Artificial Intelligence

Artificial Intelligence

- AI is one of the newest sciences.
- Work started in earnest soon after World War **II,** and the name itself was coined in 1956, Along with molecular biology.
- **AI** is regularly cited as the "field I would most like to be in" by scientists in other disciplines.
- **Definitions of artificial**
- intelligence according to eight textbooks are shown in Figure 1.1. These definitions vary along two main dimensions.

- Roughly, the ones on top are concerned with *thought processes* and *reasoning,* whereas the ones on the bottom address *behavior.*
- The definitions on the left measure success in terms of fidelity to *human* performance, whereas the ones on the right

Systems that think like humans	Systems that think rationally
"The exciting new effort to make computers think . . . *machines with minds,* in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Chamiak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Systems that act like humans	**Systems that act rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI ...is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 1.1 Some definitions of artificial intelligence, organized into four categories.

Thinking humanly: The Cognitive Modeling

- Reasons like humans do
  - Programs that behave like humans
- Requires understanding of the internal activities of the brain
  - see how humans behave in certain situations and see if you could make computers behave in that same way.
  - **Protocol analysis** (think aloud) can help in understanding how human beings solve a given problem

Example. write a program that plays chess.

- Instead of making the best possible chess-playing program, you would make one that play chess like people do.

Acting humanly: The Turing Test

Can machines act like human do? Can machines behave intelligently?

- Turing Test: Operational test for intelligent behavior
  - do experiments on the ability to achieve human-level performance,
  - Acting like humans requires AI programs to interact with people
- Suggested major components of AI: knowledge, reasoning, language understanding, learning

How to make computers act like humans?

The following sub-fields are emerged

- **Natural Language processing** (enable computers communicate in human language, English, Amharic, ..)
- **Knowledge representation** (schemes to store information, both facts and inferences, before and during interrogation)
- **Automated reasoning** (use stored information to answer questions and to draw new conclusions)
- **Machine learning** (adapt to new circumstances and accumulate knowledge)
- **Computer vision** (recognize objects based on patterns in the same way as the human visual system does)
- **Robotics** (produce mechanical device capable of controlled motion; which enable computers to see, hear & take actions)
- Is AI equals human intelligence ?

Thinking Rationally: The Laws of Thought

- A system is rational if it thinks/does the right thing through correct reasoning.

- **Aristotle**: provided the correct arguments/ thought structures that always **gave correct conclusions given correct premises**.
  - Abebe is a man; all men are mortal; therefore Abebe is mortal
  - These Laws of thought governed the operation of the mind and initiated the field of **Logic**.

Acting rationally: The rational agent

- **Doing the right thing** so as to achieve one's goal, given one's beliefs.
- **AI** is the study and construction of rational agents (an agent that perceives and acts)
- Rational action requires the ability to represent knowledge and reason with it so as to reach good decision.
  - Learning for better understanding of how the world works

History of AI

- Formally initiated in 1956 and the name AI was coined by John McCarthy.
- The advent of general purpose computers provided a vehicle for creating artificially intelligent entities.
  - Used for solving general-purpose problems
- Which one is preferred?
  - General purpose problem solving systems
  - Domain specific systems

History of AI

- Development of knowledge-based systems: the key to power
  - Performance of general-purpose problem solving methods is weak for many complex domains.
  - Use knowledge more suited to make better reasoning in narrow areas of expertise (like human experts do).
  - Early knowledge intensive systems include:
    - The **Dendral program** (1969): solved the problem of inferring molecular structure ($C_6H_{13}NO_2$).
    - **MYCIN** (1976): used for medical diagnosis.
    - etc.

History of AI

- Shifts from procedural to declarative programming paradigm.
  - Rather than telling the computer how to compute a solution, a program consists of a knowledge base of facts and relationships.
  - Rather than running a program to obtain a solution, the user asks question so that the system searches through the KB to determine the answer.

- Simulate human mind and learning behavior (Neural Network, Belief Network, Hidden Markov Models, etc.)

Intelligent Agent

- I want to build a robot that will
  - Clean my house
  - Information filtering agents
  - Cook when I don't want to
  - Wash my clothes
  - Cut my hair
  - Fix my car (or take it to be fixed)
  - Take a note when I am in a meeting
  - Handle my emails

  i.e. do the things that I don't feel like doing…

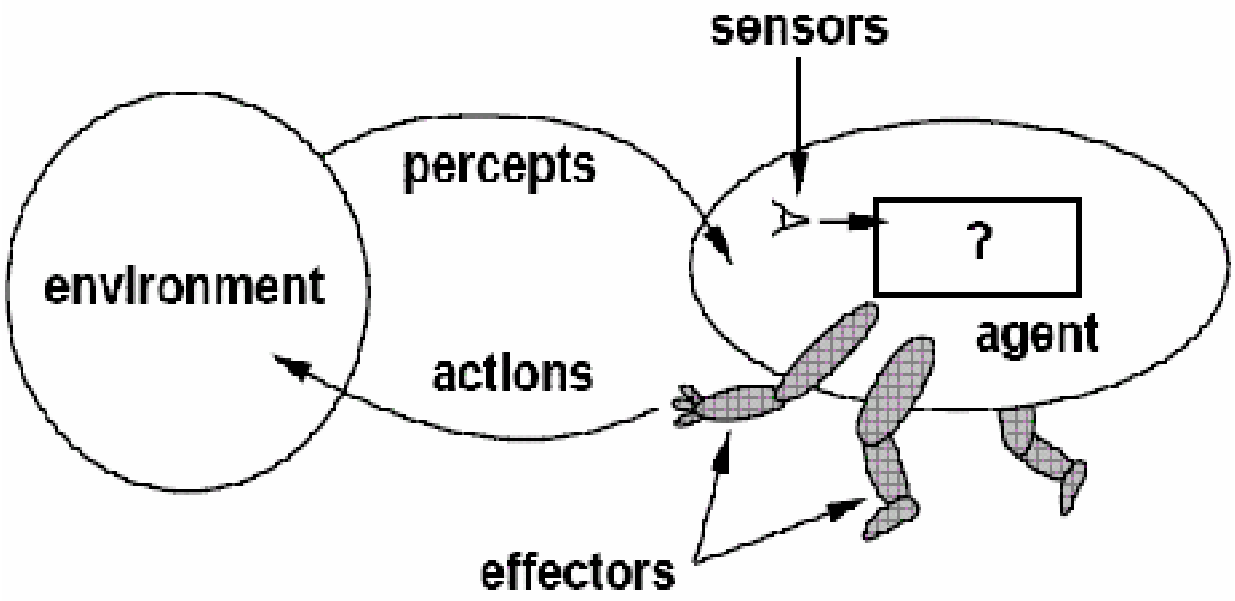
- AI is the science of building machines (agents) that act rationally with respect to a goal.

Strong AI vs. Weak AI

- Artificial Intelligence (or AI) is the concept that it is possible for a computer to think in the same sense as humans do.
- Strong AI argues that it is possible that one day a computer will be invented which can be called a mind in its fullest sense.
  - Strong AI aims to create an agent that can replicate humans intelligence completely; i.e.., it can think, reason, imagine, etc., and do all the things that we currently associate with the human brain.
- Weak AI, on the other hand, argues that computers can only appear to think and are not actually conscious in the same way as human brains are.
  - The weak AI position holds that AI should try to develop systems which have facets of intelligence, but the objective is not to build a completely sentient entity. For example, weak AI researchers see their contribution as things like expert systems used for medical diagnosis, speech recognition and data mining, which use "intelligent" models, but they do not help create a conscious agent

Agent

- Agent is something that perceives its environment through **SENSORS** and acts upon that environment through **EFFECTORS**.
- The agent is assumed to exist in an environment in which it perceives and acts
- An agent is rational since it does the right thing to achieve the specified goal.



Agent

	Human beings	Agents
Sensors	Eyes, Ears, Nose	Cameras, Scanners, Mic, infrared range finders
Effectors	Hands, Legs, Mouth	Various Motors (artificial hand, artificial leg), Speakers, Radio

Examples of agents in different types of applications

Agent type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, patient's answers	Questions, tests, treatments	Healthy patients, minimize costs	Patient, hospital
Interactive English tutor	Typed words, questions, suggestions	Write exercises, suggestions, corrections	Maximize student's score on exams	Set of students, materials
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belts with parts
Satellite image analysis system	Pixels intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery

Rationality vs. Omniscience

- Rational agent acts so as to achieve one's goals, given one's beliefs (one that does the right thing).
  - What does right thing mean? one that will cause the agent to be most successful and is expected to maximize goal achievement, given the available information
- An Omniscient agent knows the actual outcome of its actions, and can act accordingly, but in reality omniscience is impossible.
- Rational agents take action with expected success, where as omniscient agent take action with 100% sure of its success
- Is human beings Omniscient or Rational agent?

Example

- You are walking along the road to Giorgis; You see an old friend across the street. There is no traffic.
- So, being rational, you start to cross the street.
- Meanwhile a big banner falls off from above and before you finish crossing the road, you are flattened.

Were you irrational to cross the street?

- This points out that rationality is concerned with expected success, given what has been perceived.
  - Crossing the street was rational, because most of the time, the crossing would be successful, and there was no way you could have foreseen the falling banner.
  - The EXAMPLE shows that we can not blame an agent for failing to take into account something it could not perceive. Or for failing to take an action that it is incapable of taking.

Rational agent

- In summary what is rational at any given point depends on four things.
  - **Perception**: Everything that the agent has perceived so far concerning the current scenario in the environment
  - **Knowledge**: What an agent already knows about the environment
  - **Action**: The actions that the agent can perform back to the environment
  - **Performance measure**: The performance measure that defines degrees of success of the agent

- Therefore in designing an intelligent agent, one has to remember PEAS (Performance, Environment, Actuators, Sensors) framework.

Performance measure

- How do we decide whether an agent is successful or not?
  - Establish a standard of what it means to be successful in an environment and use it to measure the performance
  - A rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
- What is the performance measure for "**crossing the road**"?
- What about "**Chess Playing**"?

Assignment 1 Due Date August 10/2009

- Consider the need to design a “taxi driver agent” that serves in Addis Ababa City;
  - Identify **sensors, effectors, goals, environment** and **performance measure** that should be integrated for the agent to be successful in its operation?
  - Note: The taxi driver needs to know where it is? What else is on the road? How fast it has to drive? How to communicate with the passengers and other vehicles?

Designing an agent

- A physical agent has two parts: architecture + program
- Architecture
  - Runs the programs
  - Makes the percept from the sensors available to the programs
  - Feeds the program's action choices to the effectors
- Programs
  - Accepts percept from an environment and generates actions
    - Before designing an agent program, we need to know the possible percept and actions
  - By enabling a learning mechanism, the agent could have a degree of autonomy, such that it can reason and take decision

Program Skeleton of Agent

function SKELETON-AGENT (*percept*) **returns** action
static: *knowledge*, the agent's memory of the world

knowledge← UPDATE-KNOWLEDGE(*knowledge*,*percept*)
action ← SELECT-BEST-ACTION(*knowledge*)
knowledge← UPDATE-KNOWLEDGE (*knowledge*, *action*)
return *action*

On each invocation, the agent's knowledge base is updated to reflect the new percept, the best action is chosen, and the fact that the action taken is also stored in the knowledge base. The knowledge base persists from one invocation to the next.

NOTE: Performance measure is not part of the agent

Classes of Environments

- Actions are done by the agent on the environment. Environments provide percepts to an agent.
- Agent perceives and acts in an environment. Hence in order to design a successful agent , the designer of the agent has to understand the type of the environment it interacts with.
- Properties of Environments:
  - Fully observable vs. partially observable
  - Deterministic vs. stochastic
  - Episodic vs. non-episodic
  - Static vs. Dynamic
  - Discrete vs. continuous

Fully observable vs. partially observable

- Does the agent's sensory see the complete state of the environment?
  - If an agent has access to the complete state of the environment, then the environment is accessible or fully observable.
- An environment is effectively accessible if the sensors detect all aspects that are relevant to the choice of action.

- Taxi driving is partially observable
  - Any example of fully observable?

Deterministic vs. stochastic

- Is there a unique mapping from one state to another state for a given action?
- The environment is deterministic if the next state is completely determined by
  - the current state of the environment and
  - the actions selected by the agents.
- Taxi driving is non-deterministic (i.e. stochastic)
  - Any example of deterministic?

Episodic vs. Sequential

- Does the next “episode” depend on the actions taken in previous episodes?
- In an episodic environment, the agent's experience is divided into "episodes".
  - Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself.
- In sequential environment the current decision could affect all future decisions

- Taxi driving is sequential
  - Any example of Episodic?

Static vs. Dynamic

- Can the world change while the agent is thinking?
  - If the environment can change while the agent is on purpose, then we say the environment is dynamic for that agent
  - otherwise it is static.
- Taxi driving is dynamic
  - Any example of static?

Discrete vs. Continuous

- Are the distinct percepts & actions limited or unlimited?
  - If there are a limited number of distinct, clearly defined percepts and actions, we say the environment is discrete.
- Taxi driving is continuous
  - Any example of discrete?

Environment Types

Below are lists of properties of a number of familiar environments

Problems	Observable	Deterministic	Episodic	Static	Discrete
Crossword Puzzle	Yes	Yes	No	Yes	Yes
Part-picking robot	No	No	Yes	No	No
Web shopping program	No	No	No	No	Yes
Tutor	No	No	No	Yes	Yes
Medical Diagnosis	No	No	No	No	No
Taxi driving	No	No	No	No	No

- Hardest case: a environment that is inaccessible, sequential, non-deterministic, dynamic, continuous.

Five types of agents

Simple reflex agents

- works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.

