players playing cards and (2) one player taking the trick. FOO then uses *case analysis* to determine which steps could cause one to take points. It rules out step 1 on the basis that it knows of no intersection of the concepts take-points and play-card. But step 2 could affect taking points, so FOO UNFOLDs the definition of take-points:

Unit III - Planning, Learning and Robotics

Lecture 4

Recap

- **Hierarchical Planning**
- **Learning**
  - **Rote Learning**
  - **Learning by taking Advice**

Agenda

- **Learning**
  - **Learning in Problem Solving**
    - **Learning by Parameter Adjustments**
    - **Learning with Macro Operators**
    - **Learning by Chunking**

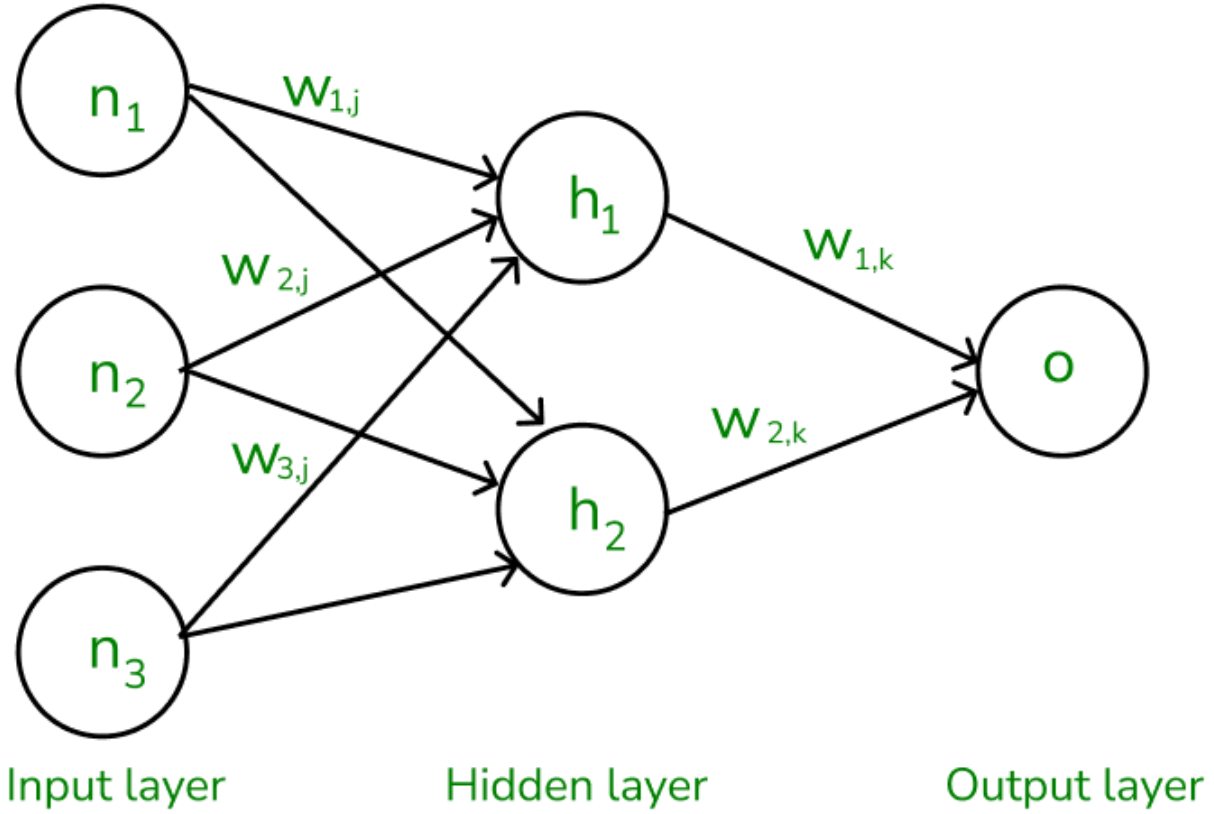
Learning

Learning in Problem Solving Learning by Parameter Adjustment

- AI programs often adjust weights (coefficients) attached to different features to optimize performance based on feedback.
- In game-playing programs, these weights help evaluate moves by combining factors into a single desirability score, while in classification programs, they help determine the category a stimulus belongs to.
- Adjusting weights involves increasing coefficients for features that predict successful outcomes and decreasing or eliminating those for poor predictors.

