ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIT 2

Prof. Merin Meleet,
Assistant Professor, Dept of ISE

CONTENTS

Adversarial search, constraint satisfaction problems, logical agents, first-order logic

- Games, Optimal decision in games, Alpha-Beta Pruning
- Defining Constraint satisfaction problems; Backtracking search for CSPs
- Knowledge-based agents; The wumpus world as an example world;
- Logic; propositional logic; Propositional theorem proving; Syntax and semantics of first-order logic; Using first-order logic;

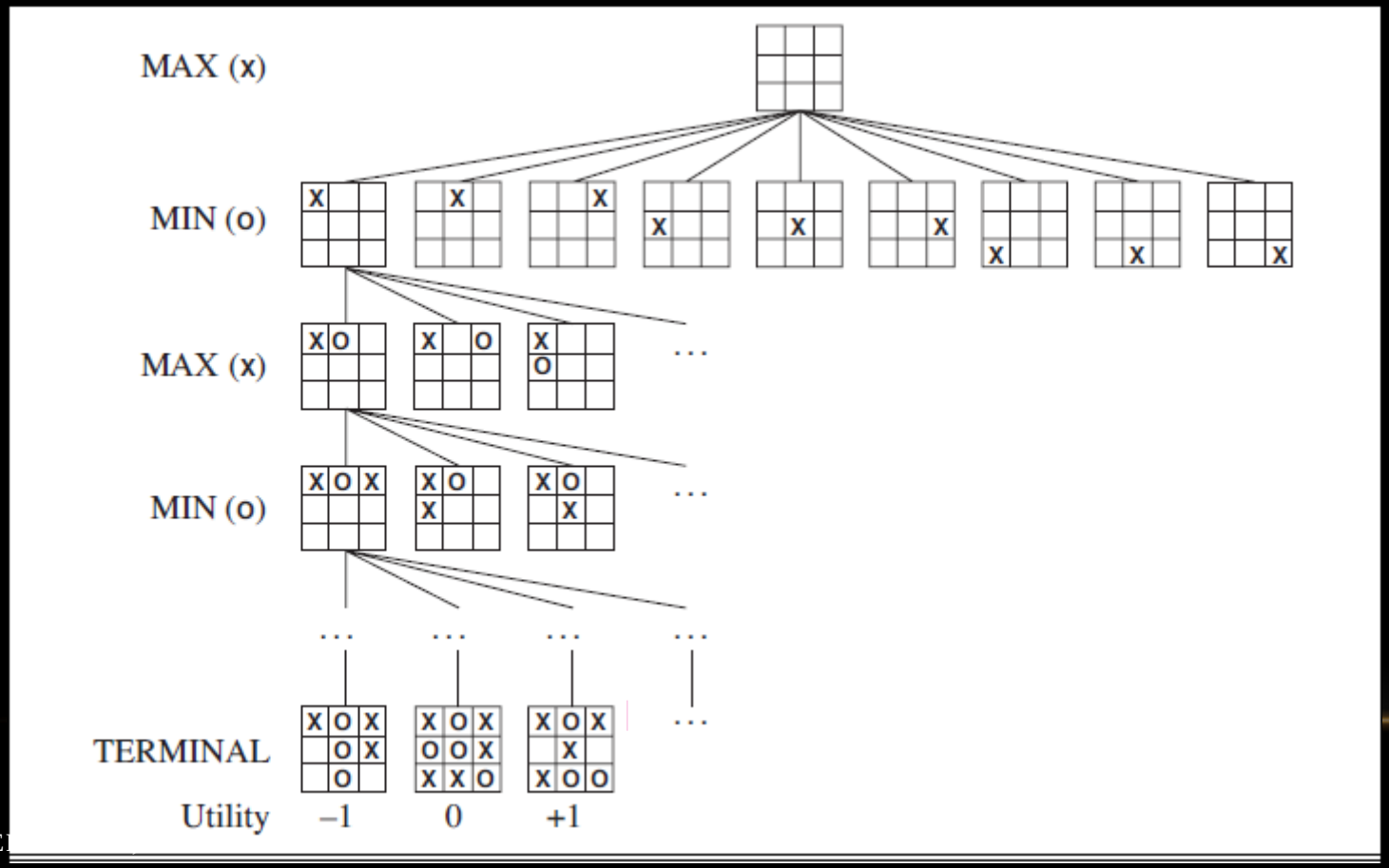
GAMES

- **Multiagent environments :** each agent needs to consider the actions of other agents and how they affect its own welfare.
- In this chapter we cover **competitive** environments, in which the agents' goals are in conflict, giving rise to **adversarial search** problems—often known as **games**.
- We begin with a definition of the optimal move and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- **Pruning** allows us to ignore portions of the search tree that make no difference to the final choice, and heuristic **evaluation functions** allow us to approximate the true utility of a state without doing a complete search.

- We first consider games with two players, whom we call MAX and MIN.
- MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser.
- A game can be formally defined as a kind of search problem with the following elements:

- S_0: The **initial state**, which specifies how the game is set up at the start.
- PLAYER(s): Defines which player has the move in a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): The **transition model**, which defines the result of a move.
- TERMINAL-TEST(s): A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- UTILITY(s, p): A **utility function** (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p. In

- The initial state, ACTIONS function, and RESULT function define the **game tree** for the game—a tree where the nodes are game states and the edges are moves.



Partial Game Tree-Example

OPTIMAL DECISIONS IN GAMES

- In a normal search problem, the optimal solution would be a sequence of actions leading to a goal state—a terminal state that is a win.
- In adversarial search, MIN has something to say about it. MAX therefore must find a contingent **strategy**, which specifies MAX's move in the initial state, then MAX's moves in the states resulting from every possible response by MIN, then MAX's moves in the states resulting from every possible response by MIN to *those* moves, and so on.

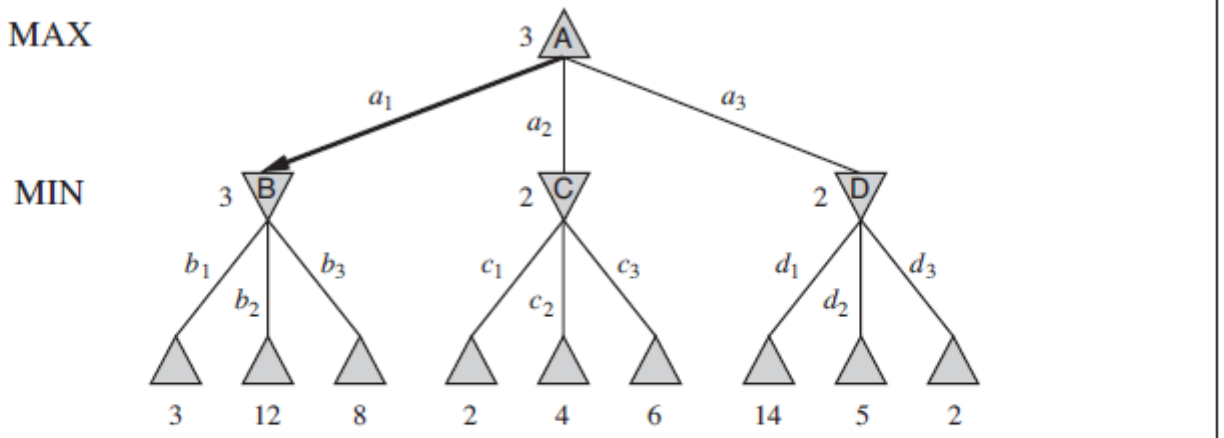


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

The possible moves for MAX at the root node are labeled a1, a2, and a3. The possible replies to a1 for MIN are b1, b2, b3, and so on. This particular game ends after one move each by MAX and MIN.