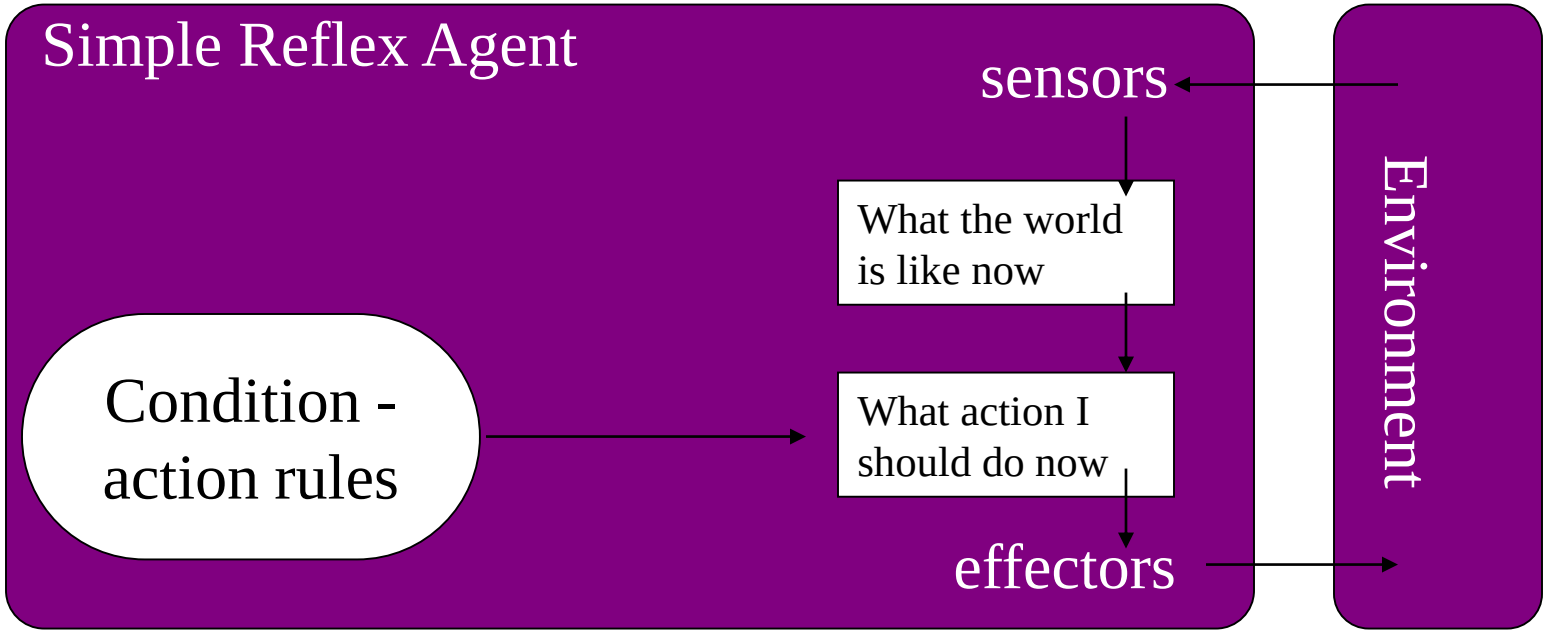
  E.g. If the car in front brakes, and its brake lights come on, then the driver should notice this and initiate braking,

  – Some processing is done on the visual input to establish the condition. If "The car in front is braking"; then this triggers some established connection in the agent program to the action "initiate braking". We call such a connection a condition-action rule written as: **If car-in-front-is breaking then initiate-braking**.

- Humans also have many such conditions. Some of which are **learned responses**. Some of which are **innate (inborn) responses**

  – Blinking when something approaches the eye.

Structure of a simple reflex agent



function SIMPLE-REFLEX-AGENT(*percept*) **returns** action
static: *rules*, a set of condition-action rules

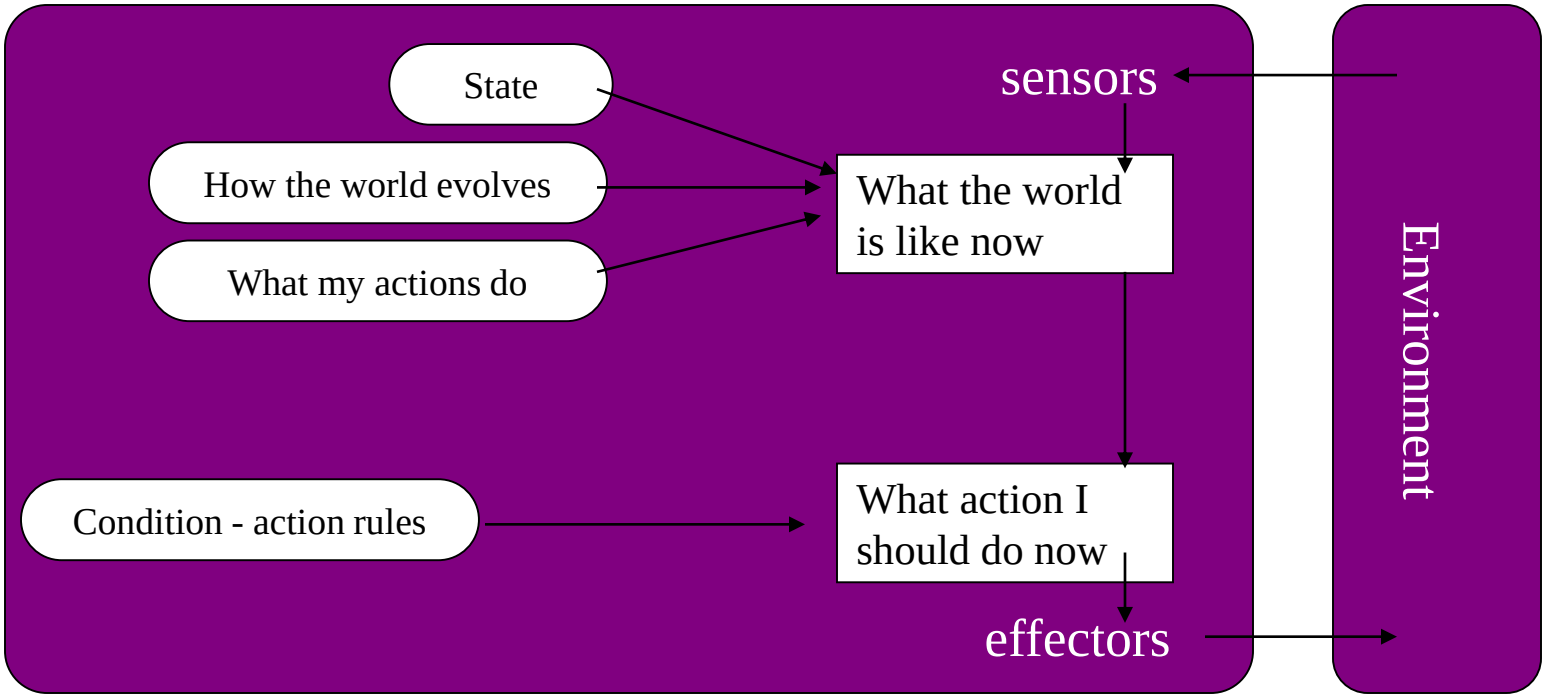
state ← INTERPRET-INPUT (*percept*)
rule ← RULE-MATCH (*state*,*rules*)
action ← RULE-ACTION [*rule*]
return *action*

Model-Based Reflex Agent

- This is a reflex agent with internal state.
  - It keeps track of the world that it can't see now.
- It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state)
  - If the car is a recent model -- there is a centrally mounted brake light. With older models, there is no centrally mounted, so what if the agent gets confused?

    Is it a parking light? Is it a brake light? Is it a turn signal light?
  - Some sort of internal state should be in order to choose an action.
  - The camera should detect two red lights at the edge of the vehicle go **ON** or **OFF** simultaneously.
- The driver should look in the rear-view mirror to check on the location of near by vehicles. In order to decide on lane-change the driver needs to know whether or not they are there. The driver sees, and there is already stored information, and then does the action associated with that rule.

Structure of Model-Based reflex agent



function REFLEX-AGENT-WITH-STATE (*percept*) **returns** action

static: *state*, a description of the current world state

rules, a set of condition-action rules

state ← UPDATE-STATE (*state*, *percept*)

rule ← RULE-MATCH (*state*, *rules*)

action ← RULE-ACTION [*rule*]

state ← UPDATE-STATE (*state*, *action*)

return *action*

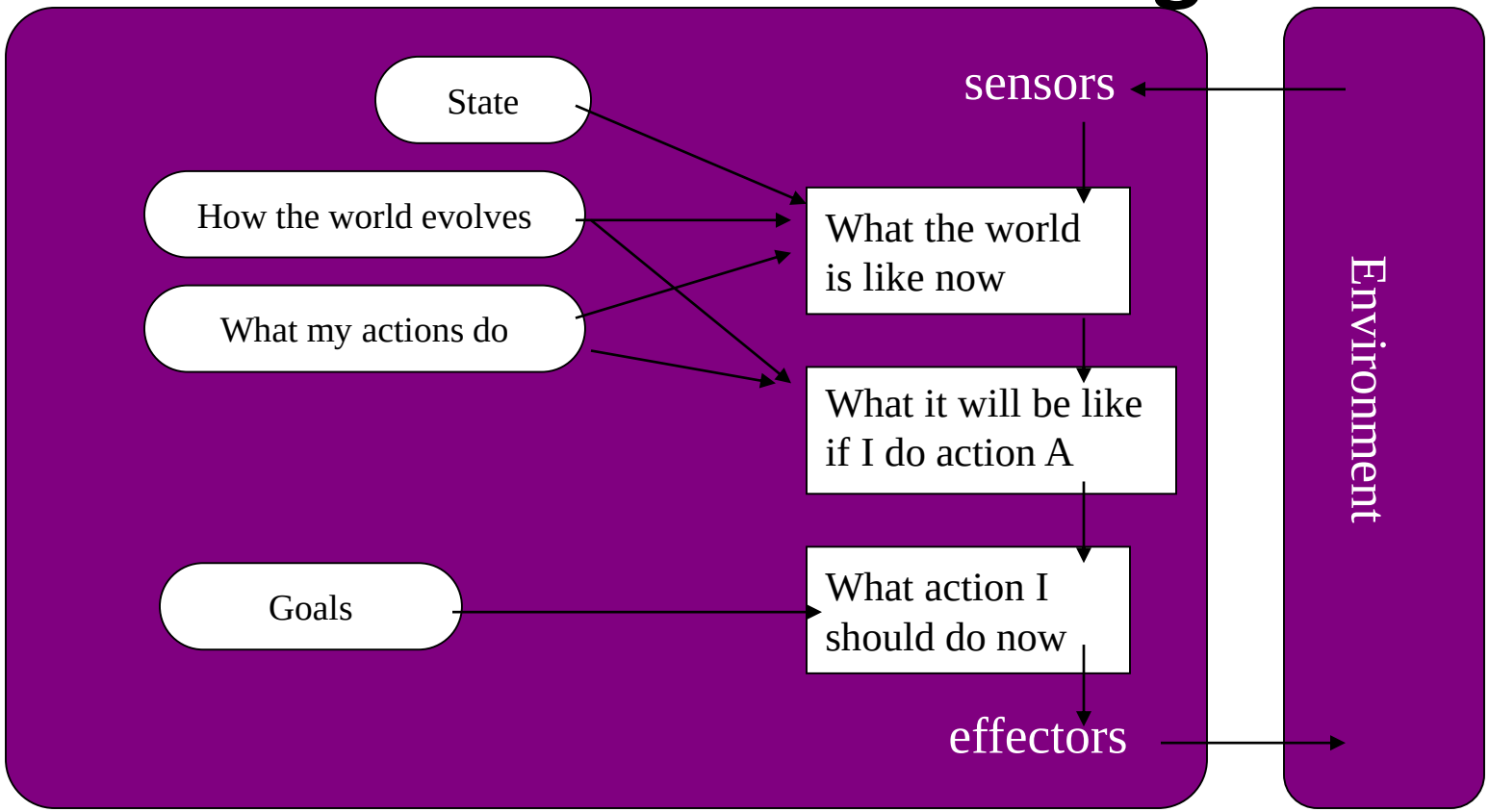
Goal based agents

- Choose actions that achieve the goal (an agent with explicit goals)
- Involves consideration of the future:
  - Knowing about the current state of the environment is not always enough to decide what to do.

For example, at a road junction, the taxi can turn left, right or go straight.

  - The right decision depends on where the taxi is trying to get to. As well as a current state description, the agent needs some sort of goal information, which describes situations that are desirable. E.g. being at the passenger's destination.
- The agent may need to consider long sequences, twists and turns to find a way to achieve a goal.

Structure of a Goal-based agent



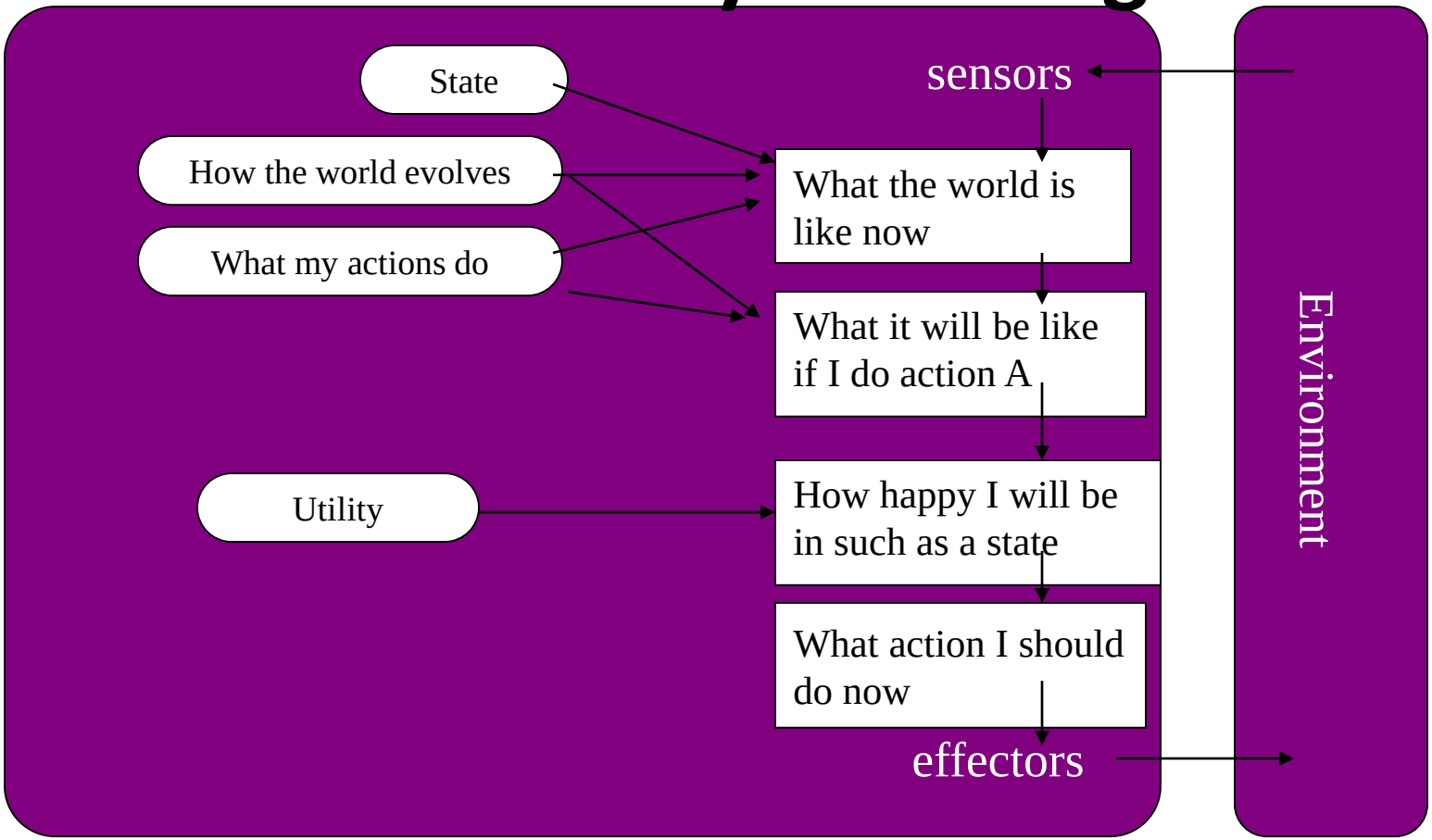
function GOAL_BASED_AGENT (*percept*) **returns** action
  state ← UPDATE-STATE (*state*, *percept*)
  action ← SELECT-ACTION [*state*, *goal*]
  state ← UPDATE-STATE (*state*, *action*)
 return *action*

Utility based agents

- Goals are not really enough to generate high quality behavior.