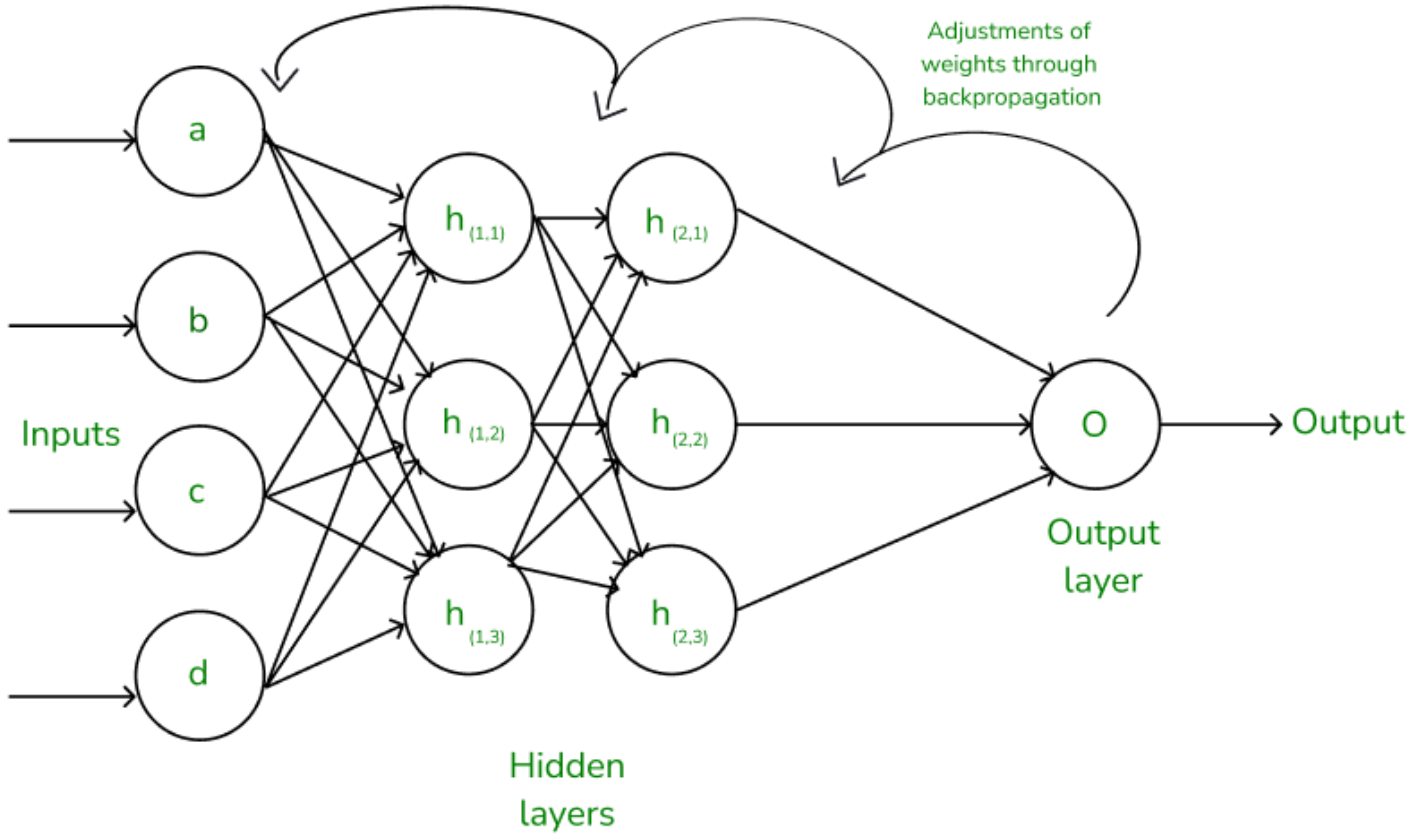
Learning

Learning in Problem Solving Learning by Parameter Adjustment



Learning

Learning in Problem Solving Learning by Parameter Adjustment



Learning

Learning in Problem Solving Learning by Parameter Adjustment

- Samuel's checkers program is an example, using a polynomial evaluation function to determine move quality by adjusting weights for sixteen features as the game progresses.
- A significant challenge, particularly in games, is determining weight adjustments since feedback is often only available after the game ends, rather than after each individual move. This is referred to as the "credit assignment problem."

Learning

Learning in Problem Solving Learning by Parameter Adjustment

- In summary, parameter adjustment allows programs to improve by optimizing feature weights based on experience, enhancing decision-making accuracy in games and classification tasks.