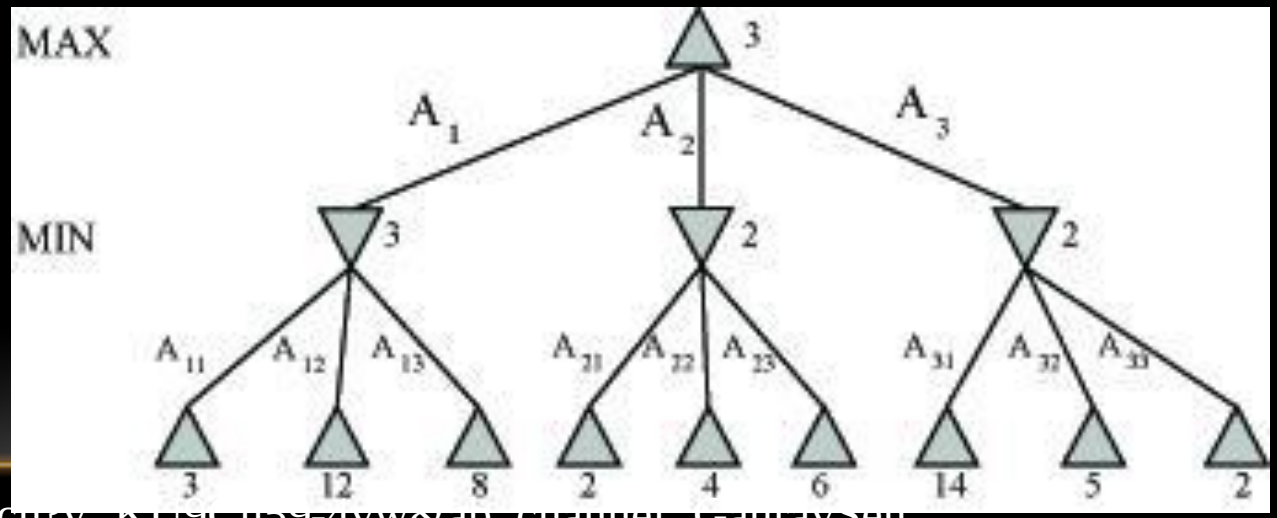
- Given a game tree, the optimal strategy can be determined from the **minimax value** of each node, which we write as MINIMAX(n).
- The minimax value of a node is the utility (for MAX) of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

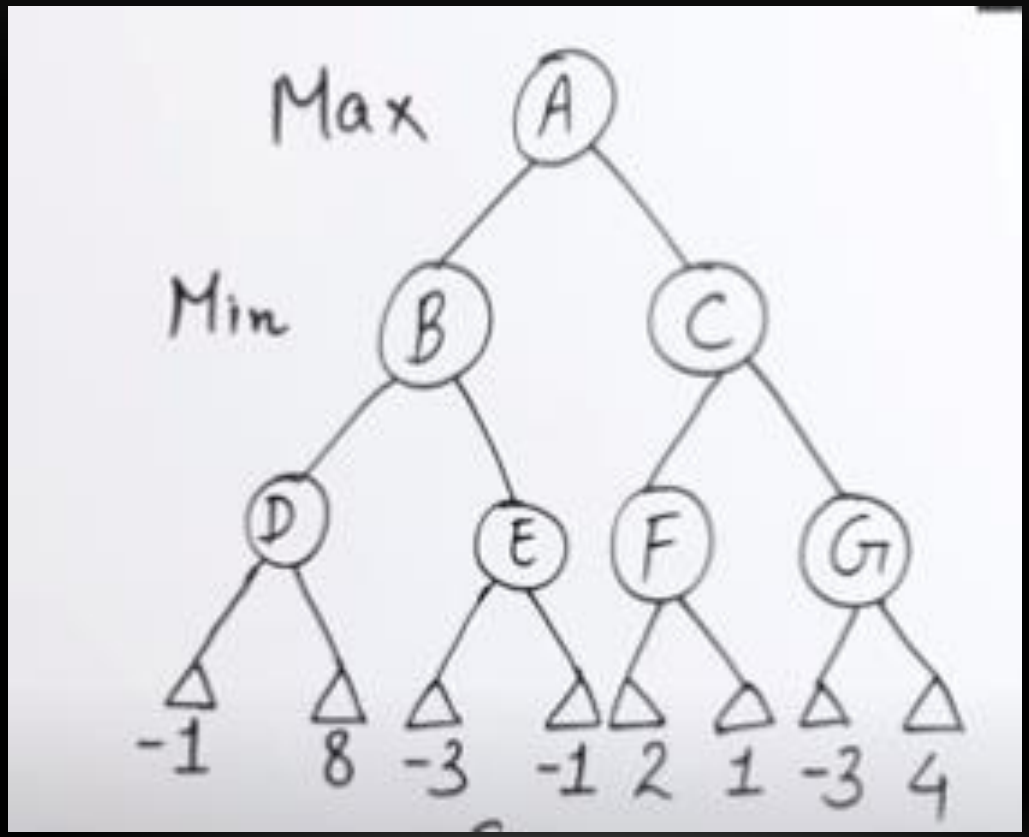
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

MINIMAX ALGORITHM

- Search the tree to the end
- Assign utility values to terminal nodes
- Find the best move for MAX (on MAX's turn), assuming:
  - MAX will make the move that maximizes MAX's utility
  - MIN will make the move that minimizes MAX's utility
- Here, MAX should make the leftmost move



