



Dr. Vishwanath Karad

**MIT WORLD PEACE  
UNIVERSITY** | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

# CS 332 Artificial Intelligence

---

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

# CS 332 Artificial Intelligence

## Teaching Scheme

**Theory:** 4 Hrs / Week

**Credits:** 2 + 1 = 3

**Practical:** 2 Hrs / Week

## Course Objectives:

- 1) To understand the concept of Artificial Intelligence (AI)
- 2) To learn various peculiar search strategies for AI
- 3) To develop a mind to solve real world problems unconventionally with optimality

## Course Outcomes:

- 1) Identify and apply suitable Intelligent agents for various AI applications
- 2) Design smart system using different informed search / uninformed search or heuristic approaches.
- 3) Identify knowledge associated and represent it by ontological engineering to plan a strategy to solve given problem.



# Syllabus

---

## **Knowledge Representation and Planning**

Propositional logic and predicate logic, Knowledge Representation structure such as frame, Conceptual dependencies, Semantic networks and script, Resolution in predicate logic, Unification algorithm, Forward and Backward chaining, Logic Programming

**Planning:** Forward and Backward planning, Goal Stack Planning, Hierarchical Planning.

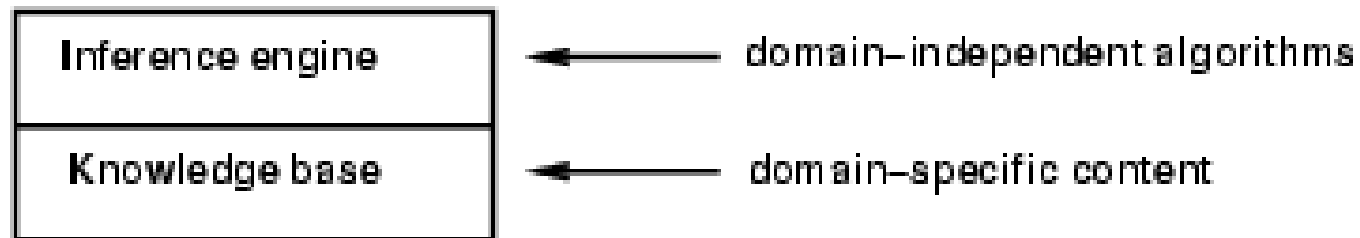
# Knowledge Representation and Planning

# Contents

---

- Propositional logic and predicate logic
- Knowledge Representation structure such as frame, Conceptual dependencies, Semantic networks and script
- Resolution in predicate logic
- Unification algorithm
- Forward and Backward chaining
- Logic Programming
- Forward and Backward planning
- Goal Stack Planning and Hierarchical Planning

# Knowledge bases



## Knowledge Representation

- **Facts:** Things we want to represent. Truth in some relevant world.
- **Representation of facts.**

# A knowledge-based agent

---

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a **sentence**.
- The sentences are expressed in a **knowledge representation language**.

The agent operates as follows:

1. It TELLS the knowledge base what it perceives.
2. It ASKS the knowledge base what action it should perform.
3. It performs the chosen action.

# Logic

- Drawing reasonable conclusions from a set of data (observations, beliefs, etc) seems key to intelligence
- Logic is a powerful and well developed approach to this and highly regarded by people
- Logic is also a strong formal system that we can program computers to use
- Maybe we can reduce any AI problem to figuring out how to represent it in logic and apply standard proof techniques to generate solutions.

Intelligent agents should have capacity for:

- **Perceiving**, that is, acquiring information from environment,
- **Knowledge Representation**, that is, representing its understanding of the world,
- **Reasoning**, that is, inferring the implications of what it knows and of the choices it has, and
- **Acting**, that is, choosing what it want to do and carry it out



# Continued...

---

Knowledge can also be represented by the symbols of logic, which is the study of the rules of exact reasoning.

- ☐ Logic is also of primary importance in expert systems in which the inference engine reasons from facts to conclusions.
- ☐ A descriptive term for logic programming and expert systems is automated reasoning systems.

# Logic in general

---

**Logics** are formal languages for representing information such that conclusions can be drawn

**Syntax** defines the sentences in the language

**Semantics** define the "meaning" of sentences

- i.e., define **truth** of a sentence in a world

E.g., the language of arithmetic

- $x+2 \geq y$  is a sentence;  $x^2+y > \{ \}$  is not a sentence
- $x+2 \geq y$  is true iff the number  $x+2$  is no less than the number  $y$
- $x+2 \geq y$  is true in a world where  $x = 7, y = 1$
- $x+2 \geq y$  is false in a world where  $x = 0, y = 6$

# Representation, reasoning, and logic

---

- The object of knowledge representation is to express knowledge in a computer-tractable form, so that agents can perform well.
- A knowledge representation language is defined by its syntax, which defines all possible sequences of symbols that constitute sentences of the language.
- Ex: Sentences in a book, bit patterns in computer memory its semantics, which determines the facts in the world to which the sentences refer.
- Each sentence makes a claim about the world.
- An agent is said to believe a sentence about the world.

# Types of Logic

---

There are a number of logical systems with different syntax and semantics.

1. Propositional logic
2. Predicate logic

# Propositional Logic

---

- Propositional logic is a symbolic logic for manipulating propositions
- Propositional logic is concerned with declarative sentences that can be classified as either true or false
- A sentence whose truth value can be determined is called a statement or proposition
- A statement is also called a closed sentence because its truth value is not open to question
- Statements that cannot be answered absolutely are called open sentences
- A compound statement is formed by using logical connectives on individual statements

# Propositional logic

---

In propositional logic (PL) an user defines a set of propositional symbols, like  $P$  and  $Q$ . User defines the semantics of each of these symbols. For example,

- $P$  means "It is hot"
- $Q$  means "It is humid"
- $R$  means "It is raining"

Examples of PL sentences:

- $(P \wedge Q) \Rightarrow R$  (here meaning "If it is hot and humid, then it is raining")
- $Q \Rightarrow P$  (here meaning "If it is humid, then it is hot")
- $Q$  (here meaning "It is humid.")

# Truth Table

p	q	$p \rightarrow q$		$\sim p$	$\sim p \vee q$
T	T	T		F	T
T	F	F		F	F
F	T	T		T	T
F	F	T		T	T

# Inference rules in Propositional logic

---

1. Idempotent rule:

$$P \wedge P \implies P$$

$$P \vee P \implies P$$

2. Commutative rule:

$$P \wedge Q \implies Q \wedge P$$

$$P \vee Q \implies Q \vee P$$

3. Associative rule:

$$P \wedge (Q \wedge R) \implies (P \wedge Q) \wedge R$$

$$P \vee (Q \vee R) \implies (P \vee Q) \vee R$$



# Continued...

---

4. Distributive Rule:

$$\mathbf{P \vee (Q \wedge R) ==> (P \vee Q) \wedge (P \vee R)}$$

$$\mathbf{P \wedge (Q \vee R) ==> (P \wedge Q) \vee (P \wedge R)}$$

5. De-Morgan's Rule:

$$\mathbf{)\neg P \vee Q) ==> \neg P \wedge \neg Q}$$

$$\mathbf{)\neg P \wedge Q) ==> \neg P \vee \neg Q}$$

6. Implication elimination:

$$\mathbf{P \rightarrow Q ==> \neg P \vee Q}$$

# Continued...

---

7. Bidirectional Implication elimination:

$$(P \Leftrightarrow Q) \Rightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$$

8. Contrapositive rule:

$$P \rightarrow Q \Rightarrow \neg P \rightarrow \neg Q$$

9. Double Negation rule:

$$\neg(\neg P) \Rightarrow P$$

10. Absorption Rule:

$$P \vee (P \wedge Q) \Rightarrow P$$

$$P \wedge (P \vee Q) \Rightarrow P$$

# Continued...

---

Example:

“I will get wet if it rains and I go out of the house”

Let Propositions be:

W : “I will get wet “

R : “it rains “

S : “I go out of the house”

$(S \wedge R) \rightarrow W$

# Pros and cons of propositional logic

---

- Propositional logic is declarative
- Propositional logic allows partial/disjunctive/negated information
- Propositional logic is compositional
- Propositional logic has very limited expressive power

# Predicate Logic

- Representing simple facts (Preposition)

“SOCRATES IS A MAN”

SOCRATESMAN -----1

“PLATO IS A MAN”

PLATOMAN -----2

- Fails to capture relationship between Socrates and man. We do not get any information about the objects involved

Ex: if asked a question : “who is a man?” we cannot get

answer. Using Predicate Logic however we can represent above

facts as: Man(Socretes) and Man(Plato)

1. Marcus was a man. man(Marcus)

# Continued...

## Representation and Mapping

- Spot is a dog

$\text{dog}(\text{Spot})$

- Every dog has a tail

$\forall x: \text{dog}(x) \rightarrow \text{hastail}(x)$



$\text{hastail}(\text{Spot})$

Spot has a tail

# Quantifiers

---

## 1. Universal quantifier ( $\forall$ )

- $\forall x$ : means “for all”  $x$
- It is used to represent phrase “for all”.
- It says that something is true for all possible values of a variable.
- Ex. “John loves everyone”

$\forall x$ : loves(John ,  $x$ )

# Continued...

---

## 2. Existential quantifier ( $\exists$ ):

- Used to represent the fact “ there exists some”
- Ex:
- “some people like reading and hence they gain good knowledge”  
 $\exists x: \{ [\text{person}(x) \wedge \text{like}(x, \text{reading})] \rightarrow \text{gain}(x, \text{knowledge}) \}$
- “lord Haggins has a crown on his head”
- $\exists x: \text{crown}(x) \wedge \text{onhead}(x, \text{Haggins})$



# Continued...

## 3. Nested Quantifiers

- We can use both  $\forall$  and  $\exists$  separately
- Ex: “everybody loves somebody”

$\forall x: \exists y: \text{loves} (x, y)$

- Connection between  $\forall$  and  $\exists$
- “everyone dislikes garlic”

$\forall x: \neg \text{like} (x, \text{garlic})$

➤ This can be also said as:

“there does not exist someone who likes garlic”

$\neg \exists x: \text{like} (x, \text{garlic})$

# Continued...

---

All Romans were either loyal to Caesar or hated him.

$\forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})$

4. Every one is loyal to someone.

$\forall x: \exists y: \text{loyalto}(x, y) \quad \exists y: \forall x: \text{loyalto}(x, y)$

5. People only try to assassinate rulers they are not loyal to.

$\forall x: \forall y: \text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y)$

$\rightarrow \neg \text{loyalto}(x, y)$

# Inference Methods

---

- Unification (prerequisite)
- Forward Chaining
- Backward Chaining
- Logic Programming (Prolog)
- Resolution
  - Transform to CNF (Chomsky normal form )
  - Generalization of Prop. Logic resolution

# Forward Chaining

---

- Forward Chaining
  - Start with atomic sentences in the KB and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made.
  - $P$  implies  $Q$  and  $P$  is asserted to be true, so therefore  $Q$  must be true
- Given a new fact, generate all consequences
- Assumes all rules are of the form
- Each rule & binding generates a new fact
- This new fact will “trigger” other rules
- Keep going until the desired fact is generated

# Example Knowledge Base

---

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy America, has some missiles, and all of its missiles were sold to it by Col. West, who is an American.

