Unit 4

Knowledge and Reasoning
Table of Contents

Knowledge Representation & Reasoning

- The <u>second</u> most important concept in AI
- If we are going to act rationally in our environment, then we must have some way of describing that environment and drawing inferences from that representation.
    - how do we describe what we know about the world ?
    - how do we describe it *concisely* ?
    - how do we describe it so that we can get hold of the right piece of knowledge when we need it ?
    - how do we generate new pieces of knowledge ?
    - how do we deal with *uncertain* knowledge ?

Knowledge Representation & Reasoning

$$\begin{tikzcd}
 & \text{Knowledge} \arrow[dl, no head] \arrow[dr, no head] & \\
\text{Declarative} & & \text{Procedural}
\end{tikzcd}$$

- Declarative knowledge deals with factoid questions (what is the capital of India? Etc.)
- Procedural knowledge deals with “How”
- Procedural knowledge can be embedded in declarative knowledge

Planning

Given a set of goals, construct a sequence of actions that achieves those goals:

- often very large search space
- but most parts of the world are independent of most other parts
- often start with goals and connect them to actions
- no necessary connection between order of planning and order of execution
- what happens if the world changes as we execute the plan and/or our actions don’t produce the expected results?

Learning

- If a system is going to act truly appropriately, then it must be able to change its actions in the light of experience:
  - how do we generate new facts from old ?
  - how do we generate new concepts ?
  - how do we learn to distinguish different situations in new environments ?

What is knowledge representation?

•Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.

•It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.

•It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent?

Following are the kind of knowledge which needs to be represented in AI systems:

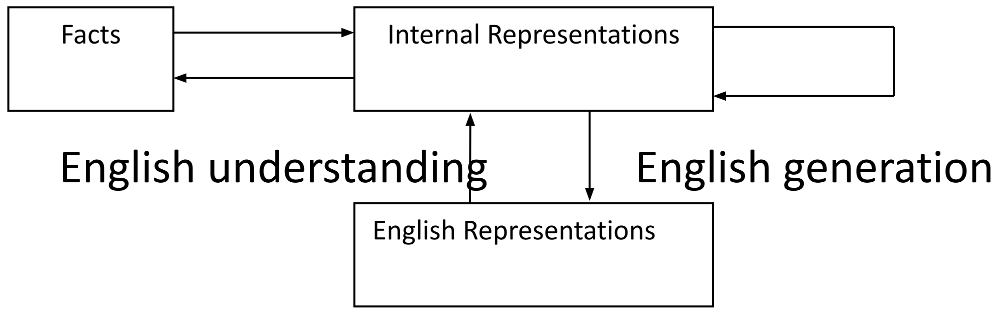
- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Approaches to knowledge Representation

- Representational adequacy the ability to represent all of the kinds of knowledge that are needed in that domain.
- Inferential Adequacy: - the ability to manipulate the representation structures in such a way as to derive new structures corresponding to new knowledge inferred from ol.
- Inferential Efficiency: - the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanism in the most promising directions.
- Acquisitioned Efficiency: - the ability to acquire new information easily. The simplest case involves direct insertion by a person of new knowledge into the database.

Knowledge Representation Issues

- It becomes clear that particular knowledge representation models allow for more specific more powerful problem solving mechanisms that operate on them.
- Examine specific techniques that can be used for representing & manipulating knowledge within programs.
- **Representation & Mapping**
- Facts :- truths in some relevant world
- These are the things we want to represent.
- Representations of facts in some chosen formalism.
- Things we are actually manipulating. Structuring these entities is as two levels.
- The knowledge level, at which facts concluding each agents behavior & current goals are described.



Knowledge and Reasoning
Table of Contents

A KNOWLEDGE-BASED AGENT

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a **sentence**.
- The sentences are expressed in a **knowledge representation language**.
- The agent operates as follows:
  1. It TELLs the knowledge base what it perceives.
  2. It ASKs the knowledge base what action it should perform.
  3. It performs the chosen action.

Requirements for a Knowledge-Based Agent

1. \what it already knows" [McCarthy '59]

A knowledge base of beliefs.

2. \it must rst be capable of being told" [McCarthy '59]

A way to put new beliefs into the knowledge base.

3. \automatically deduces for itself a suciently wide class of immediate consequences" [McCarthy '59]

A reasoning mechanism to derive new beliefs from ones already in the knowledge base.

ARCHITECTURE OF A KNOWLEDGE-BASED AGENT

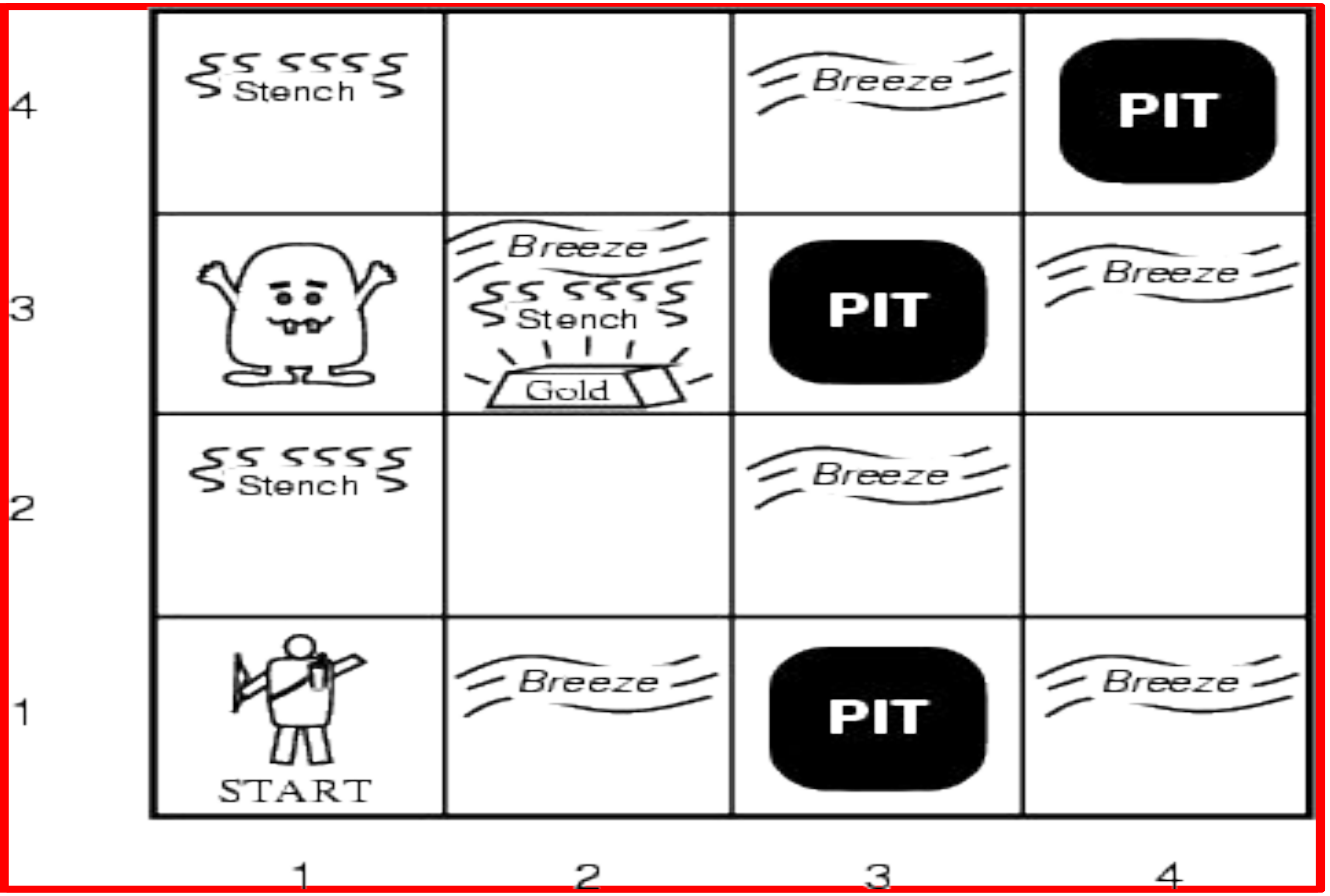
- **Knowledge Level.**
  - The most abstract level: describe agent by saying what it knows.
  - Example: A taxi agent might know that the Golden Gate Bridge connects San Francisco with the Marin County.
- **Logical Level.**
  - The level at which the knowledge is encoded into sentences.
  - Example: Links(GoldenGateBridge, SanFrancisco, MarinCounty).
- **Implementation Level.**
  - The physical representation of the sentences in the logical level.
  - Example: ‘(links goldengatebridge sanfrancisco marincounty)