https://www.youtube.com/watch?v=KU9Cn59-4vw&ab_channel=GauravSen


Max
A
Min
B
C
D
E
F
G
-1
8
-3
-1
2
1
-3
4

ALPHA–BETA PRUNING

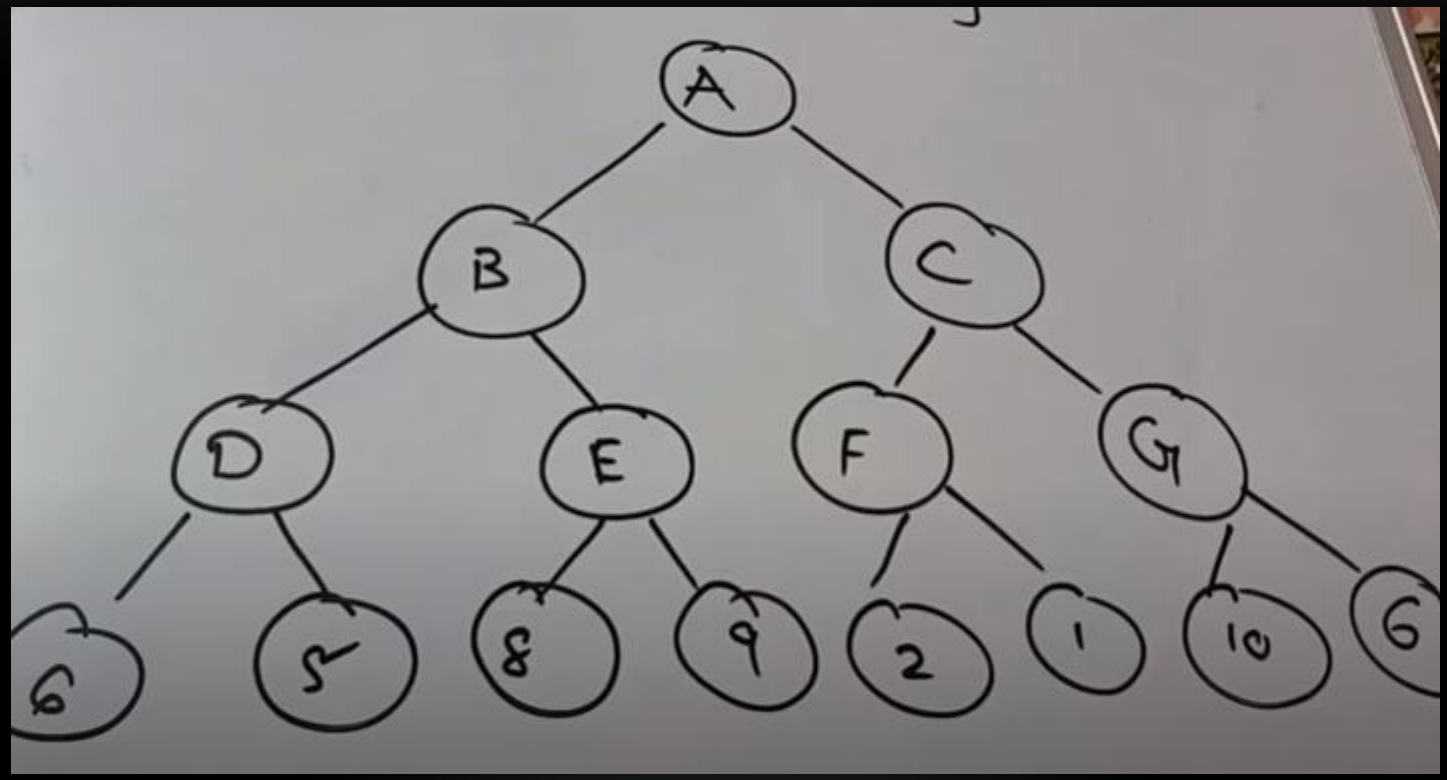
- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree
- The trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree
- we can borrow the idea of **pruning** to eliminate large parts of the tree from consideration.
- The particular technique we examine is called **alpha– beta pruning**
- When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

- https://www.youtube.com/watch?v=X_rfBIjqd-I&ab_channel=BeingPassionateLearner

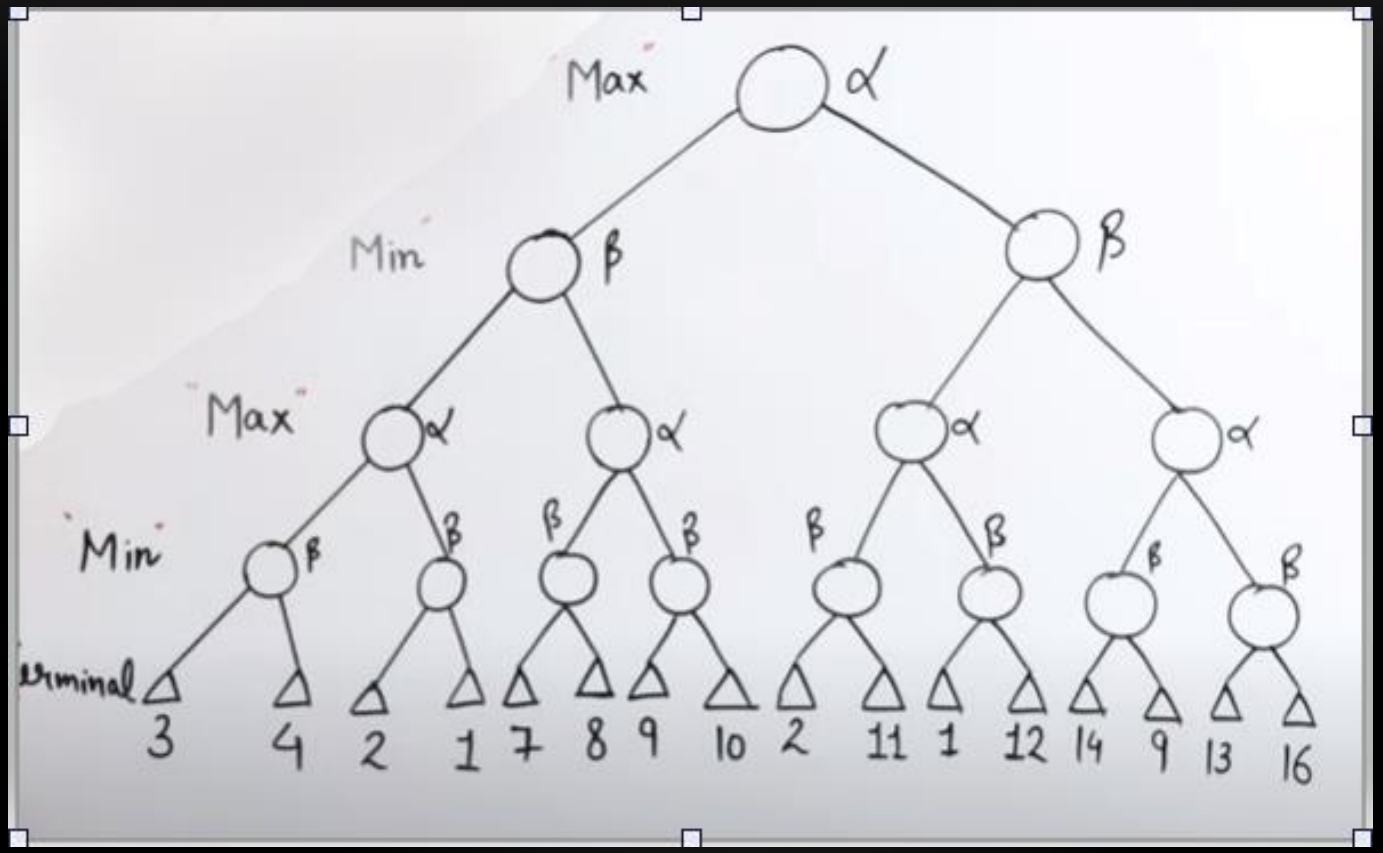
SOME PROPERTIES AND TERMINOLOGY

- The effectiveness of alpha–beta pruning is highly dependent on the order in which the states are examined.
- The best moves are often called **killer moves** and to try them first is called the killer move heuristic.
- In many games, repeated states occur frequently because of **transpositions**—different permutations of the move sequence that end up in the same position.