  For e.g., there are many action sequences that will get the taxi to its destination, thereby achieving the goal. Some are quicker, safer, more reliable, or cheaper than others. We need to consider Speed and safety
- When there are several goals that the agent can aim for, non of which can be achieved with certainty. Utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.
- An agent that possesses an explicit utility function can make rational decisions.

Structure of a utility-based agent



function UTILITY_BASED_AGENT (*percept*) **returns** action

state ← UPDATE-STATE (*state*, *percept*)
action ← SELECT-OPTIMAL_ACTION [*state*, *goal*]
state ← UPDATE-STATE (*state*, *action*)
return *action*

Problem solving

Problem

- It is a gap between what actually is and what is desired.
  - A problem exists when an individual becomes aware of the existence of an obstacle which makes it difficult to achieve a desired goal or objective.
- A number of problems are addressed in AI, both:
  - **Toy problems**: are problems that are useful to test and demonstrate methodologies.
    - Can be used by researchers to compare the performance of different algorithms
    - e.g. 8-puzzle, n-queens, vacuum cleaner world, towers of Hanoi, …
  - **Real-life problems:** are problems that have much greater commercial/economic impact if solved.
    - Such problems are more difficult and complex to solve, and there is no single agreed-upon description
    - E.g. Route finding, Traveling sales person, etc.

Solving a problem

Formalize the problem: Identify the collection of information that the agent will use to decide what to do.

- Define states
  - States describe distinguishable stages during the problem-solving process
  - Example- What are the various states in **route finding** problem?
    - The various places including the location of the agent
- Define the available operators/rules for getting from one state to the next
  - Operators cause an action that brings transitions from one state to another by applying on a current state
- Suggest a suitable representation for the **problem space/state space**
  - Graph, table, list, set, … or a combination of them

State space of the problem

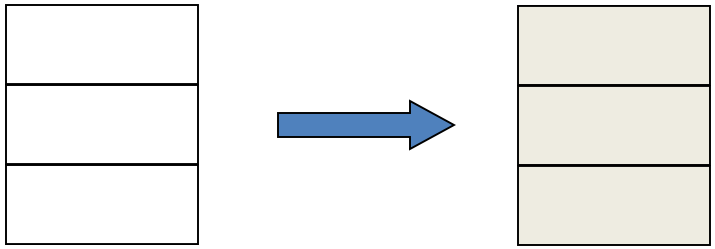
- The **state space** defines the set of all relevant states reachable from the initial state by (any) sequence of actions through iterative application of all permutations and combinations of operators
- **State space** (also called **search space/problem space)** of the problem includes the various states
  - Initial state: defines where the agent starts or begins its task
  - Goal state: describes the situation the agent attempts to achieve
  - Transition states: other states in between initial and goal states

Example – Find the state **space** for route finding problem where the agent wants to go from **Poly-Campus** to Peda-Campus.

- Think of the states reachable from the initial state until we reach to the goal state.

Example: **Coloring problem**

– There are 3 rectangles. Both are initially white. The problem is to change all rectangles with white color to black color. Color change is one rectangle at a time. Define state space for coloring problem?



The 8 puzzle problem

- Arrange the tiles so that all the tiles are in the correct positions. You do this by moving tiles or space. You can move a tile/space up, down, left, or right, so long as the following conditions are met:
  A) there's no other tile blocking you in the direction of the movement; and

  B) you're not trying to move outside of the boundaries/edges.

1	2	3
8	4	5
7	6	

1	2	3
8		4
7	6	5

Assignment 2 August 20

Missionary-and-cannibal problem:

Three missionaries and three cannibals are on one side of a river that they wish to cross. There is a boat that can hold one or two people. Find an action sequence that brings everyone safely to the opposite bank (i.e. Cross the river). But you must never leave a group of missionaries outnumbered by cannibals on the same bank (in any place).

1. Identify the set of states and operators
2. Show using suitable representation the state space of the problem

Knowledge and types of problems

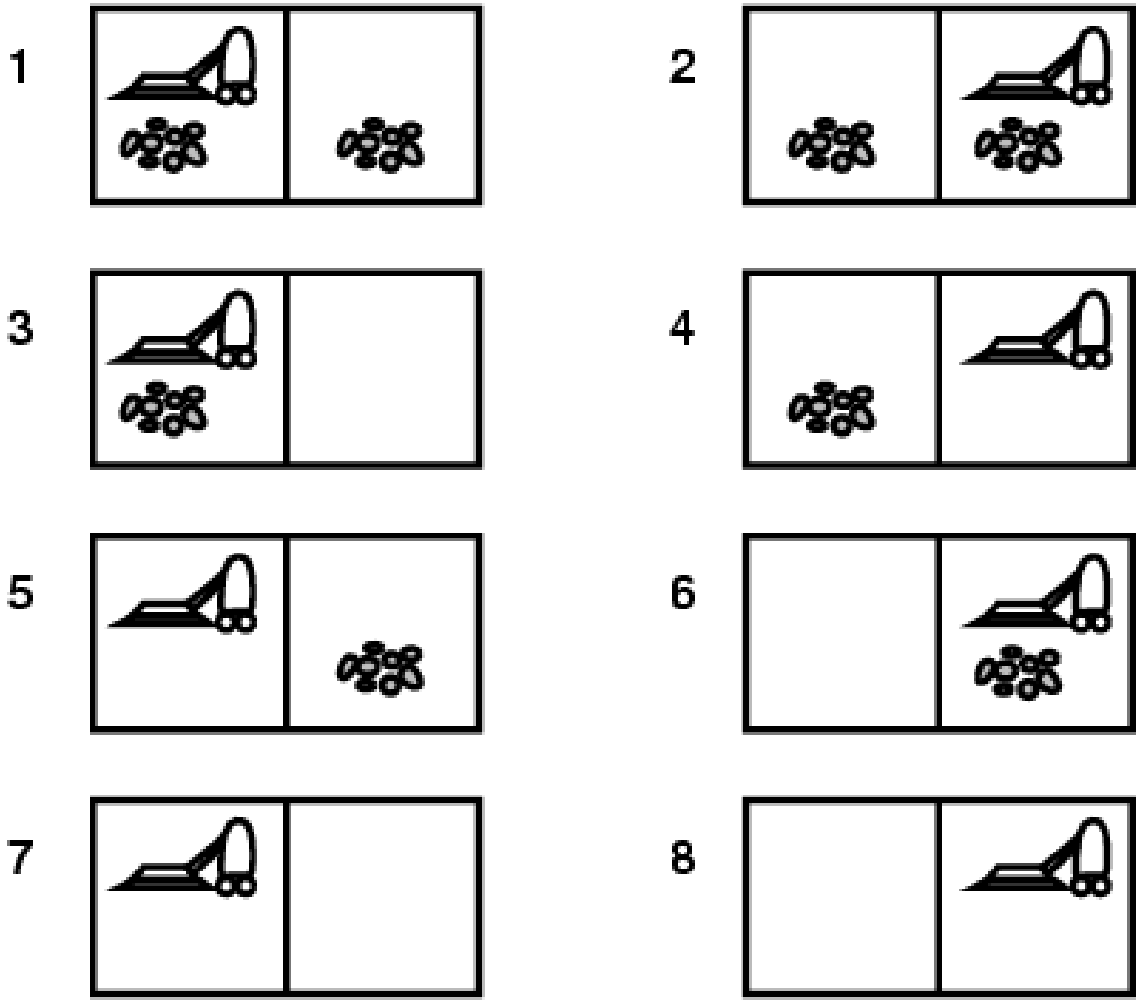
- There are **four types** of problems:
  - Single state problems (Type 1)
  - Multiple state problems (Type 2)
  - Exploration problems (Type 3)
  - Contingency Problems (Type 4)
- This classification is based on the level of knowledge that an agent can have concerning its action and the state of the world
- Thus, the first step in formulating a problem for an agent is to see what knowledge it has concerning
  - The effects of its action on the environment
  - Accessibility of the state of the world
- This knowledge depends on how it is connected to the environment via its percepts and actions

Example: Vacuum world problem

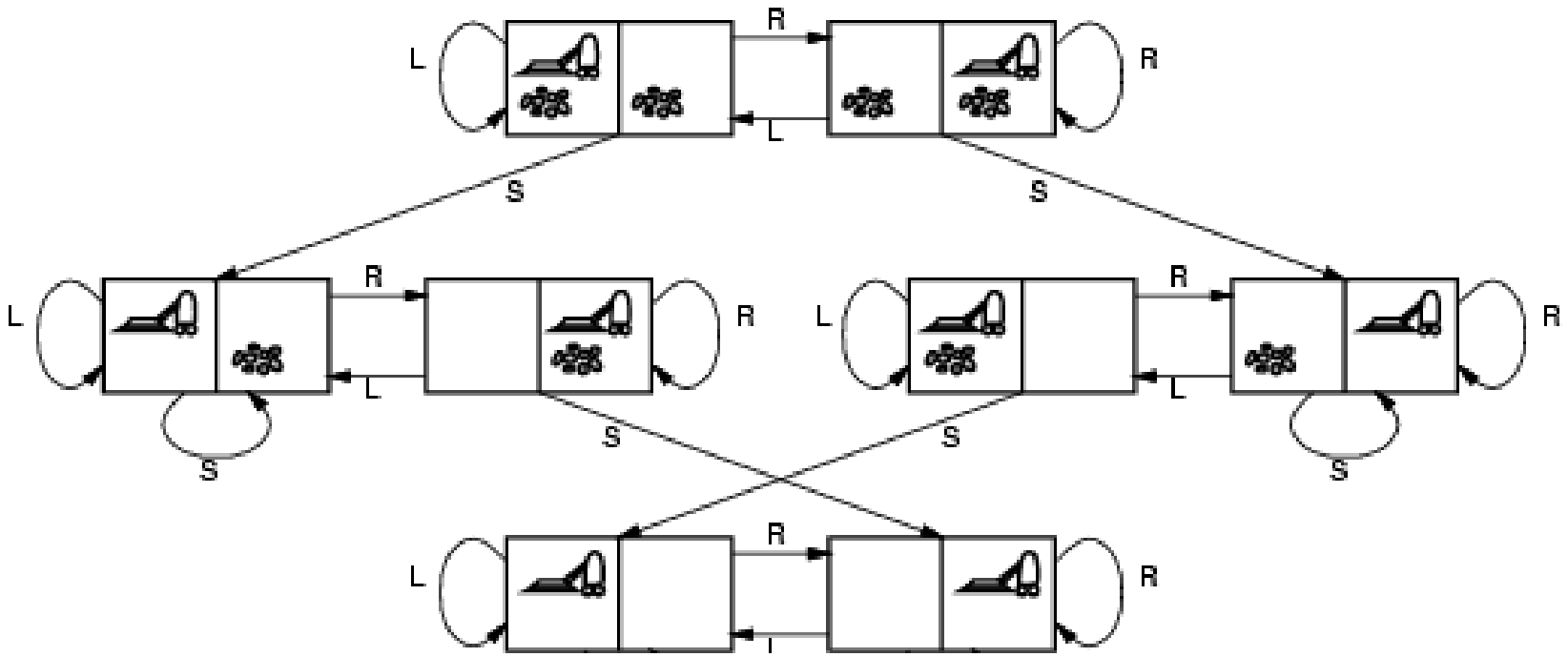
To simplify the problem (rather than the full version), let;

- The world has only **two** *locations*
  - *Each location may or may not contain dirt*
  - The agent may be in one location or the other
- Eight possible *world states*
- Three possible actions (*Left, Right, Suck)*
  - **Suck** operator clean the dirt
  - **Left** and **Right** operators move the agent from location to location
- *Goal*: to clean up all the dirt

Clean House Task



Vacuum Cleaner state Space



Single state problem

- **Fully observable:** The world is accessible to the agent
  - It can determine its exact state through its sensors
  - The agent's sensor knows which state it is in
- **Deterministic:** The agent knows exactly the effect of its actions
  - It can then calculate exactly which state it will be in after any sequence of actions
- Action sequence is completely planned

Example - **Vacuum cleaner world**

- What will happen if the agent is initially at state = 5 and formulates action sequence - [Right, Suck]?
- Agent calculates and knows that it will get to a goal state
  - Right → {6}
  - Suck → {8}

Multiple state problems

- **Partially observable:** The agent has limited access to the world state
  - It might not have sensors to get full access to the environment states or as an extreme, it can have no sensors at all (due to lack of percepts)
- **Deterministic:** The agent knows exactly what each of its actions do
  - It can then calculate which state it will be in after any sequence of actions
- If the agent has full knowledge of how its actions change the world, but does not know of the state of the world, it can still solve the task

Example - **Vacuum cleaner world**

- Agent's initial state is one of the 8 states: {1,2,3,4,5,6,7,8}
- Action sequence: {right, suck, left, suck}
- Because agent knows what its actions do, it can discover and reach to goal state.

Right → [2.4.6.8.]	Suck → {4,8}
Left → {3,7}	Suck → {7}

Contingency Problems

- **Partially observable:** The agent has limited access to the world state
- **Non-deterministic:** The agent is ignorant of the effect of its actions
- Sometimes ignorance prevents the agent from finding a guaranteed solution sequence.
- Suppose the agent is in **Murphy's law world**
  - The agent has to sense during the execution phase, since things might have changed while it was carrying out an action. This implies that
    - the agent has to compute a tree of actions, rather than a linear sequence of action
  - Example - **Vacuum cleaner world:**
    - action 'Suck' deposits dirt on the carpet, but only if there is no dirt already. Depositing dirt rather than sucking returns from ignorance about the effects of actions

Contingency Problems (cont…)

- Example - **Vacuum cleaner world**
  - What will happen given initial state {1,3}, and action sequence: [Suck, Right, Suck]?