Prove that Col. West is a criminal.

## Example Knowledge Base

- It is a crime for an American to sell weapons to hostile nations  
 $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- Nono...has some missiles  
 $\exists x Owns(Nono, x) \wedge Missiles(x)$   
 $Owns(Nono, M_1) \text{ and } Missile(M_1)$
- All of its missiles were sold to it by Col. West  
 $\forall x Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- Missiles are weapons  
 $Missile(x) \Rightarrow Weapon(x)$

# Example Knowledge Base

---

An enemy of America counts as “hostile”

$Enemy(x, America) \Rightarrow Hostile(x)$

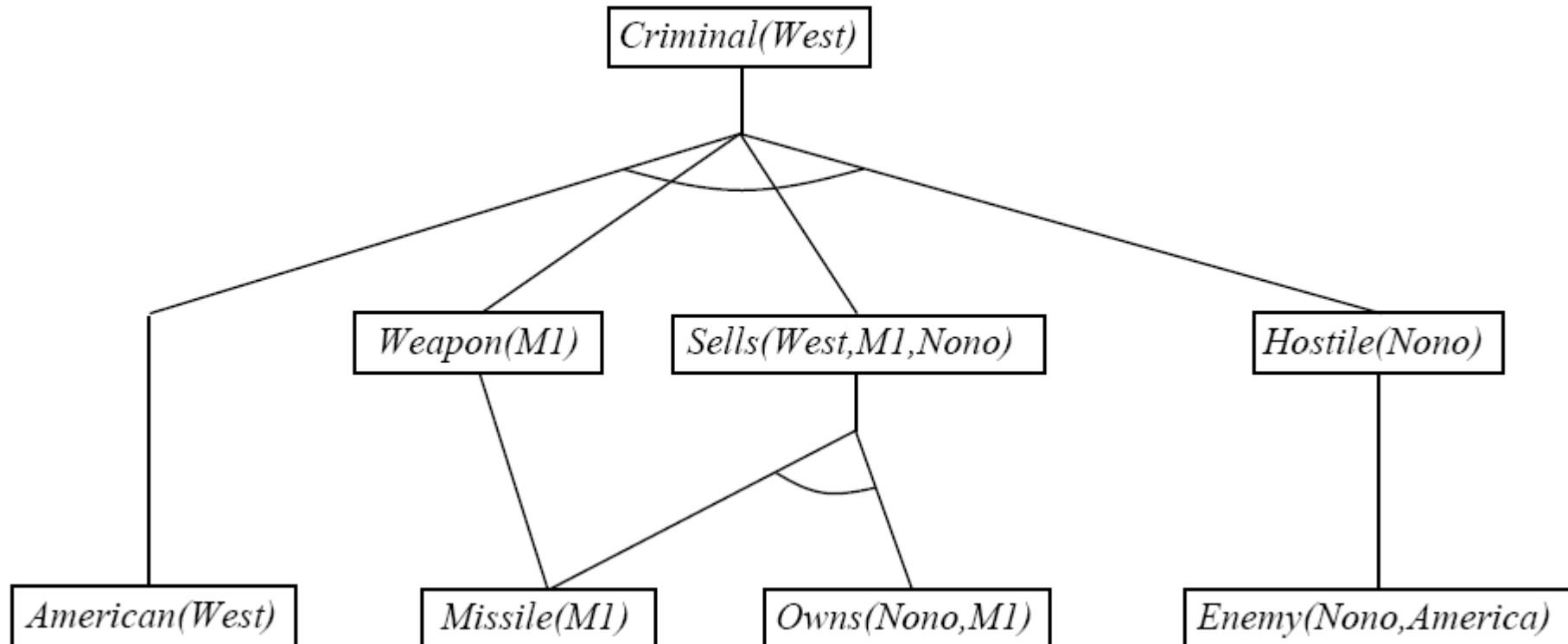
Col. West who is an American

$American(Col. West)$

The country Nono, an enemy of America

$Enemy(Nono, America)$

# Example Knowledge Base



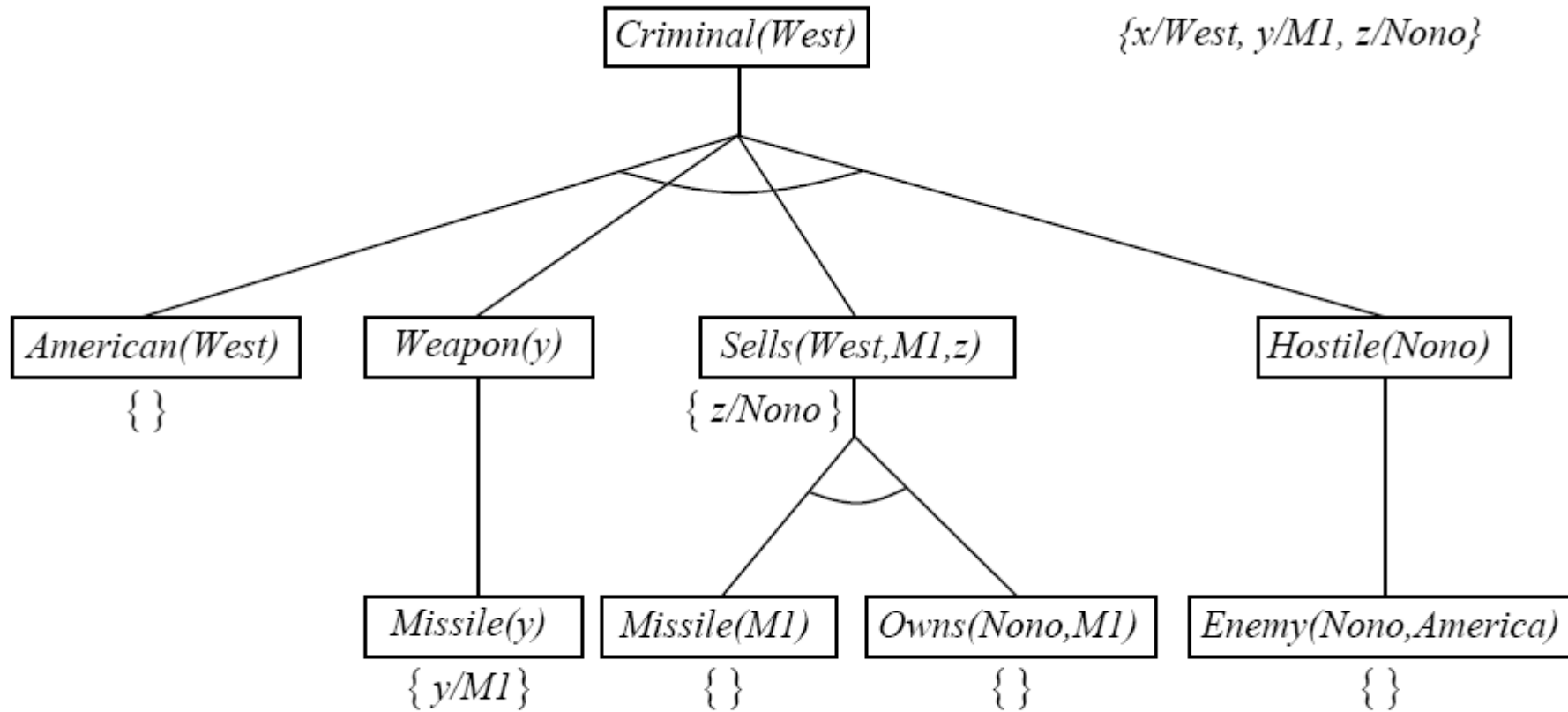


# Backward Chaining

---

- Consider the item to be proven a goal
- Find a rule whose head is the goal (and bindings)
- Apply bindings to the body, and prove these (subgoals) in turn
- If you prove all the subgoals, increasing the binding set as you go, you will prove the item.
- Logic Programming (cprolog, on CS)

# Backward Chaining Example



# Unification

- It's a matching procedure that compares two literals and discovers whether there exists a set of substitutions that can make them identical.

- E.g. 1

Hate( marcus , **X**) Hate (marcus , **caesar**)

caesar/ X

e.g. 2.

Hate(X,Y) Hate( john, Z) could be unified as:

John/X and y/z

---

$\text{UNIFY}(p, q) = \text{unifier } \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$

$\forall x: \text{knows}(\text{John}, x) \rightarrow \text{hates}(\text{John}, x)$

$\text{knows}(\text{John}, \text{Jane})$

$\forall y: \text{knows}(y, \text{Leonid})$

$\forall y: \text{knows}(y, \text{mother}(y))$

$\forall x: \text{knows}(x, \text{Elizabeth})$

$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(\text{John}, \text{Jane})) = \{\text{Jane}/x\}$

$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(y, \text{Leonid})) = \{\text{Leonid}/x, \text{John}/y\}$

$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(y, \text{mother}(y))) = \{\text{John}/y, \text{mother}(\text{John})/x\}$

