Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

CS 332 Artificial Intelligence

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

CS 332 Artificial Intelligence

Teaching Scheme **Credits: 2 + 1 = 3**

Theory: 4 Hrs / Week **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.

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

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.

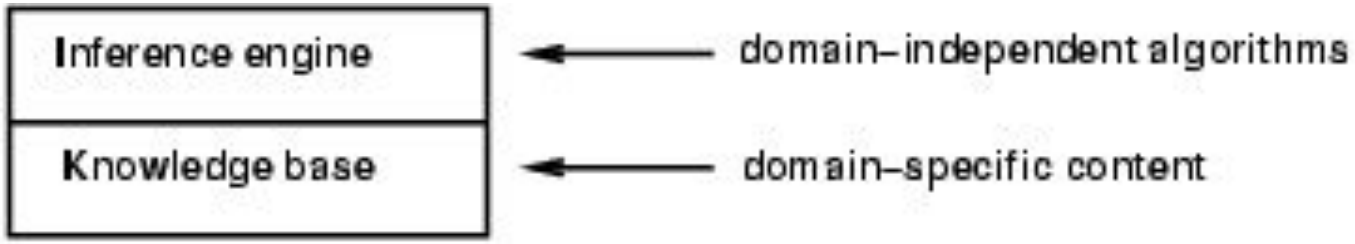
MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

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:

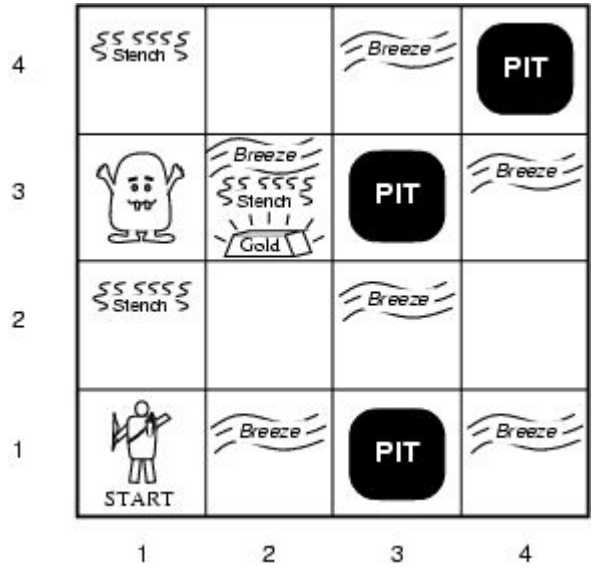
1. It TELLs the knowledge base what it perceives.
2. It ASKs the knowledge base what action it should perform.
3. It performs the chosen action.

The Wumpus World environment

- The agent explores a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
- Some rooms contain bottomless pits that trap any agent that wanders into the room.
- The Wumpus can fall into a pit too, so avoids them Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten

Wumpus World PEAS description

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow
- Environment
  - Squares adjacent to wumpus are smelly
    Squares adjacent to pit are breezy
    Glitter iff gold is in the same square
    Shooting kills wumpus if you are facing it
    Shooting uses up the only arrow
    Grabbing picks up gold if in same square
    Releasing drops the gold in same square
- Sensors: Stench, Breeze, Glitter, Bump, Scream
  Left turn, Right turn, Forward, Grab, Release, Shoot



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 for 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

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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; $x2+y > \{\}$ 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
3. Elaborat with example Propositional logic and Predicate logic
4.

99

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

- The symbols of propositional calculus are the propositional symbols:

P, Q, R, S, …

- Truth symbols:

true, false

and connectives:

$\wedge$, $\vee$, $\neg$, $\rightarrow$, $\equiv$

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:

o (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")

o Q => P (here meaning "If it is humid, then it is hot")

o Q (here meaning "It is humid.")

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Truth Table

p	q	p → q		~p	~p v 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 ∧ P ==> P

P ∨ P ==> P

2. Commutative rule:

P ∧ Q ==> Q ∧ P

P ∨ Q ==> Q ∨ P

3.Associative rule:

P ∧ (Q ∧ R) ==> (P ∧ Q) ∧ R

P ∨ (Q ∨ R) ==> (P ∨ Q) ∨ R

Continued…

4. Distributive Rule:

P ∨ (Q ∧ R) ==> (P ∨ Q) ∧ (P ∨ R)

P ∧ (Q ∨ R) ==> (P ∧ Q) ∨ (P ∧ R)

5. De-Morgan's Rule:

¬(P ∨ Q) ==> ¬P ∧ ¬Q

¬ (P ∧ Q) ==> ¬ P ∨ ¬Q

6. Implication elimination:

P □ Q => ¬P ∨ Q

Continued…

7. Bidirectional Implication elimination:

(P ⇔ Q) ==> (P □ Q) ∧ (Q □ P)

8. Contrapositive rule:

P □ Q => ¬P □ ¬Q

9. Double Negation rule:

¬(¬ P) => P

10. Absorption Rule:

P ∨ (P ∧ Q) => P

P ∧ (P ∨ Q) => P

$\neg P \vee Q$ and $P \rightarrow Q$

P	Q	$\neg P$	$\neg P \vee Q$	$P \Rightarrow Q$	$(\neg P \vee Q) = (P \Rightarrow Q)$
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

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 ∧ R) □ 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

dog(Spot)

- Every dog has a tail

$\forall x: dog(x) \rightarrow hastail(x)$

hastail(Spot)

Spot has a tail

Quantifiers

1. Universal quantifier (∀)

- ∀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”

∀x: loves(John , x)

Continued…

2. Existential quantifier (∃):

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

Continued…

3. Nested Quantifiers

- We can use both ∀ and ∃ separately
- Ex: “ everybody loves somebody ”

∀x: ∃y: loves (x , y)

- Connection between ∀ and ∃
- “ everyone dislikes garlic”

∀ x: ¬ like (x , garlic)

 This can be also said as:

“there does not exists someone who likes garlic”

¬ ∃x: like (x, garlic)

Continued…

All Romans were either loyal to Caesar or hated him.

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

4. Every one is loyal to someone.

$\forall x: \exists y: loyalto(x, y)$ $\exists y: \forall x: loyalto(x, y)$

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

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

$\rightarrow \neg loyalto(x, y)$

Inference Methods

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

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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
  $American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$
- Nono…has some missiles
  $\exists$ x Owns(Nono, x) $\land$ Missiles(x)
  $Owns(Nono, M_1)$ *and* $Missle(M_1)$
- All of its missiles were sold to it by Col. West
  $\forall x\ Missle(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- Missiles are weapons
  $Missle(x) \Rightarrow Weapon(x)$

Example Knowledge Base

An enemy of America counts as “hostile”

Enemy(x, America) ⇒ Hostile(x)

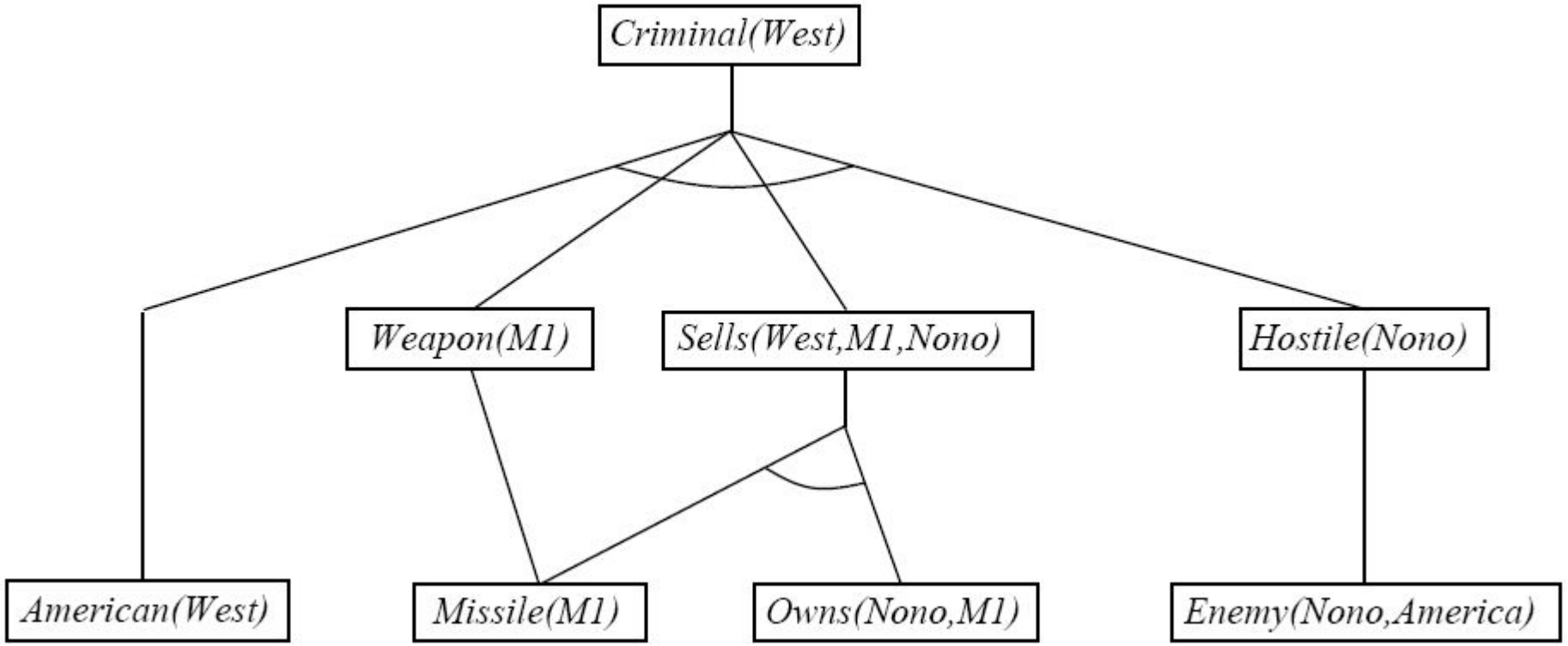
Col. West who is an American

American(Col. West)