    {1,3} → {5,7} → {6,8} → {8,6} (failure)
- Is there a way to solve this problem?
  - Thus, solving this problem requires **local sensing,** i.e. sensing the execution phase,
  - Start from one of the states {1,3}, and take improved action sequence [Suck, Right, Suck (only if there is dirt there)]
- Many problems in the real world are contingency problems (exact prediction is impossible)
  - For this reason many people keep their eyes open while walking around or driving.

Exploration problem

- The agent has no knowledge of the environment
  - **World Partially observable** : No knowledge of states (environment)
    - Unknown state space (no map, no sensor)
  - **Non-deterministic**: No knowledge of the effects of its actions
  - Problem faced by (intelligent) agents (like, newborn babies)
- This is a kind of problem in the real world rather than in a model, which may involve significant danger for an ignorant agent. If the agent survives, it learns about the environment
- The agent must experiment, learn and build the model of the environment through its results, gradually, discovering
  - What sort of states exist and What its action do
  - Then it can use these to solve subsequent (future) problems
- Example: in solving Vacuum cleaner world problem the agent learns the state space and effects of its action sequences say: [suck, Right]

Well-defined problems and solutions

To define a problem, we need the following elements: states, operators, goal test function and cost function.

- **The Initial state:** is the state that the agent starts in or begins with.
  - **Example-** the initial state for each of the following:
    - Coloring problem
      - All rectangles white
    - Route finding problem
      - The point where the agent start its journey (SidistKilo?)
    - 8-puzzle problem ??

Operators

- The set of possible actions available to the agent, i.e.
  - –which state(s) will be reached by carrying out the action in a particular state
  - –**A Successor function S(x)**
    - Is a function that returns the set of states that are reachable from a single state by **any** single action/operator
    - Given state x, S(x) returns the set of states reachable from x by any single action
- Example:
  - –Coloring problem: Paint with black color

    paint (color w, color b)
  - –Route finding problem: Drive through cities/places

    drive (place x, place y)
  - – 8-puzzle ??

Goal test function

- The agent execute to determine if it has reached the goal state or not
  - Is a function which determines if the state is a goal state or not

- Example:
  - Route finding problem: Reach Addis Ababa airport on time

    → IsGoal(x, Addis_Ababa)

  - Coloring problem: All rectangles black

    → IsGoal(rectangle[], n)

  - 8-puzzle ??

Path cost function

- A function that assigns a cost to a path (sequence of actions).
  - Is often denoted by **g.** Usually, it is the sum of the costs of the individual actions along the path (from one state to another state)
  - Measure path cost of a sequence of actions in the state space. For example, we may prefer paths with fewer or less costly actions
- Example:
  - Route finding problem:
    - Path cost from initial to goal state
  - Coloring problem:
    - One for each transition from state to state till goal state is reached
  - 8-puzzle?

Steps in problem solving

- **Goal formulation**
  - is a step that specifies exactly what the agent is trying to achieve
  - This step narrows down the **scope** that the agent has to look at
- **Problem formulation**
  - is a step that puts down the **actions** and **states** that the agent has to consider **given a goal** (avoiding any redundant states), like:
    - the initial state
    - the allowable actions etc…
- **Search**
  - is the process of looking for the **various sequence of actions** that lead to a goal state, evaluating them and choosing the **optimal** sequence.
- **Execute**
  - is the final step that the agent executes the chosen sequence of actions to get it to the solution/goal

Assignment (End Class)

Consider one of the following problem:

- **Missionary-and-cannibal problem**
- **Towers of Hanoi**
- **Tic-Tac-Toe**
- **Travelling sales person problem**
- **8-queens**

1. Identify the set of states and operators and construct the state space of the problem
2. write goal test function
3. Determine path cost

Uninformed Search

Searching

- Examine different possible sequences of actions & states, and come up with **specific sequence** of operators/actions that will take you from the initial state to the goal state
  - *Given a state space with **initial state** and **goal state,** find* optimal ***sequence of actions** leading through a **sequence of states** to the **final goal state***

- **Searching in State Space**
  - choose min-cost/max-profit node/state in the state space,
  - test the state to know whether we reached to the goal state or not,
  - if not, expand the node further to identify its successor.

Search Tree

- The searching process is like building the search tree that is super imposed over the state space
  - A search tree is a representation in which nodes denote paths and branches connect paths. The node with no parent is the **root node**. The nodes with no children are called **leaf nodes**.

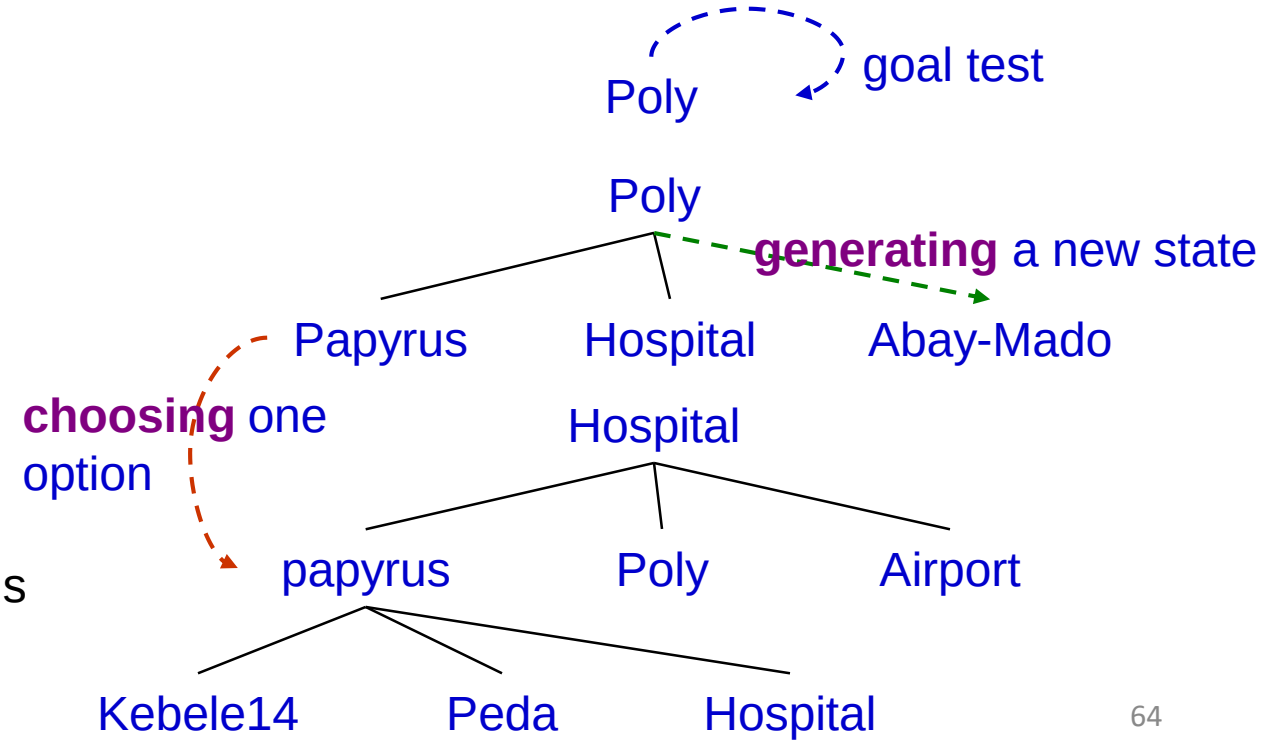
Example: Route finding Problem

- Partial **search tree** for route finding from Poly to Kebele14

(a) The initial state

(b) After **expanding** poly

(c) After expanding papyrus



Search algorithm

- Two functions needed for conducting search
  - **Generator (or successors) function**: Given a state and action, produces its successor states (in a state space)
  - **Tester (or IsGoal) function**: Tells whether given state S is a goal state IsGoal(S) → True/False
  - IsGoal and Successors functions depend on problem domain.
- Two lists maintained during searching
  - **OPEN list**: stores the nodes we have seen but not explored
  - **CLOSED list**: the nodes we have seen and explored
  - Generally, search proceeds by examining each node on the OPEN list, performing some expansion operation that adds its children to the OPEN list, and moving the node to the CLOSED list.
- **Merge function**: Given successor nodes, it either append, prepend or arrange based on evaluation cost
- **Path cost**: function assigning a numeric cost to each path; either from initial node to current node and/or from current node to goal node

Search algorithm

- **Input**: a given problem (Initial State + Goal state + transit states and operators)
- **Output**: returns optimal sequence of actions to reach the goal.

```
function GeneralSearch (problem, strategy)
    open = (initialState); //put initial state in the List
    closed = {}; //maintain list of nodes examined earlier
    while (not (empty (open)))
        f = remove_first(open);
        if IsGoal (f) then return (f);
        closed = append (closed, f);
        succ = Successors (f)
        left = not-in-closed (succ, closed );
        open = merge (rest(open), left); //append or prepend l to open list
    end while
    return ('fail')
end  GeneralSearch
```

Algorithm Evaluation: Completeness and Optimality

- Is the search strategies find solutions to problems with no solutions
  - Is it produces incorrect solutions to problems
- **Completeness**
  - Is the algorithm guarantees in finding a solution whenever one exists
  - Think about the density of solutions in space and evaluate whether the searching technique guaranteed to find *all* solutions or not.
- **Optimality**
  - is the algorithm finding an optimal solution; i.e. the one with minimum cost
    - how good is our solution?

Algorithm Evaluation: Time & Space Tradeoffs

- With many computing projects, we worry about:
  - Speed versus memory
  - Time complexity: how long does it take to find a solution
  - Space complexity: how much space is used by the algorithm
- Fast programs can be written
  - But they use up too much memory
- Memory efficient programs can be written
  - But they are slow
- We consider various search strategies
  - In terms of their memory/speed tradeoffs

Searching Strategies

- Search strategy gives the order in which the search space is examined
- **Uninformed (= blind) search**
  - they do not need domain knowledge that guide them to the right direction towards the goal
  - Have no information about the number of steps or the path cost from the current state to the goal
  - It is important for problems for which there is no additional information to consider
- **Informed (= heuristic) search**
  - have problem-specific knowledge (knowledge that is true from experience)
  - Have knowledge about how far are the various state from the goal
  - Can find solutions more efficiently than uninformed search

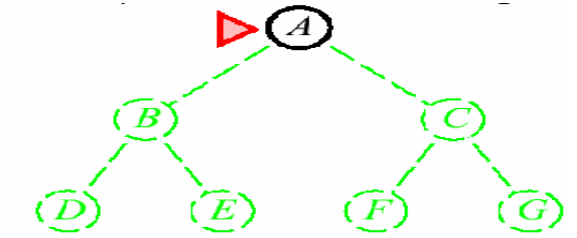
Search Methods:

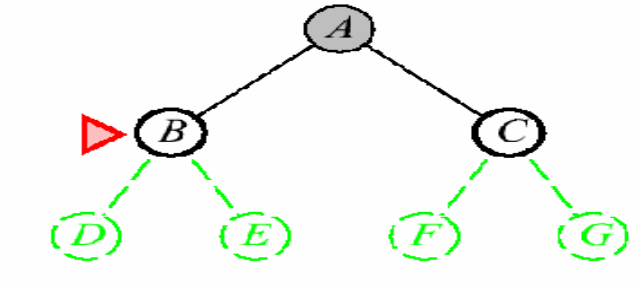
- *Uninformed search*
  - Breadth first
  - Depth first
  - Uniform cost, …
  - Depth limited search
  - Iterative deepening
  - etc.

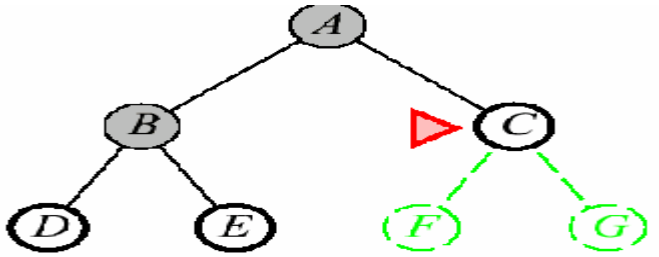
- Informed search
  - Greedy search
  - A*-search
  - Iterative improvement,
  - Constraint satisfaction
  - etc.

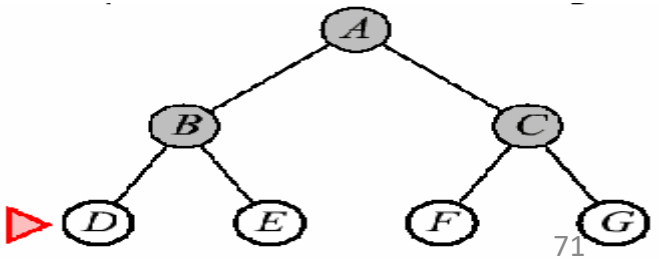
Breadth first search

- Expand shallowest unexpanded node,
  - i.e. expand all nodes on a given level of the search tree before moving to the next level
- **Implementation**: use **queue** data structure to store the list:
  - Expansion: put successors at the end of queue
  - Pop nodes from the front of the queue
- Properties:
  - Takes space: keeps every node in memory
  - Optimal and complete: guarantees to fin solution







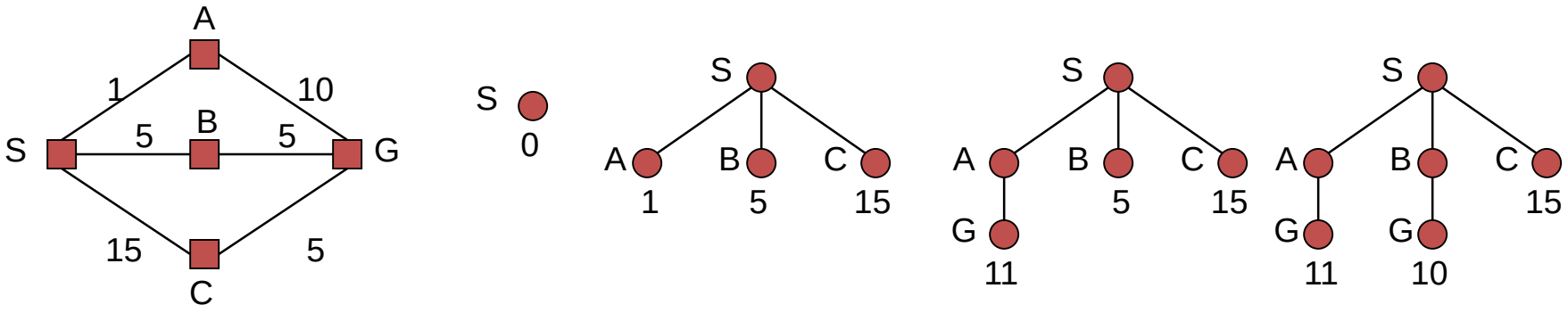


Algorithm for Breadth first search

```
function BFS (problem){
        open = (C_0); //put initial state C_0 in the List
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
           l = not-in-set (Successors (f), closed );
            open = merge ( rest(open), l); //append to the list
         }
        return ('fail')
    }
```