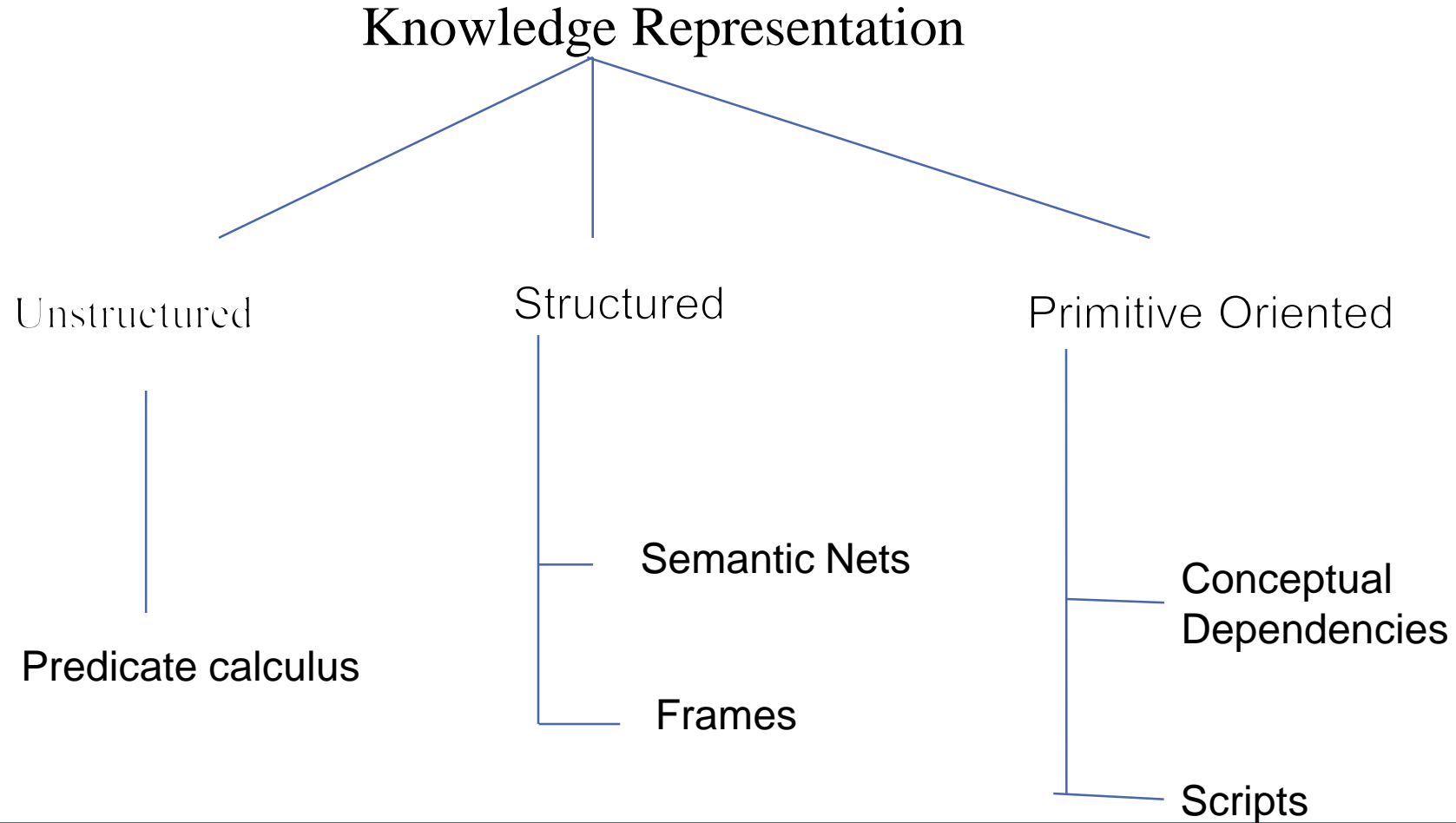
$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(x, \text{Elizabeth})) = \text{FAIL}$

# Knowledge Representation structure

---

- Knowledge types :
  1. Declarative
  2. Procedural
- Declarative knowledge deals with factoid questions (what is the capital of India? Who won the Wimbledon in 2005? Etc.)
- Procedural knowledge deals with “How”
- Procedural knowledge can be embedded in declarative knowledge

# Classification



# Predicate Calculus

---

By Gottlob Frege

Key points

- Simplest type of representation
- Fully logic based
- Deduction, Abduction and Induction
- Resolution and Refutation

Application: In rule-based systems

# Frames

---

- By Marvin Minsky in 1970
- Evolution of Frame System
- Definition- A collection of attributes and associated values that describe some entity in the world
- Differs from semantic nets in a way that frames may involve procedural embedding in place of values of attributes. (which are called as fillers)

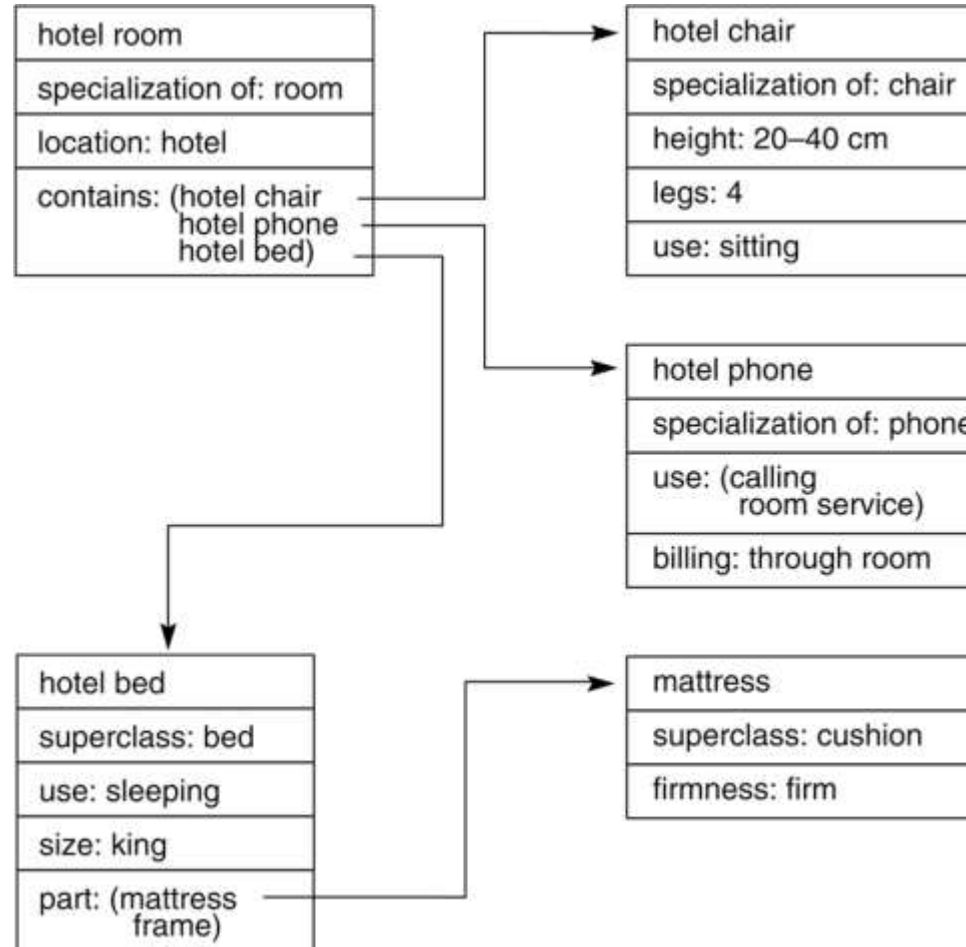


# Continued...

---

- A frame represents a concept.
- a frame system represents an organization of knowledge about a set of related concepts.
- A frame has slots that denote properties of objects. Some slots have *default* fillers, some are empty (may be filled when more becomes known about an object).
- Frames are linked by relations of specialization/generalization and by many ad-hoc relations.

# Continued...



# Conceptual Dependency (CD)

- CD theory was developed by Schank in 1973 to 1975 to represent the meaning of NL sentences.
  - It helps in drawing inferences
  - It is independent of the language
- CD representation of a sentence is not built using words in the sentence rather built using conceptual primitives which give the intended meanings of words.
- CD provides **structures** and specific **set of primitives** from which representation can be built.

# Primitive Acts of CD theory

---

- ATRANS            Transfer of an abstract relationship (i.e. give)
- PTRANS           Transfer of the physical location of an object (e.g., go)
- PROPEL           Application of physical force to an object (e.g. push)
- MOVE            Movement of a body part by its owner (e.g. kick)
- GRASP            Grasping of an object by an action (e.g. throw)
- INGEST           Ingesting of an object by an animal (e.g. eat)
- EXPEL            Expulsion of something from the body of an animal (e.g. cry)
- MTRANS           Transfer of mental information (e.g. tell)
- MBUILD           Building new information out of old (e.g decide)
- SPEAK            Producing of sounds (e.g. say)
- ATTEND           Focusing of a sense organ toward a stimulus  
(e.g. listen)

# Some of Conceptualizations of CD

---

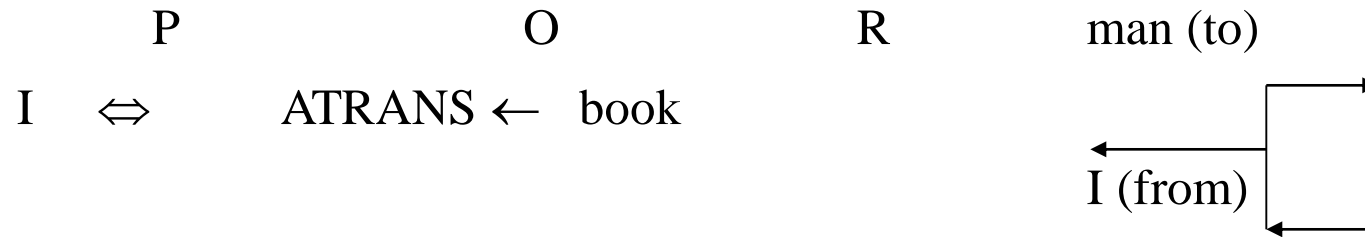
- Dependency structures are themselves conceptualization and can serve as components of larger dependency structures.
- The dependencies among conceptualization correspond to semantic relations among the underlying concepts.
- We will list the most important ones allowed by CD.
- Remaining can be seen from the book.

# Conceptual category

- There are four conceptual categories
  - ACT      Actions {one of the CD primitives}
  - PP      Objects {picture producers}
  - AA      Modifiers of actions {action aiders}
  - PA      Modifiers of PP's {picture aiders}

# Example

- I gave a book to the man. CD representation is as follows:



- It should be noted that this representation is same for different saying with same meaning. For example
  - I gave the man a book,
  - The man got book from me,
  - The book was given to man by me etc.