The country Nono, an enemy of America

Enemy(Nono, America)

Example Knowledge Base



Backward Chaining

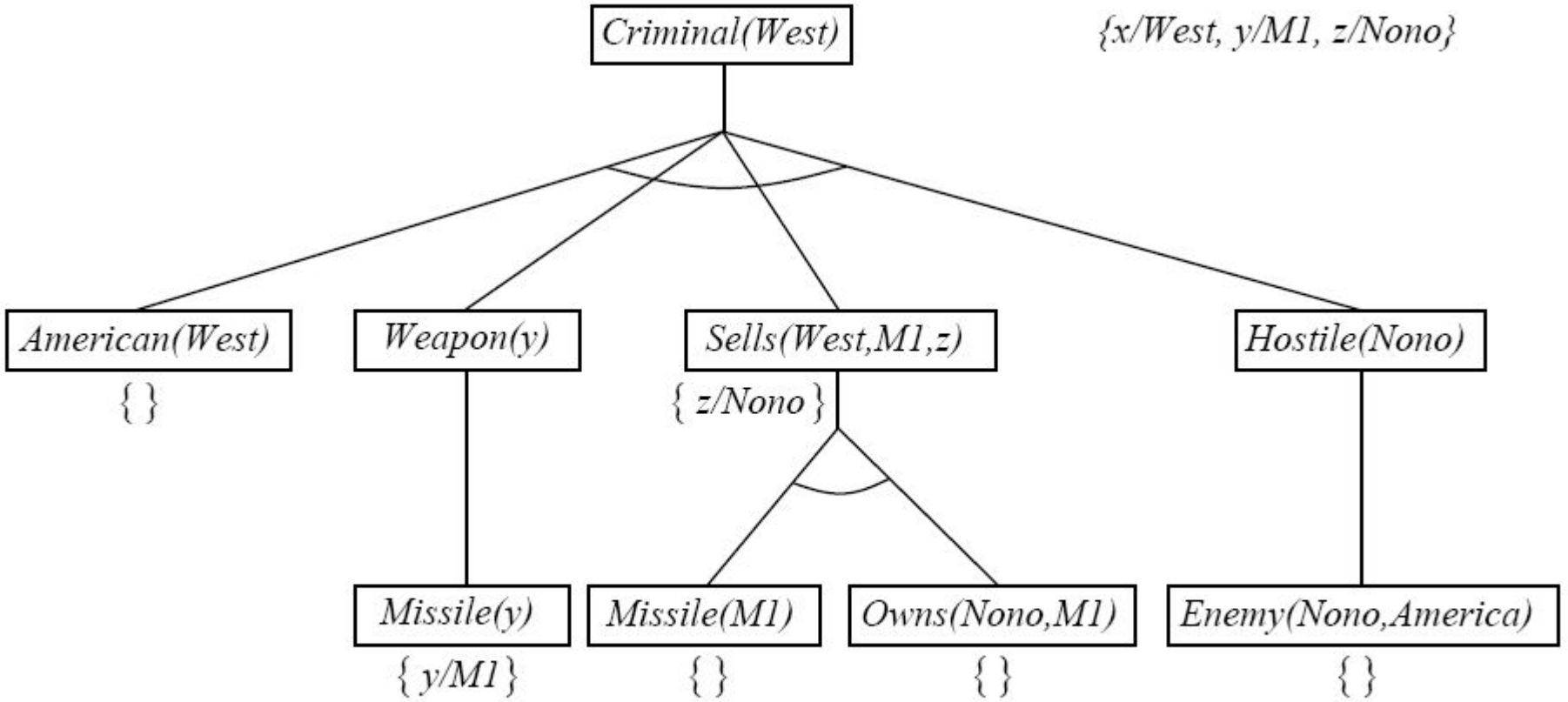
- Consider the item to be proven a goal
- Find a rule whose head is the goal (and bindings)
- Apply bindings to the body, and prove these (subgoals) in turn
- If you prove all the subgoals, increasing the binding set as you go, you will prove the item.
- Logic Programming (cprolog, on CS)

Forward Chaining Algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
  new $\leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

Backward Chaining Example



MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Backward Chaining Algorithm

function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions

inputs: KB, a knowledge base
$goals$, a list of conjuncts forming a query
θ, the current substitution, initially the empty substitution $\{\}$

local variables: ans, a set of substitutions, initially empty

if $goals$ is empty **then return** $\{\theta\}$

$q' \leftarrow$ SUBST(θ, FIRST($goals$))

for each r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
and $\theta' \leftarrow$ UNIFY(q, q') succeeds

$ans \leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta', \theta$)) $\cup$ ans

return ans

Logic Programming

Logic Programming

- Identify problem
- Assemble information
- Encode information in KB
- Encode problem instance as facts
- Ask queries
- Find false facts

Ordinary Programming

- Identify problem
- Assemble information
- Figure out solution
- Program Solution
- Encode problem instance as data
- Apply program to data
- Debug procedural errors

Logic Programming

- Basis: backward chaining with Horn clauses + lots of bells and whistles
  - Widely used in Europe and Japan
    - Basis of 5th Generation Languages and Projects
- Compilation techniques -> 60 million LIPS
- Programming = set of clauses

  head :- literal1, …, literaln

  criminal(X) :- american(X), weapon(X), sells(X, Y, Z), hostile(Z)

Logic Programming

- Rule Example
  puton(X,Y) :- cleartop(X), cleartop(Y), takeoff(X,Y).
- Capital letters are variables
- Three parts to the rule
  - Head (thing to prove)
  - Neck :-
  - Body (subgoals, separated by ,)
- Rules end with .

Logic Programming

- Efficient unification by open coding
- Efficient retrieval of matching clauses by direct linking
- Depth-first, left-to-right, backward chaining
- Built-in predicate for arithmetic e.g. X is Y*Z+2
- Closed-world assumption (“negation as failure”)
  - e.g. given alive(X) :- not dead(X).
  - alive(Joe) succeeds if dead(joe) fails

Logic Programming

- These notes are for gprolog
- To read a file, consult('file').
- To enter data directly, consult(user).
- Every statement must end in a period. If you forget, put it on the next line.
- To prove a fact, enter the fact directly at the command line. gprolog will respond Yes, No, or give you a binding set. If you want another answer, type ; otherwise return.
- Trace(predicate) or trace(all) will allow you to watch the backward chaining process.

RESOLUTION IN PREDICATE LOGIC

- Two literals are contradictory if one can be unified with the negation of the other.
- For example man(x) and man (Himalayas) are contradictory since man(x) and man(Himalayas) can be unified.
- In predicate logic unification algorithm is used to locate pairs of literals that cancel out.
- It is important that if two instances of the same variable occur, then they must be given identical substitutions.

Resolution Algorithm

Let f be a set of given statements and S is a statement to be proved.

1. Covert all the statements of F to clause form.

2. Negate S and convert the result to clause form. Add it to the set of clauses obtained in 1.

3. Repeat until either a contradiction is found or no progress can be made or a predetermined amount of effort has been expended. a) Select two clauses. Call them parent clauses.

b) Resolve them together. The resolvent will be the disjunction of all of these literals of both clauses. If there is a pair of literals T1 and T2 such that one parent clause contains Ti and the other contains T2 and if T1 and T2 are unifiable, then neither t1 nor T2 should appear in the resolvent. Here Ti and T2 are called complimentary literals.

C) If the resolvent is the empty clause , then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

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

UNIFY(p, q) = unifier θ where SUBST(θ, p) = SUBST(θ, q)

∀x: knows(John, x) → hates(John, x)

knows(John, Jane)

∀y: knows(y, Leonid)

∀y: knows(y, mother(y))

∀x: knows(x, Elizabeth)

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

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

UNIFY(knows(John, x), knows(y, mother(y))) = {John/y,

mother(John)/x}

UNIFY(knows(John, x), knows(x, Elizabeth)) = FAIL

Algorithm

UNIFY (L1, L2)

1. if L1 or L2 is an atom part of same thing do

(a) if L1 or L2 are identical then return NIL

(b) else if L1 is a variable then do
- (i) if L1 occurs in L2 then return F else return (L2/L1)
- © else if L2 is a variable then do
- (i) if L2 occurs in L1 then return F else return (L1/L2)
- else return F.

2. If length (L!) is not equal to length (L2) then return F.

3. Set SUBST to NIL

(at the end of this procedure , SUBST will contain all the substitutions used to unify L1 and L2).

4. For I = 1 to number of elements in L1 do

i) call UNIFY with the i th element of L1 and I'th element of L2, putting the result in S

ii) if S = F then return F

iii) if S is not equal to NIL then do

(A) apply S to the remainder of both L1 and L2

(B) SUBST := APPEND (S, SUBST) return SUBST.