Uniform cost Search

- The goal of this technique is to find the shortest path to the goal in terms of cost.
  - It modifies the BFS by always expanding least-cost unexpanded node
- Implementation: nodes in list keep track of total path length from start to that node
  - List kept in priority queue ordered by path cost



- Properties:
  - This strategy finds the cheapest solution provided the cost of a path must never decrease as we go along the path

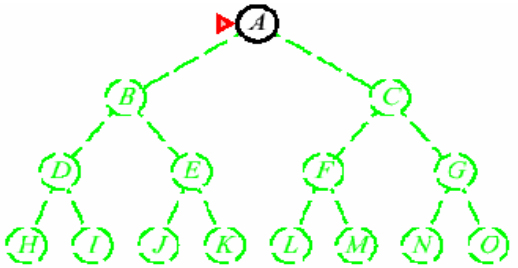
    g(successor(n)) ≥ g(n), for every node n
  - Takes space since it keeps every node in memory

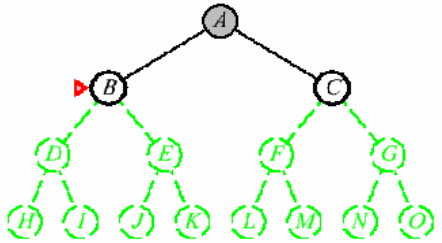
Algorithm for Uniform Cost search

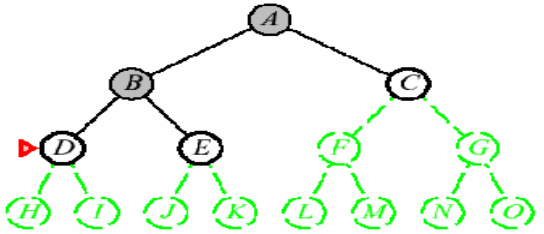
```
function uniform_cost (problem){
        open = (C_0); //put initial state C_0 in the List
        g(s) = 0;
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
           l = not-in-set (Successors (f), closed );
            g(f,l_i) = g(f) + c(f,l_i);
           open = merge(rest(open), l, g(f,l_i)); //keep the open list sorted in
           ascending order by edge cost
            }
        return ('fail')
    }
```

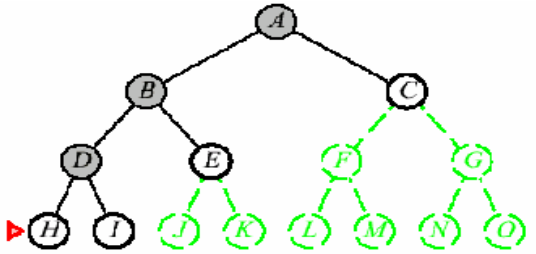
Depth-first search

- Expand one of the node at the deepest level of the tree.
  - –Only when the search hits a non-goal dead end does the search go back and expand nodes at shallower levels
- **Implementation**: treat the list as **stack**
  - –Expansion: push successors at the top of stack
  - –Pop nodes from the top of the stack
- Properties
  - –Incomplete and not optimal: fails in infinite-depth spaces, spaces with loops.
    - Modify to avoid repeated states along the path
  - –Takes less space (Linear): Only needs to remember up to the depth expanded









Algorithm for Depth first search

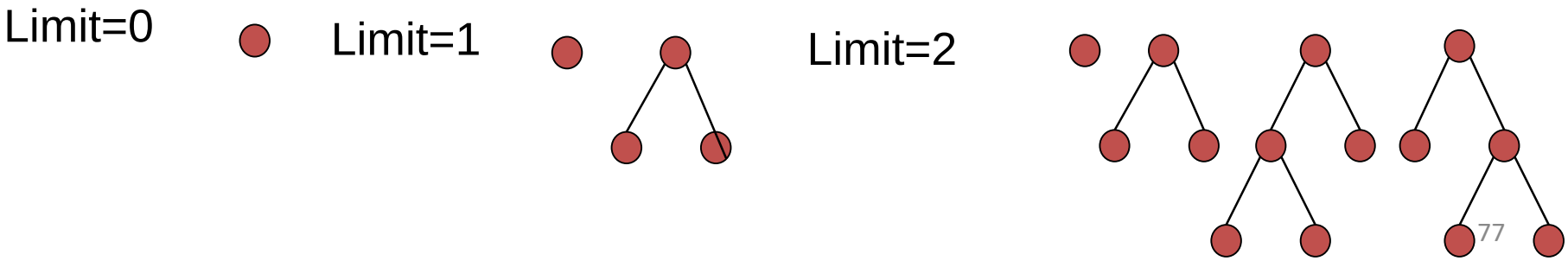
```
function DFS (problem){
        open = (C_0); //put initial state C_0 in the List
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
           l = not-in-set (Successors (f), closed );
            open = merge ( rest(open), l); //prepend to the list
         }
        return ('fail')
    }
```

Iterative Deepening Search (IDS)

- IDS solves the issue of choosing the best depth limit by trying all possible depth limit:
  - Perform depth-first search to a bounded depth *d*, starting at *d = 1* and increasing it by 1 at each iteration.

Example: for route finding problem we can take the diameter of the state space. In our example, at most 9 steps is enough to reach any node

- This search combines the benefits of DFS and BFS
  - DFS is efficient in space, but has no path-length guarantee
  - BFS finds min-step path towards the goal, but requires memory space
  - IDS performs a sequence of DFS searches with increasing depth-cutoff until goal is found

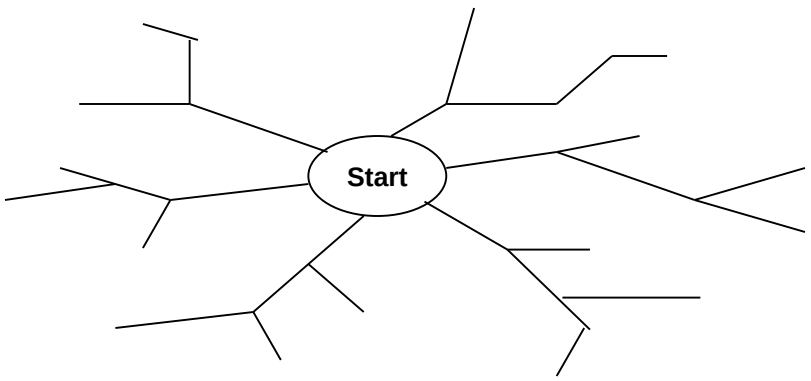


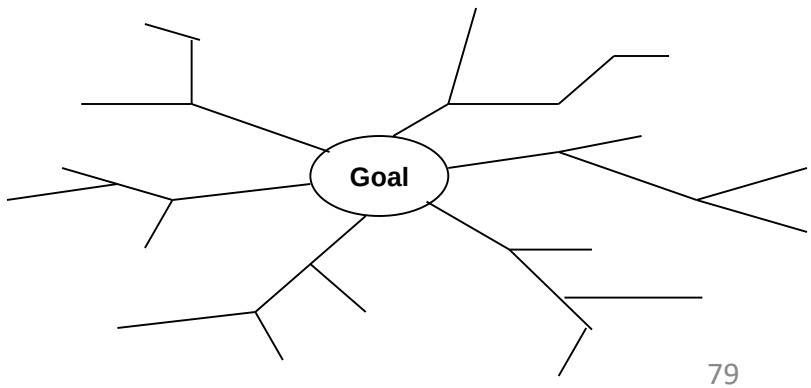
Algorithm for IDS

```
function IDS (problem){
      open = (C_0); //put initial state C_0 in the List
      closed = {}; /maintain list of nodes examined earlier
      while (not reached maxDepth) {
       while (not (empty (open))) {
            f = remove_first(open);
            if (IsGoal (f)) then return (f);
            closed = append (closed, f);
           l = not-in-set (Successors (f), closed);
            if (depth(l) < maxDepth) then
                     open = merge (rest(open), l); //prepend to the list
        }
       }
       return ('fail')
}
```

Bidirectional Search

- Simultaneously search both forward from the initial state to the goal and backward from the goal to the initial state, and stop when the two searches meet somewhere in the middle
  - Requires an explicit goal state and invertible operators (or backward chaining).
  - Decide what kind of search is going to take place in each half using BFS, DFS, uniform cost search, etc.





Bidirectional Search

- Advantages:
  - Only need to go to half depth
  - It can enormously reduce time complexity, but is not always applicable
- Difficulties
  - Do you really know solution? Unique?
  - Cannot reverse operators
  - Memory requirements may be important: Record all paths to check they meet
    - Memory intensive
- Note that if a heuristic function is inaccurate, the two searches might miss one another.

Comparing Uninformed Search

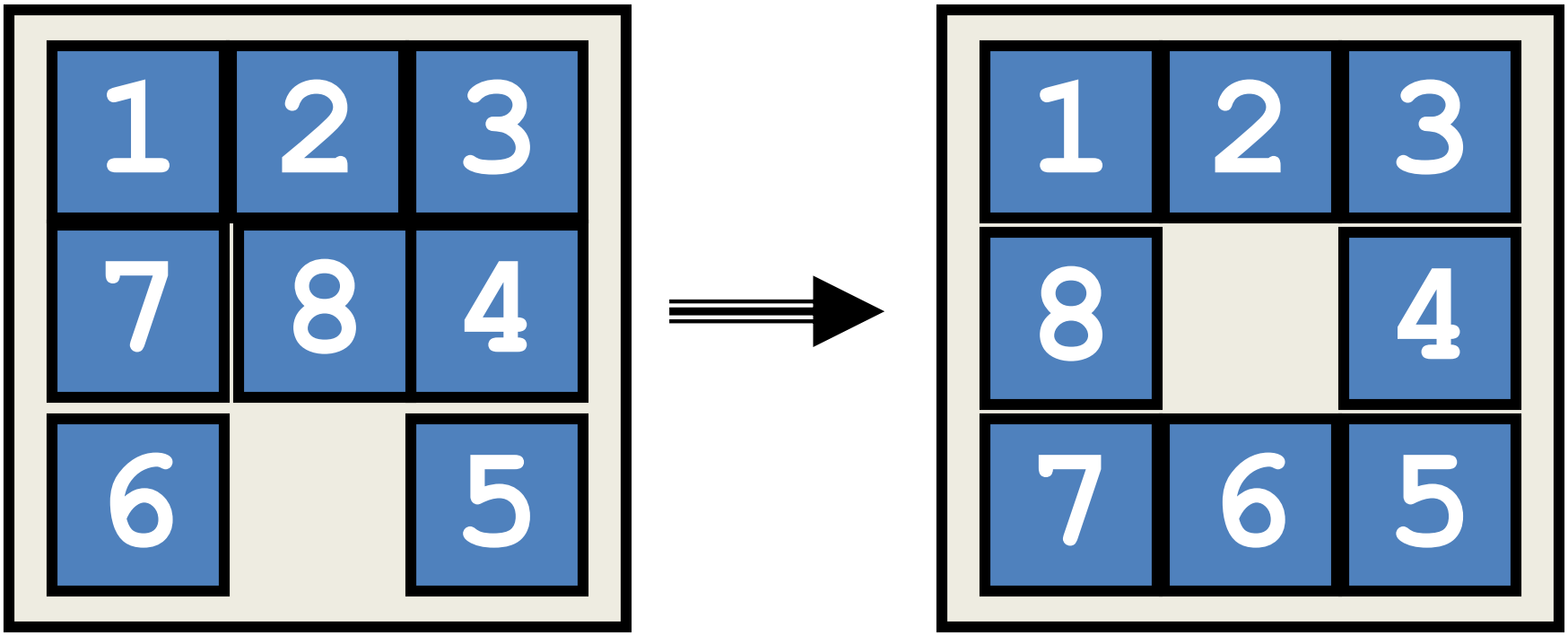
Strategies	**Complete**	**Optimal**	**Time complexity**	**Space complexity**
Breadth first search	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	$O(bm)$
Uniform cost search	yes	yes	$O(b^d)$	$O(b^d)$
Depth limited search	if l >= d	no	$O(b^l)$	$O(bl)$
Iterative deepening search	yes	yes	$O(b^d)$	$O(bd)$
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

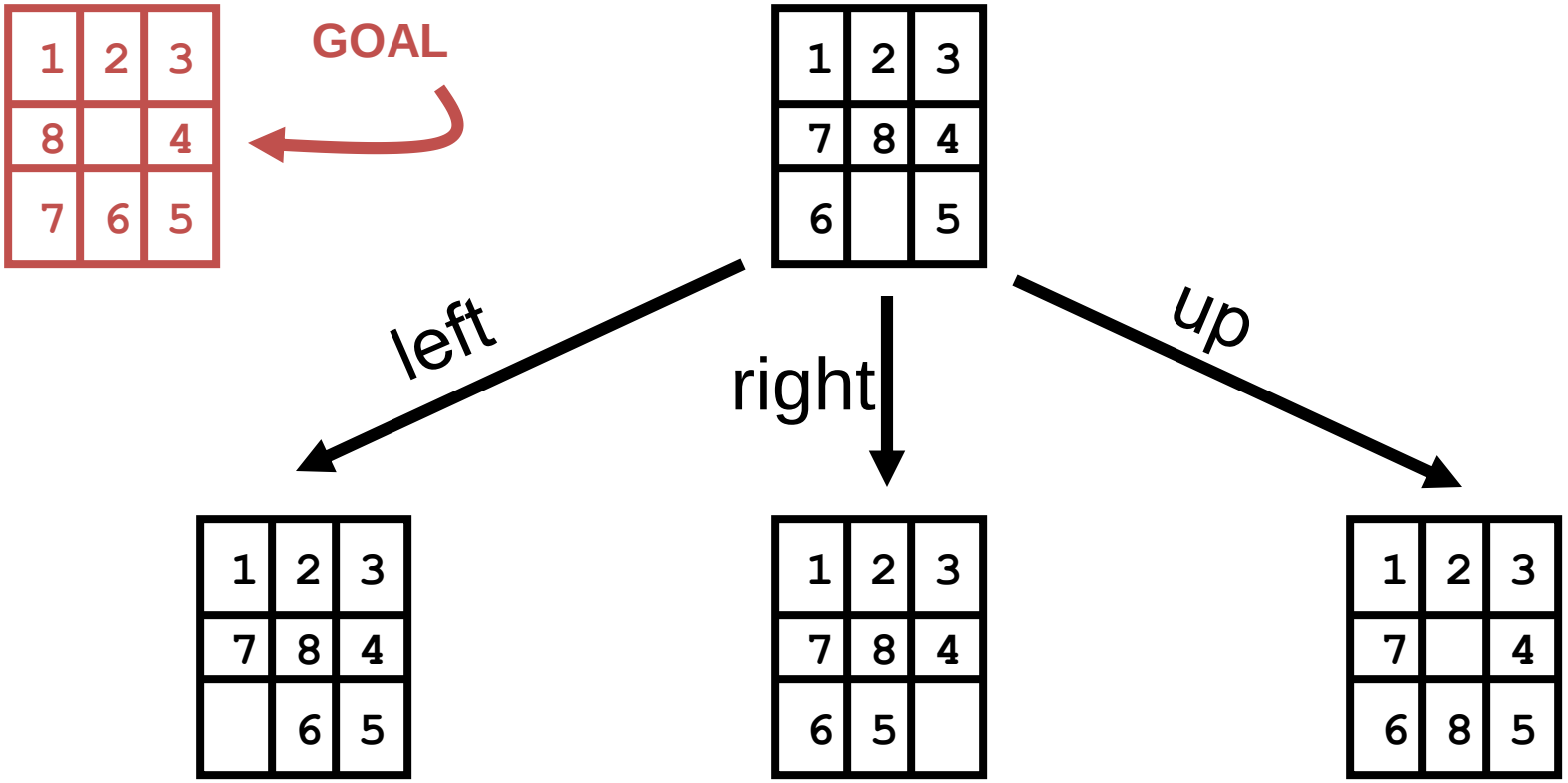
- b is branching factor,
- d is depth of the shallowest solution,
- m is the maximum depth of the search tree,
- l is the depth limit

Heuristic Search? Informed Search

- blind search are not always possible (they require too much time or memory).
- *Weak* techniques can be effective if applied correctly on the right kinds of tasks.
  - Typically require domain specific information.