ANOTHER EXAMPLE



HOMEWORK


Max
α
Min
β
β
Max
α
α
α
α
Min
β
β
β
β
β
β
β
β
Terminal
3
4
2
1
7
8
9
10
2
11
1
12
14
9
13
16

CONSTRAINT SATISFACTION PROBLEMS

WHAT ARE CSP

- We use a **factored representation** for each state: a set of variables, each of which has a value.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called a **constraint satisfaction problem**, or CSP.
- The main idea is to eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints

DEFINING CONSTRAINT SATISFACTION PROBLEMS

- A constraint satisfaction problem consists of three components, X,D, and C:
    - X is a set of variables, {X1, . . . ,Xn}.
    - D is a set of domains, {D1, . . . ,Dn}, one for each variable.
    - C is a set of constraints that specify allowable combinations of values.

- Each domain Di consists of a set of allowable values, {v1, . . . , vk} for variable Xi.
- Each constraint Ci consists of a pair <scope, rel> , where scope is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on.

- A relation can be represented as an explicit list of all tuples of values that satisfy the constraint, or as an abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation.
- For example, if X1 and X2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as <(X1,X2), [(A,B), (B,A)]> or as <(X1,X2), X1 != X2>.
- To solve a CSP, we need to define a state space and the notion of a solution.
- Each state in a CSP is defined by an **assignment** of values to some or all of the variables, {Xi = vi,Xj = vj , . . .}.
- An assignment that does not violate any constraints is called a **consistent** or legal assignment.
- A **complete assignment** is one in which every variable is assigned, and a **solution** to a CSP is a consistent, complete assignment.
- A **partial assignment** is one that assigns values to only some of the variables.

EXAMPLE : MAP COLOURING PROBLEM

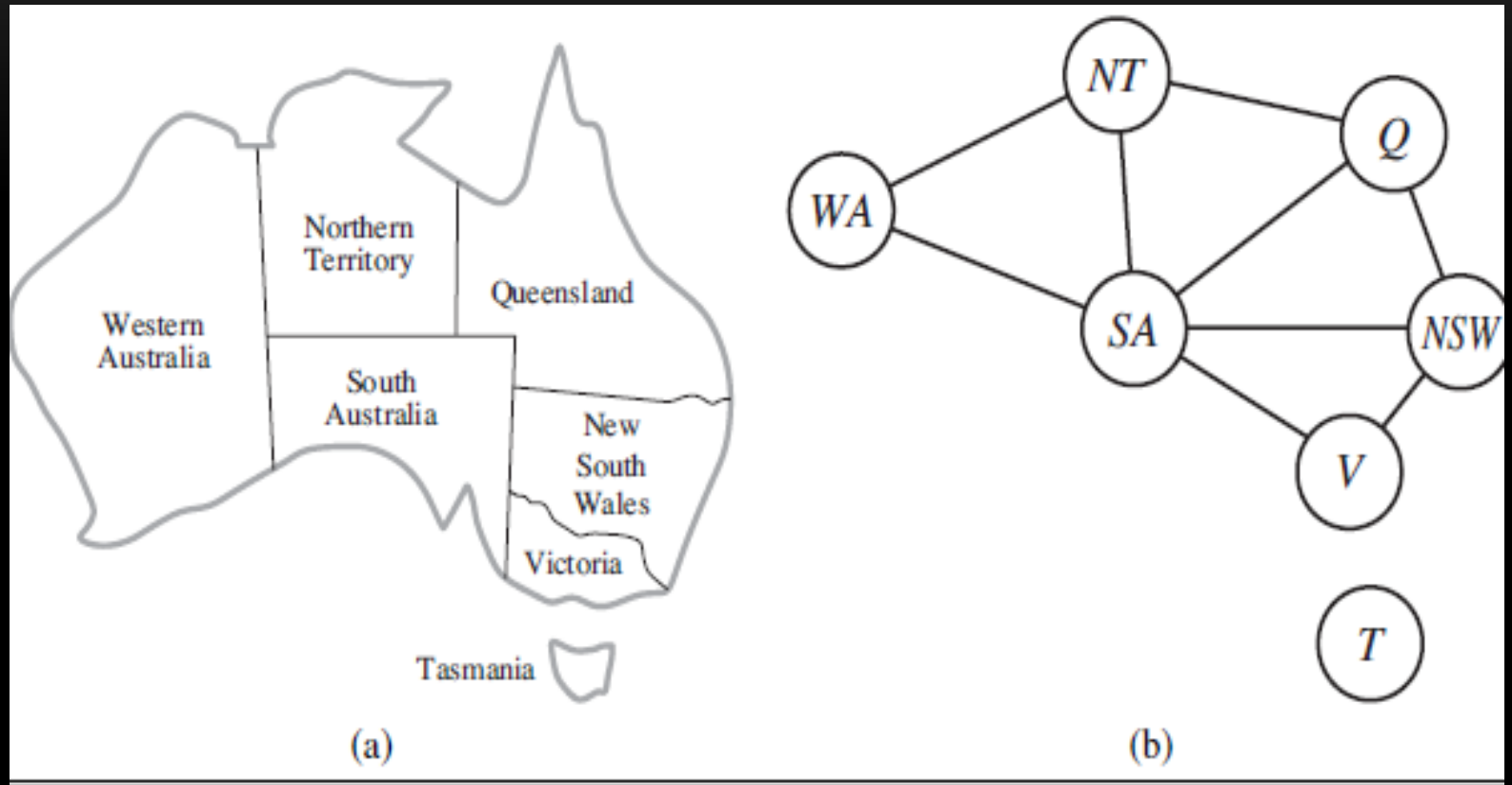


Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

- We are given the task of coloring each region either red, green, or blue in such a way that no neighboring regions have the same color.
- To formulate this as a CSP, we define the variables to be the regions

  X = {WA,NT,Q,NSW, V,SA, T} .
- The domain of each variable is the set

  Di = {red , green, blue}.
- The constraints require neighboring regions to have distinct colors. Since there are nine places where regions border, there are nine constraints:

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$

Here we are using abbreviations; $SA \neq WA$ is a shortcut for $\langle (SA, WA), SA \neq WA \rangle$, where $SA \neq WA$ can be fully enumerated in turn as

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}.$$

There are many possible solutions to this problem, such as

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red\}.$$

- visualize a CSP as a **constraint graph**
- The nodes of the graph correspond to variables of the problem, and a link connects any two variables that participate in a constraint.