Knowledge Representation structure

- Knowledge types :

  1. Declarative

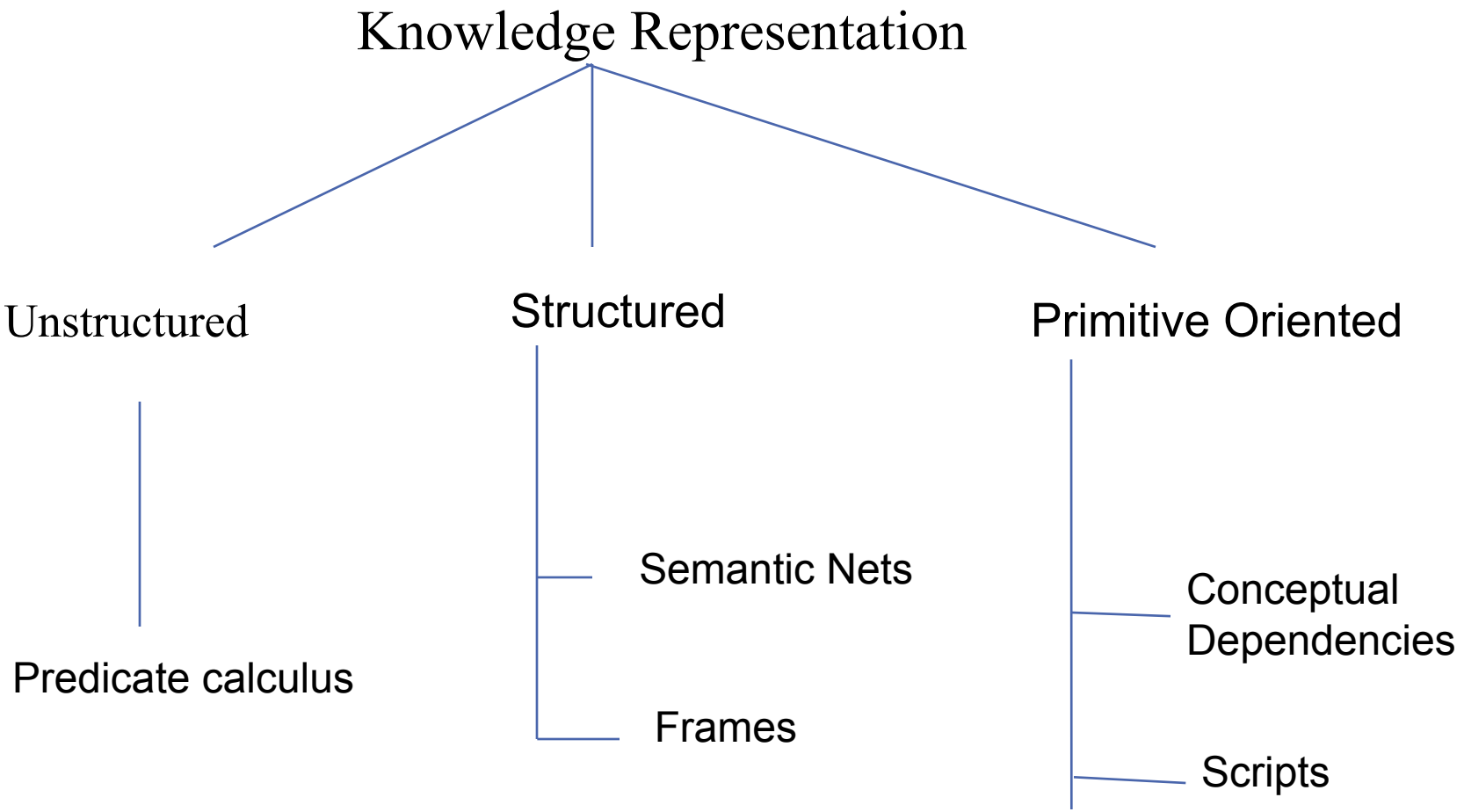
  2. Procedural

•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

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.

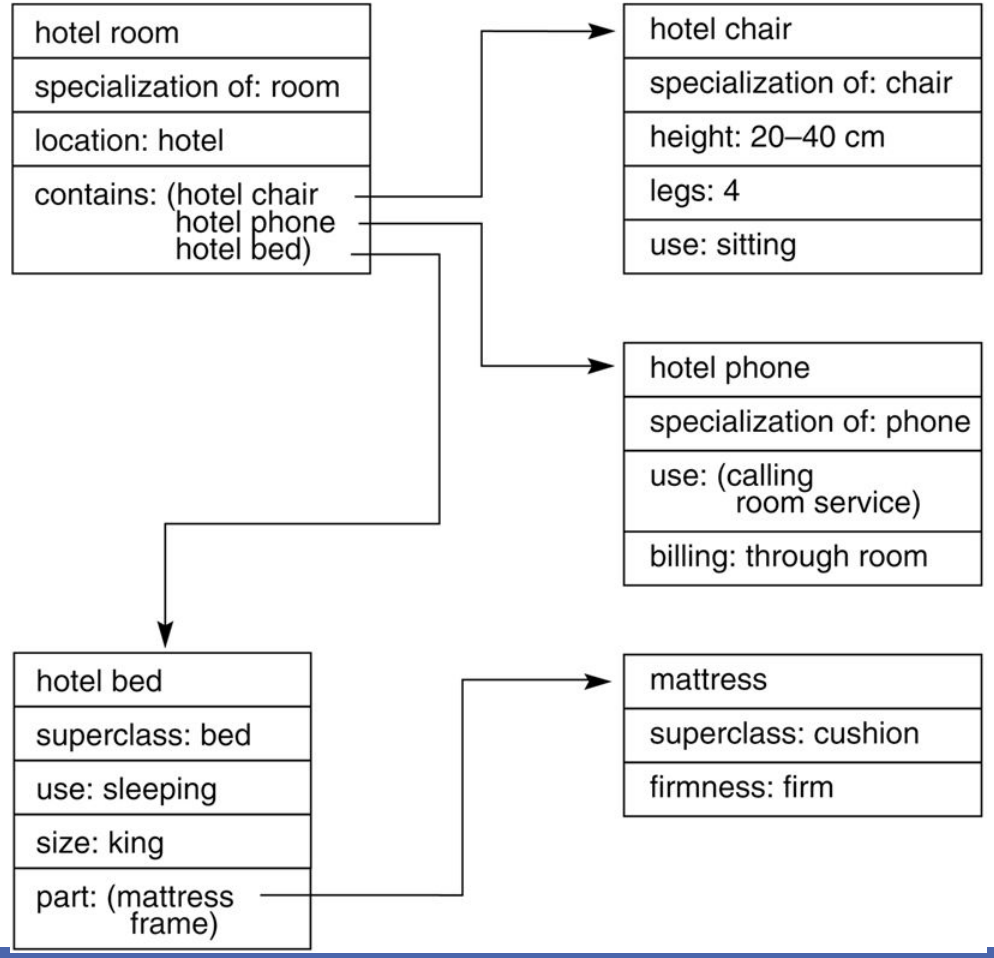
Frames

- 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…


hotel room
specialization of: room
location: hotel
contains: (hotel chair
hotel phone
hotel bed)
hotel chair
specialization of: chair
height: 20–40 cm
legs: 4
use: sitting
hotel phone
specialization of: phone
use: (calling
room service)
billing: through room
hotel bed
superclass: bed
use: sleeping
size: king
part: (mattress
frame)
mattress
superclass: cushion
firmness: firm

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}

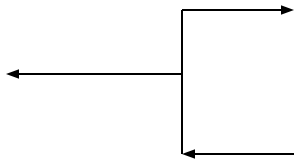
Example

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

P O R man (to)

I ⇔ ATRANS ← book

I (from)