Learning

Learning in Problem Solving Learning with Macro operators

- Sequences of actions that can be treated as a whole are called macro-operators.
- Macro operators were used in the early problem-solving system STRIPS.

```c
#include <stdio.h>

#define PI 3.14159  // Defining a constant macro
#define AREA_OF_CIRCLE(r) (PI * (r) * (r))  // Defining a macro function
```

Learning

Learning in Problem Solving Learning with Macro operators

Suppose we are given an initial blocks world situation in which ON(C, B) and ON(A, Table) are both true.

STRIPS can achieve the goal ON(A, B) by devising a plan with the four steps

1. UNSTACK(C, B),
2. PUTDOWN(C),
3. PICKUP(A),
4. STACK(A, B).

STRIPS now builds a MACROP with preconditions ON(C, B), ON(A, Table) and postconditions ON(C, Table), ON(A, B). The body of the MACROP consists of the four steps above mentioned.

Learning

Learning in Problem Solving Learning with Macro operators

Generalizing MACROOPs we get

1. UNSTACK(x1,x2)
2. PUTDOWN(x1),
3. PICKUP(x1),
4. STACK(x1,x2).

Learning

Learning in Problem Solving Learning by Chunking

- Chunking is a cognitive process used by the SOAR system to learn and solve problems.
- It involves breaking down complex problems into smaller, more manageable chunks.
- These chunks are stored in long-term memory and can be reused to solve similar problems in the future.

Learning

Learning in Problem Solving Learning by Chunking

- Chunking allows SOAR to learn from experience and improve its performance over time.
- However, chunking can be computationally expensive, especially when dealing with large, complex problems.

Learning

Learning from Examples Induction

- Induction refers to a reasoning or process of deriving general principles from specific observations.
- It is widely used in various fields, such as mathematics, science, and logic, and carries slightly different meanings depending on the context.

Learning

Learning from Examples Induction

- The task of constructing class definitions is called Concept Learning, Or Induction.

Learning

Learning from Examples Induction

- Classification is the process of assigning, to a particular input, the name of a class to which it belongs.
- The classes from which the classification procedure can choose can be described in a variety of ways.
- Their definition will depend on the use to which they are put.
- Classification is an important component of many problem solving tasks.

Learning

Learning from Examples Induction

- Before classification can be done, the classes it will use must be defined:
  - Isolate a set of features that are relevant to the task domain. Define each class by a weighted sum of the values of these features. For example, a task such as weather prediction involves parameters that can be measured, including rainfall and the location of cold fronts.
  - Isolate a set of features that are relevant to the task domain. Define each class as a structure composed of these features. Ex: classifying animals, various features can be such things as color, length of neck, etc

Unit III - Planning, Learning and Robotics

Lecture 5

Recap

- **Learning**
  - **Learning in Problem Solving**
    - **Learning by Parameter Adjustments**
    - **Learning with Macro Operators**
    - **Learning by Chunking**

Agenda

- Learning
  - Learning from Examples
    - Induction
      - Winston's learning Problem
      - Version Spaces
      - Decision Trees
  - Explanation Based learning
  - Discovery
  - Analogy

Learning

Learning from Examples Induction Winston's Learning Program

- An early structural concept learning program.
- This program operates in a simple blocks world domain.
- Its goal was to construct representations of the definitions of concepts in blocks domain.
- For example, it learned the concepts House, Tent and Arch.
- A near miss is an object that is not an instance of the concept in question but that is very similar to such instances.