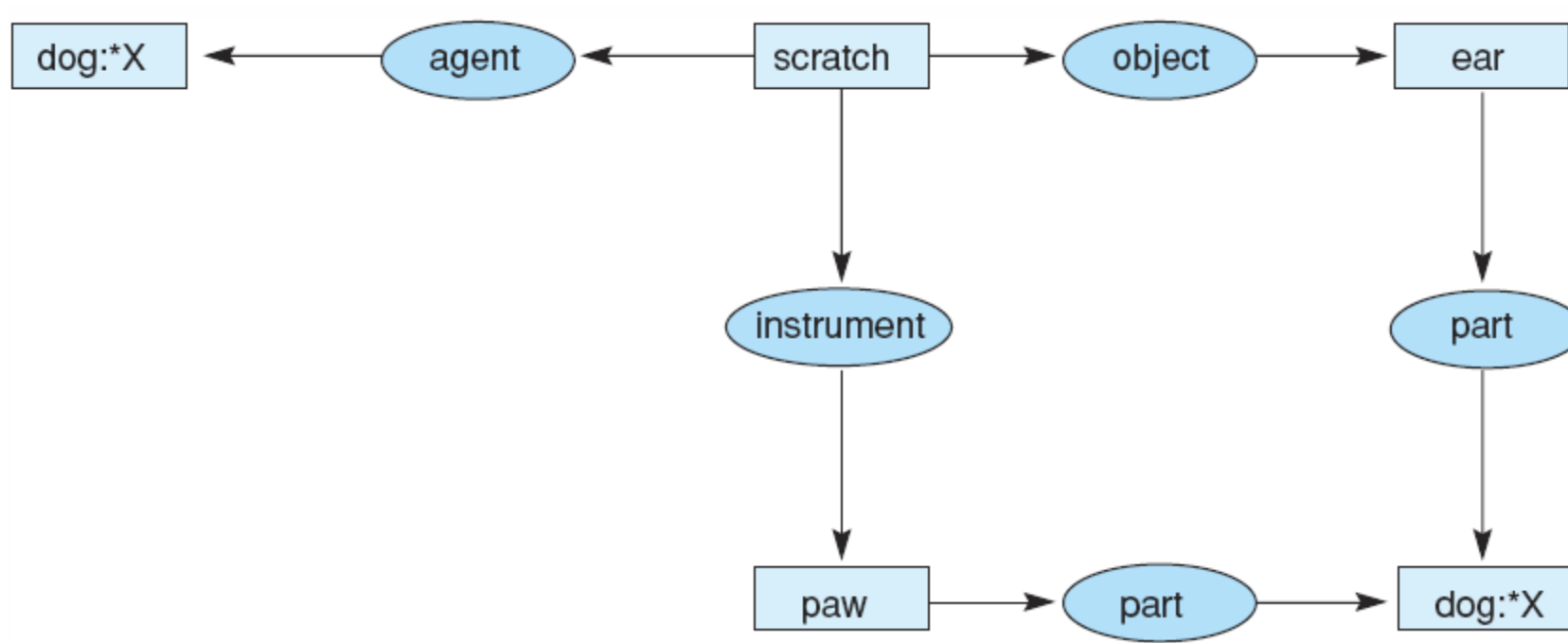
# Few conventions

---

- Arrows indicate directions of dependency
  - Double arrow indicates two way link between actor and action.
- O – for the object case relation
- R – for the recipient case relation
- P – for past tense
- D - destination



## Conceptual graph of the sentence “The dog scratches its ear with its paw.”



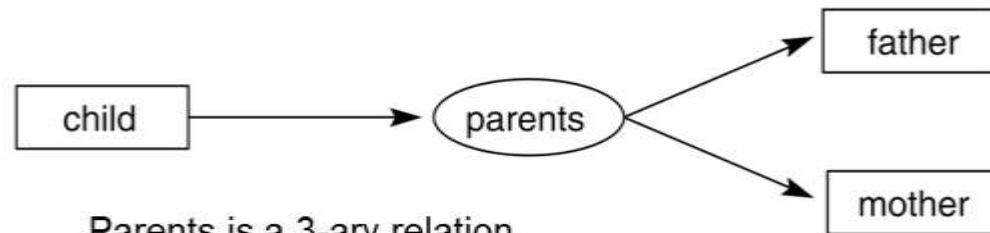
# Conceptual Graph



Flies is a 1-ary relation.



Color is a 2-ary relation.



Parents is a 3-ary relation.

# Semantic nets

---

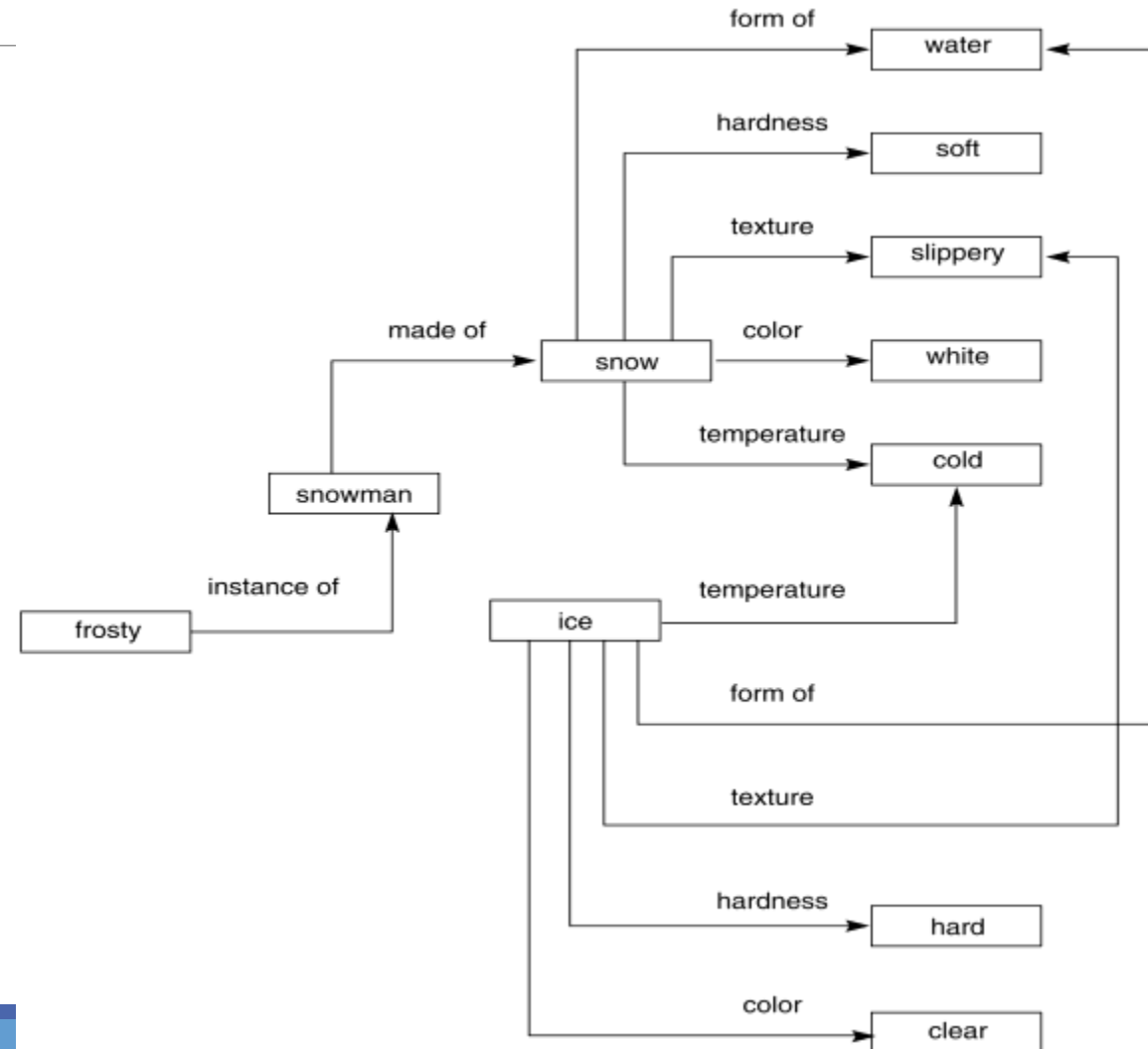
- A semantic network is an irregular graph that has concepts in vertices and relations on arcs.
- Relations can be ad-hoc, but they can also be quite general, for example, “is a” (**ISA**), “a kind of” (**AKO**), “an instance of”, “part of”.
- Relations often express physical properties of objects (colour, length, and lots of others).
- Most often, relations link two concepts.

# Continued...

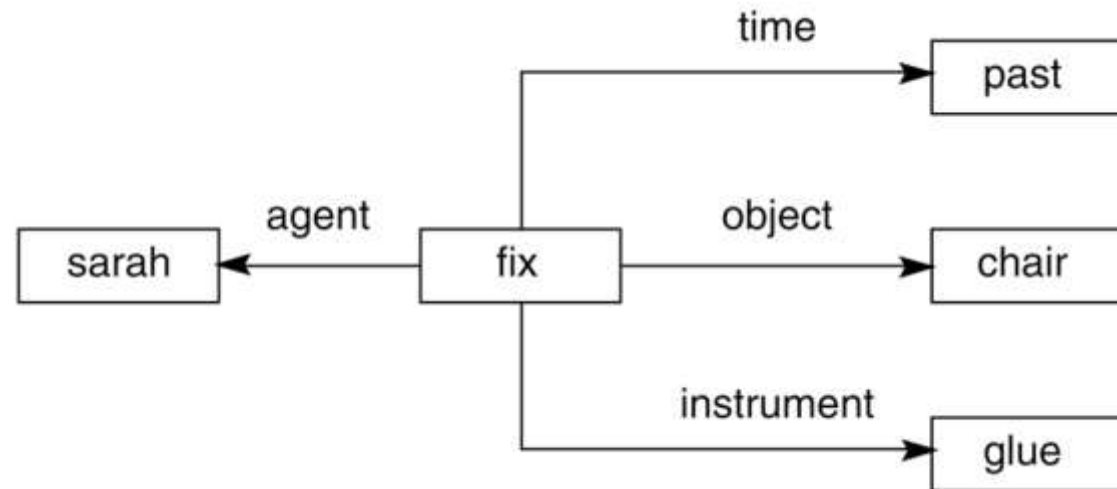
---

- General semantic relations help represent the meaning of simple sentences in a systematic way.
- A sentence is centered on a verb that *expects* certain arguments.
- For example, verbs usually denotes actions (with *agents*) or states (with passive *experiencers*, for example, “he dreams” or “he is sick”).

# Continued...



# Continued...



# What is planning in AI?

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

# Motivation

- 
- Many AI agents operate in an environment. Two issues:
  - The agent wants to find a “good” sequence of actions that will take it from an initial to a goal state
  - Every action of the agent changes some part of the state of the environment, *keeping the remaining part unchanged*.
  - After every action, how to find which part changed, and which did not? Frame problem



# *A planning agent*

---

- An agent interacts with the world via perception and actions.
- Perception involves sensing the world and assessing the situation creating some internal representation of the world
- Actions are what the agent does in the domain. Planning involves reasoning about actions that the agent intends to carry out
- *Planning* is the reasoning side of acting
- This reasoning involves the representation of the world that the agent has, as also the representation of its actions.
- Hard constraints where the objectives *have to* be achieved completely for success
- The objectives could also be soft constraints, or *preferences*, to be achieved as much as possible

# Example

- A robot has to pick up an object fallen on the floor at point A and keep it at point B on the floor in a room.
- State: positions of objects in the room + own position
- Robot actions: move one step LEFT, RIGHT, FORWARD, BACK, pick up object from floor, put down the object in its arm on the floor.
- Robot needs to find a “safe” way from point A to B, by exploring the environment
- Robot has a map of the entire room; knows all the objects in the room, and their positions.

# Example: Travel Planning

- 
- “Please plan an economy class round trip air-travel from Pune to France lasting 9 days covering at least 3 different locations. I want to spend at least 2 days in each location. I am interested in nature, history and art.”