THE WUMPUS WORLD ENVIRONMENT

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

A TYPICAL WUMPUS WORLD

- The agent always starts in the field [1,1].
- The task of the agent is to find the gold, return to the field [1,1] and climb out of the cave.



AGENT IN A WUMPUS WORLD: PERCEPTS

- The agent perceives
  - a stench in the square containing the Wumpus and in the adjacent squares (not diagonally)
  - a breeze in the squares adjacent to a pit
  - a glitter in the square where the gold is
  - a bump, if it walks into a wall
  - a woeful scream everywhere in the cave, if the wumpus is killed
- The percepts are given as a five-symbol list. If there is a stench and a breeze, but no glitter, no bump, and no scream, the percept is

  [Stench, Breeze, None, None, None]

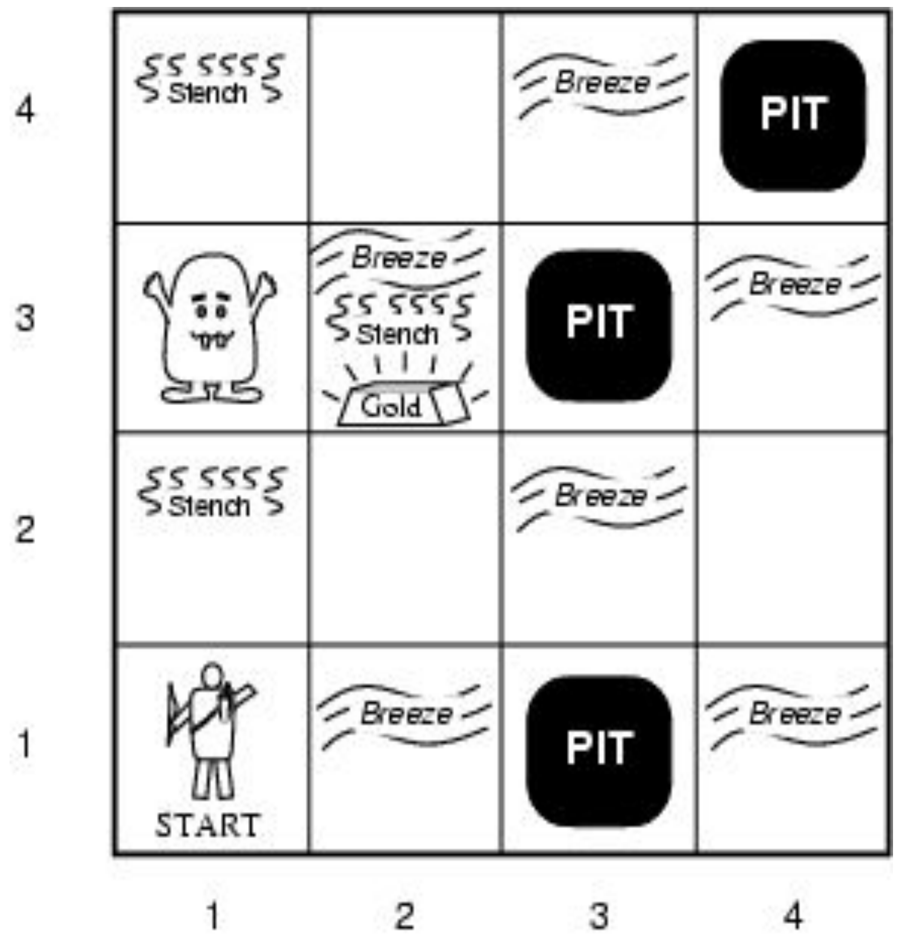
WUMPUS WORLD ACTIONS

- **go forward**
- **turn right** 90 degrees
- **turn left** 90 degrees
- **grab**: Pick up an object that is in the same square as the agent
- **shoot**: Fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits and kills the wumpus or hits the outer wall. The agent has only one arrow, so only the first Shoot action has any effect
- **climb** is used to leave the cave. This action is only effective in the start square
- **die**: This action automatically and irretrievably happens if the agent enters a square with a pit or a live wumpus