VARIATIONS ON THE CSP FORMALISM

- The simplest kind of CSP involves variables that have **discrete**, **finite domains**. Map- coloring problems and scheduling with time limits are both of this kind
- A discrete domain can be **infinite**, such as the set of integers or strings
- Constraint satisfaction problems with **continuous domains are common in real worlds**
- A **binary constraint** relates two variables. For example, SA = NSW is a binary constraint. A binary CSP is one with only binary constraints; it can be represented as a constraint graph
- **Constraint hypergraph** : A hypergraph consists of ordinary nodes (the circles in the figure) and hypernodes (the squares), which represent n-ary constraints

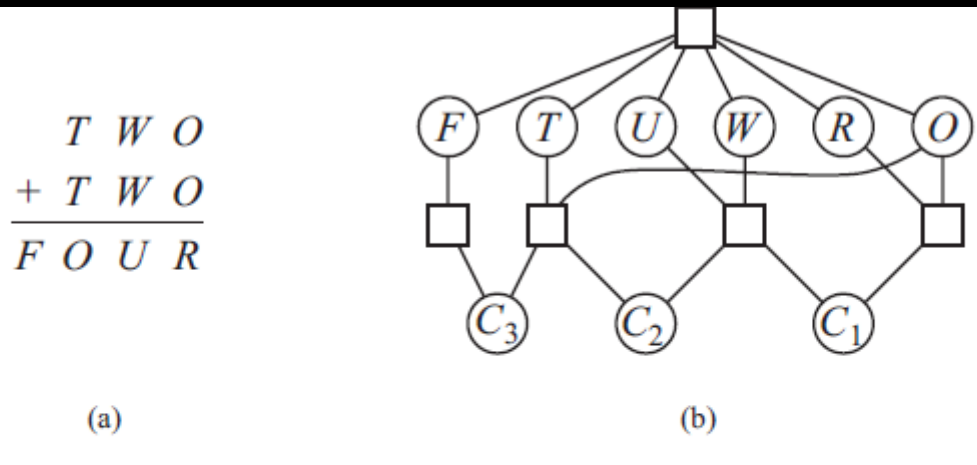


Figure 6.2 (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns.

BACKTRACKING SEARCH FOR CSPS

- Sudoku Problems
- A problem is commutative if the order of application of any given set of actions has no effect on the outcome.
- CSPs are commutative because when assigning values to variables, we reach the same partial assignment regardless of order.
- Therefore, we need only consider a *single* variable at each node in the search tree.
- The term **backtracking search** is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign
- It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution.
- If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

Figure 6.5 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or k-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

VARIABLE AND VALUE ORDERING

- 1. Which variable should be assigned next (SELECT-UNASSIGNED-VARIABLE), and in what order should its values be tried (ORDER-DOMAIN-VALUES)?

The backtracking algorithm contains the line

$$var \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(csp) \ .$$

- The simplest strategy for SELECT-UNASSIGNED-VARIABLE is to choose the next unassigned variable in order, {X1,X2, . . .}.
- This static variable ordering seldom results in the most efficient search.
- The intuitive idea—choosing the variable with the fewest "legal" values—is called the **minimum remaining-values** (MRV) heuristic.
- It also has been called the "most constrained variable" or "fail-first" heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.

INTERLEAVING SEARCH AND INFERENCE

2.What inferences should be performed at each step in the search (INFERENCE)

- One of the simplest forms of inference is called **forward checking**.
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y ’s domain any value that is inconsistent with the value chosen for X.

INTELLIGENT BACKTRACKING: LOOKING BACKWARD

3. When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

- a very simple policy for what to do when a branch of the search fails: back up to the preceding variable and try a different for it. This is called **chronological backtracking** because the *most recent* decision point is revisited.
- A more intelligent approach to backtracking is to backtrack to a variable that might fix the problem—a variable that was responsible for making one of the possible values of SA impossible. (in the Australian Map problem)
- **conflict set** for SA.
- A backjumping algorithm that uses conflict sets defined in this way is called **conflict-directed backjumping**.

CHAPTER 7

KNOWLEDGE BASED AGENTS

Prof. Merin Meleet

INTRODUCTION

- How the intelligence of humans is achieved—not by purely reflex mechanisms but by processes of **reasoning** that operate on internal **representations** of knowledge.
- In AI, this approach to intelligence is embodied in **knowledge-based agents**.

TERMINOLOGY

- The central component of a knowledge-based agent is its **knowledge base**, or KB.
- A knowledge base is a set of **sentences**. (Here "sentence" is used as a technical term.)
- Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world.
- Sometimes we dignify a sentence with the name **axiom**, when the sentence is taken as given without being derived from other sentences.
- **Inference**—deriving new sentences from old.
- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told (or TELLed) to the knowledge base previously.

- Like all our agents, it takes a percept as input and returns an action.
- The agent maintains a knowledge base, KB, which may initially contain some **background knowledge**