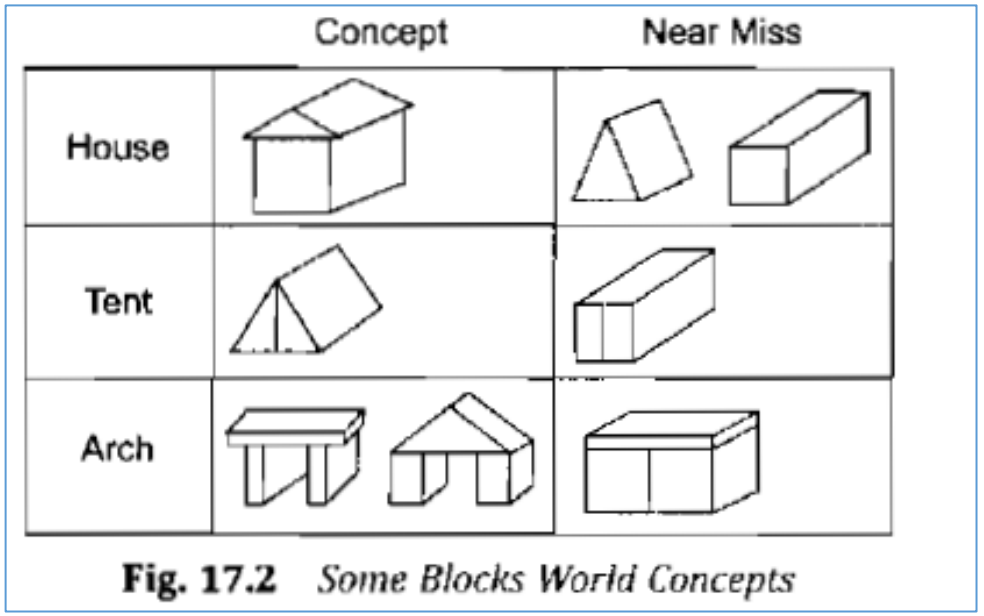


Fig. 17.2 *Some Blocks World Concepts*

Learning

Learning from Examples Induction Winston's Learning Program

Basic approach of Winston's Program

1. Begin with a structural description of one known instance of the concept. Call that description the concept defintion.
2. Examine descriptions of other known instances of the concepts. Generalize th definition to include them.
3. Examine the descriptions of near misses of the concept. Restrict the definition to exclude these.

Learning

Learning from Examples Induction Version Spaces

- The goal of version spaces is to produce a description that is consistent with all positive examples but no negative examples in the training set.
- This is another approach to concept learning.
- Version spaces work by maintaining a set of possible descriptions and evolving that set as new examples and near misses are presented.
- The version space is simply a set of descriptions, so an initial idea is to keep an explicit list of those descriptions.
- Version space consists of two subsets of the concept space.
- One subset called G contains most general descriptions consistent with the training examples . The other subset contains the most specific descriptions consistent with the training examples.
- The algorithm for narrowing the version space is called the Candidate elimination algorithm.

Learning

Learning from Examples Induction Version Spaces

Algorithm: Candidate Elimination

- Given: A representation language and a set of positive and negative examples expressed in that language.
- Compute : A concept description that is consistent with all the positive examples and none of the negative examples.

1. Initialize G to contain one element
2. Initialize S to contain one element: the first positive element.
3. Accept new training example. If it is a positive example, first remove from G any descriptions that do not cover the example. Then update the set S to contain most specific set of descriptions in the version space that cover the example and the current elements of the S set. Inverse actions for negative example
4. If S and G are both singleton sets, then if they are identical, output their values and halt.

Learning

Learning from Examples Induction Version Spaces

Key Concepts

1. **General Boundary (G):** Represents the most general hypotheses consistent with the data.
2. **Specific Boundary (S):** Represents the most specific hypotheses consistent with the data.
3. **Version Space (VS):** The space between these boundaries, i.e., all hypotheses that lie between the general and specific boundaries.

Learning

Learning from Examples Induction Version Spaces

Example: Fruit Classification

Suppose we are classifying fruits based on their **color** and **shape**. The attributes are:

- **Color:** {Red, Yellow, Green}
- **Shape:** {Round, Long}

Training Data:

1. **Example 1:** A round, red fruit is a **positive example** (e.g., apple).

Learning

Learning from Examples Induction Decision Trees

- This is a third approach to concept learning.
- To classify a particular input, we start at the top of the tree and answer questions until we reach a leaf, where the specification is stored.
- ID3 is a program example for Decision Trees.
- ID3 uses iterative method to build up decision trees, preferring simple trees over complex ones, on the theory that simple trees are more accurate classifiers of future inputs.
- It begins by choosing a random subset of the training examples.
- This subset is called the window.
- The algorithm builds a decision tree that correctly classifies all examples in the win do.

Learning

Explanation Based Learning

- Learning complex concepts using Induction procedures typically requires a substantial number of training instances.
- But people seem to be able to learn quite a bit from single examples.
- We don’t need to see dozens of positive and negative examples of fork(chess) positions in order to learn to avoid this trap in the future and perhaps use it to our advantage.
- What makes such single-example learning possible? The answer is knowledge.

Learning

Explanation Based Learning

- Much of the recent work in machine learning has moved away from the empirical, data intensive approach described in the last section toward this more analytical knowledge intensive approach.
- A number of independent studies led to the characterization of this approach as explanation-base learning(EBL).