# Planning Types

---

1. STRIPS
2. Forward and Backward State Space Planning
3. Goal Stack Planning
4. Plan Space Planning
5. A Unified Framework For Planning

# Planning in Blocks World

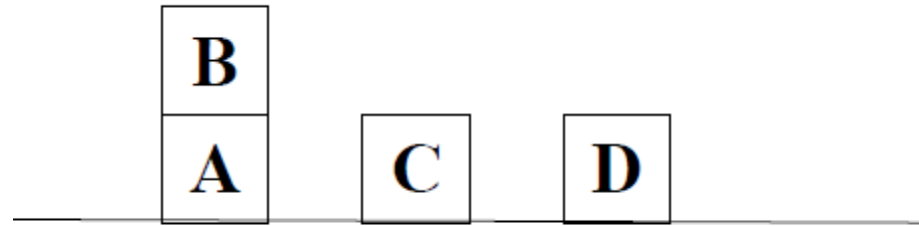
- There is a flat surface on which blocks can be placed.
- There are many square blocks, all of the same size.
- Each block has a unique ID (we use A, B, C, ... as IDs)
- Blocks can be placed on top of each other.
- There is a robot arm that can perform several actions related to moving the blocks around
- The arm can hold at most one block at a time.
- Assumption:  
Each block can have at most 1 other block on top of it.

# Predicates

A given “situation” (“state”) can be described using a formula made up of the following 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 robot arm is holding block A
5. **ARMEMPTY**:the robot arm is holding nothing

# Example

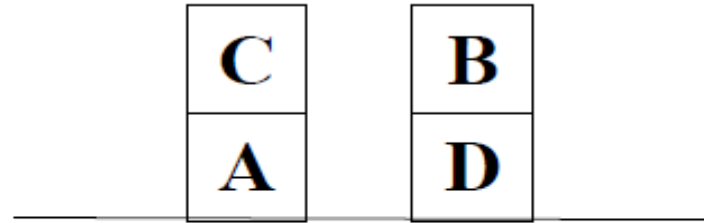


$ON(B, A) \wedge$   
 $ONTABLE(A) \wedge$   
 $ONTABLE(C) \wedge$   
 $ONTABLE(D) \wedge$   
 $ARMEMPTY$

Each state description is a “grounded”  
formula (does not contain  
any variables)

# Another example

---



$\text{ON}(\text{C}, \text{A}) \wedge$   
 $\text{ONTABLE}(\text{A}) \wedge$   
 $\text{ON}(\text{B}, \text{D}) \wedge$   
 $\text{ONTABLE}(\text{D}) \wedge$   
 $\text{ARMEMPTY}$



# Actions the Robot Arm Can Perform

- Each action (operator) transforms one state into another.
- Each action has associated with it:

**PRECONDITION:** list of predicates that must be TRUE before the action can be applied

**ADD:** list of predicates that become TRUE after the actions applied

**DELETE:** list of predicates that become FALSE after the action is applied

Any predicate not included in either the ADD or the DELETE list is assumed to be unaffected by the action

# Actions...

**UNSTACK( $x, y$ ):** pickup block  $x$  from its current position on block  $y$ .

–PRECONDITION:  $ARMEMPTY \wedge ON(x, y) \wedge CLEAR(x)$

–DELETE:  $ARMEMPTY \wedge ON(x, y)$

–ADD:  $HOLDING(x) \wedge CLEAR(y)$

–Note:  $CLEAR(x)$  is still true, even after the action occurs!

**STACK( $x, y$ ):** place block  $x$  on block  $y$ .

–PRECONDITION:  $CLEAR(y) \wedge HOLDING(x)$

–DELETE:  $CLEAR(y) \wedge HOLDING(x)$

–ADD:  $ARMEMPTY \wedge ON(x, y) \wedge CLEAR(x)$

# Actions...

---

**PICKUP( $x$ ):** pickup block  $x$  from the table and hold it.

–PRECONDITION:  $\text{ARMEMPTY} \wedge \text{CLEAR}(x) \wedge \text{ONTABLE}(x)$

–DELETE:  $\text{ARMEMPTY} \wedge \text{ONTABLE}(x)$

–ADD:  $\text{HOLDING}(x)$

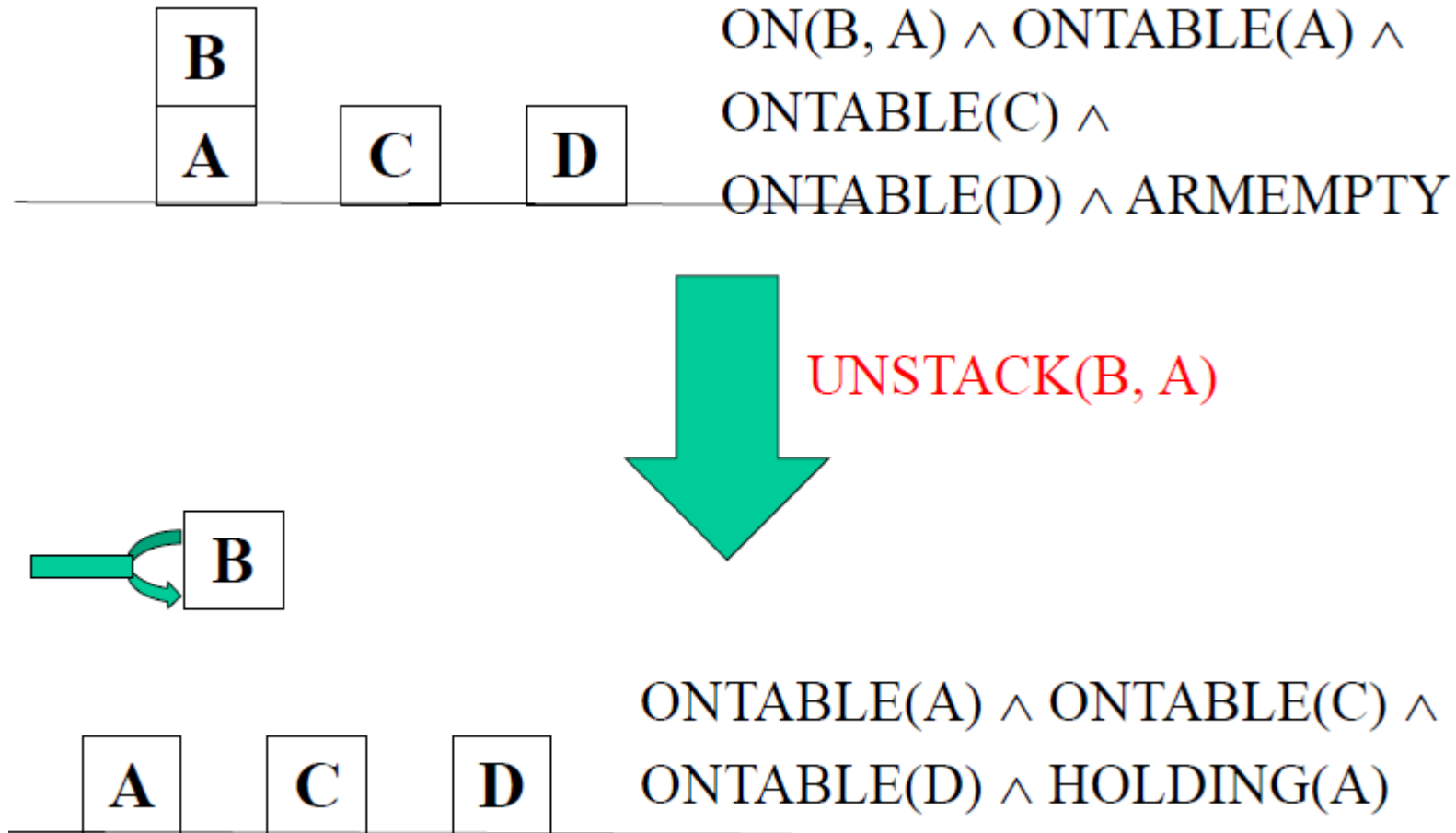
**PUTDOWN( $x$ ):** putdown block  $x$  on the table

–PRECONDITION:  $\text{HOLDING}(x)$

–DELETE:  $\text{HOLDING}(x)$

–ADD:  $\text{ARMEMPTY} \wedge \text{ONTABLE}(x)$

# Continued...



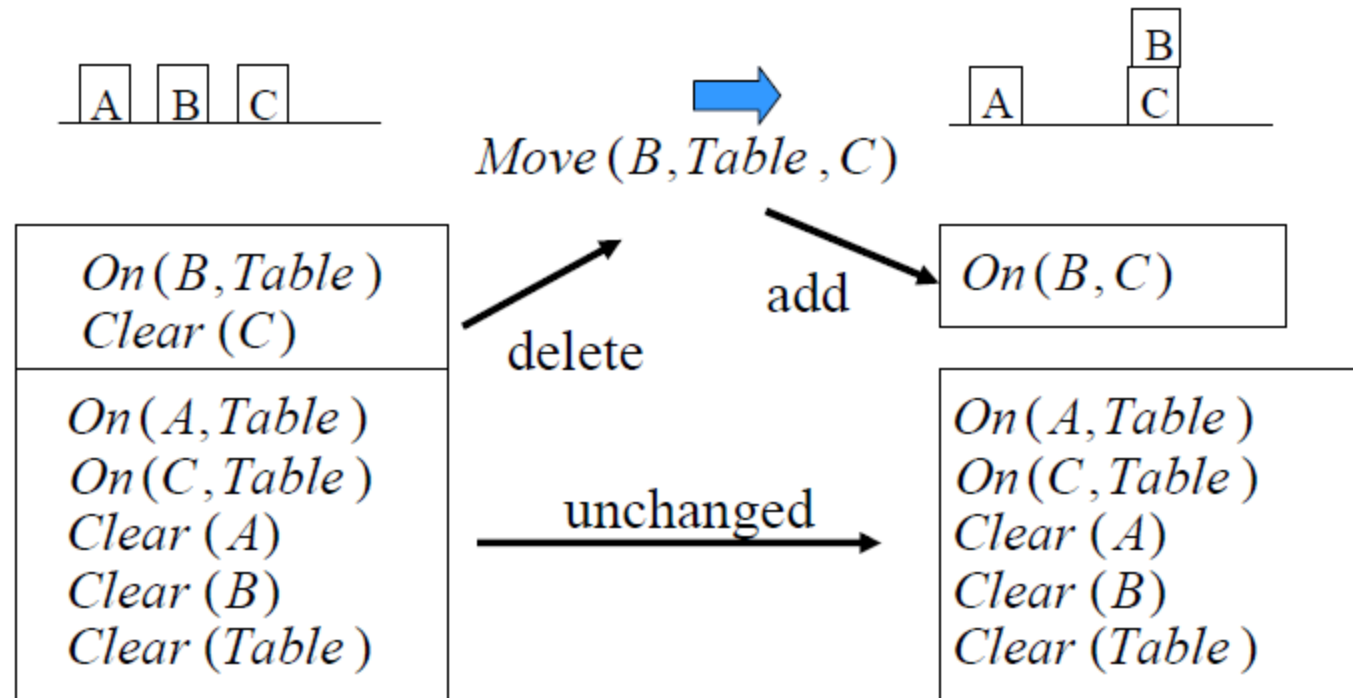
# The Planning Problem

---

Given an initial state  $S_0$ , and a final (goal) state  $S_1$ ,  
identify the “best” **sequence** of actions that will transform  $S_0$  into  $S_1$ .  
Each action can only be one of: UNSTACK, STACK, PICKUP, PUTDOWN.