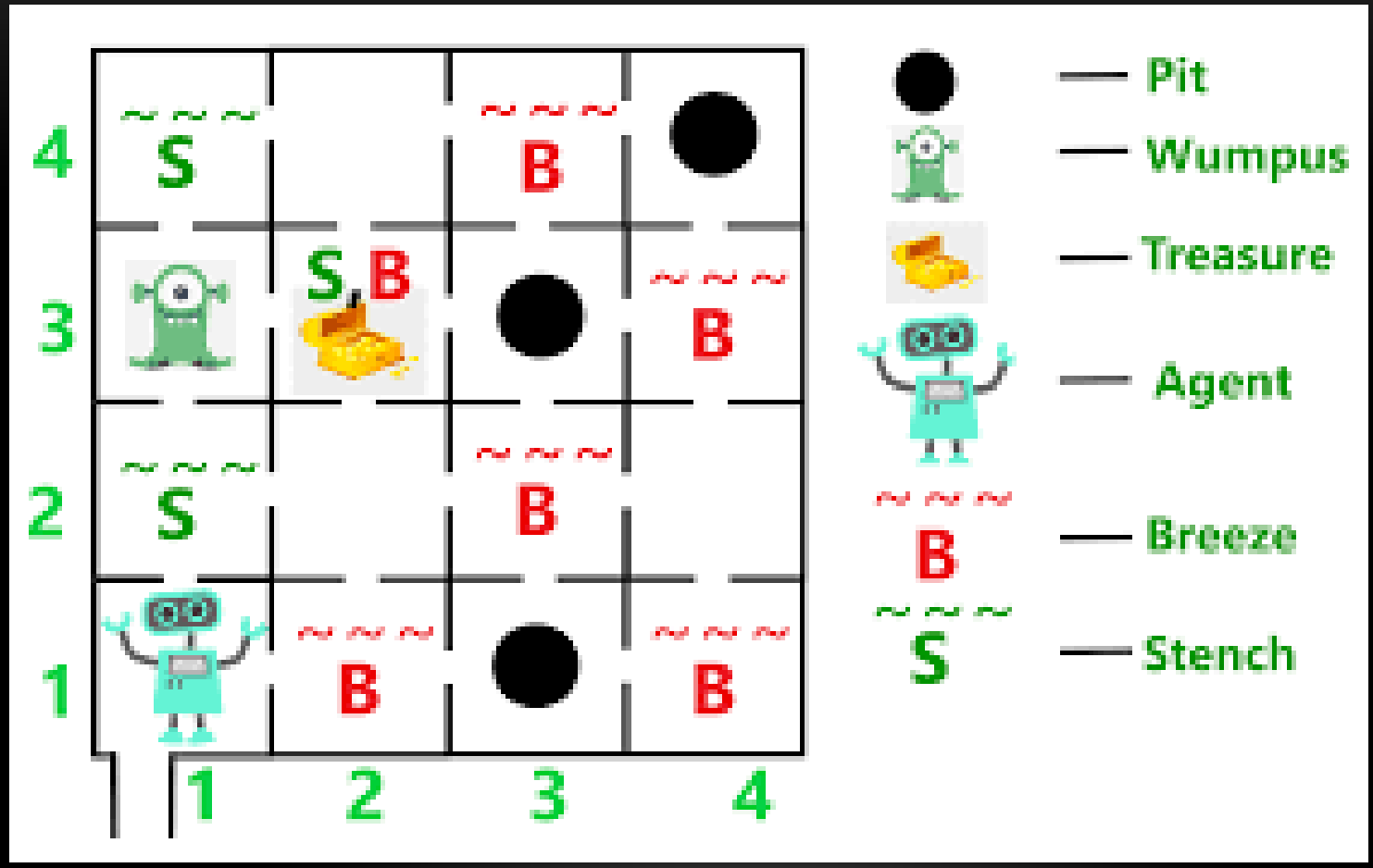
```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

- Each time the agent program is called, it does three things.
  - First, it TELLs the knowledge base what it perceives.
  - Second, it ASKs the knowledge base what action it should perform.
  - In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.
  - Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action.
- ***MAKE-PERCEPT-SENTENCE*** constructs a sentence asserting that the agent perceived the given percept at the given time.
- ***MAKE-ACTION-QUERY*** constructs a sentence that asks what action should be done at the current time.
- ***MAKE-ACTION-SENTENCE*** constructs a sentence asserting that the chosen action was executed.

THE WUMPUS WORLD



https://thiagodnf.github.io/wumpus-world-simulator/

TASK DEFINITION :

- **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.
- **Environment**: A 4×4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square.

- **Actuators**: The agent can move *Forward*, *TurnLeft* by 90◦, or *TurnRight* by 90◦. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and bumps into a wall, then the agent does not move. The action *Grab* can be used to pick up the gold if it is in the same square as the agent. The action *Shoot* can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first *Shoot* action has any effect. Finally, the action *Climb* can be used to climb out of the cave, but only from square [1,1].
- • **Sensors**: The agent has five sensors, each of which gives a single bit of information: – In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a *Stench*. – In the squares directly adjacent to a pit, the agent will perceive a *Breeze*. – In the square where the gold is, the agent will perceive a *Glitter*. – When an agent walks into a wall, it will perceive a *Bump*.

- When the wumpus is killed, it emits a woeful *Scream* that can be perceived anywhere in the cave.
- The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].

- The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].

- The first percept is [None, None, None, None, None], from which the agent can conclude that its neighboring squares, [1,2] and [2,1], are free of dangers—they are OK. <see 1st fig>

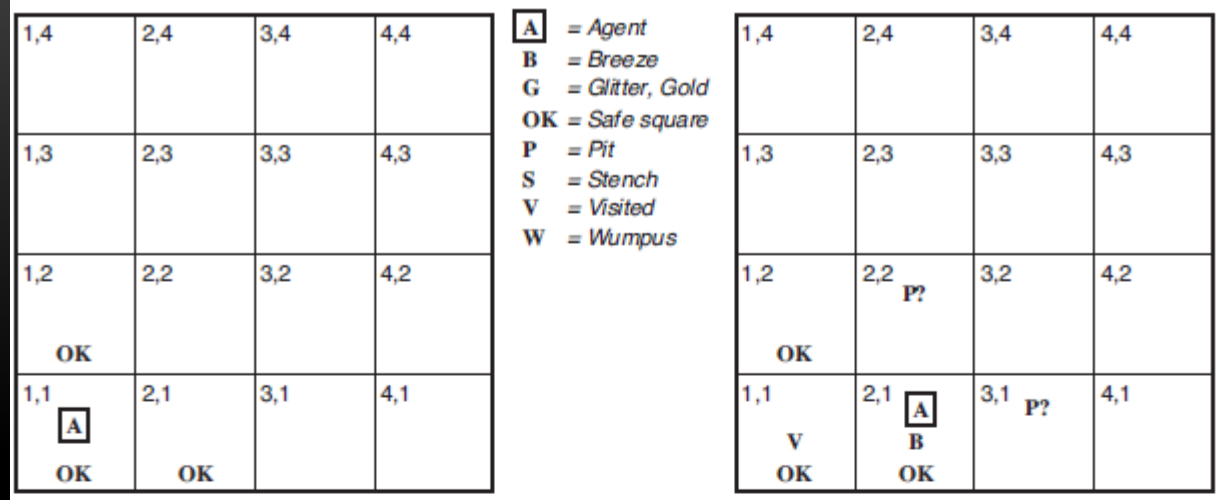


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After one move, with percept [*None*, *Breeze*, *None*, *None*, *None*].

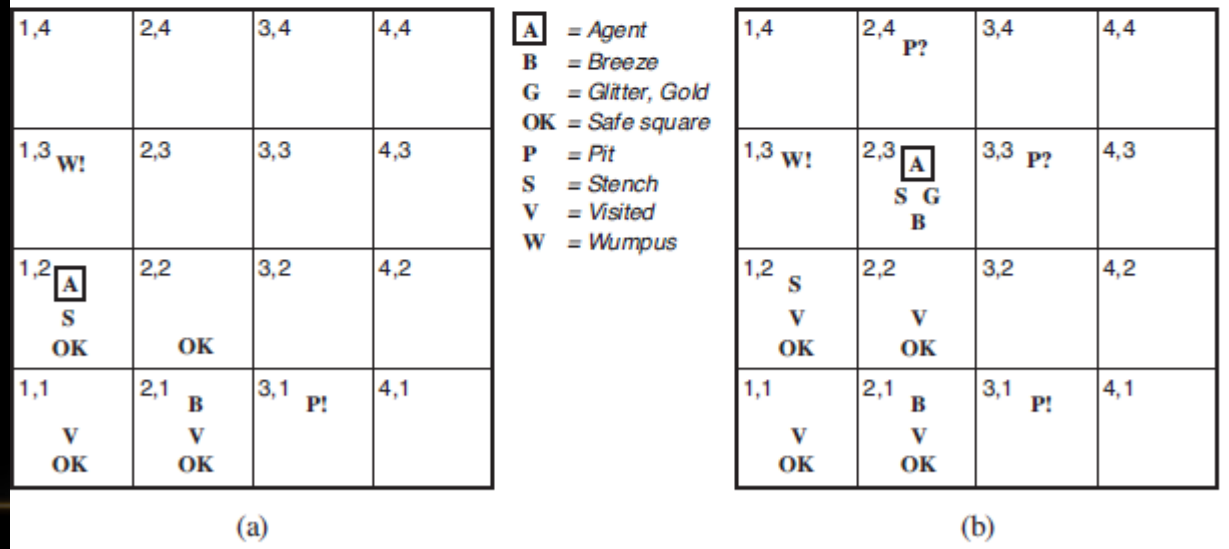


Figure 7.4 Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench*, *None*, *None*, *None*, *None*]. (b) After the fifth move, with percept [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

LOGIC

- The sentences are expressed according to the syntax of the representation language, which specifies all the sentences that are well formed.
- The notion of syntax is clear enough in ordinary arithmetic: “x + y = 4” is a well-formed sentence, whereas “x4y+ =” is not.
- A logic must also define the **semantics** or meaning of sentences. The semantics defines the **truth** of each sentence with respect to each **possible world**
- we use the term **model** in place of “possible world.”
- Models are mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence
- If a sentence α is true in model m, we say that m **satisfies** α or sometimes m **is a model of** α. We use the notation M(α) to mean the set of all models of α.

- The relation of logical **entailment** between sentences—the idea that a sentence *follows logically* from another sentence. In mathematical notation, we write

$\alpha \models \beta$ <read as entails>

- The formal definition of entailment is this:

$\alpha \models \beta$ if and only if, in every model in which α is true, β is also true. Using the notation just introduced, we can write

$\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.

PROPOSITIONAL LOGIC: A VERY SIMPLE LOGIC

- The **syntax** of propositional logic defines the allowable sentences.
- The **atomic sentences c**onsist of a single **proposition symbol**.
- Each such symbol stands for a proposition that can be true or false.
- Eg: we use $W_{1,3}$ to stand for the proposition that the wumpus is in [1,3].
- **Complex sentences** are constructed from simpler sentences, using parentheses and **logical connectives**.
- There are five connectives in common use:

CONNECTIVES

- $\neg$ (not). A sentence such as $\neg W_{1,3}$ is called the **negation** of $W_{1,3}$. A **literal** is either an atomic sentence (a **positive literal**) or a negated atomic sentence (a **negative literal**).
- $\wedge$ (and). A sentence whose main connective is $\wedge$, such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction**; its parts are the **conjuncts**. (The $\wedge$ looks like an "A" for "And.")
- $\vee$ (or). A sentence using $\vee$, such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction** of the **disjuncts** $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$. (Historically, the $\vee$ comes from the Latin "vel," which means "or." For most people, it is easier to remember $\vee$ as an upside-down $\wedge$.)