Example: 8 Puzzle





Which move is best?

8 Puzzle Heuristics

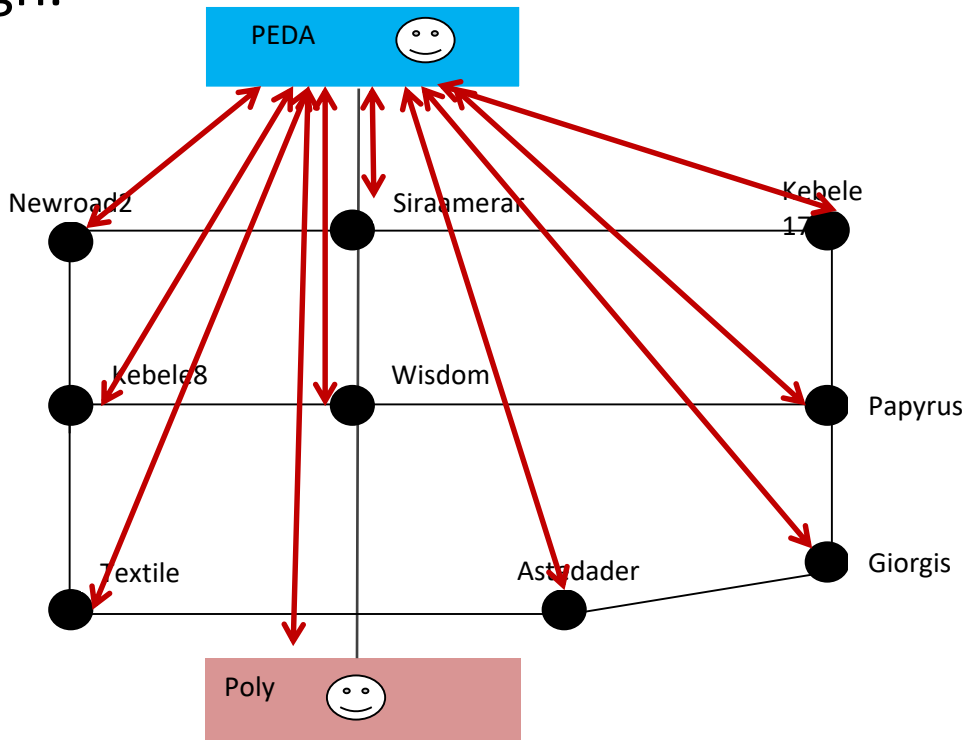
- Blind search techniques used an arbitrary ordering (priority) of operations.
- Heuristic search techniques make use of domain specific information - a heuristic.
- What heurisitic(s) can we use to decide which 8-puzzle move is “best” (worth considering first).

Heuristic s Information /Manhattan Distance/

- Number of tiles in the *incorrect* position.
  - This can also be considered a lower bound on the number of moves from a solution!
  - The “best” move is the one with the lowest number returned by the heuristic.
  - Is this heuristic more than a heuristic (is it always correct?).
    - Given any 2 states, does it always order them properly with respect to the minimum number of moves away from a solution?

Heuristic For Sales Man Problem

- The straight line distance between the goal node and different places in which the sales man person possibly can pass through.



Evaluation-function driven search

- A search strategy can be defined in terms of **a node evaluation function**
- **Evaluation function**
  - Denoted
  - Defines the desirability of a node to be expanded next
- **Evaluation-function driven search: expand the node (state) with the best evaluation-function value**
- **Implementation: priority queue with nodes in the decreasing** order of their evaluation function value

Best-first search

Best-first search

- incorporates a **heuristic function,** $h(n)$, into the evaluation function $f(n)$ to guide the search.

Heuristic function:

- Measures a potential of a state (node) to reach a goal
- Typically in terms of some distance to a goal estimate

Example of a heuristic function:

- Assume a shortest path problem with city distances on connections
- Straight-line distances between cities give additional information we can use to guide the search

Uniform cost search

- **Uniform cost search (Dijkstra's shortest path):**
  - A special case of the evaluation-function driven search

$$f(n) = g(n)$$

- **Path cost function** $g(n)$;
  - path cost from the initial state to n

- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - **weights or costs** associated with operators (links).

- **Note:** Uniform cost search relies on the problem definition only
  - It is an uninformed search method

Best-first search

- incorporates a **heuristic function,** $h(n)$, into the evaluation function $f(n)$ to guide the search.
- **heuristic function:** measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):

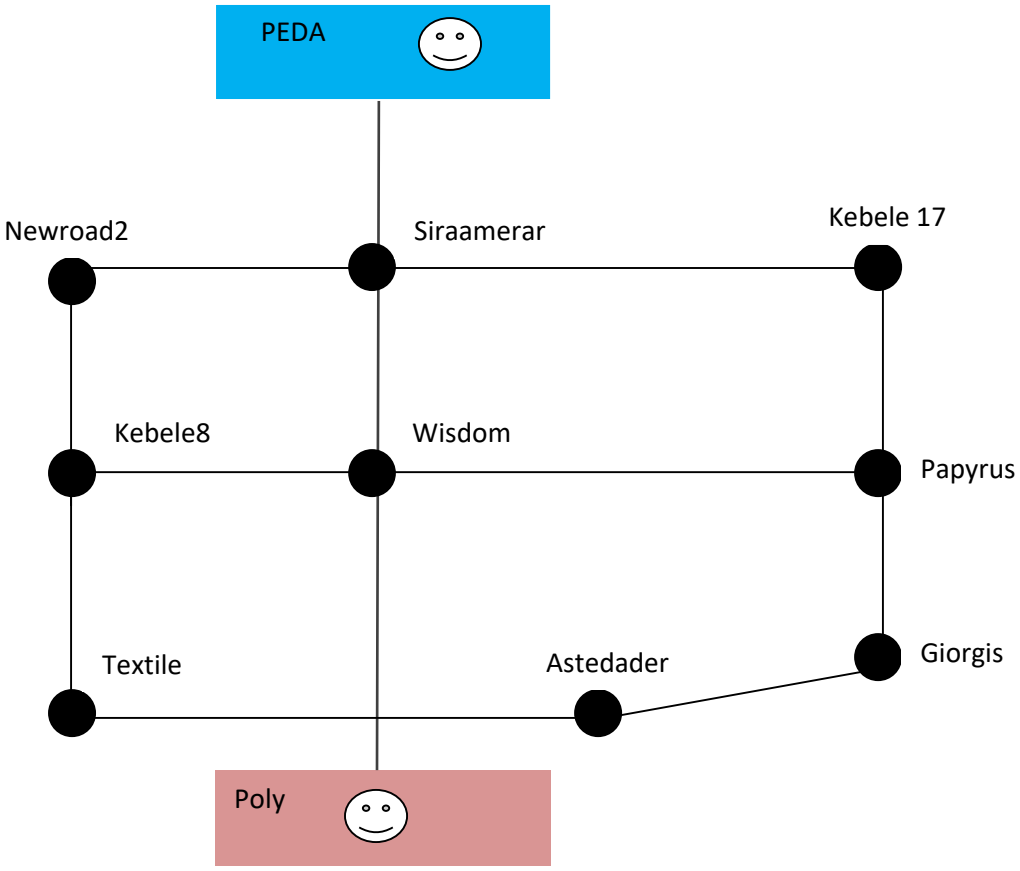
- **Greedy search**

$$f(n) = h(n)$$

- **A* algorithm**

$$f(n) = g(n) + h(n)$$

Exercises Greedy Search



Goal	Nodes	Disance
Peda	Poly	780m
Peda	textile	800m
Peda	astedader	760m
Peda	giorgis	890m
Peda	papyrus	650m
Peda	Kebele 17	520m
Peda	siramerar	200m
Peda	Kebele 8	700m
Peda	wisdom	510m
Peda	newroad1	310m

Properties of greedy search

- **Completeness:** **No**.

  We can loop forever. Nodes that seem to be the best choices can lead to cycles. Elimination of state repeats can solve the problem.

- **Optimality:** **No.**

  Even if we reach the goal, we may be biased by a bad heuristic estimate. Evaluation function disregards the cost of the path built so far.

- **Time complexity:** $O(b^m)$

  Worst case !!! But often better!

- **Memory (space) complexity:** $O(b^m)$

  Often better!

A* search

- The problem with the greedy search is that it can keep expanding paths that are already very expensive.
- The problem with the uniform-cost search is that it uses only past exploration information (path cost), no additional information is utilized
- **A* search**

$$f(n) = g(n) + h(n)$$

$g(n)$ - cost of reaching the state

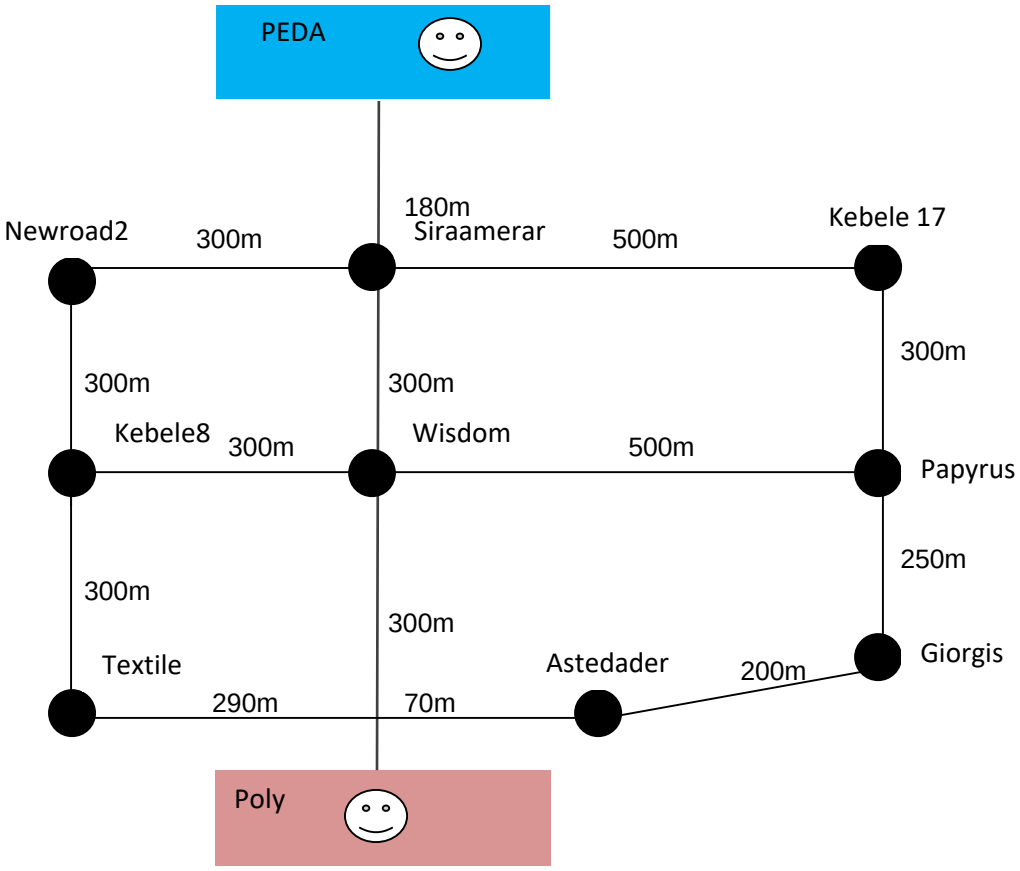
$h(n)$ - estimate of the cost from the current state to a goal

$f(n)$ - estimate of the path length

- **Additional A*condition**: admissible heuristic

$$h(n) \leq h^*(n) \quad \text{for all } n$$

Exercises A* search



Goal	Nodes	Disance
Peda	Poly	780m
Peda	textile	800m
Peda	astedader	760m
Peda	giorgis	890m
Peda	papyrus	650m
Peda	Kebele 17	520m
Peda	siramerar	200m
Peda	Kebele 8	700m
Peda	wisdom	510m
Peda	newroad1	310m

Properties of A* search

- **Completeness: Yes.**
- **Optimality: Yes (with the admissible heuristic)**
- **Time complexity:**
  - **Order roughly the number of nodes with $f(n)$ smaller than the cost of the optimal path g^***
- **Memory (space) complexity:**
  - **Same as time complexity (all nodes in the memory)**

Optimality of A*

- In general, a heuristic function $h(n)$:

  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$

- **Admissible heuristic condition**
  - **Never overestimate the distance to the goal !!!**

$$h(n) \leq h^*(n) \quad \text{for all } n$$

Example: the straight-line distance in the travel problem never overestimates the actual distance

Is A* search with an admissible heuristic is optimal ??