- 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

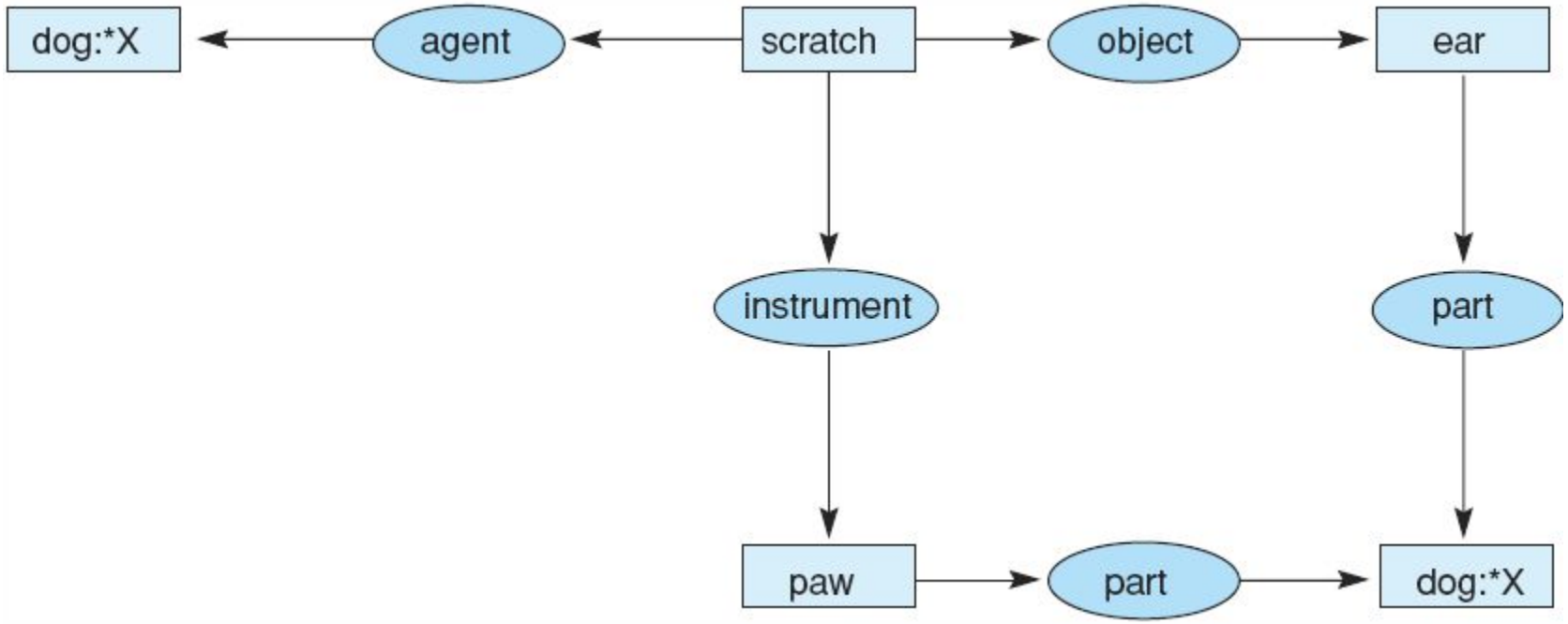
  R – for the recipient case relation

  P – for past tense

  D - destination

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

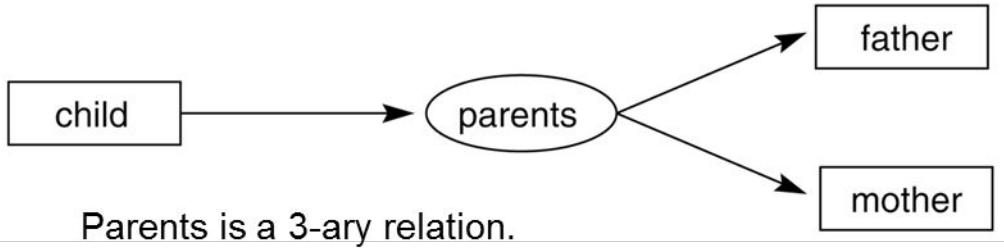
Conceptual graph of the sentence “The dog scratches its ear with its paw.”



Conceptual Graph

Flies is a 1-ary relation.

Color is a 2-ary relation.

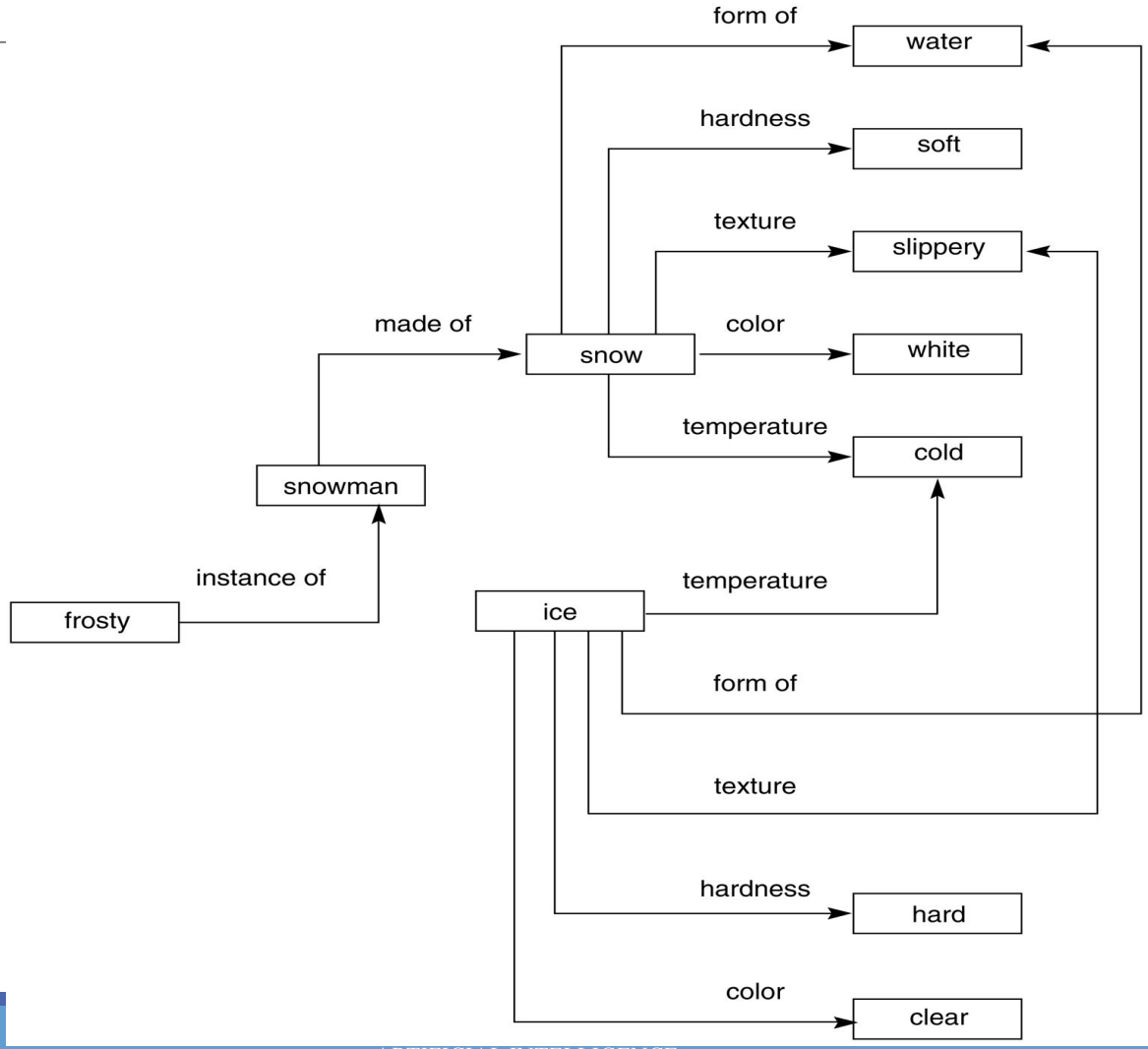


Parents is a 3-ary relation.

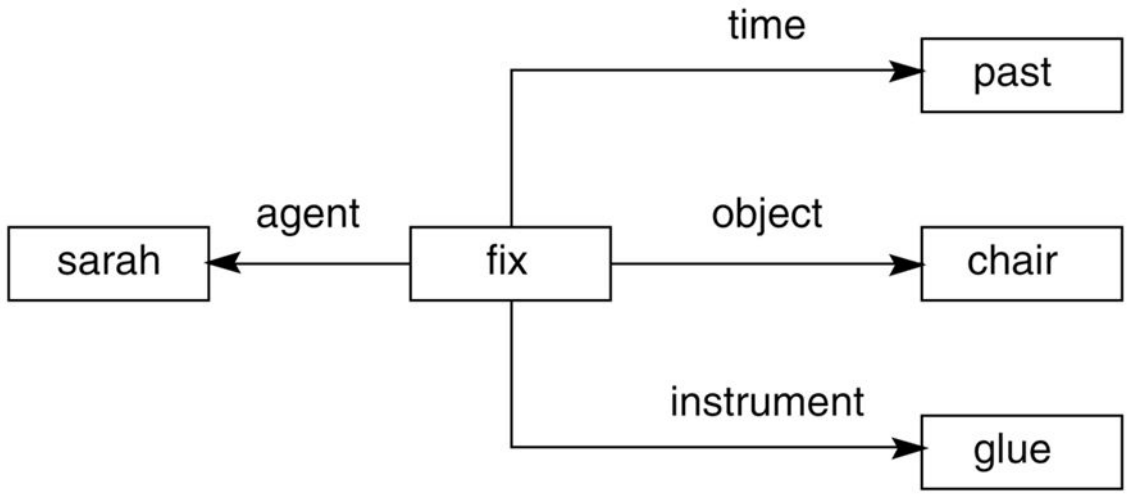
Continued…

- 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…


form of
water
hardness
soft
texture
slippery
made of
snow
color
white
temperature
cold
snowman
instance of
frosty
temperature
ice
form of
texture
hardness
hard
color
clear

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:

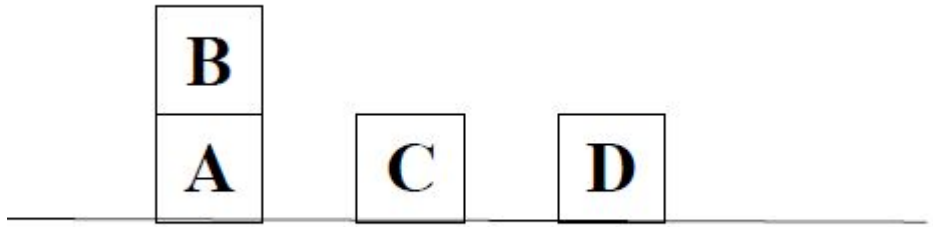
Each block can have at most 1 other block on top of it.