ILLUSTRATIVE EXAMPLE: WUMPUS WORLD

- **Performance measure**
  - gold +1000,
  - death -1000

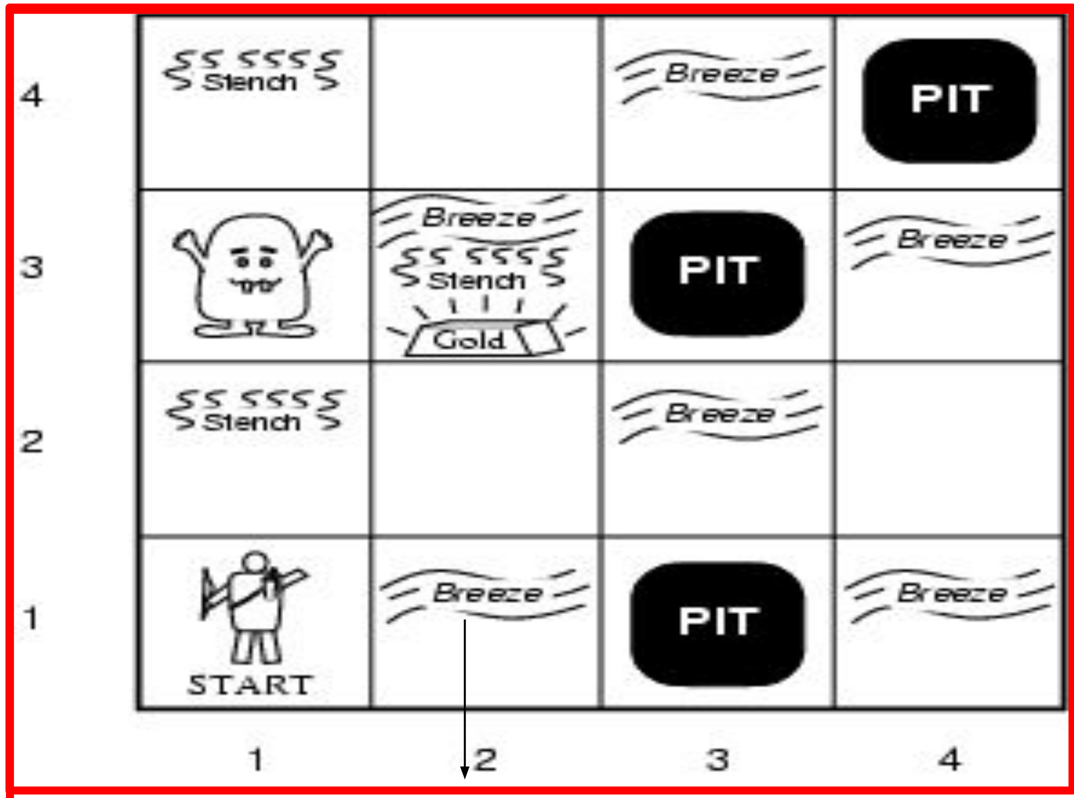
  (falling into a pit or being eaten by the wumpus)
  - -1 per step, -10 for using the arrow
- **Environment**
  - Rooms / squares connected by doors.
  - Squares adjacent to wumpus are smelly
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square
  - Randomly generated at start of game. Wumpus only senses current room.
- **Sensors:** Stench, Breeze, Glitter, Bump, Scream [perceptual inputs]
- **Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot



WUMPUS WORLD CHARACTERIZATION

Fully Observable	No – only local perception
Deterministic	Yes – outcomes exactly specified
Static	Yes – Wumpus and Pits do not move
Discrete	Yes
Single-agent?	Yes – Wumpus is essentially a “natural feature.”

EXPLORING A WUMPUS WORLD



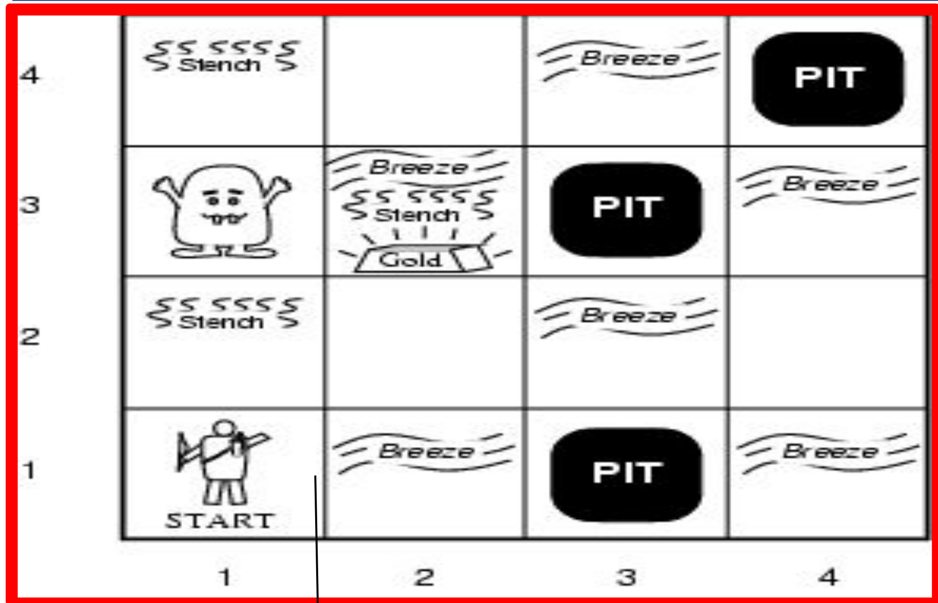
The knowledge base of the agent consists of the rules of the Wumpus world plus the percept "nothing" in [1,1]

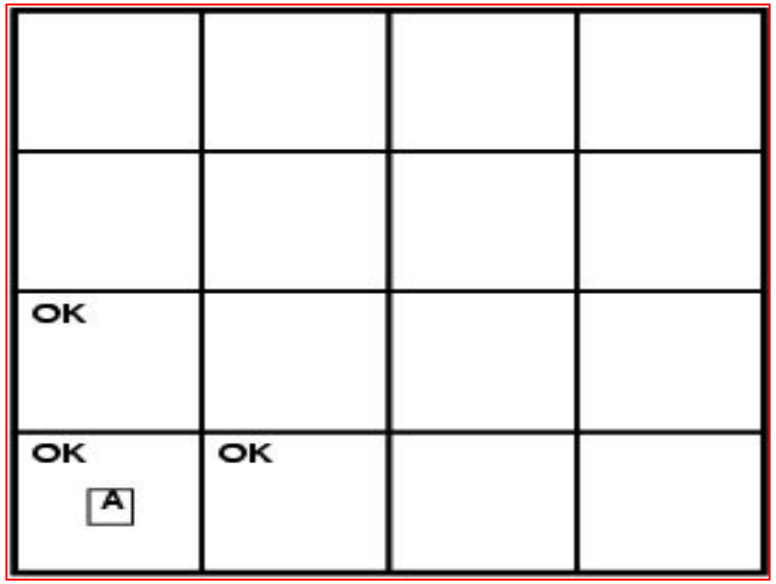
Boolean percept feature values: <0, 0, 0, 0, 0>