Knowledge Representation

Prepositional Logic

Knowledge-based agent

Any Knowledge base agent should have two important component

- **Knowledge base**
- **Inference engine**

Knowledge base (KB):

– A set of sentences that describe facts about the world in some formal (representational) language. It is hard two represent all general knowledge in one KB. Most of the time it is **domain specific**

- **Inference engine:**

– A set of procedures that use the representational language to infer new facts from known ones or answer a variety of KB queries. Inferences typically require search. It is **domain independent**

Example: MYCIN

- MYCIN: an expert system for diagnosis of bacterial infections
- **Knowledge base represents**

– Facts about a specific patient case

– Rules describing relations between entities in the bacterial infection domain

If	1. The stain of the organism is gram-positive, and 2. The morphology of the organism is coccus, and 3. The growth conformation of the organism is chains
Then	the identity of the organism is streptococcus

- **Inference engine:**

– manipulates the facts and known relations to answer diagnostic queries (consistent with findings and rules)

Knowledge representation

- The objective of knowledge representation is to express the knowledge about the world in a computer tractable form
- Key aspects of knowledge representation languages:

– **Syntax: describes how sentences are formed in the** language

– **Semantics: describes the meaning of sentences, what is it.** the sentence refers to in the real world

– **Computational aspect: describes how sentences and**

objects are manipulated in concordance with semantically conventions

Many KB systems rely on some variant of logic

Logic

A formal language for expressing knowledge and ways of reasoning.

Logic is defined by:

- **A set of sentences**: A sentence is constructed from a set of primitives according to syntax rules.
- **A set of interpretations**: An interpretation gives a semantic to primitives. It associates primitives with values.
- **The valuation (meaning) function *V***

– Assigns a value (typically the truth value) to a given sentence under some interpretation

V : sentence × interprétation → {True , False }

Example of logic

Language of numerical constraints:

- **A sentence:**
- **An interpretation:**
- **Valuation (meaning) function V:**

$x + 3 \leq z$

x , z - variable symbols (primitives in the language)

$x = 5, z = 2$

Variables mapped to specific real numbers

$V (x + 3 \leq z ,$ ***I) is False for I: x = 5, z = 2***

is ***True for I: x = 5, z = 10***

I:

Types of logic

- Different types of logics possible:

– Propositional logic

– First-order logic

– Temporal logic

– Numerical constraints logic

– Map-coloring logic

In the following:

- **Propositional logic.**

– Formal language for making logical inferences

– Foundations of **propositional logic: George Boole (1854)**

- **Propositional logic P:**

– defines a language for symbolic reasoning

- **Proposition: a statement that is either true or false**
- Examples of propositions:

– *Abay is the largest river in Ethiopia.*

– *Ethiopia is in Europe.*

– *It rains outside.*

– *2 is a prime number and 6 is a prime*

– *How are you? Not a proposition.*

Propositional logic. Syntax

- **Formally propositional logic P:**

– Is defined **by Syntax + interpretation + semantics of P**

Syntax:

- **Symbols (alphabet) in P:**

– **Constants: *True, False***

– **Propositional symbols**

Examples:

- *P*
- *Abay is the largest river in Ethiopia.,*
- *It rains outside, etc.*

– **A set of connectives:**

¬,∧ ,∨ ,⇒ ,⇔

Propositional logic. Syntax

Sentences in the propositional logic:

- **Atomic sentences:**

– **Constructed from constants and propositional symbols**

– True, False are (atomic) sentences

– or *Light in the room is on, It rains outside are* (atomic) sentences

- **Composite sentences:**

– **Constructed from valid sentences via connectives**

– If are sentences then

or

are sentences

P , Q

A, B

$\neg A$ $(A \wedge B)$ $(A \vee B)$ $(A \Rightarrow B)$ $(A \Leftrightarrow B)$

$(A \vee B) \wedge (A \vee \neg B)$

Propositional logic. Semantics.

The semantic gives the meaning to sentences.

the semantics in the propositional logic is defined by:

1. Interpretation of propositional symbols and constants

– Semantics of atomic sentences

2. Through the meaning of connectives

– Meaning (semantics) of composite sentences

Semantic: propositional symbols

A **propositional symbol**

- a statement about the world that is either true or false

Examples:

– *Abay is The largest river in Ethiopia*

– *It rains outside*

– *Light in the room is on*

- An **interpretation maps symbols to one of the two values:**

True (T), or False (F), depending on whether the symbol is satisfied in the world

I*: Light in the room is on -> True, It rains outside -> False*

I'*: Light in the room is on -> False, It rains outside -> False*

The **meaning (value) of the propositional symbol for a specific** interpretation is given by its interpretation

I*: Light in the room is on -> True, It rains outside -> False*

V(Light in the room is on, ***I) = True***

I'*: Light in the room is on -> False, It rains outside -> False*

V(Light in the room is on, ***I') = False***

Semantics: constants

- **The meaning (truth) of constants:**

– True and False constants are always (under any interpretation) assigned the corresponding ***True,False***

$$V(True, \mathbf{I}) = \textbf{\textit{True}}$$
$$V(False, \mathbf{I}) = \textbf{\textit{False}}$$

} For any interpretation $\mathbf{I}$

Semantics: composite sentences.

- **The meaning (truth value) of complex propositional sentences.**

– Determined using the standard rules of logic:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
True	*True*	*False*	*True*	*True*	*True*	*True*
True	*False*	*False*	*False*	*True*	*False*	*False*
False	*True*	*True*	*False*	*True*	*True*	*False*
False	*False*	*True*	*False*	*False*	*True*	*True*

Translation

Assume the following sentences:

- It is not sunny this afternoon and it is colder than yesterday. $\neg p \wedge q$
- We will go swimming only if it is sunny. $r \rightarrow p$
- If we do not go swimming then we will take a canoe trip. $\neg r \rightarrow s$
- If we take a canoe trip, then we will be home by sunset. $s \rightarrow t$

Denote:

- p = It is sunny this afternoon
- q = it is colder than yesterday
- r = We will go swimming
- s= we will take a canoe trip
- t= We will be home by sunset

Model, validity and satisfiability

• A **model (in logic): An interpretation is a model for a set of** sentences if it assigns true to each sentence in the set.

• A sentence is **satisfiable if it has a model;**

– There is at least one interpretation under which the sentence can evaluate to True.

• A sentence is **valid if it is *True in all interpretations***

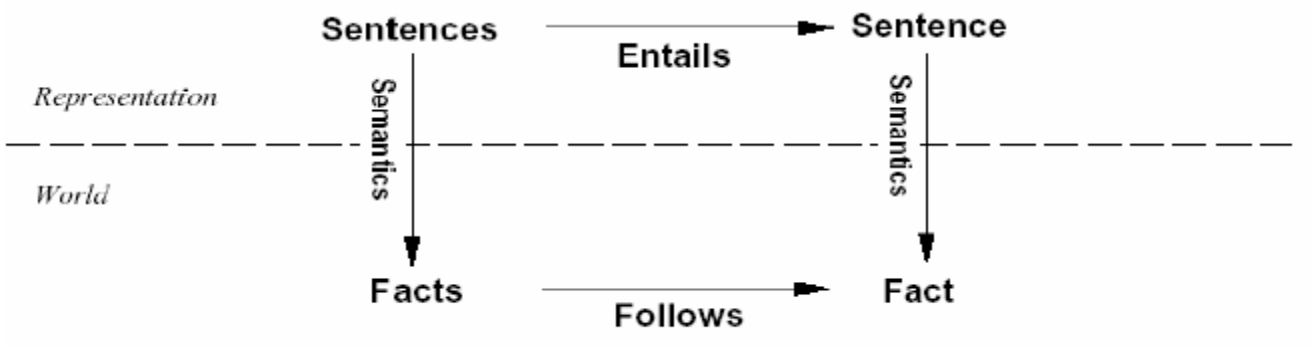
– i.e., if its negation is **not satisfiable (leads to contradiction)**

		Satisfiable sentence		Valid sentence
P	Q	$P \vee Q$	$(P \vee Q) \wedge \neg Q$	$((P \vee Q) \wedge \neg Q) \Rightarrow P$
True	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
False	*True*	*True*	*False*	*True*
False	*False*	*False*	*False*	*True*

Entailment

- **Entailment reflects the relation of one fact in the world**

following from the others



- Entailment $KB \models \alpha$
- Knowledge base KB entails sentence if and only if is true in all worlds where KB is true

Sound and complete inference.

Inference is a process by which conclusions are reached.

- We want to implement the inference process on a computer !!

Assume an **inference procedure *i that***

- derives a sentence α from the KB : $KB \vdash_i \alpha$

Properties of the inference procedure in terms of entailment

- **Soundness: An inference procedure is sound**

$$\text{If} \quad KB \vdash_i \alpha \text{ then it is true that} \quad KB \models \alpha$$

- **Completeness: An inference procedure is complete**

$$\text{If} \quad KB \models \alpha \text{ then it is true that} \quad KB \vdash_i \alpha$$

Logical inference problem:

- **Given:**

– a knowledge base KB (a set of sentences) and

– a sentence α (called **a theorem**)

- **Does a KB semantically entail** α ? $KB \models \alpha$?

In other words: In all interpretations in which sentences in the KB are true, is also α true?

Question: Is there a procedure (program) that can decide this problem in a finite number of steps?

Answer: Yes. Logical inference problem for the propositional logic is **decidable.**

Solving logical inference problem

In the following:

How to design the procedure that answers:

$$KB \models \alpha \ ?$$

Three approaches:

- **Truth-table approach**
- **Inference rules**
- **Conversion to the inverse SAT problem**

– **Resolution-refutation**

Truth-table approach

Problem: $KB \models \alpha$?

- We need to check all possible interpretations for which the KB is true (models of KB) whether α is true for each of them

Truth table:

- enumerates truth values of sentences for all possible interpretations (assignments of True/False to propositional symbols)

Example:

		KB		α	
P	Q	$P \vee Q$	$P \Leftrightarrow Q$	$(P \vee \neg Q) \wedge Q$	
True	*True*	*True*	*True*	*True*	✓
True	*False*	*True*	*False*	*False*	
False	*True*	*True*	*False*	*False*	
False	*False*	*False*	*True*	*False*	

Truth-table approach

$KB = (A \vee C) \wedge (B \vee \neg C)$ $\quad \alpha = (A \vee B)$

A	B	C	$A \vee C$	$(B \vee \neg C)$	KB	α
True	*True*	*True*	*True*	*True*	*True*	*True*
True	*True*	*False*	*True*	*True*	*True*	*True*
True	*False*	*True*	*True*	*False*	*False*	*True*
True	*False*	*False*	*True*	*True*	*True*	*True*
False	*True*	*True*	*True*	*True*	*True*	*True*
False	*True*	*False*	*False*	*True*	*False*	*True*
False	*False*	*True*	*True*	*False*	*False*	*False*
False	*False*	*False*	*False*	*True*	*False*	*False*

KB entails α

- The **truth-table approach** is **sound and complete** for the propositional logic!!

Limitation of the truth table approach.

$$KB \models \alpha \ ?$$

Problem with the truth table approach:

- the truth table is **exponential** in the number of propositional symbols (we checked all assignments)
- KB is true only on a small subset interpretations

How to make the process more efficient?

Inference rules approach.

$$KB \models \alpha \ ?$$

Problem with the truth table approach:

- the truth table is **exponential** in the number of propositional symbols (we checked all assignments)
- KB is true on only a smaller subset

How to make the process more efficient?

Solution: check only entries for which KB is *True*.

This is the idea behind the inference rules approach

Inference rules:

- Represent sound inference patterns repeated in inferences
- Can be used to generate new (sound) sentences from the existing ones

Inference rules for logic

- **Modus ponens**

$$\frac{A \Rightarrow B, \quad A}{B}$$

⟸ **premise**

⟸ **conclusion**

- If both sentences in the premise are true then conclusion is true.
- The modus ponens inference rule is **sound.**
  - We can prove this through the truth table.

A	B	$A \Rightarrow B$
False	*False*	*True*
False	*True*	*True*
True	*False*	*False*
True	*True*	*True*

Inference rules for logic

- **And-elimination**

$$\frac{A_1 \wedge A_2 \wedge \quad A_n}{A_i}$$

- **And-introduction**

$$\frac{A_1, A_2, \quad A_n}{A_1 \wedge A_2 \wedge \quad A_n}$$

- **Or-introduction**

$$\frac{A_i}{A_1 \vee A_2 \vee \ldots A_i \vee \quad A_n}$$

Inference rules for logic

- **Elimination of double negation** $\dfrac{\neg\neg A}{A}$

- **Unit resolution** $\dfrac{A \vee B, \quad \neg A}{B}$

- **Resolution** $\dfrac{A \vee B, \quad \neg B \vee C}{A \vee C}$

A special case of

- All of the above inference rules **are sound.** We can prove this through the truth table, similarly to the **modus ponens** case.

Example. Inference rules approach.

KB: $P \wedge Q \quad P \Rightarrow R \quad (Q \wedge R) \Rightarrow S$ **Theorem:** S

1. $P \wedge Q$
2. $P \Rightarrow R$
3. $(Q \wedge R) \Rightarrow S$
4. P — **From 1 and And-elim**
5. R — **From 2,4 and Modus ponens**
6. Q — **From 1 and And-elim**
7. $(Q \wedge R)$ — **From 5,6 and And-introduction**
8. S — **From 7,3 and Modus ponens**

Proved: S

First Order Logic

- Representing knowledge with symbolic representation has many draw backs, since the representation doesn’t have any facility that can represents objects and relations between objects, variables and quantifiers in addition to prepositions.

- FOL represents objects and relations between objects, variables and quantifiers in addition to prepositions.
- Example

Every elephant is gray

- in prepositional logic we can represent with some symbol either G or Q as far as the sentences have either a true or false value. Or in prepositional logic we can represent only facts. How ever in FOL the sentence can be represented as :
  - $\forall x$ elephant(x)→gray(x).

There is a white dog

  - $\exists x$ dog(X) $\wedge$ white(X)

- It makes it possible to say that a certain property holds of all objects, of some objects, or of no object.
- In predicates logic there are three additional notations.
- Terms: terms in first-order logic are used to represent objects or individuals.
- **Terms can be :**
  - a constant (designate specific object) For e.g. A, B, Smith, Blue, john kebede etc,
  - variable (designate unspecified object): x, y, z, etc, and
  - Functions (designate a specific object related in a certain way to another object, or, objects):Father Of, Colour Of, sqrroot.
  - Function can also return values
  - Connectives retain connectives in prepositional logic

- **Predicates:** Predicates is defined as a relation that binds two atoms have a value of true or false. Used to relate one object with the other.
- A predicate can take arguments, which are terms.
  - A predicate with one argument expresses a property of an object for e.g. Student(Bob).
  - A predicate with two or more arguments expresses a relation between objects for e.g .likes(Bob, Mary).
  - Predicate with no arguments is just a simple proposition logic.