- $\Rightarrow$ (implies). A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as $\supset$ or $\rightarrow$.
- $\Leftrightarrow$ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional**. Some other books write this as $\equiv$.

SEMANTICS

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- In propositional logic, a model simply fixes the **truth value**—true or false—for every proposition symbol.
- For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is

$$m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\} .$$

With three proposition symbols, there are $2^3 = 8$ possible models

- Atomic sentences are easy:
  - True is true in every model and False is false in every model.
  - The truth value of every other proposition symbol must be specified directly in the model.
- For complex sentences, we have five rules, which hold for any subsentences P and Q in any model m (here “iff” means “if and only if”):
  - •¬P is true iff P is false in m.
  - P ∧ Q is true iff both P and Q are true in m.
  - P ∨ Q is true iff either P or Q is true in m.
  - P ⇒ Q is true unless P is true and Q is false in m.
  - •P ⇔ Q is true iff P and Q are both true or both false in m.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*.

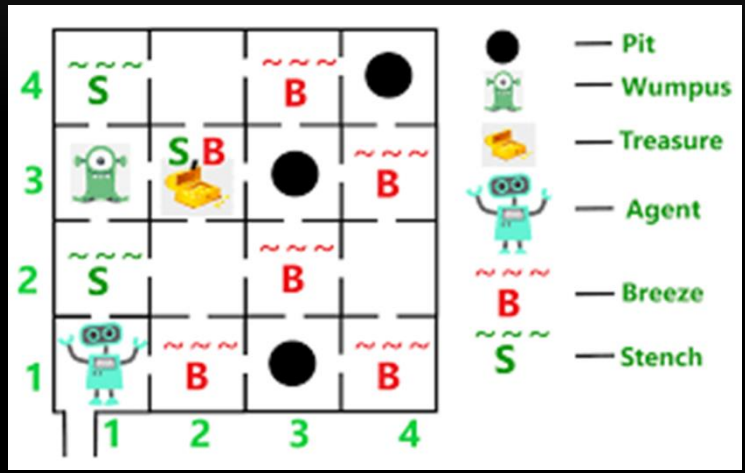
- For example, a square is breezy *if* a neighboring square has a pit, and a square is breezy *only if* a neighboring square has a pit.
- So we need a biconditional,

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}) ,$$

where $B_{1,1}$ means that there is a breeze in [1,1].

A SIMPLE KNOWLEDGE BASE

- Now we can construct a knowledge base for the wumpus world.
- For now, we need the following symbols for each [x, y] location:
  - $P_{x,y}$ is true if there is a pit in [x, y].
  - $W_{x,y}$ is true if there is a wumpus in [x, y], dead or alive.
  - $B_{x,y}$ is true if the agent perceives a breeze in [x, y].
  - $S_{x,y}$ is true if the agent perceives a stench in [x, y].
- We label each sentence Ri so that we can refer to them:
  - There is no pit in [1,1]: $R_1 : \neg P_{1,1}$.



- • A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:
  - $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
  - $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.

PROPOSITIONAL THEOREM PROVING

- How entailment can be done by **theorem proving**—applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.
- The first concept is **logical equivalence**: two sentences α and β are logically equivalent if they are true in the same set of models.
- We write this as $\alpha \equiv \beta$. For example, we can easily show (using truth tables) that $P \wedge Q$ and $Q \wedge P$ are logically equivalent;
- An alternative definition of equivalence is as follows: any two sentences α and β are equivalent only if each of them entails the other:

$$\alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha .$$

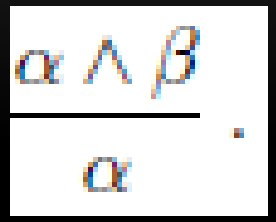
- The second concept we will need is **validity**.
- A sentence is valid if it is true in *all* models.
- For example, the sentence $P \vee \neg P$ is valid.
- Valid sentences are also known as **tautologies**—they are *necessarily* true.
- Because the sentence True is true in all models, every valid sentence is logically equivalent to True.
- From our definition of entailment, we can derive the **deduction theorem**, which was known to the ancient Greeks:
- *For any sentences* α *and* β, $\alpha \models \beta$ *if and only if the sentence* $(\alpha \Rightarrow \beta)$ *is valid.*
- The final concept we will need is **satisfiability**.
- A sentence is satisfiable if it is true in, or satisfied by, *some* model

- **inference rules** that can be applied to derive a **proof**—a chain of conclusions that leads to the desired goal.
- The best-known rule is called **Modus Ponens** (Latin for *mode that affirms*) and is written

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}.$$

- The notation means that, whenever any sentences of the form α ⇒ β and α are given, then the sentence β can be inferred. For example, if (WumpusAhead ∧WumpusAlive) ⇒ Shoot and (WumpusAhead ∧ WumpusAlive) are given, then Shoot can be inferred.

- Another useful inference rule is **And-Elimination**, which says that, from a conjunction, any of the conjuncts can be inferred:



- For example, from (WumpusAhead ∧ WumpusAlive), WumpusAlive can be inferred

- To find a sequence of steps that constitutes a proof, we just need to define a proof problem as follows:
  - • INITIAL STATE: the initial knowledge base.
  - • ACTIONS: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
  - • RESULT: the result of an action is to add the sentence in the bottom half of the inference rule.
  - • GOAL: the goal is a state that contains the sentence we are trying to prove.

RESOLUTION METHOD / ALGORITHM

- The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause (which indicates contradiction).
- Resolution method is also called **Proof by Refutation**.
- Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal.
- As a result, we have to conclude that the original goal is true.