None, none, none, none, none

Stench, Breeze, Glitter, Bump, Scream

EXPLORING A WUMPUS WORLD





None, none, none, none, none

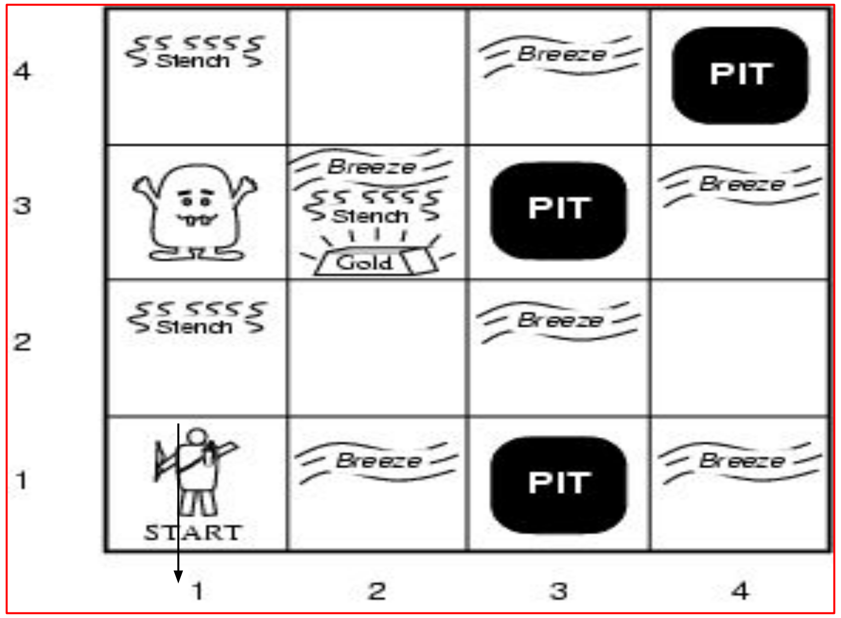
Stench, Breeze, Glitter, Bump, Scream

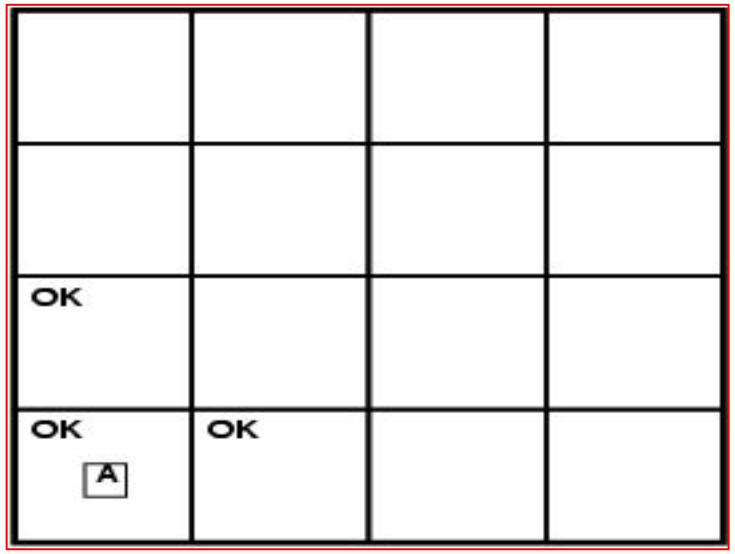
World "known" to agent at time = 0.

T=0 The KB of the agent consists of the rules of the Wumpus world plus the percept "nothing" in [1,1].
By inference, the agent's knowledge base also has the information that [2,1] and [1,2] are okay.
Added as propositions.

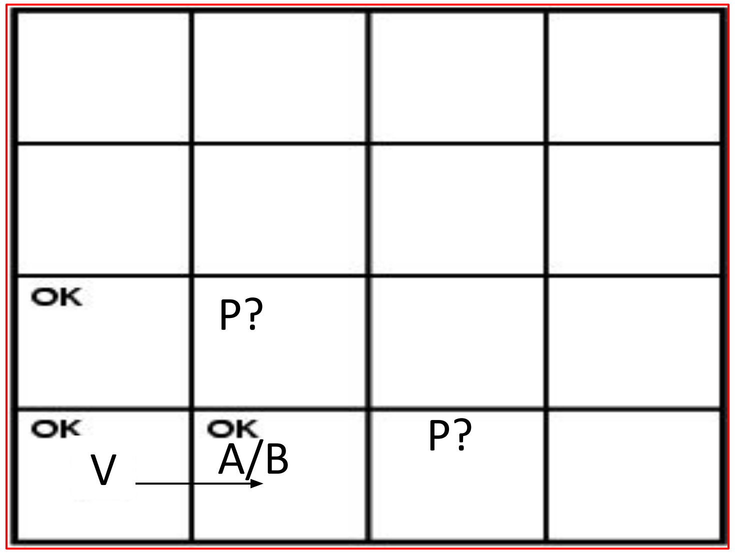
EXPLORING A WUMPUS WORLD

T = 0





T = 1



None, breeze, none, none, none

None, none, none, none, none

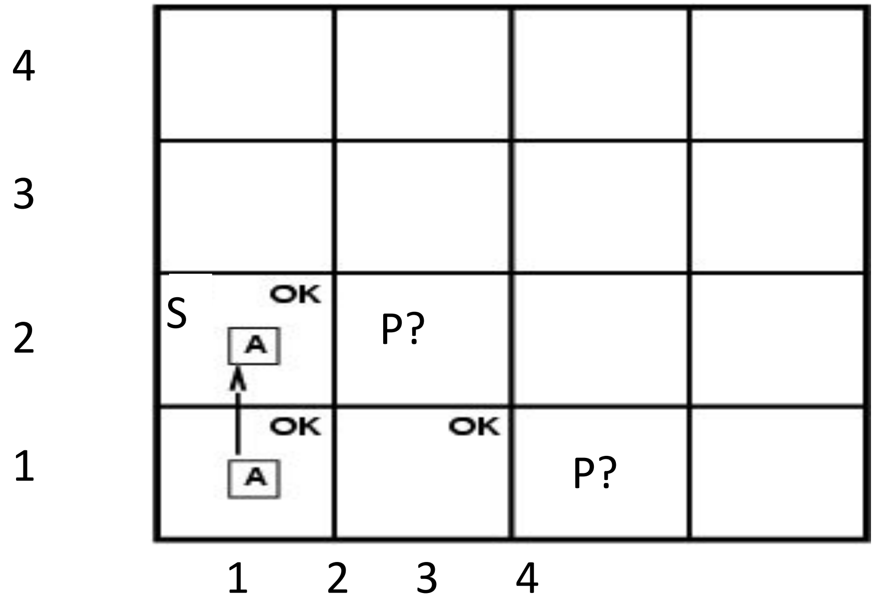
Stench, Breeze, Glitter, Bump, Scream

A – agent
V – visited
B - breeze

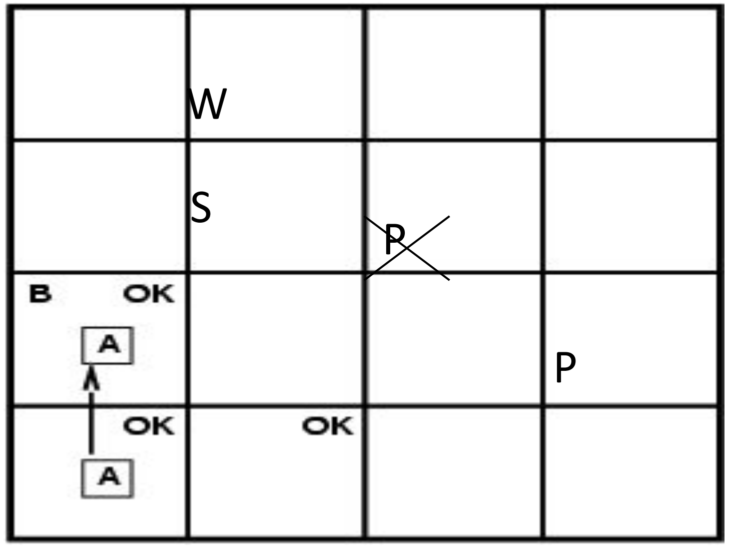
Where next?