Quantifiers: specify whether some or all objects satisfy properties of relations between objects .

Two standard quantifier : Universal(For all quantifier)

- For all quantifier
  - $\forall$=all quantifier and $\forall$ X stands for all X
  - For e.g. $\forall$x P(x) means “for all x, P of x is true”. Example: $\forall$ Person Happy (Person)
  - For all Person, Happy of person is true
  - that means everyone is happy.

- Universal quantifier make statement about every object

  ∀<variables><sentence>

Example

- Every one at Computer Science is smart.

  ∀X at(X,Computer Science)→ smart(X)

- All cats are mamal

  ∀X cat(X)→ mammal(X)

∀X sentence P is true iff P is true with X being each possible object in a given universe.

- The above statement is equivalent to the conjunction

  At(Jonas, Computer Science)→smart(Jonas) ^
  At(alemu, Computer Science)→smart(alemu) ^
  At(abebe, Computer Science)→smart(abebe) ^
  At(aster, Computer Science)→smart(aster) ^……..

Common Mistakes to avoid

Typically → is the main connective with ∀ Quantifier

Common Mistakes: The use of ^ as the main connective with ∀

- **Example**

  Correct: ∀ *x* (*StudiesAt*(*x*;*BDU*))*Smart*(*x*))

  "Everyone who studies at BDU is smart"

- Wrong: $\forall x$ (*StudiesAt*(*x*;*BDU*) ^ *Smart*(*x*))
- “Everyone studies at BDU and is smart”, i.e.,
- “Everyone studies at Koblenz and everyone is smart”

Existential Quantifier: Makes statement about some object in the universe

- ∃ <variables><sentence>
- Existential Quantifier: if the statement is ∃ x P(x) means
- “there exists at least one x for which P of x is true”.
- Example: ∃x Happy(x),
- If the universe of discourse is people, then this means there is at least one happy person

Example: FOL Sentence

- "Every rose has a thorn"

$$\forall X.\big(rose(X) \rightarrow \exists Y.(has(X,Y) \wedge thorn(Y))\big)$$

- For all X
  - if (X is a rose)
  - then there exists Y
    - (X has Y) and (Y is a thorn)

- Every one at computer science Department is smart.

  ∃x at(X,Computer Science) ^ samrt(X)

- Spot has sister who is cat

  ∃x sister(Spot,X) ^ cat(X)

  ∃x Sentence P is true if and only if P is true with X being some possible object. ∃x at(X,Computer Science) ^ samrt(X) equivalent to the disjunction

  At(jones, Computer Science) Samrt(Jones) ^

  at(Kebede,Computer Science) smart(Kebede) ^……

Common mistakes to avoid

Note

^ is the main connective with ∃

Common mistake →Using) as the main connective with ∃

Example

Correct: ∃ *x* (*StudiesAt*(*x*;*computer science*) ^ *Smart*(*x*))

“There is someone who studies computer science and is smart”

Nested Quantifier

- The two quantifier may nested one over the other.
- Example for all Children Y and Parent X if X is a parent of Y then Y is a child of X.
- $\forall XY\ \text{parent}(X,Y) \rightarrow \text{child}(Y,X)$
- $\exists\ x\ \forall Y\ \text{loves}(X,Y)$

  May be interpreted as . There is a person X who laves every one of Y in the world
- $\forall Y\ \exists\ X\ loves(Y,X)$

  Every one in the world have some x to love in the world

Property of Quantifier

- $\forall X \forall Y = \forall Y \forall X$
- $\exists X \exists Y = \exists Y \exists X$
- $\exists X \forall Y$ is not equal to $\exists Y \forall X$

Eaxmples

- $\exists x \forall y Loves(x; y)$
- **“There is a person who loves everyone in the world”**
- $\forall y \exists x Loves(x; y)$
- **“Everyone in the world is loved by at least one person”**

Quantifier Duality

- $\forall x Likes(x,$Bread$)$ **is the same as** $\neg \exists x \neg Likes(x, Bread)$

  "Every body likes Bread" is equivalent to the expression "there is no one who dislike Bread"
- $\exists xLikes(x,Injera)$ **is the same as** $\neg\forall\neg Likes(x,Injera)$
- "There is some one who likes injera is equivalent to the expression " not all person dislikes injera"

Sentence structure

- In the first order formal language the basic unit is predicate(argument/terms) structure called sentence of predicate facts.
- Terms refer to objects
- Example – likes(aster, chocolate)= mulu likes chocolate
- Tall(abebe)= abebe is tall
- Terms or arguments can be any of constant symbols such as Mulu variable symbol such as X function such as Motherof(abebe), fatherof(fatherof(aster)

Example represent the following in first order logic

- Everything in some garden is lovely.
- Every onle likes Cake
- Peter has some friends.
- John palys Piano or Violinn
- Some people Likes snake.
- Winston Didn’t write hamlet
- Aster has an engineer Brother.
- Every soldier has its own gun

Inference Rules

- All inference rules that works for Prepositional logic also works for First Order Logic

First Order Logic Inference rules

- **Variable substitutions**

Variables in the sentences can be substituted with terms. (terms = constants, variables, functions)

Substitution:

Is represented by a mapping from variables to terms

$\{x_1/ t_1, x_2/ t_2,\text{K}\}$

Application of the substitution to sentences

SUBST({*x* / *Sam*, *y* / *Pam*}, *Likes*(*x*, *y*)) = *Likes*(*Sam*, *Pam*)

SUBST ({*x* / *z*, *y* / *fatherof* (*John*)}, *Likes* (*x*, *y*)) =

Likes (*z*, *fatherof* (*John*))

- **Universal elimination**

$$\frac{\forall\ x\ f(x)}{f(a)}$$

a - is a constant symbol

– substitutes a variable with a constant symbol

$\forall\ x$ *Likes*(*x*, *IceCream*) *Likes*(*Ben*, *IceCream*)

- **Existential elimination.**

$$\frac{\exists\ x\ f(x)}{f(a)}$$

– Substitutes a variable with a constant symbol that does not appear elsewhere in the KB

$\exists\ x$ *Kill*(*x*,*Victim*) *Kill*(*Murderer*,*Victim*)

Inference with resolution rule

- **Proof by refutation:**
  - Prove that $KB \ , \emptyset \ \alpha$ is **unsatisfiable**
  - resolution is **refutation-complete**

- **Main procedure (steps):**
  1. Convert $KB \ , \emptyset \ \alpha$ to CNF with ground terms and universal variables only
  2. Apply repeatedly the resolution rule while keeping track and consistency of substitutions
  3. Stop when empty set (contradiction) is derived or no more new resolvents (conclusions) follow

Conversion to CNF

1. Eliminate implications, equivalences

$$(p \Rightarrow q) \rightarrow (\neg p \vee q)$$

2. Move negations inside (DeMorgan's Laws, double negation)

$$\neg(p \wedge q) \rightarrow \neg p \vee \neg q$$
$$\neg(p \vee q) \rightarrow \neg p \wedge \neg q$$

$$\neg \forall x\ p \rightarrow \exists x\ \neg p$$
$$\neg \exists x\ p \rightarrow \forall x\ \neg p$$
$$\neg\neg\ p \rightarrow p$$

3. Standardize variables (rename duplicate variables)

$$(\forall x\ P(x)) \vee (\exists x\ Q(x)) \rightarrow (\forall x\ P(x)) \vee (\exists y\ Q(y))$$

4. Move all quantifiers left (no invalid capture possible)

$$(\forall x\ P(x)) \vee (\exists y\ Q(y)) \rightarrow \forall x\ \exists y\ P(x) \vee Q(y)$$

5. Skolemization (removal of existential quantifiers through elimination)

- If no universal quantifier occurs before the **existential quantifier**, replace the **variable with a new constant symbol**

$$\exists y \; P(A) \vee Q(y) \rightarrow P(A) \vee Q(B)$$

- If a universal quantifier precede the existential quantifier replace the variable with a function of the "universal" variable

$$\forall x \; \exists y \; P(x) \vee Q(y) \rightarrow \forall x \;\; P(x) \vee Q(F(x))$$

$F(x)$ - **a special function**
- **called Skolem function**

6. Drop universal quantifiers (all variables are universally quantified)

$$\forall x \;\; P(x) \vee Q(F(x)) \rightarrow P(x) \vee Q(F(x))$$

7. Convert to CNF using the distributive laws

$$p \vee (q \wedge r) \rightarrow (p \vee q) \wedge (p \vee r)$$

The result is a CNF with variables, constants, functions

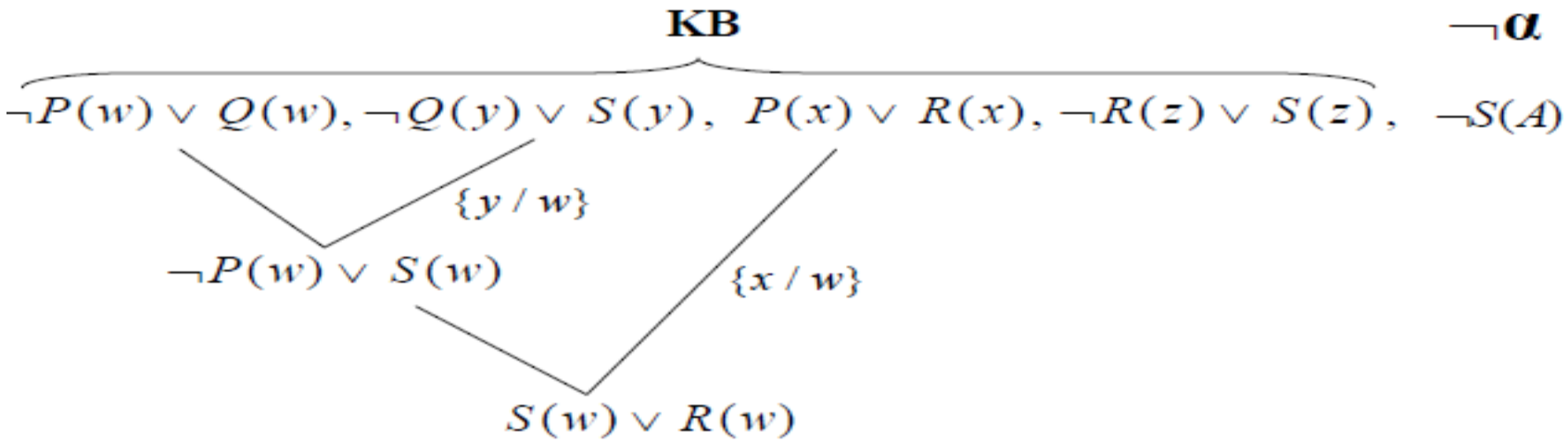
Resolution example

$$\overbrace{\neg P(w) \vee Q(w), \neg Q(y) \vee S(y), \; P(x) \vee R(x), \neg R(z) \vee S(z)}^{\text{KB}}, \quad \overbrace{\neg S(A)}^{\neg \alpha}$$

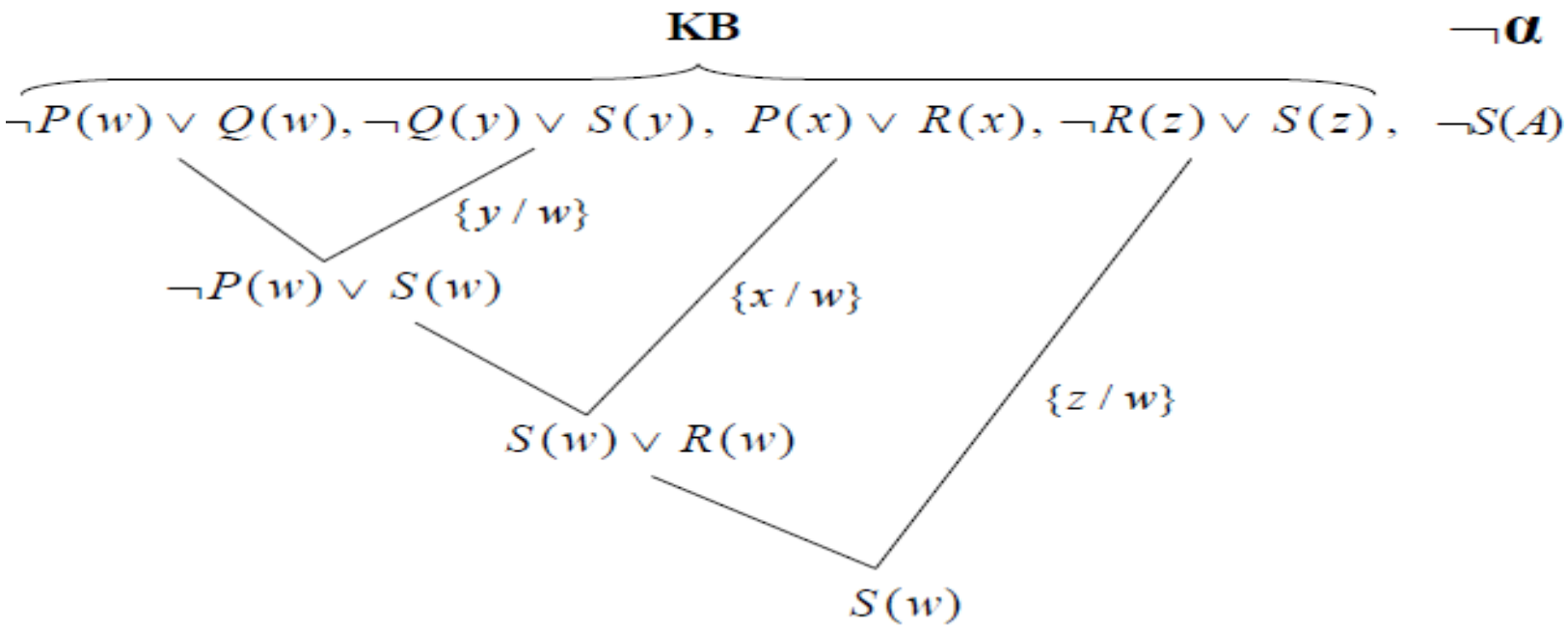
$$\overbrace{\neg P(w) \vee Q(w), \neg Q(y) \vee S(y), \; P(x) \vee R(x), \neg R(z) \vee S(z)}^{\text{KB}}, \quad \overbrace{\neg S(A)}^{\neg \alpha}$$

$$\{y / w\}$$

$$\neg P(w) \vee S(w)$$


KB
$\neg\alpha$
$\neg P(w) \vee Q(w)$, $\neg Q(y) \vee S(y)$, $P(x) \vee R(x)$, $\neg R(z) \vee S(z)$, $\neg S(A)$
$\{y / w\}$
$\neg P(w) \vee S(w)$
$\{x / w\}$
$S(w) \vee R(w)$

Resolution example



Resolution example