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

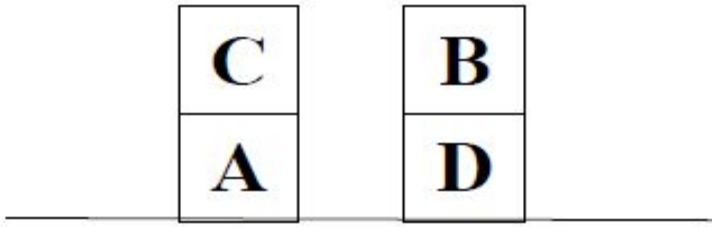
Example



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

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

Another example



ON(C, A) ∧
ONTABLE(A) ∧
ON(B, D) ∧
ONTABLE(D) ∧
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: ARMEMPTY $\wedge$ CLEAR(x) $\wedge$ ONTABLE(x)

–DELETE: ARMEMPTY $\wedge$ ONTABLE(x)

–ADD: HOLDING(x)

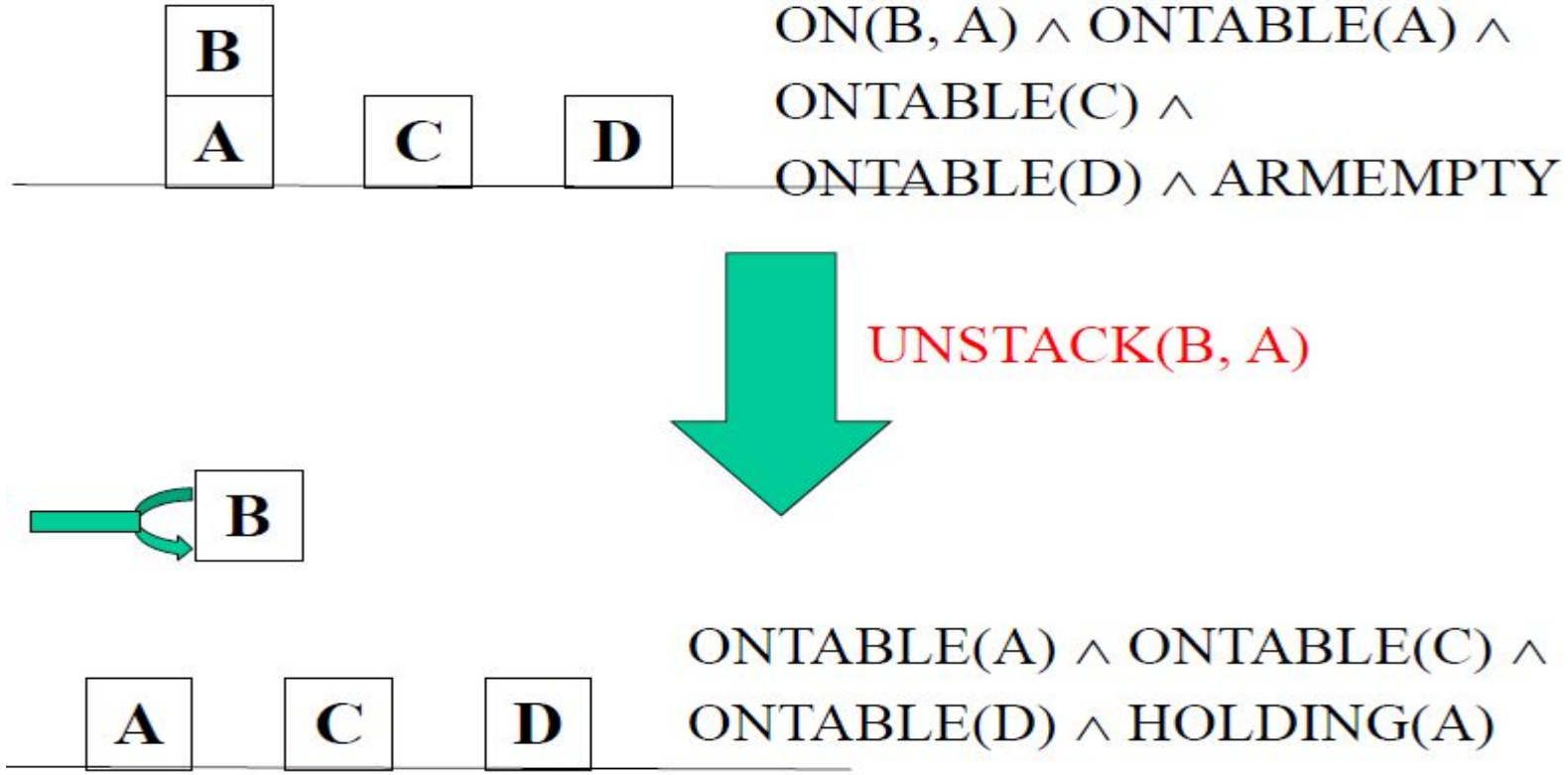
PUTDOWN(x):putdown block x on the table

–PRECONDITION: HOLDING(x)

–DELETE: HOLDING(x)

–ADD: ARMEMPTY $\wedge$ ONTABLE(x)

Continued…


B
A
C
D
ON(B, A) ∧ ONTABLE(A) ∧
ONTABLE(C) ∧
ONTABLE(D) ∧ ARMEMPTY
UNSTACK(B, A)
B
A
C
D
ONTABLE(A) ∧ ONTABLE(C) ∧
ONTABLE(D) ∧ HOLDING(A)

The Planning Problem

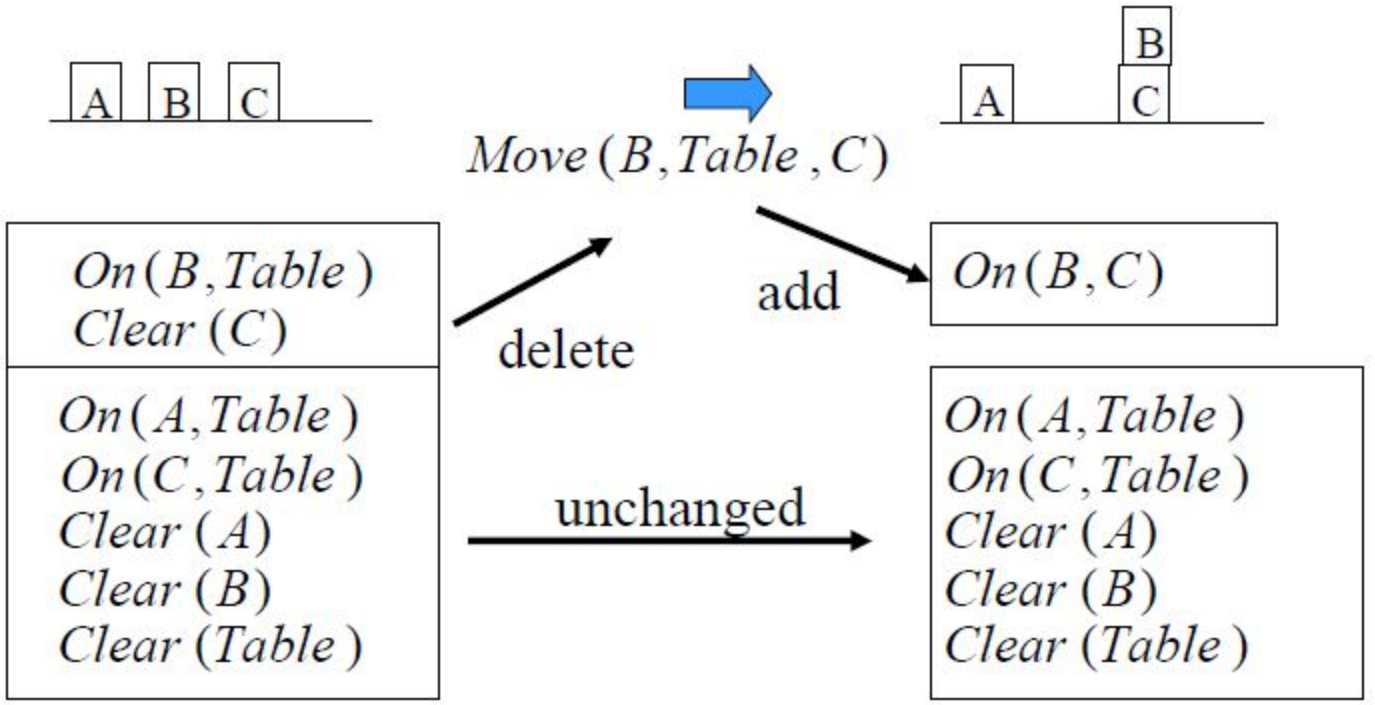
Given an initial state S0, and a final (goal) state S1,

identify the “best” **sequence**of actions that will transform S0 into S1.

Each action can only be one of: UNSTACK, STACK, PICKUP, PUTDOWN.