@ T = 1 What follows?
Pit(2,2) or Pit(3,1)

EXPLORING A WUMPUS WORLD



T=3



Stench, none, none, none, none

Stench, Breeze, Glitter, Bump, Scream

Where is Wumpus?

Wumpus cannot be in (1,1) or in (2,2) (Why?) Wumpus in (1,3)
Not breeze in (1,2) no pit in (2,2); but we know there is
pit in (2,2) or (3,1) pit in (3,1)

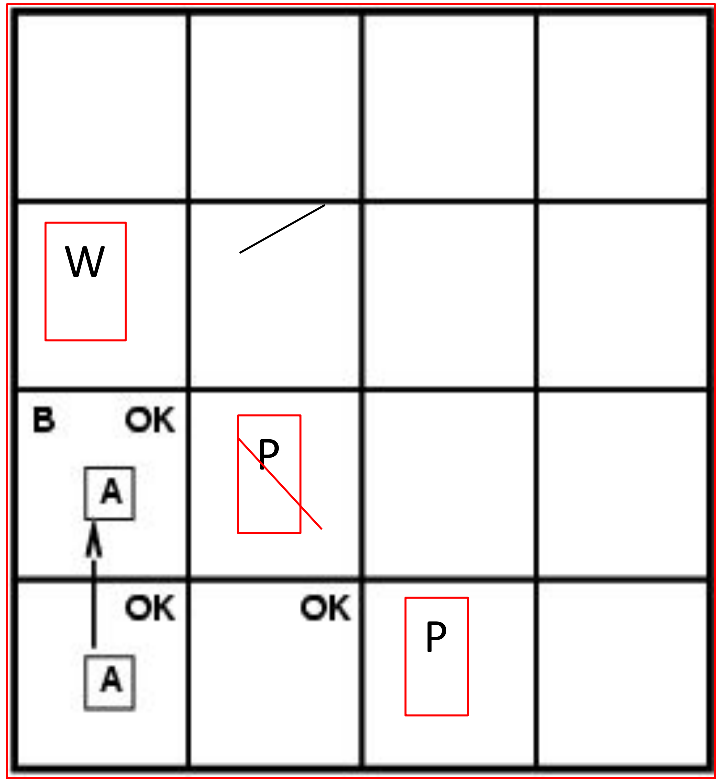
EXPLORING A WUMPUS WORLD

We reasoned about the possible states the Wumpus world can be in, given our percepts and our knowledge of the rules of the Wumpus world.
I.e., the content of KB at T=3.

What follows is what holds true in all those worlds that satisfy what is known at that time T=3 about the particular Wumpus world we are in.

Example property: P_in_(3,1)

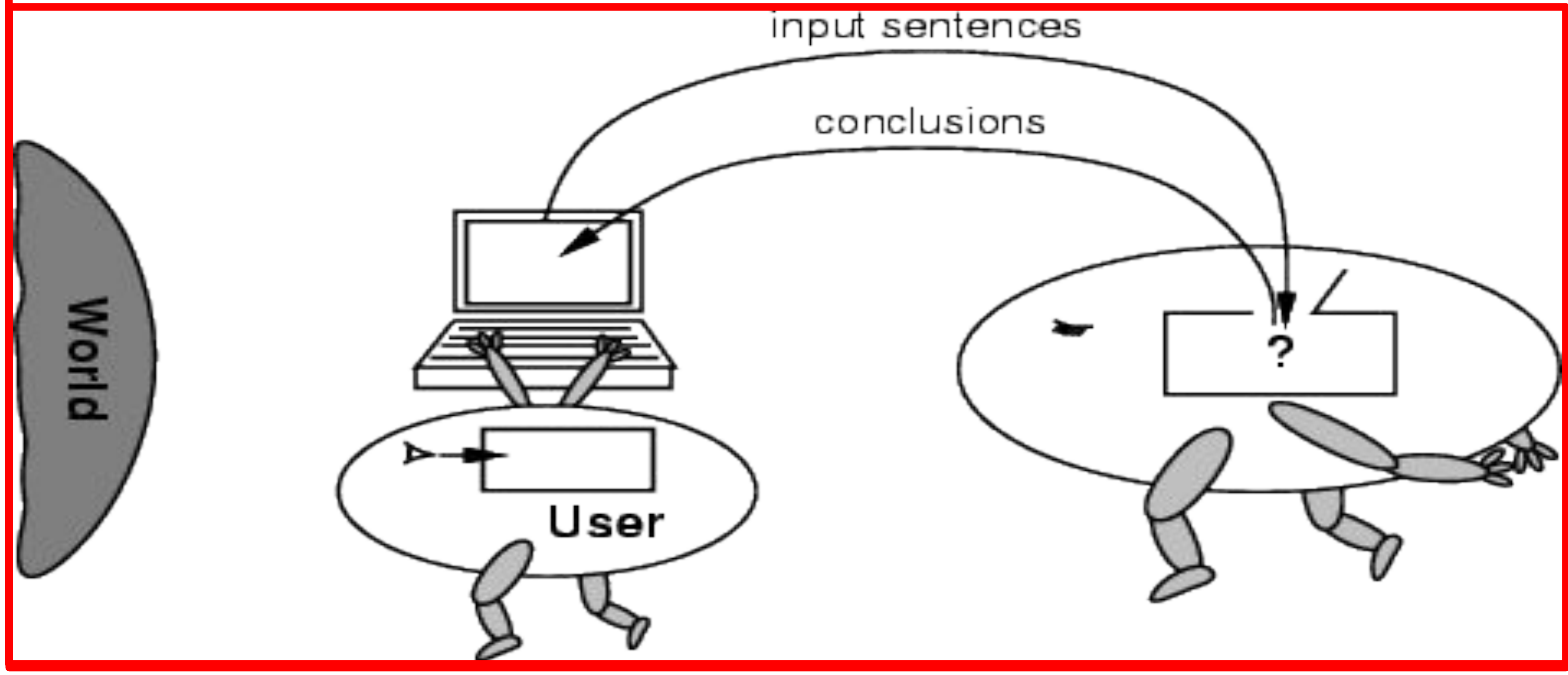
Models(KB) $\subseteq$ Models(P_in_(3,1))



Essence of logical reasoning:
Given *all we know*, Pit_in_(3,1) holds.
("The world cannot be different.")