- That is, to show that KB |= α, we show that (KB ∧ ¬α) is unsatisfiable.
- We do this by proving a contradiction.

Method

- First, (KB ∧ ¬α) is converted into CNF.
- Then, the resolution rule is applied to the resulting clauses.
- Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
- The process continues until one of two things happens:
    - there are no new clauses that can be added, in which case KB does not entail α; or,
    - two clauses resolve to yield the *empty* clause, in which case KB entails α.

NOTE: A sentence expressed as a conjunction of clauses is said to be in **conjunctive normal form** or **CNF**

HORN CLAUSES AND DEFINITE CLAUSES

- **DEFINITE CLAUSE :** a disjunction of literals of which *exactly one is positive*.
- For example, the clause (¬L1,1 ∨¬Breeze ∨B1,1) is a definite clause, whereas (¬B1,1 ∨ P1,2 ∨ P2,1) is not.
- **HORN CLAUSE:** Slightly more general is the **Horn clause**, which is a disjunction of literals of which *at most one is positive*.
- So all definite clauses are Horn clauses, as are clauses with no positive literals; these are called **goal clauses**

SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC

- Models of a logical language are the formal structures that constitute the possible worlds under consideration.
- Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined.
- Thus, models for propositional logic link proposition symbols to predefined truth values.

SAMPLE CASE STUDY

- Figure shows a model with five objects:
  - Richard the Lionheart, King of England from 1189 to 1199;
  - his younger brother, the evil King John, who ruled from 1199 to 1215;
  - the left legs of Richard and John;
  - a crown.
- The objects in the model may be *related* in various ways.
- In the figure, Richard and John are brothers. Formally speaking, a relation is just the set of **tuples** of objects that are related. (A tuple is a collection of objects arranged in a fixed order and is written with angle brackets surrounding the objects.)
- Thus, the brotherhood relation in this model is the set

  { Richard the Lionheart, King John, King John, Richard the Lionheart }

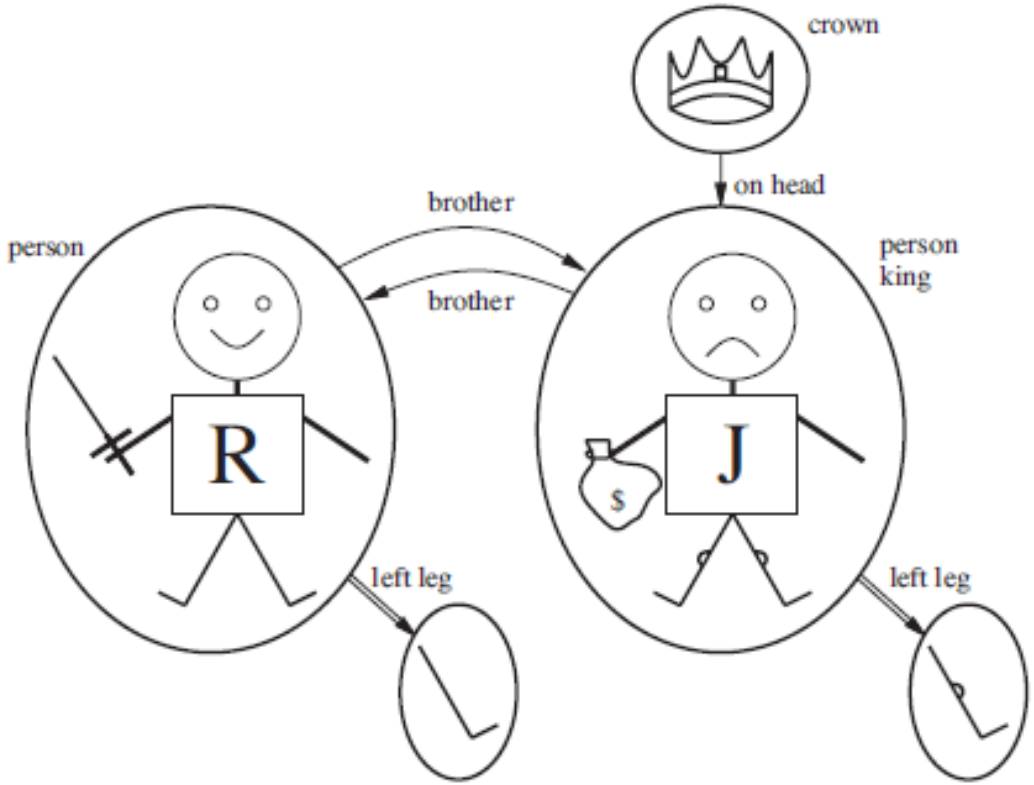


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

The crown is on King John's head, so the "on head" relation contains just one tuple, the crown, King John. The "brother" and "on head" relations are binary relations—that is, they relate pairs of objects.

The model also contains unary relations, or properties: the "person" property is true of both Richard and John; the "king" property is true only of John (presumably because Richard is dead at this point); and the "crown" property is true only of the crown.

SYMBOLS AND INTERPRETATIONS

- The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions.
- The symbols, therefore, come in three kinds: **constant symbols**, which stand for objects; **predicate symbols**, which stand for relations; and **function symbols**, which stand for functions.
- A **term** is a logical expression that refers to an object.
- Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object.
- For example, in English we might use the expression “King John’s left leg” rather than giving a name to his leg.

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
&\mid \neg\ \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term}, \ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\\
\text{OPERATOR PRECEDENCE} &: \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

- An **atomic sentence** (or **atom** for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as

  Brother (Richard , John).

Complex sentences

- We can use **logical connectives** to construct more complex sentences, with the same syntax and semantics to express properties of entire collections of objects, instead of enumerating the objects by name.
- **QUANTIFIER** : to express properties of entire collections of objects, instead of enumerating the objects by name.
- First-order logic contains two standard quantifiers, called *universal* and *existential*.
  - **Universal quantification (∀)**
  - **Existential quantification (∃)**

- **Nested Quantifiers**
- In other cases we will have mixtures. “Everybody loves somebody” means that for every person, there is someone that person loves:

∀ x ∃ y Loves(x, y)

USING FIRST-ORDER LOGIC

- **DOMAIN**
- In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge
- Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called **assertions**.

  TELL(KB, King(John)) .

  TELL(KB, Person(Richard))
- We can ask questions of the knowledge base using ASK.

  ASK(KB, King(John)) returns true.

Questions asked with ASK are called queries or goals.

THE KINSHIP DOMAIN

- domain of family relationships, or kinship.
- This domain includes facts such as “Elizabeth is the mother of Charles”.
- Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother , Sister , Child , Daughter, Son, Spouse etc
- For example, one’s mother is one’s female parent:

$$\forall\, m, c \;\; \text{Mother}(c) = m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c) .$$

- USE OF SETS
- APPLICATIONS IN WUMPUS WORLD

END OF UNIT 2