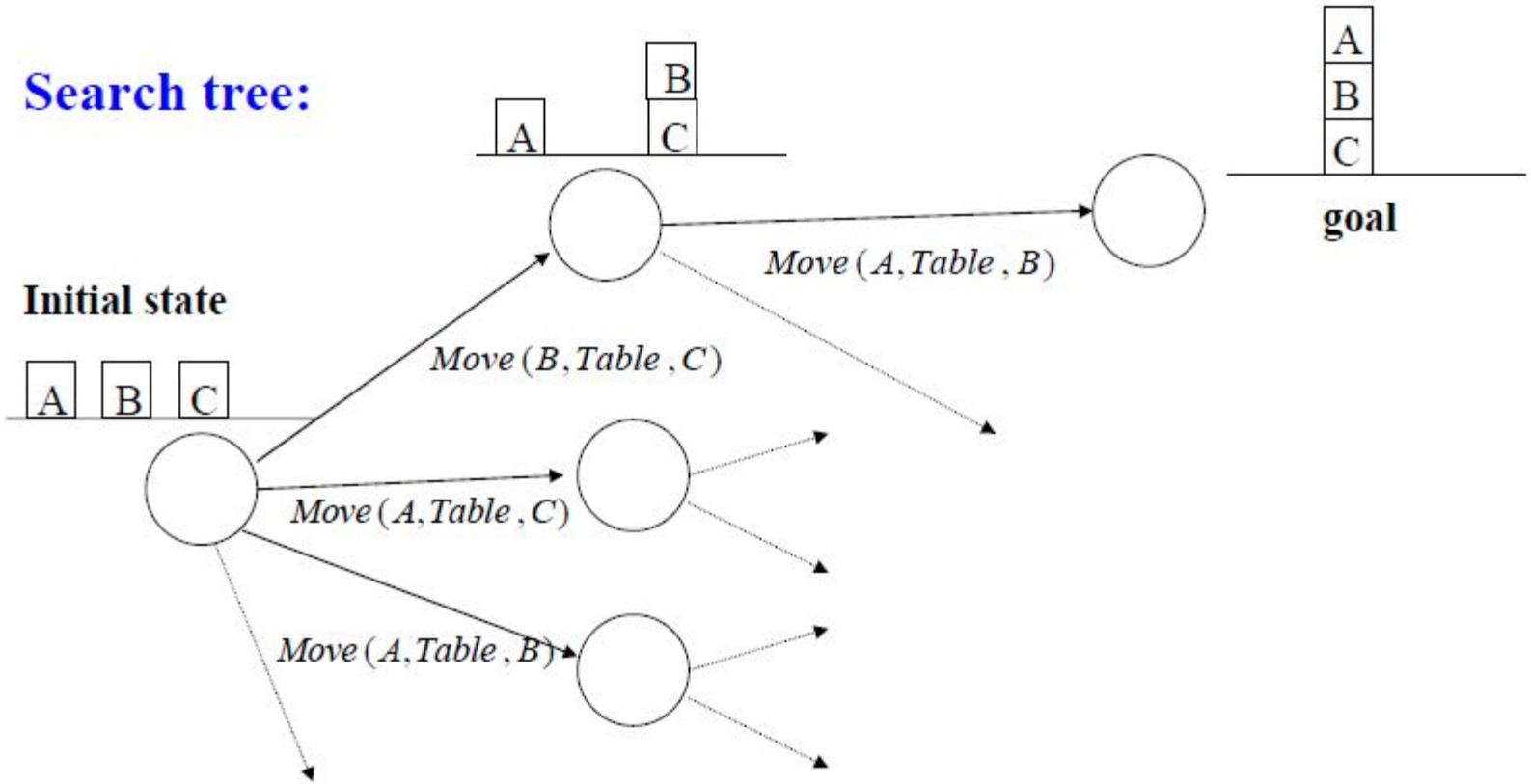
Forward search (goal progression)

- **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



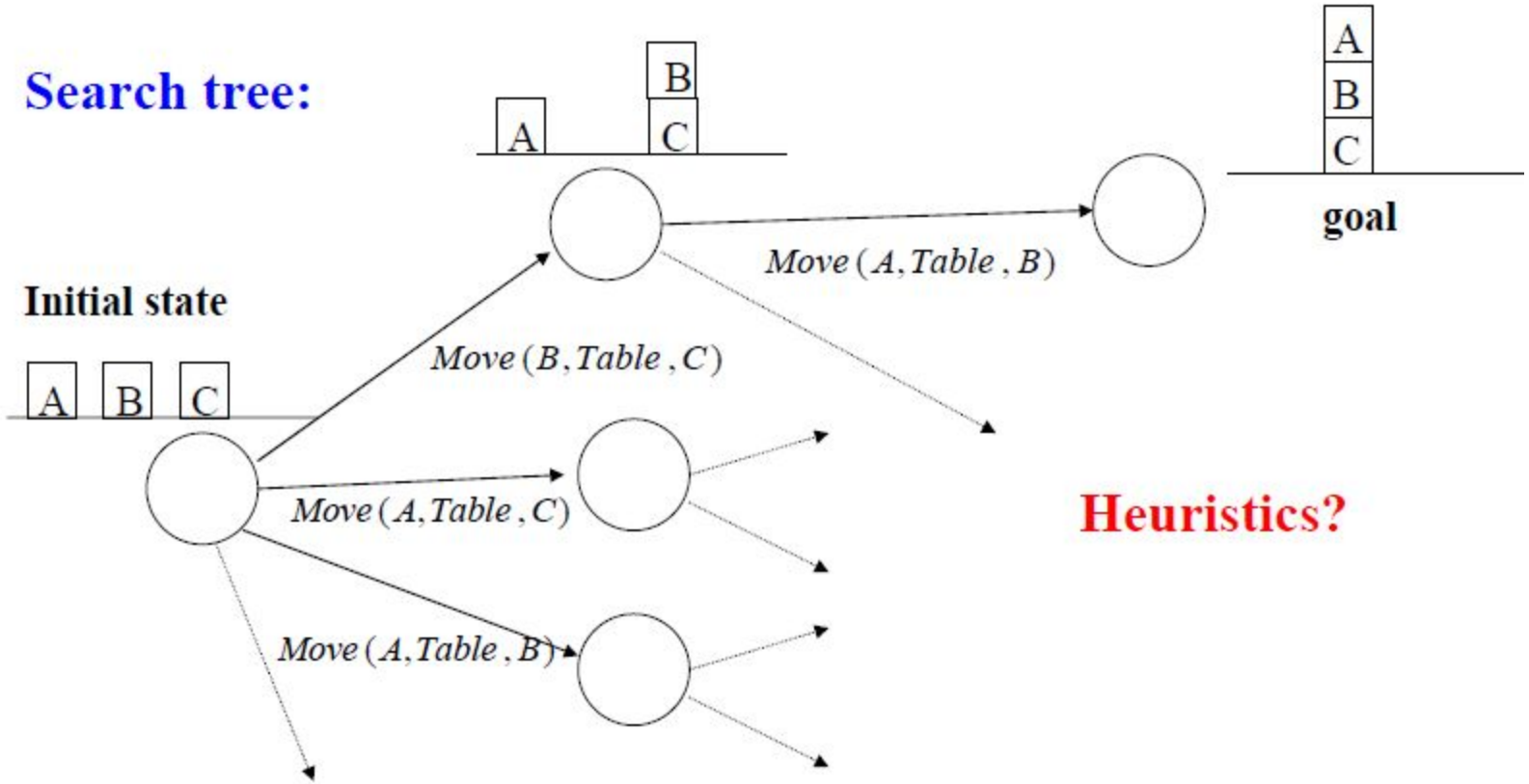
Forward search (goal progression)

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



Forward search (goal progression)

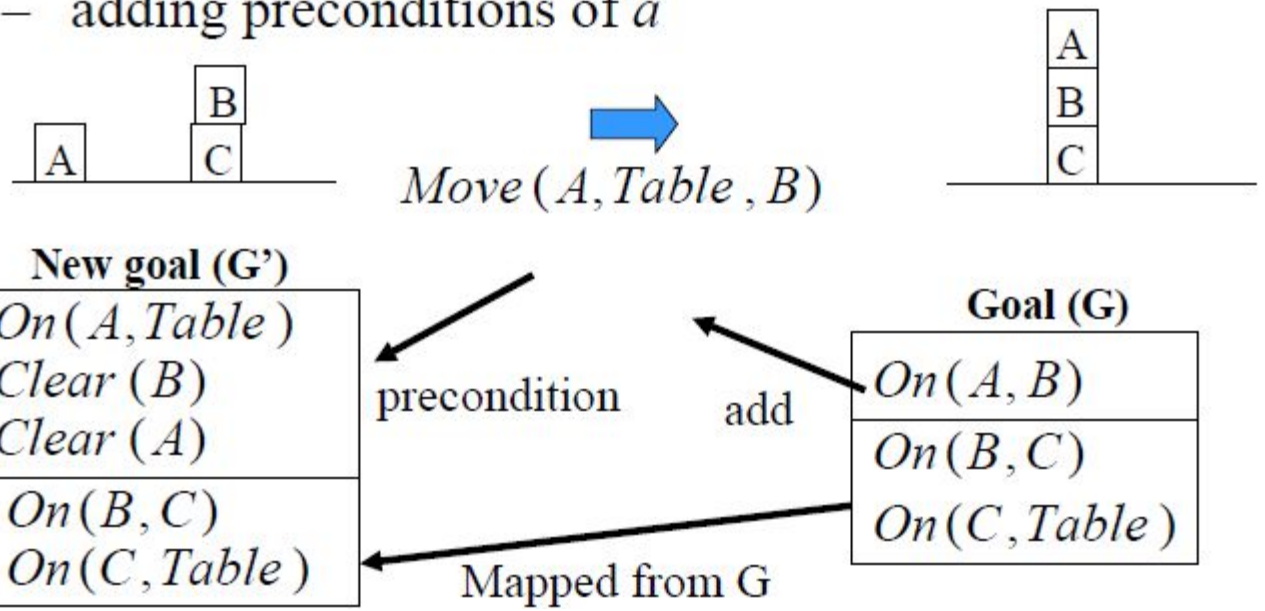
- Use operators to generate new states to search
- Check new states whether they satisfy the goal