Learning

Explanation Based Learning

- An EBL system attempts to learn from a single example x by explaining why x is an example of the target concept.
- The explanation is then generalized, and then system's performance is improved through the availability of this knowledge.

Learning

Explanation Based Learning

We can think of EBL programs as accepting the following as input:

- ✓ A training example
- ✓ A goal concept: A high-level description of what the program is supposed to learn
- ✓ An operational criterion- A description of which concepts are usable.
- ✓ A domain theory: A set of rules that describe relationships between objects and actions in a domain

Learning

Explanation Based Learning

- From this EBL computes a generalization of the training example that is sufficient to describe the goal concept, and also satisfies the operationality criterion.

Learning

Discovery

Discovery is a restricted form of learning in which one entity acquires knowledge without the help of a teacher.

1. Theory-Driven Discovery
2. Data-Driven Discovery
3. Clustering

Learning

Discovery AM : Theory-Driven Discovery

- Discovery is certainly learning. More clearly than other kinds of learning, problem solving.
- Suppose that we want to build a program to discover things in maths, such a program would have to rely heavily on the problem-solving techniques.
- AM is written by Lenat and it worked from a few basic concepts of set theory to discover a good deal of standard number theory.
- AM exploited a variety of general-purpose AI techniques. It used a frame system to represent mathematical concepts. One of the major activities of AM is to create new concepts and fill in their slots.

Learning

Discovery AM : Theory-Driven Discovery

- AM uses Heuristic search, guided by a set of 250 heuristic rules representing hints about activities that are likely to lead to “interesting” discoveries.
- In one run AM discovered the concept of prime numbers. How did it do it?
- Having stumbled onto the natural numbers, AM explored operations such as addition, multiplication and their inverses. It created the concept of divisibility and noticed that some numbers had very few divisors

Learning

Discovery Bacon: Data-Driven Discovery

- AM showed how discovery might occur in a theoretical setting.
- Scientific discovery has inspired several computer models.
- Langley et al presented a model of data-driven scientific discovery that has been implemented as a program called BACON (named after Sir Francis Bacon, a philosopher of science)

Learning

Discovery Bacon: Data-Driven Discovery

- BACON begins with a set of variables for a problem.
- For example in the study of the behavior of gases, some variables are p, the pressure on the gas, V, the volume of the gas, n, the amount of gas in moles, and T the temperature of the gas.
- Physicists have long known a law, called ideal gas law, that relates these variables.
- BACON is able to derive this law on its own.

Learning

Discovery Bacon: Data-Driven Discovery

- First, BACON holds the variables n and T constant, performing experiments at different pressures p1, p2, and p3.
- BACON notices that as the pressure increases, the volume V decreases.
- For all values, n,p, V and T, pV/nT = 8.32 which is ideal gas law as shown by BACON.
- BACON has been used to discover a wide variety of scientific laws such as Kepler's thirdlaw, Ohm's law, the conservation of momentum, and Joule's law.
- BACON's discovery procedure is state space search.
- A better understanding of the science of scientific discovery may lead one day to programs that display true creativity. Much more work must be done in areas of science that BACON does not model.

Learning

Discovery Clustering

- Clustering is very similar to induction.
- In Inductive learning a program learns to classify objects based on the labeling provided by a teacher.
- In clustering, no class labeling is provided.
- The program must discover for itself the natural classes that exist for the objects, in addition to a method for classifying instances.

Learning

Discovery Clustering

- AUTOCLASS is one program that accepts several training cases and hypothesizes a set of classes.
- For any given case, the program provides a set of probabilities that predict into which classes the case is likely to fall.
- In one application, AUTOCLASS found meaningful new classes of stars from their infrared spectral data.

Unit III - Planning, Learning and Robotics

Lecture 6

Recap

- Learning
  - Learning from Examples
    - Induction
      - Winston's learning Problem
      - Version Spaces
      - Decision Trees
  - Explanation Based learning
  - Discovery

Agenda

- **Analogy**
- **Formal Learning Theory**

Learning

Analogy

(A comparison between two things that shows a way in which they are similar)

- Analogy is a fundamental tool for human reasoning and language.
- Underlying analogies are complex mappings between seemingly different concepts.
- To grasp an analogy, we need to identify key properties and recognize relevant mappings.
- The space of possible analogies is vast, making it difficult to identify the most relevant ones.