## Forward search (goal progression)

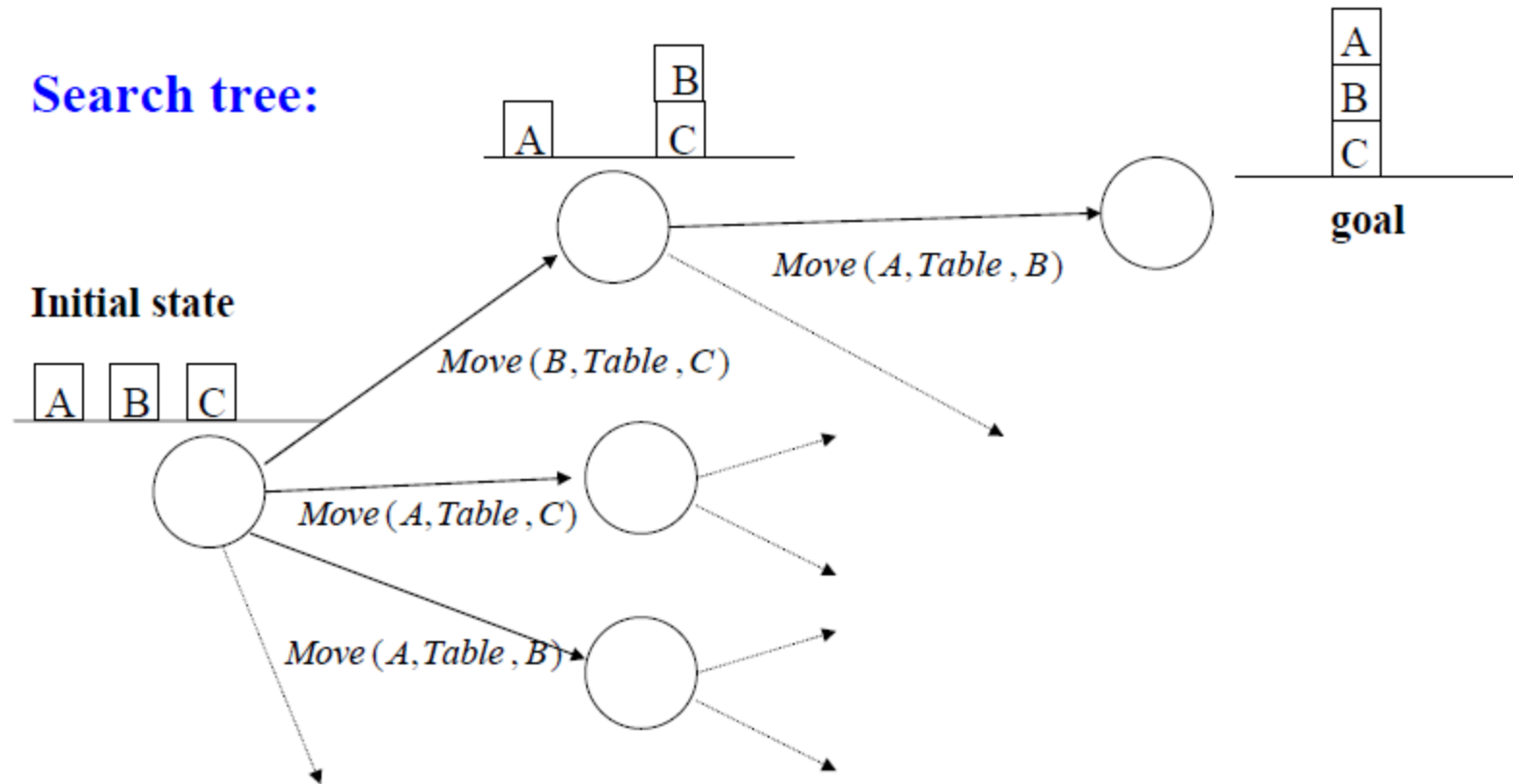
- **Idea:** Given a state  $s$ 
  - Unify the preconditions of some operator  $a$  with  $s$
  - Add and delete sentences from the add and delete list of an operator  $a$  from  $s$  to get a new state



## Forward search (goal progression)

- Use operators to generate new states to search
- Check new states whether they satisfy the goal

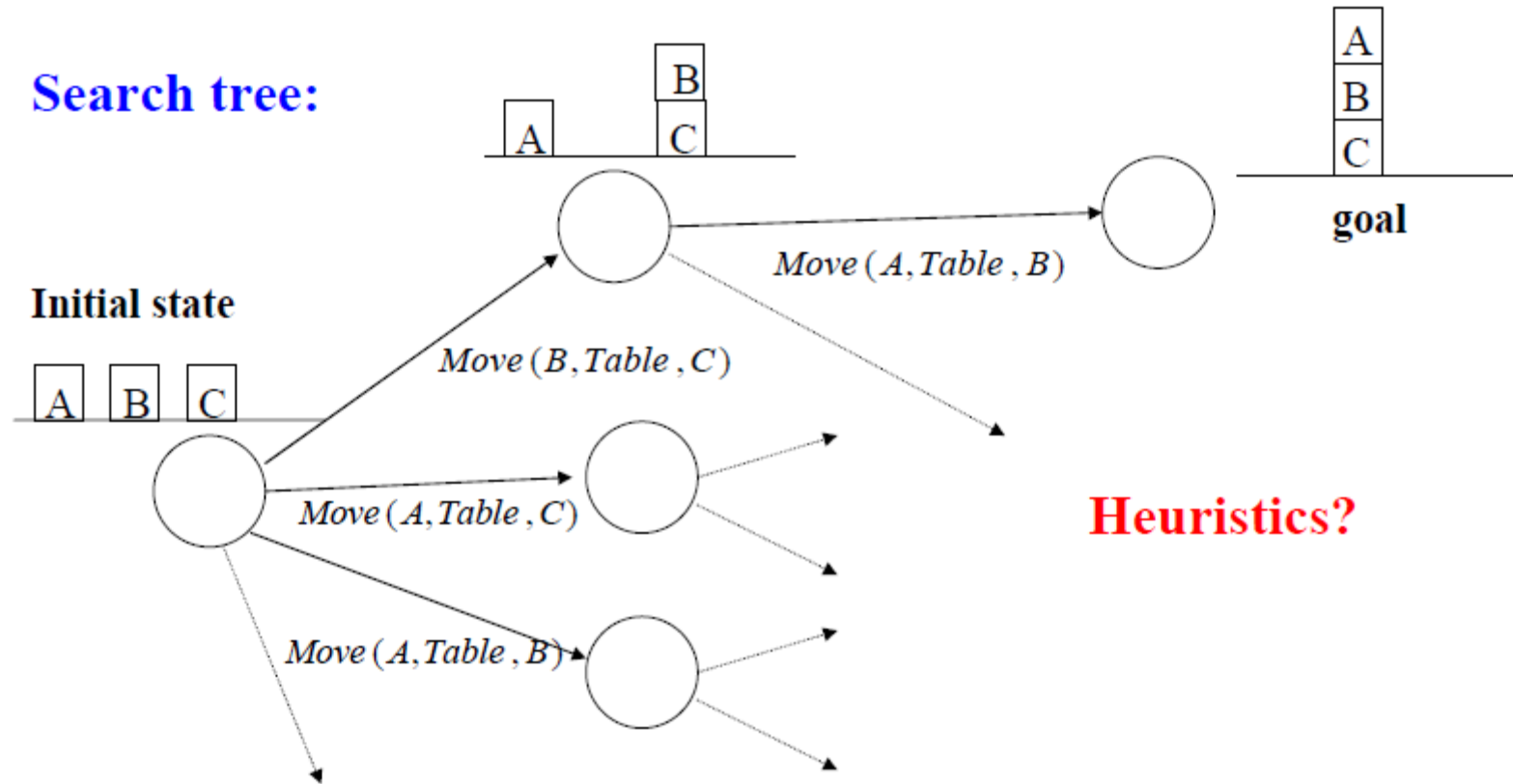
Search tree:



## Forward search (goal progression)

- Use operators to generate new states to search
- Check new states whether they satisfy the goal

Search tree:

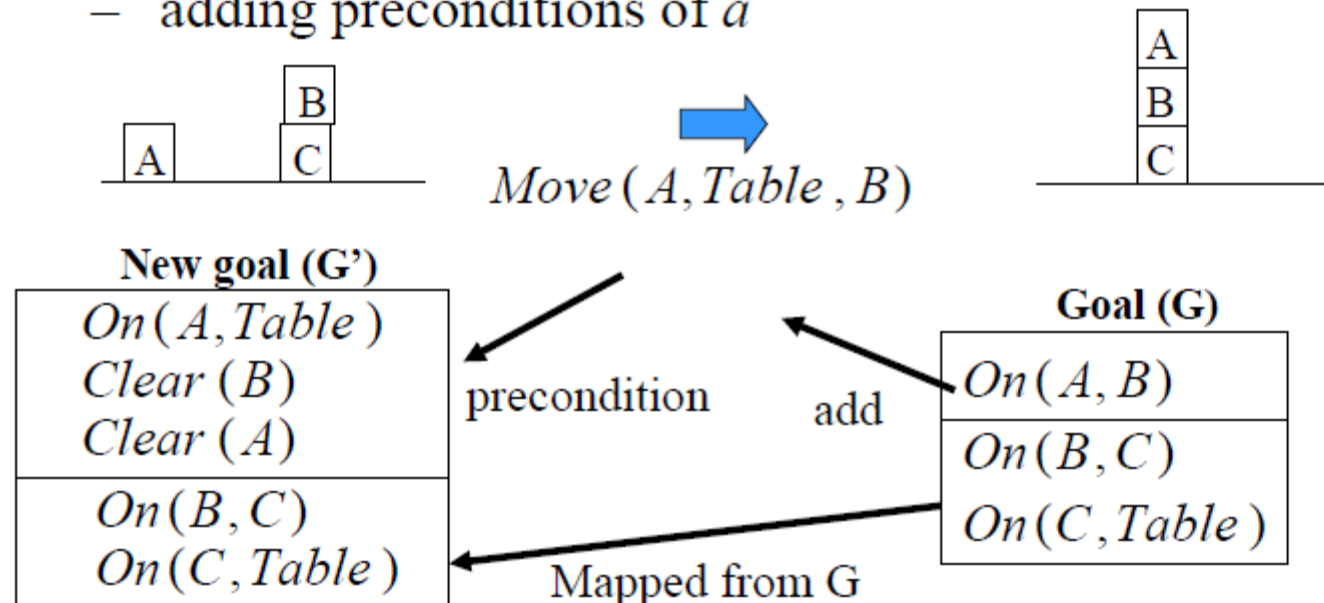




## Backward search (goal regression)

**Idea:** Given a goal  $G$

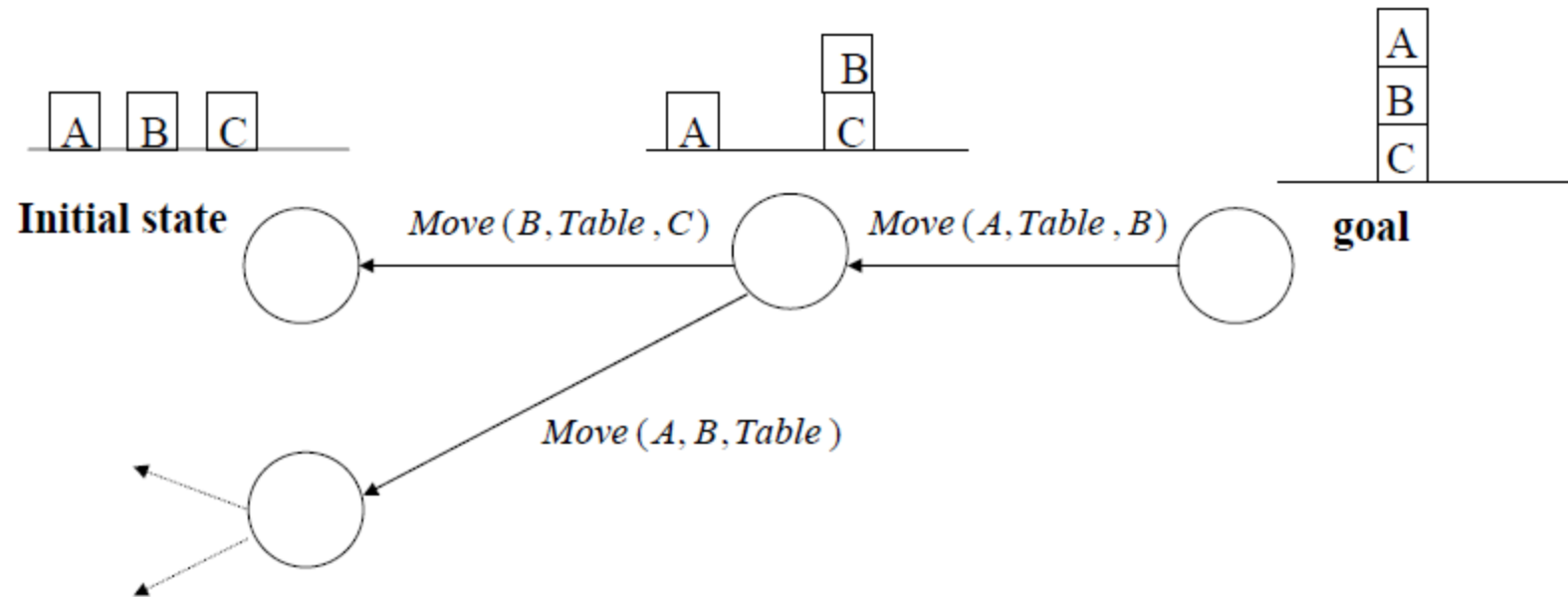
- Unify the add list of some operator  $a$  with a subset of  $G$
- If the delete list of  $a$  does not remove elements of  $G$ , then the goal regresses to a new goal  $G'$  that is obtained from  $G$  by:
  - deleting add list of  $a$
  - adding preconditions of  $a$



## Backward search (goal regression)

- Use operators to generate new goals
- Check whether the initial state satisfies the goal

### Search tree:



# Forward planning

**Start** with the **initial state** (current world state).

**Select actions** whose **preconditions** match with the current state description (before-action state) through unification.

**Apply actions** to the current world state. Generate possible **follow-states** (after-action states) according to the add and delete lists of the action description.

**Repeat** for every generated new world state.

**Stop** when a state is generated which includes the **goal formula**.

# Backward planning

---

**Start** with a given **goal state description**.

**Check unsatisfied (pre)conditions** of the goal description.

**Select and apply actions** which have those conditions as effects.

**Proceed** in the same way **backwards**, trying to fulfill preconditions of each new action by **recursively choosing actions** which achieve those preconditions.

**Stop** when the **precondition is part of the initial state**, and thus a sequence of actions is found leading from the initial state to the goal state.

# Goal Stack Planning

---

- Problem-solving is searching and moving through a state space.
- Planning is searching for successful paths through a state space
  - ✓ Planning = problem solving in advance.
  - ✓ Planning is important if solutions cannot be undone.
  - ✓ If the universe is not predictable, then a plan can fail dynamic plan revision.

# Goal Stack planning

$\Pi := \phi$  // plan is initially empty

---

$C :=$  start state //  $C$  is the current state

Push the goal state on the stack

Push all its subgoals on the stack (in any order)

Repeat until the stack is empty:

$X :=$  Pop the top of the stack

IF  $X$  is a compound goal THEN

push its subgoals which are unsatisfied in  $C$  on the stack

ELSE IF  $X$  a single goal FALSE in  $C$  THEN

push an action  $Q$  that satisfies  $X$

push all preconditions of  $Q$

ELSE IF  $X$  is an action THEN

execute  $X$  in current state  $C$ , change the new current state  $C$

using the action's effects, add  $X$  to plan  $\square$

ELSE IF  $X$  is a goal which is TRUE in current state  $C$  THEN NOTHING

# Goal Stack Planning algorithm

---

1. Push goal in the stack
  - if top is compound goal, push to stack
  - If top is single unsatisfied goal replace it by an action
  - Push action precondition
  - If top is action then pop
  - If top is satisfied goal then complete

# Rules

---

R1 : pickup(x)

armempty, OnTable(x), clear(x)

R2: putdown(x)

holding(x)

R3:stack(x,y)

holding(x)

clear(y)

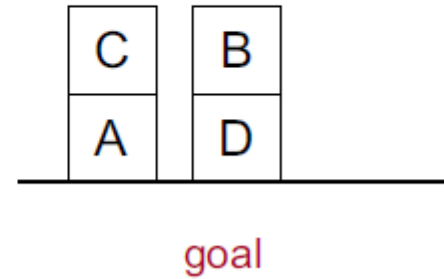
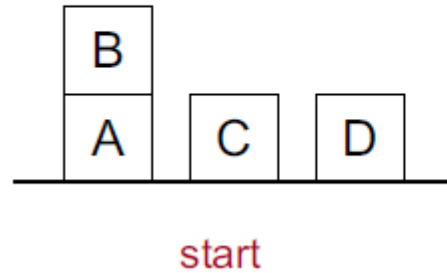
R4: unstack(x,y)

armempty, on(x,y), clear(x)



# Continued...

The blocks world



Planning = generating a sequence of actions to achieve the goal from the start

# Continued...

---

➤ Actions:

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

# Continued...

---

➤ Conditions and results:

1. ON(A, B)
2. ONTABLE(A)
3. CLEAR(A)
4. HOLDING(A)
5. ARMEMPTY

# Hierarchical planning

---

**hierarchical decomposition** : HIERARCHICAL an idea that pervades almost all DECOMPOSITION attempts to manage complexity.

For example, complex software is created from a hierarchy of subroutines or object classes; armies operate as a hierarchy of units; governments and corporations have hierarchies of departments, subsidiaries, and branch offices.

The key benefit of hierarchical structure is that, at each level of the hierarchy, a computational task, military mission, or administrative function is reduced to a *small* number of activities at the next lower level, so the computational cost of finding the correct way to arrange those activities for the current problem is small

# Hierarchical Planning

## Principle

- hierarchical organization of 'actions'
- complex and less complex (or: abstract) actions
- lowest level reflects directly executable actions

## Procedure

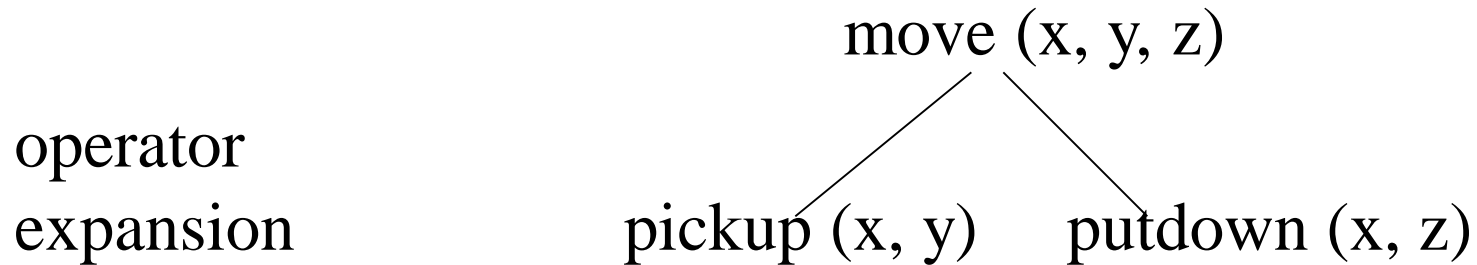
- planning starts with complex action on top
- plan constructed through action decomposition
- substitute complex action with plan of less complex actions (pre-defined plan schemata; or learning of plans/plan abstraction)
- overall plan must generate effect of complex action