Backward search (goal regression)

Idea: Given a goal G

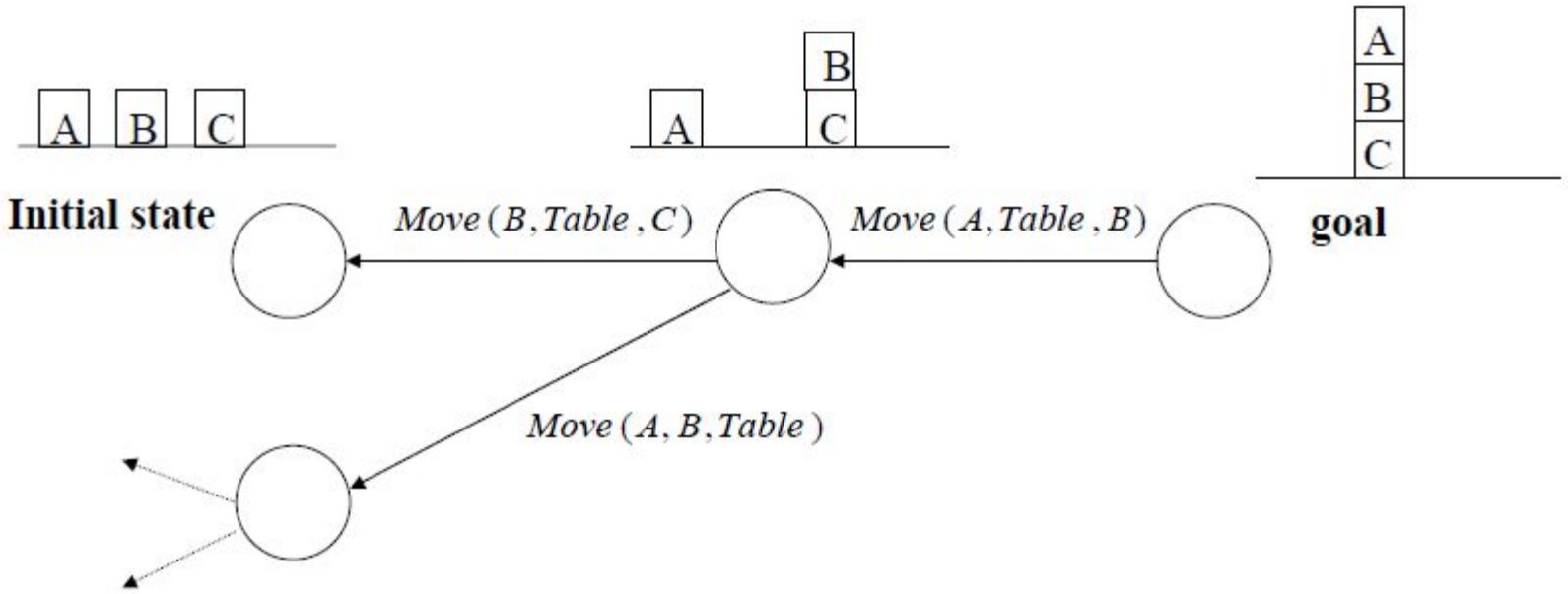
- Unify the add list of some operator a with a subset of G
- If the delete list of a does not remove elements of G, then the goal regresses to a new goal G' that is obtained from G by:
  - deleting add list of a
  - adding preconditions of a



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 initital 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

Π:= φ // 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 □

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)

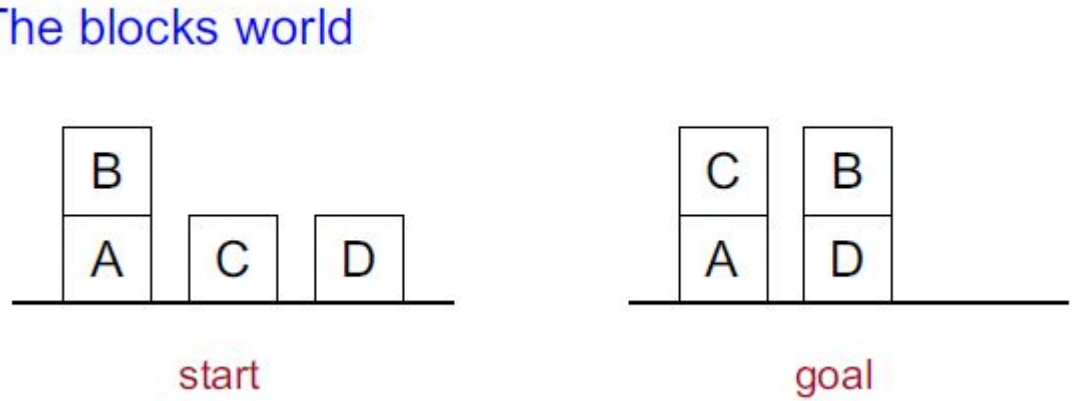
holding(x)

clear(y)

R4: unstack(x,y)

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

Continued...



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

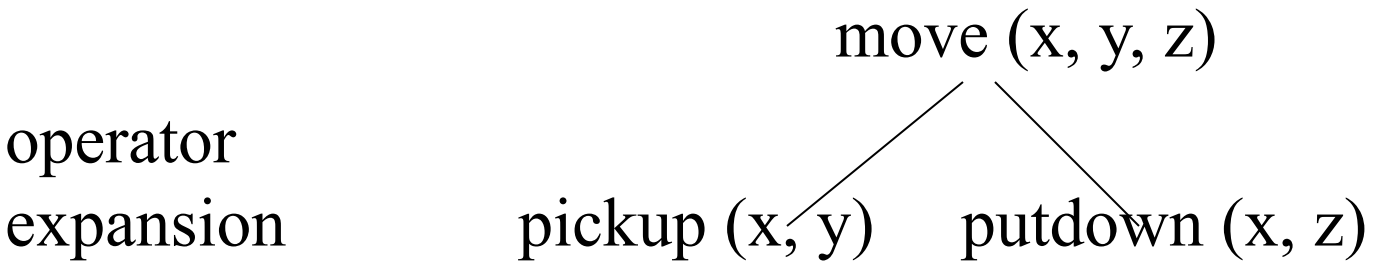
- planning starts with complex action on top
- plan constructed through action decomposition
- substitute complex action with plan of less complex actions (pre-defined plan schemata; or learning of plans/plan abstraction)
- overall plan must generate effect of complex action

Continued...

Hierarchical Planning / Plan Decomposition

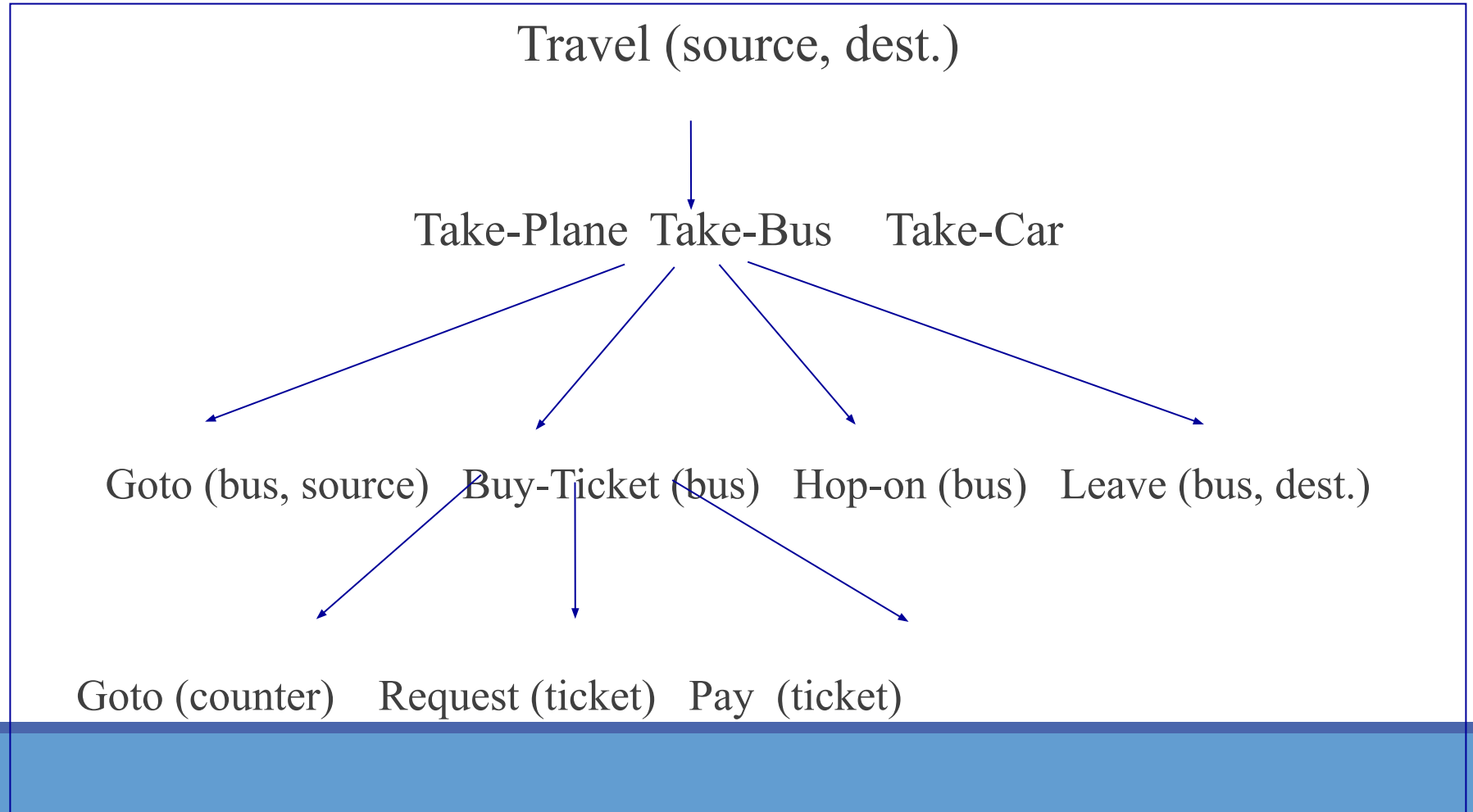
Plans are organized in a hierarchy. Links between nodes at different levels in the hierarchy denote a decomposition of a “complex action” into more primitive actions (operator expansion).