Learning

Analogy

- Analogy plays a crucial role in learning and problem-solving, as humans often solve new problems by drawing on past experiences.
- AI systems that can understand and use analogies would be more intelligent and adaptable.
- **Two Types of Analogical Problem-Solving:**
  - Transformational Analogy and
  - Derivational Analogy

Learning

Analogy Transformational Analogy

- It involves adapting a solution to a previously solved problem to solve a new, similar problem.
- The process includes identifying the core concepts and relationships in the old solution and mapping them onto the new problem.
- This mapping often involves substitutions, where elements from the old solution are replaced with corresponding elements from the new problem.
- The goal is to reuse the underlying structure and reasoning of the old solution to solve the new problem efficiently.

Learning

Analogy Transformational Analogy

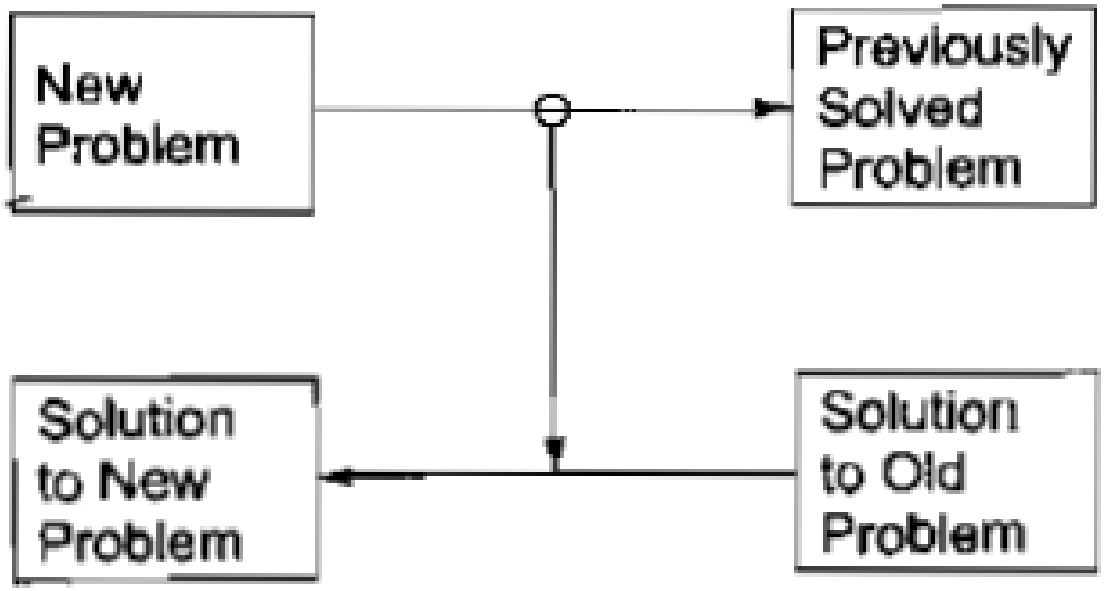


Fig. 17.19 *Transformational Analogy*

Example:

A + B = C,
and you are given
C - B = ?,

Learning

Analogy Derivational Analogy

- It considers not only the final solution of a previously solved problem but also the process of how that solution was reached.
- It takes into account the detailed history of a problem-solving episode, called its derivation.
- This allows for a more nuanced and flexible approach to analogy, as it can reuse not just the final solution but also the steps and reasoning involved in reaching it.