NO INDEPENDENT ACCESS TO THE WORLD

- The reasoning agent often gets its knowledge about the facts of the world as a sequence of logical sentences and must draw conclusions only from them without independent access to the world.
- Thus it is very important that the agent’s reasoning is sound!



SUMMARY OF KNOWLEDGE BASED AGENTS

- Intelligent agents need knowledge about the world for making good decisions.
- The knowledge of an agent is stored in a knowledge base in the form of **sentences** in a knowledge representation language.
- A knowledge-based agent needs a **knowledge base** and an **inference mechanism**. It operates by storing sentences in its knowledge base, inferring new sentences with the inference mechanism, and using them to deduce which actions to take.
- A **representation language** is defined by its syntax and semantics, which specify the structure of sentences and how they relate to the facts of the world.
- The **interpretation** of a sentence is the fact to which it refers. If this fact is part of the actual world, then the sentence is true.

Knowledge and Reasoning
Table of Contents

What is a Logic?

- A language with concrete rules
  - No ambiguity in representation (may be other errors!)
  - Allows unambiguous communication and processing
  - Very unlike natural languages e.g. English
- Many ways to translate between languages
  - A statement can be represented in different logics
  - And perhaps differently in same logic
- **Expressiveness** of a logic
  - How much can we say in this language?
- Not to be confused with logical reasoning
  - Logics are languages, reasoning is a process (may **use** logic)

Syntax and Semantics

- Syntax
  - Rules for constructing legal sentences in the logic
  - Which symbols we can use (English: letters, punctuation)
  - How we are allowed to combine symbols
- Semantics
  - How we interpret (read) sentences in the logic
  - Assigns a meaning to each sentence
- Example: “All lecturers are seven foot tall”
  - A valid sentence (syntax)
  - And we can understand the meaning (semantics)
  - This sentence happens to be false (there is a counterexample)

Propositional Logic

- Syntax
  - Propositions, e.g. “it is wet”
  - Connectives: and, or, not, implies, iff (equivalent)

    $\wedge \quad \vee \quad \neg \quad \rightarrow \quad \leftrightarrow$
  - Brackets, T (true) and F (false)
- Semantics (Classical AKA Boolean)
  - Define how connectives affect truth
    - “P and Q” is true if and only if P is true and Q is true
  - Use **truth tables** to work out the truth of statements

Predicate Logic

- Propositional logic combines atoms
  - An atom contains no propositional connectives
  - Have no structure (today_is_wet, john_likes_apples)
- **Predicates** allow us to talk about objects
  - Properties: is_wet(today)
  - Relations: likes(john, apples)
  - True or false
- In predicate logic each atom is a predicate
  - e.g. first order logic, higher-order logic

First Order Logic

- More expressive logic than propositional
  - Used in this course (Lecture 6 on representation in FOL)
- **Constants** are objects: john, apples
- **Predicates** are properties and relations:
  - likes(john, apples)
- **Functions** transform objects:
  - likes(john, fruit_of(apple_tree))
- **Variables** represent any object: likes(X, apples)
- **Quantifiers** qualify values of variables
  - True for all objects (Universal): $\forall$X. likes(X, apples)
  - Exists at least one object (Existential): $\exists$X. likes(X, apples)

Example: FOL Sentence

- “Every rose has a thorn”

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

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

Example: FOL Sentence

- "On Mondays and Wednesdays I go to John's house for dinner"

$$\forall X.((is\_mon(X) \vee is\_wed(X)) \rightarrow eat\_meal(me, houseOf(john), X))$$

- Note the change from "and" to "or"
  - Translating is problematic

Higher Order Logic

- More expressive than first order
- Functions and predicates are also objects
  - Described by predicates: binary(addition)
  - Transformed by functions: differentiate(square)
  - Can quantify over both
- E.g. define red functions as having zero at 17

$$\forall F.(red(F) \leftrightarrow F(0) = 17)$$

- Much harder to reason with

Beyond True and False

- Multi-valued logics
  - More than two truth values
  - e.g., true, false & unknown
  - **Fuzzy logic** uses probabilities, truth value in [0,1]
- Modal logics
  - Modal operators define mode for propositions
  - **Epistemic logics** (belief)
    - e.g. □p (necessarily p), ◊p (possibly p), …
  - **Temporal logics** (time)
    - e.g. □p (always p), ◊p (eventually p), …

Knowledge and Reasoning
Table of Contents

Knowledge and Reasoning
Table of Contents

Propositional logic

- Propositional logic consists of:
  - The logical values true and false (T and F)
  - Propositions: “Sentences,” which
    - Are atomic (that is, they must be treated as indivisible units, with no internal structure), and
    - Have a single logical value, either true or false
  - Operators, both unary and binary; when applied to logical values, yield logical values
    - The usual operators are and, or, not, and implies

Truth tables

- Logic, like arithmetic, has operators, which apply to one, two, or more values (operands)
- A truth table lists the results for each possible arrangement of operands
  - Order is important: ***x op y*** may or may not give the same result as ***y op x***
- The rows in a truth table list all possible sequences of truth values for n operands, and specify a result for each sequence
  - Hence, there are 2^n rows in a truth table for n operands

Unary operators

- There are four possible unary operators:

X	Constant true, (T)
T	T
F	T

X	Identity, (X)
T	T
F	F

X	Constant false, (F)
T	F
F	F

X	**Negation, ¬X**
T	F
F	T

- Only the last of these (negation) is widely used (and has a symbol,¬ ,for the operation

Combined tables for unary operators

X	Constant T	Constant F	Identity	¬X
T	T	F	T	F
F	T	F	F	T

Binary operators

- There are sixteen possible binary operators:

X	Y																
T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F

- All these operators have names, but I haven't tried to fit them in
- Only a few of these operators are normally used in logic

Useful binary operators

- Here are the binary operators that are traditionally used:

X	Y	AND $X \land Y$	OR $X \lor Y$	IMPLIES $X \Rightarrow Y$	BICONDITIONAL $X \Leftrightarrow Y$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

- Notice in particular that material implication ($\Rightarrow$) only approximately means the same as the English word “implies”
- All the other operators can be constructed from a combination of these (along with unary not, $\neg$)

Logical expressions

- All logical expressions can be computed with some combination of and ($\wedge$), or ($\vee$), and not ($\neg$) operators
- For example, logical implication can be computed this way:

X	Y	$\neg X$	$\neg X \vee Y$	$X \Rightarrow Y$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

- Notice that $\neg X \vee Y$ is equivalent to $X \Rightarrow Y$

X	Y	¬X	X <=Y> = X =>Y AND Y => X	X <=Y> = X =>Y AND Y => X	X <=Y> = X =>Y AND Y => X
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	T	F	F
F	F	T	T	T	T