# Continued...

## Hierarchical Planning / Plan Decomposition

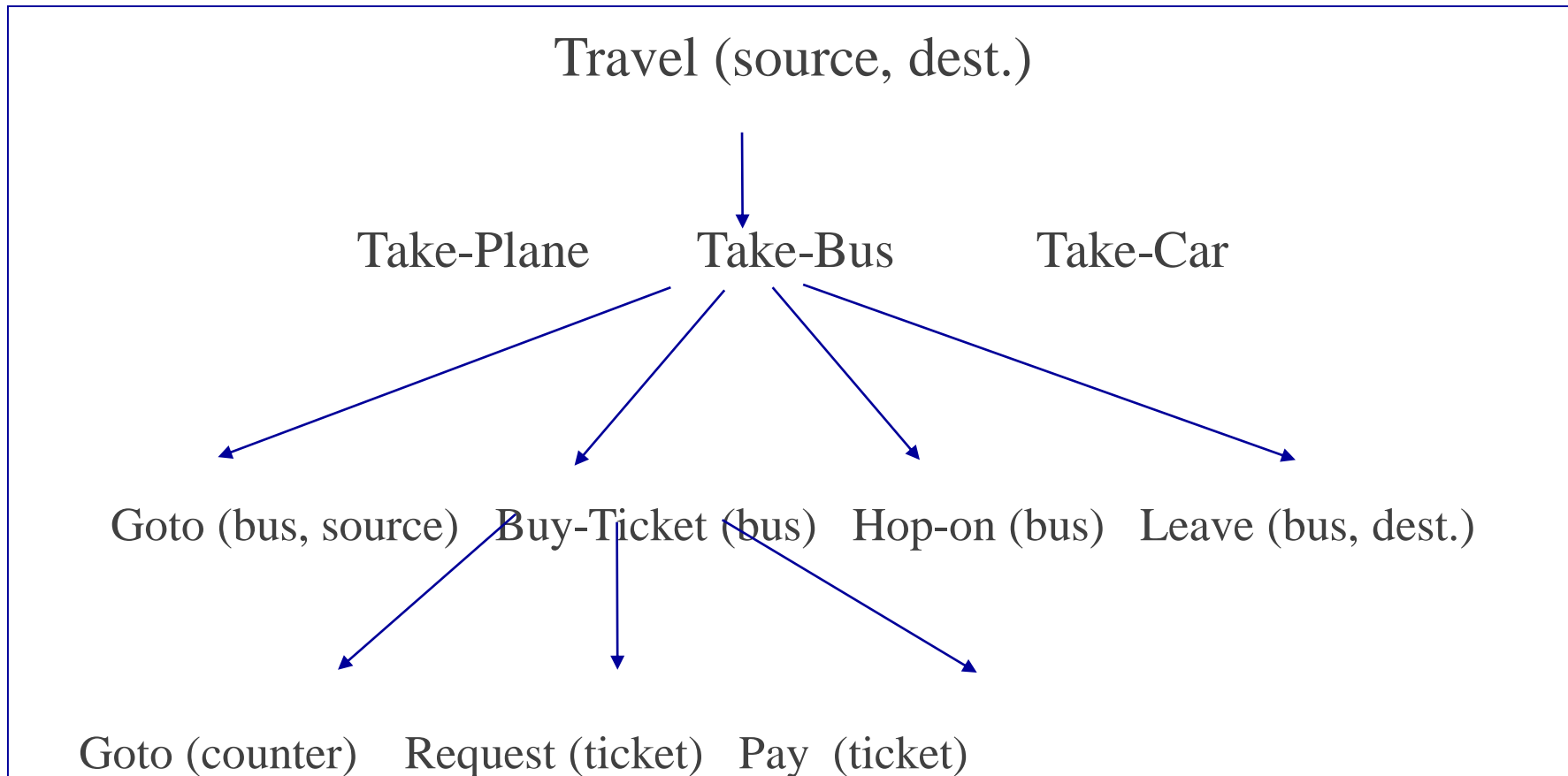
Plans are organized in a hierarchy. Links between nodes at different levels in the hierarchy denote a decomposition of a “complex action” into more primitive actions (operator expansion).

### Example:

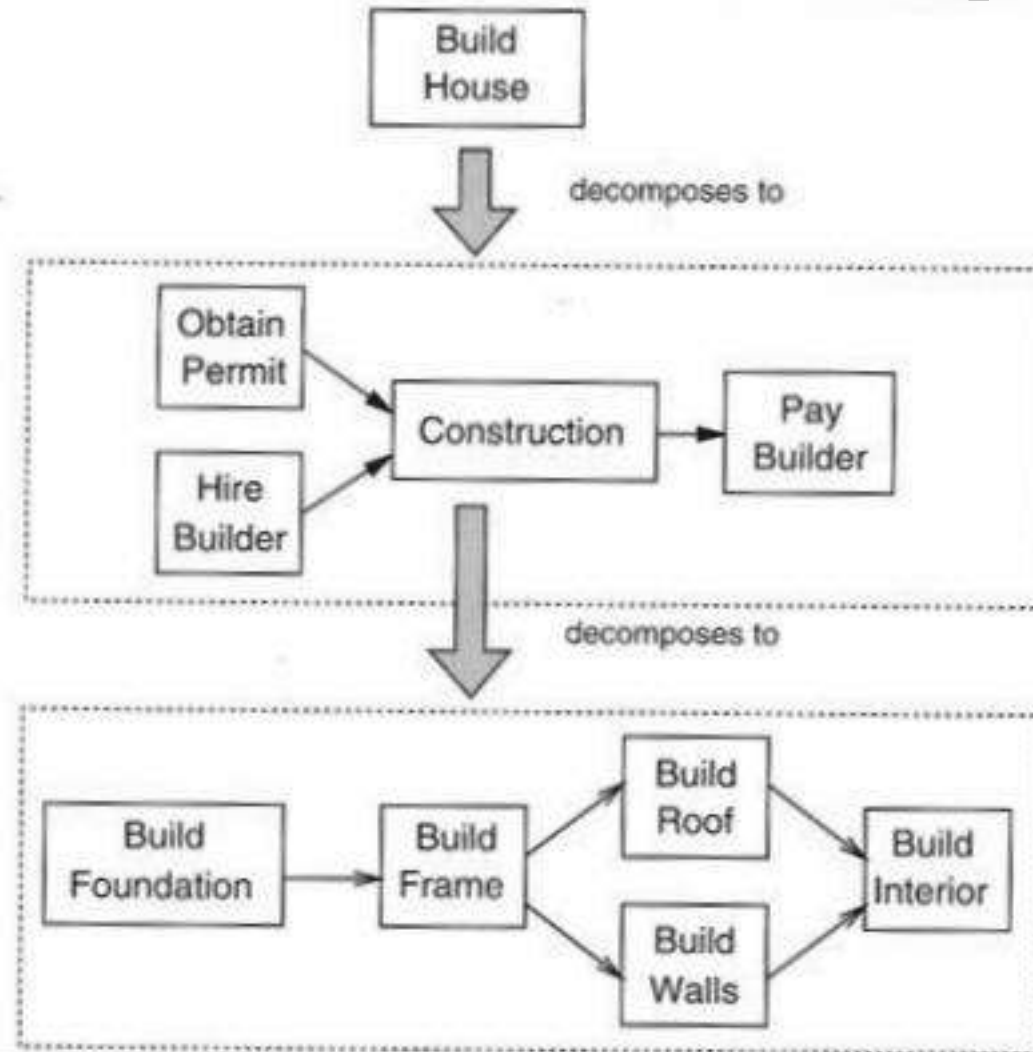


*The lowest level corresponds to executable actions of the agent.*

# Hierarchical Plan - Example



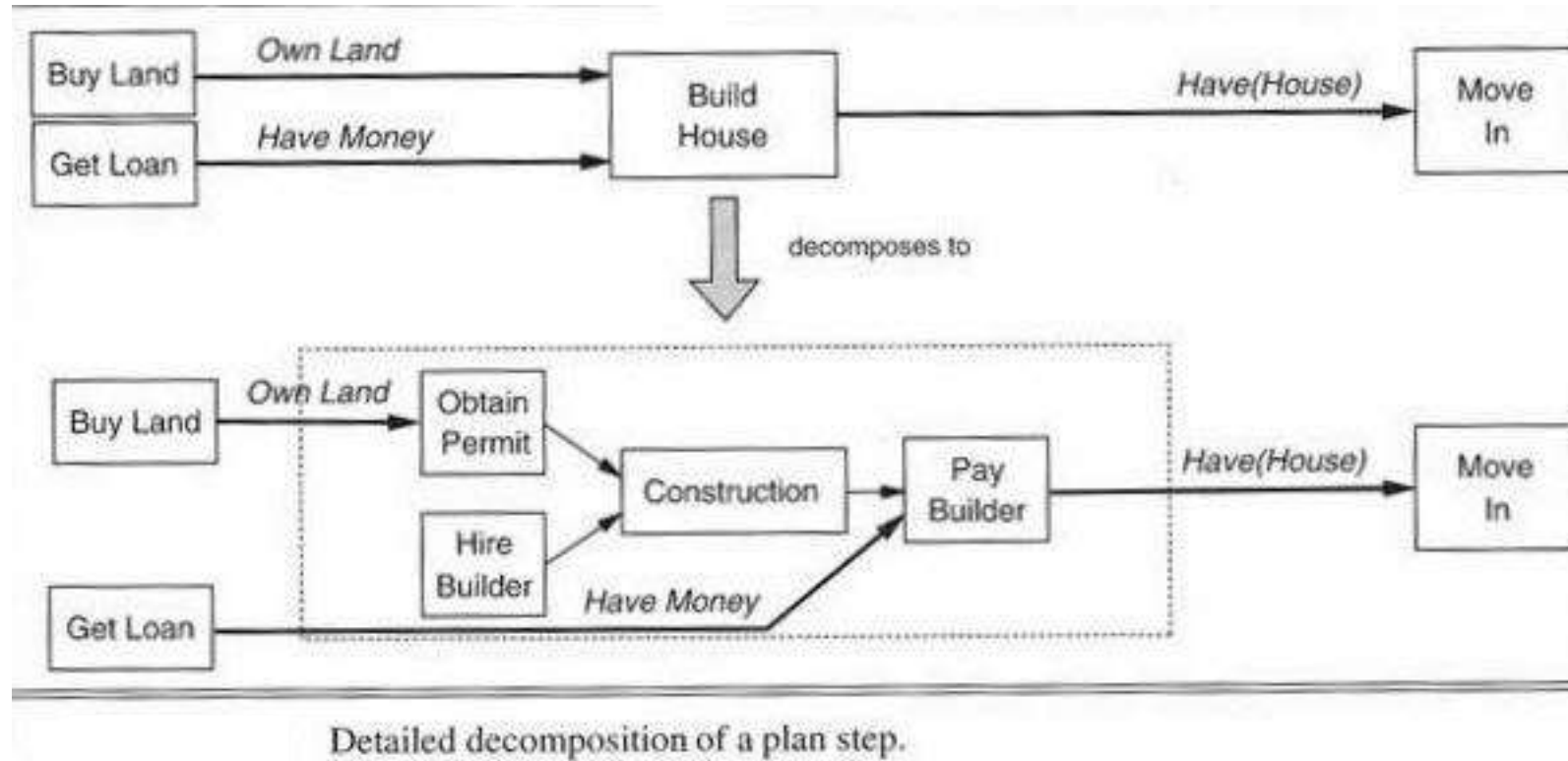
# Hierarchical Plan - Example



Hierarchical decomposition of the operator *Build House* into a partial-order plan.



# Hierarchical Plan - Example





**THANK  
YOU FOR  
LISTENING  
ANY  
QUESTION ?**