Learning

Analogy Derivational Analogy

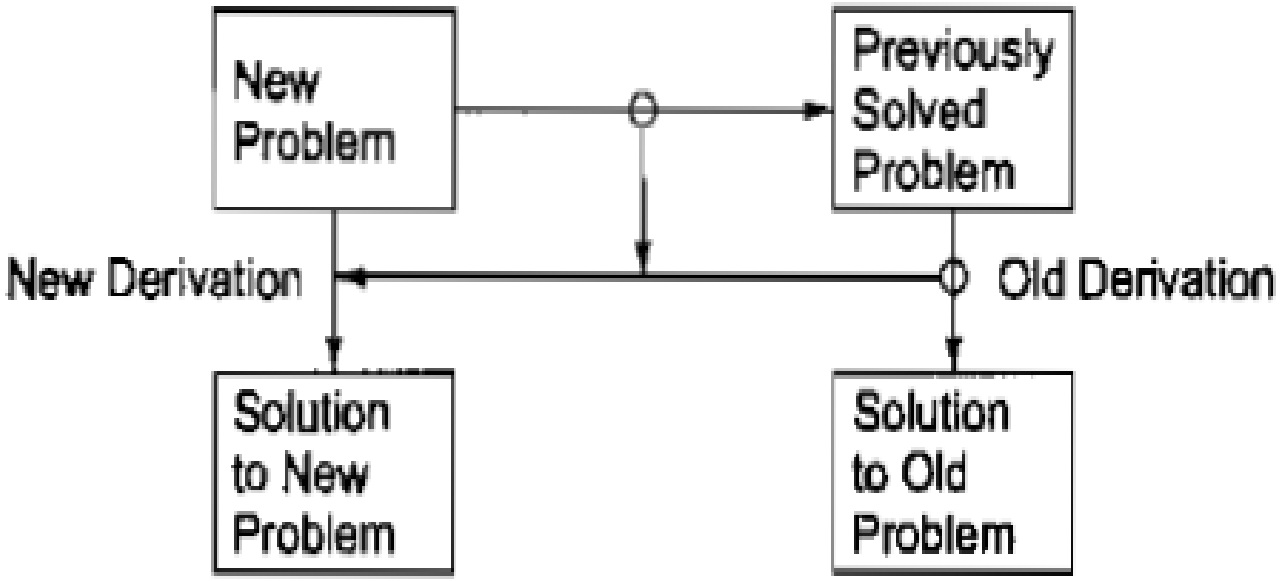


Fig. 17.21 *Derivational Analogy*

Example:

If you solved a math problem step-by-step, and a similar problem appears, you recall **how** you solved it before (the logic, not just the answer) and use the same reasoning process.

Formal Learning Theory

- Inductive learning has attracted significant attention from mathematicians and computer scientists.
- Valiant's "theory of the learnable" classifies problems based on their learnability.
- A device learns a concept by producing an algorithm to classify future examples.
- The complexity of learning depends on error tolerance, number of features, and rule size.
- Conjunctive feature descriptions are learnable with a small number of training examples.
- Learning from positive examples only is possible with error tolerance.

Neural Net Learning and Genetic Learning

Early Machine Learning: The initial attempts in machine learning aimed to imitate animal learning at a neural level.

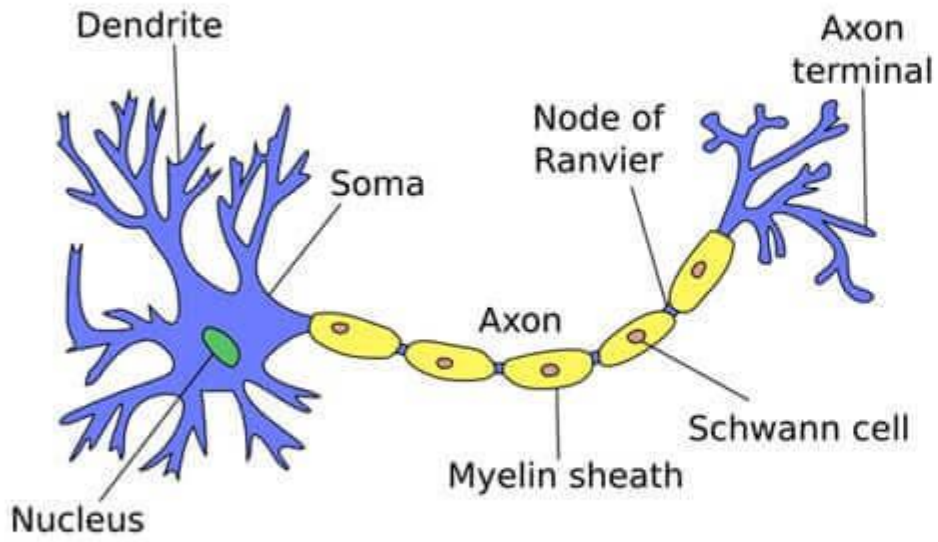
Symbolic Manipulation vs. Neural Networks: These early efforts differed from the symbolic manipulation methods.