Another example

- Exclusive or (xor) is true if exactly one of its operands is true

X	Y	¬X	¬Y	¬X ∧ Y	X ∧ ¬Y	(¬X∧Y)∨(X∧¬Y)	X xor Y
T	T	F	F	F	F	F	F
T	F	F	T	F	T	T	T
F	T	T	F	T	F	T	T
F	F	T	T	F	F	F	F

- Notice that (¬X∧Y)∨(X∧¬Y) is equivalent to X xor Y

Given a propositional logic, check if it is valid

- (X-> Y o (x v y))
- Valid

X	Y	X->Y	X v Y	X->Y or X v Y
0	0	1	0	1
0	1	0	1	1
1	0	1	1	1
1	1	1	1	1

World

- A world is a collection of prepositions and logical expressions relating those prepositions
- Example:
  - Propositions: JohnLovesMary, MaryIsFemale, MaryIsRich
  - Expressions:
    MaryIsFemale ∧ MaryIsRich ⇒ JohnLovesMary
- A proposition “says something” about the world, but since it is atomic (you can’t look inside it to see component parts), propositions tend to be very specialized and inflexible

Models

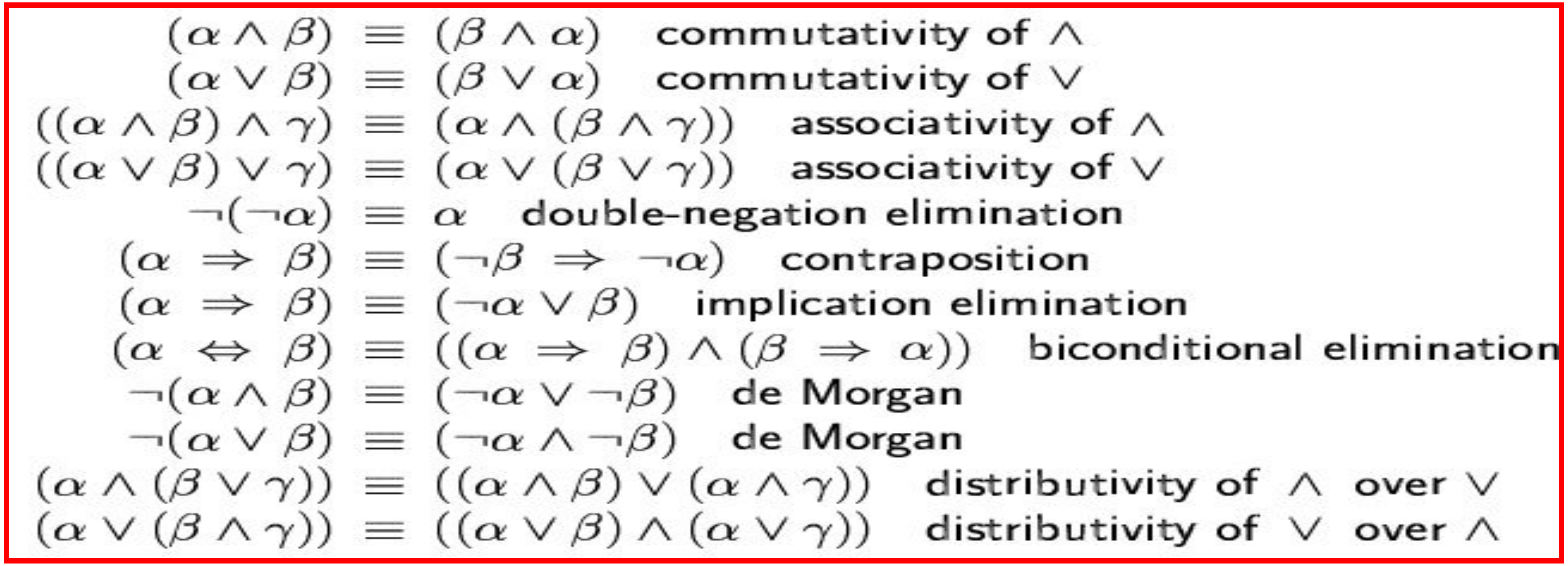
A model is an assignment of a truth value to each proposition, for example:

- JohnLovesMary: T, MaryIsFemale: T, MaryIsRich: F

- An expression is satisfiable if there is a model for which the expression is true
  - For example, the above model satisfies the expression
    MaryIsFemale ∧ MaryIsRich ⇒ JohnLovesMary
- An expression is valid if it is satisfied by *every* model
  - This expression is *not* valid:
    MaryIsFemale ∧ MaryIsRich ⇒ JohnLovesMary
    because it is not satisfied by this model:
    JohnLovesMary: F, MaryIsFemale: T, MaryIsRich: T
  - But this expression *is* valid:
    MaryIsFemale ∧ MaryIsRich ⇒ MaryIsFemale

Inference rules in propositional logic

- Here are just a few of the rules you can apply when reasoning in propositional logic:



$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv$	$(\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv$	$(\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv$	$(\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv$	$(\neg\alpha \lor \neg\beta)$	de Morgan
$\neg(\alpha \lor \beta)$	$\equiv$	$(\neg\alpha \land \neg\beta)$	de Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv$	$((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv$	$((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Implication elimination

- A particularly important rule allows you to get rid of the implication operator, ⇒ :
    - $X \Rightarrow Y \equiv \neg X \lor Y$
- We will use this later on as a necessary tool for simplifying logical expressions
- The symbol $\equiv$ means “is logically equivalent to”

Conjunction elimination

- Another important rule for simplifying logical expressions allows you to get rid of the conjunction (**and**) operator, ∧ :
- This rule simply says that if you have an **and** operator at the top level of a fact (logical expression), you can break the expression up into two separate facts:
  - MaryIsFemale ∧ MaryIsRich
    - becomes:
  - MaryIsFemale
  - MaryIsRich

Inference by computer

- To do inference (reasoning) by computer is basically a *search* process, taking logical expressions and applying inference rules to them
  - Which logical expressions to use?
  - Which inference rules to apply?
- Usually you are trying to "prove" some particular statement
- Example:
  - it_is_raining V it_is_sunny
  - it_is_sunny ⇒ I_stay_dry
  - A => B = NOT A V B
  - NOT(it_is_sunny) V I_stay_dry
  - it_is_rainy ⇒ I_take_umbrella
  - NOT(it_is_rainy) V I_take_umbrella
  - I_take_umbrella ⇒ I_stay_dry
  - NOT(I_take_umbrella) V I_stay_dry

  - To prove: I_stay_dry

Knowledge and Reasoning
Table of Contents

Reasoning Patterns

- Inference in propositional logic is NP-complete!
- However, inference in propositional logic shows monoticity:
  - Adding more rules to a knowledge base does not affect earlier inferences

Forward and backward reasoning

- Situation: You have a collection of logical expressions (premises), and you are trying to prove some additional logical expression (the conclusion)
- You can:
  - Do forward reasoning: Start applying inference rules to the logical expressions you have, and stop if one of your results is the conclusion you want
  - Do backward reasoning: Start from the conclusion you want, and try to choose inference rules that will get you back to the logical expressions you have
- With the tools we have discussed so far, *neither* is feasible

Example

- Given:
  - it_is_raining ∨ it_is_sunny
  - it_is_sunny ⇒ I_stay_dry
  - it_is_raining ⇒ I_take_umbrella
  - I_take_umbrella ⇒ I_stay_dry
- You can conclude:
  - it_is_sunny ∨ it_is_raining
  - I_take_umbrella ∨ it_is_sunny
  - ¬I_stay_dry ⇒ I_take_umbrella
  - Etc., etc. ... there are *just too many* things you can conclude!