Example:

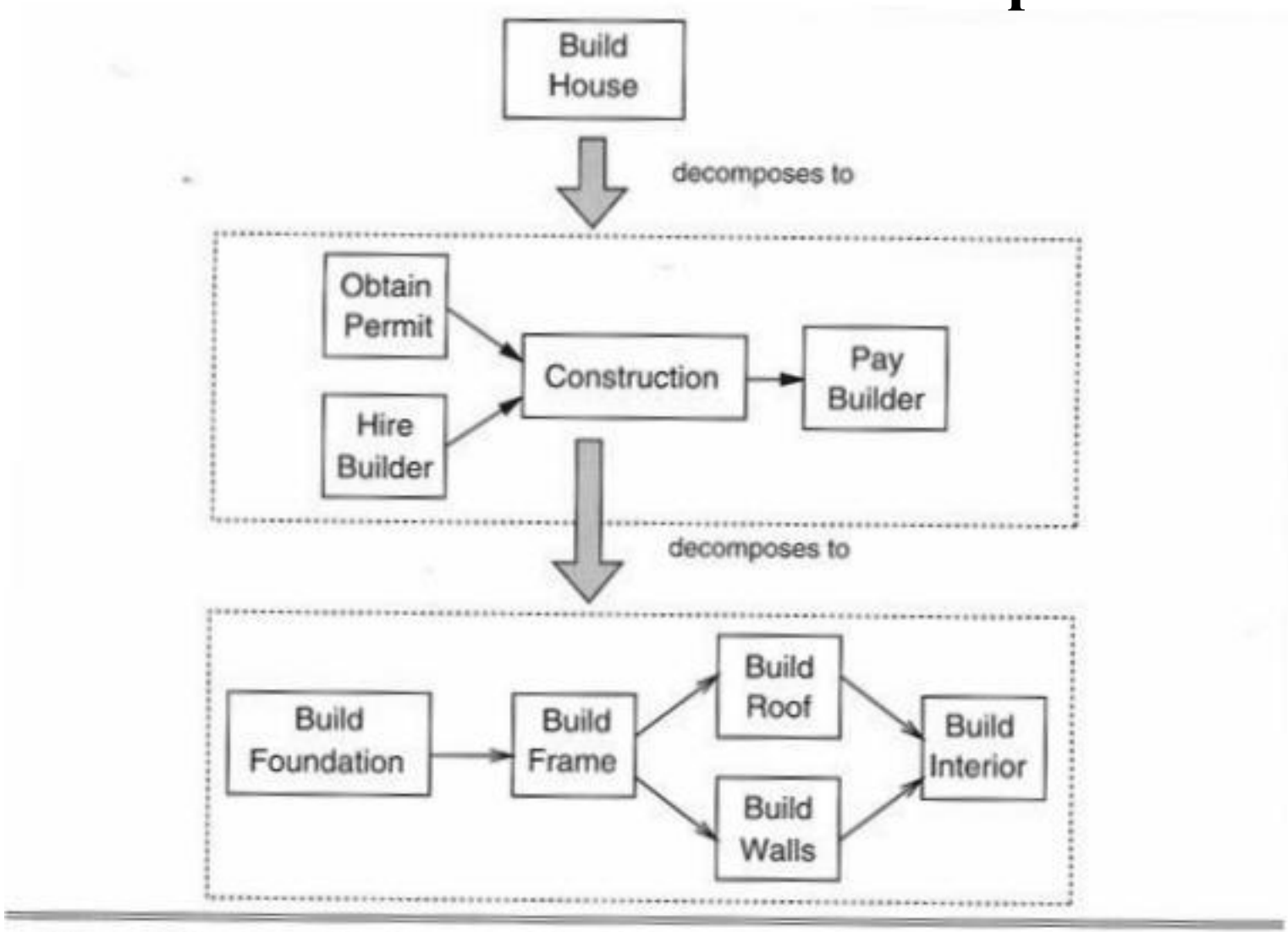


The lowest level corresponds to executable actions of the agent.

Hierarchical Plan - Example

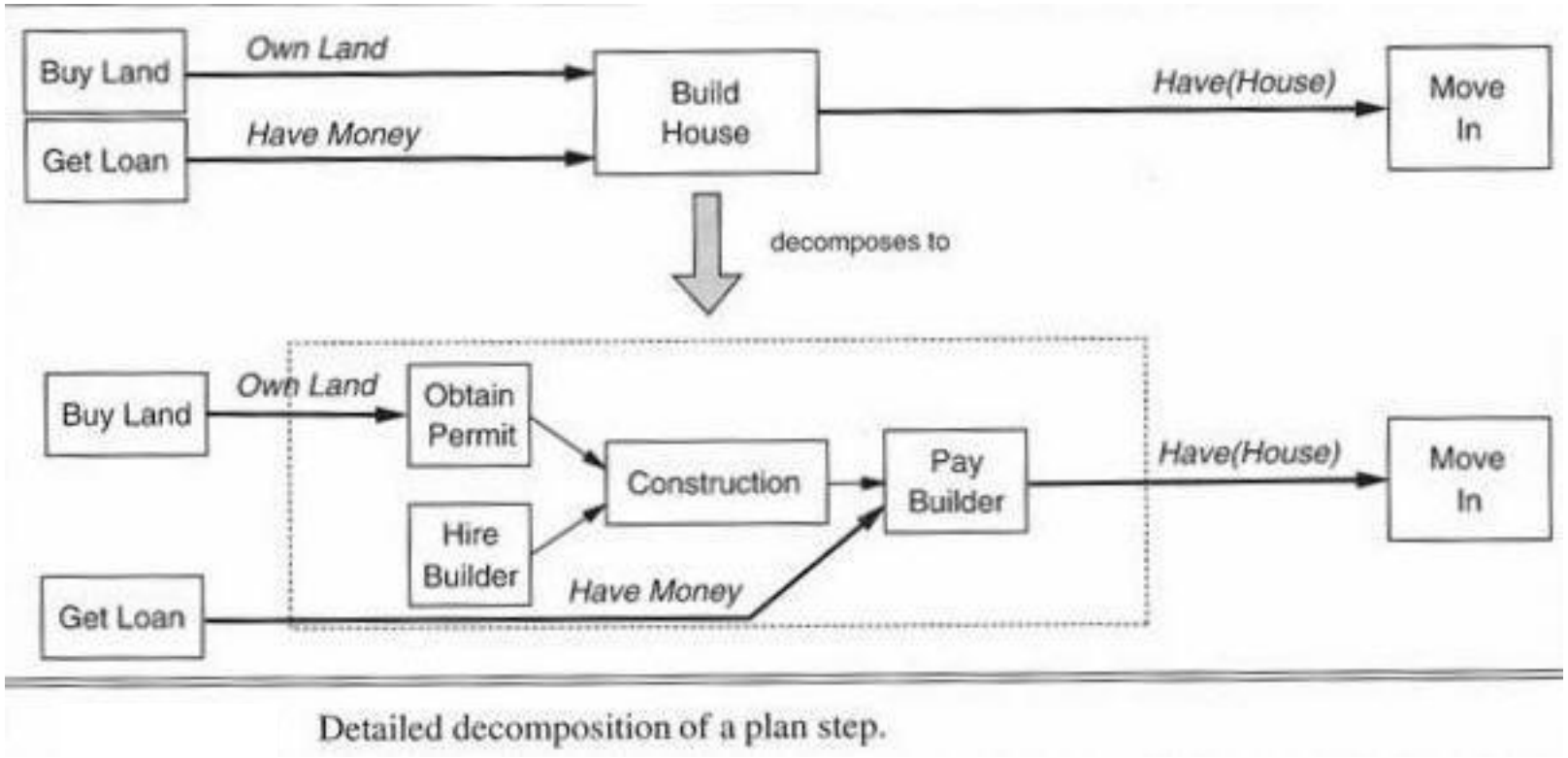


Hierarchical Plan - Example



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

Hierarchical Plan - Example



Detailed decomposition of a plan step.

MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

THANK YOU FOR LISTENING ANY QUESTION?