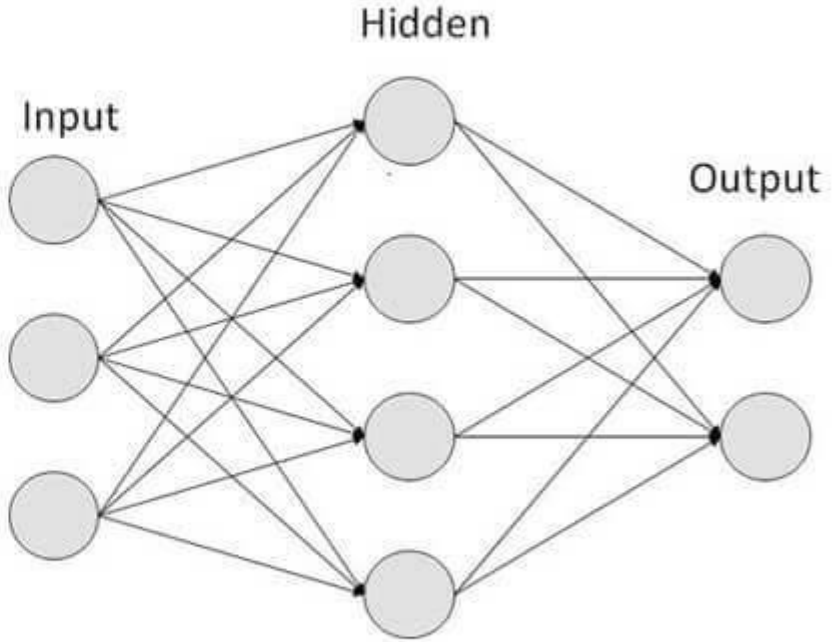
Neuron-Based Learning: Collections of idealized neurons were presented with stimuli and trained through reward and punishment.

Neural Network

- A **neural network** is a computational model inspired by the structure and function of the human brain.
- It is a fundamental building block of **deep learning** and is designed to recognize patterns, make predictions, or solve complex tasks by mimicking the way neurons in the brain process information.

Neural Network





Neural Network

Key Components of a Neural Network:

- Neurons (Nodes)
- Layers
- Weights and Biases
- Activation Function
- Connections

Robotics

- Robotics is a multidisciplinary field that involves the design, construction, operation, and use of robots.
- It integrates principles from mechanical engineering, electrical engineering, computer science, and artificial intelligence to create machines capable of performing tasks autonomously or semi-autonomously.
- Robotics plays a crucial role in various industries, including manufacturing, healthcare, agriculture, and space exploration.

Robot Hardware

Robot hardware encompasses the physical components that enable a robot to interact with its environment. Key hardware elements include:

Actuators: Motors or hydraulic systems that control movement (e.g., joints, wheels, arms).

Sensors: Devices that gather data from the robot's surroundings (e.g., cameras, lidar, accelerometers, and temperature sensors).

Power Supply: Batteries or other energy sources that provide the necessary power for robot functions.

Controller: The computational unit that processes sensor data and commands actuators

Robotic Perception

Robotic perception refers to the ability of a robot to interpret sensory data from its environment. This involves:

Computer Vision: Interpreting visual data from cameras and other imaging systems.

Sensor Fusion: Combining data from multiple sensors (e.g., lidar, radar, and cameras) to create a unified representation of the environment.

Simultaneous Localization and Mapping (SLAM): Techniques that allow robots to map their environment and localize themselves within it in real-time.

Robotic Software Archiectures

Robotic software architecture is the structure that defines how software components communicate and function together in a robot. Common architectures include:

Reactive Architectures: Simple control schemes based on sensory inputs that react to the environment without high-level planning (e.g., subsumption architecture).

Deliberative Architectures: Systems that involve higher-level reasoning and planning, often using AI techniques like decision trees or reinforcement learning.

Hybrid Architectures: Combine reactive and deliberative systems to balance the responsiveness of the robot with its ability to plan ahead.

Robotic Application Domains

Robotics has diverse applications across many fields:

Manufacturing: Automation of repetitive tasks, assembly lines, and quality control.

Healthcare: Surgical robots, rehabilitation devices, and robotic prosthetics.

Agriculture: Precision farming, crop monitoring, and harvesting robots.

Exploration: Robots used in space exploration, underwater exploration, and hazardous environments.

Service and Assistance: Robots designed for customer service, elderly care, and personal assistance.

Unit Summary

Unit 3 – 13 Hrs

Planning, Learning and Robotics

Planning: The Blocks World, Components of a Planning System, Goal Stack Planning, Nonlinear Planning Using Constraint Posting, Hierarchical Planning, Other Planning Techniques.

Learning: Rote Learning, learning by Taking Advice, Learning in Problem-solving, Learning from Examples: Induction, Explanation-based Learning, Discovery, Analogy, Formal Learning Theory. Learning in Neural and Belief Networks' How the Brain Works, Neural Networks, perceptions.

Robotics: Introduction, Robot Hardware, Robotic Perception, Robotic Software Architectures, Application Domains.

Thank You