Predicate calculus

- Predicate calculus is also known as "First Order Logic" (FOL)
- Predicate calculus includes:
  - All of propositional logic
    - Logical values true, false
    - Variables x, y, a, b,...
    - Connectives ¬, ⇒, ∧, ∨, ⇔
  - Constants KingJohn, 2, Villanova,...
  - Predicates Brother, >,...
  - Functions Sqrt, MotherOf,...
  - Quantifiers ∀, ∃

Constants, functions, and predicates

- A constant represents a "thing"--it has no truth value, and it does not occur "bare" in a logical expression
  - Examples: DavidMatuszek, 5, Earth, goodIdea
- Given zero or more arguments, a function produces a constant as its value:
  - Examples: motherOf(DavidMatuszek), add(2, 2), thisPlanet()
- A predicate is like a function, but produces a truth value
  - Examples: greatInstructor(DavidMatuszek), isPlanet(Earth), greater(3, add(2, 2))

Universal quantification

- The universal quantifier, $\forall$, is read as “for each” or “for every”
  - Example: $\forall x,\ x^2 \geq 0$ (for all x, x^2 is greater than or equal to zero)
- Typically, $\Rightarrow$ is the main connective with $\forall$:

  $\forall x,\ at(x, Villanova) \Rightarrow smart(x)$

  means “Everyone at Villanova is smart”
- Common mistake: using $\wedge$ as the main connective with $\forall$:

  $\forall x,\ at(x, Villanova) \wedge smart(x)$

  means “Everyone is at Villanova and everyone is smart”
- If there are no values satisfying the condition, the result is true
  - Example: $\forall x,\ isPersonFromMars(x) \Rightarrow smart(x)$ is true

Existential quantification

- The existential quantifier, $\exists$, is read "for some" or "there exists"
  - Example: $\exists x,\ x^2 < 0$ (there exists an x such that x^2 is less than zero)
- Typically, $\wedge$ is the main connective with $\exists$:

  $\exists x,\ at(x,Villanova) \wedge smart(x)$

  means "There is someone who is at Villanova and is smart"
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

  $\exists x,\ at(x,Villanova) \Rightarrow smart(x)$

  This is true if there is someone at Villanova who is smart...

  ...but it is also true if there is someone who is *not* at Villanova

  By the rules of material implication, the result of $F \Rightarrow T$ is T

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$ LOVES (X,Y)
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$

- $\exists x\ \forall y$ is *not* the same as $\forall y\ \exists x$
- $\exists x\ \forall y\ Loves(x,y)$
  - "There is a person who loves everyone in the world"
  - More exactly: $\exists x\ \forall y\ (person(x) \land person(y) \Rightarrow Loves(x,y))$
- $\forall y\ \exists x\ Loves(x,y)$
  - "Everyone in the world is loved by at least one person"

- **Quantifier duality**: each can be expressed using the other
- $\forall x\ Likes(x,IceCream)$ $\neg\exists x\ \neg Likes(x,IceCream)$
- $\exists x\ Likes(x,Broccoli)$ $\neg\forall x\ \neg Likes(x,Broccoli)$

Parentheses

- Parentheses are often used with quantifiers
- Unfortunately, everyone uses them differently, so don’t be upset at any usage you see
- Examples:
  - $(\forall x)\ person(x) \Rightarrow likes(x,iceCream)$
  - $(\forall x)\ (person(x) \Rightarrow likes(x,iceCream))$
  - $(\forall x)\ [\ person(x) \Rightarrow likes(x,iceCream)\ ]$
  - $\forall x,\ person(x) \Rightarrow likes(x,iceCream)$
  - $\forall x\ (person(x) \Rightarrow likes(x,iceCream))$
- I prefer parentheses that show the scope of the quantifier
  - $\exists x\ (x > 0)\ \land\ \exists x\ (x < 0)$

More rules

- Now there are numerous additional rules we can apply!
- Here are two exceptionally important rules:
  - $\neg\forall x, p(x) \quad \Rightarrow \quad \exists x, \neg p(x)$
    "If not every x satisfies $p(x)$, then there exists a x that does not satisfy $p(x)$"
  - $\neg\exists x, p(x) \quad \Rightarrow \quad \forall x, \neg p(x)$
    "If there does not exist an x that satisfies $p(x)$, then all x do not satisfy $p(x)$"
- In any case, the search space is *just too large* to be feasible
- This was the case until 1970, when J. Robinson discovered resolution

Knowledge and Reasoning
Table of Contents

Logic by computer was infeasible

- Why is logic so hard?
  - You start with a large collection of facts (predicates)
  - You start with a large collection of possible transformations (rules)
    - Some of these rules apply to a single fact to yield a new fact
    - Some of these rules apply to a pair of facts to yield a new fact
  - So at every step you must:
    - Choose some rule to apply
    - Choose one or two facts to which you might be able to apply the rule
      - If there are n facts
        - There are n potential ways to apply a single-operand rule
        - There are n * (n - 1) potential ways to apply a two-operand rule
    - Add the new fact to your ever-expanding fact base
  - The search space is huge!

The magic of resolution

- Here’s how resolution works:
  - You transform each of your facts into a particular form, called a clause (this is the tricky part)
  - You apply a *single rule,* the resolution principle, to a pair of clauses
    - Clauses are closed with respect to resolution--that is, when you resolve two clauses, you get a new clause
  - You add the new clause to your fact base
- So the number of facts you have grows linearly
  - You still have to choose a pair of facts to resolve
  - You never have to choose a rule, because there’s only one

The fact base

- A fact base is a collection of “facts,” expressed in predicate calculus, that are presumed to be true (valid)
- These facts are implicitly “**and**ed” together
- Example fact base:
  - seafood(X) ⇒ likes(John, X) (where X is a variable)
  - seafood(shrimp)
  - pasta(X) ⇒ ¬likes(Mary, X) (where X is a *different* variable)
  - pasta(spaghetti)
- That is,
  - (seafood(X) ⇒ likes(John, X)) ∧ seafood(shrimp) ∧
    (pasta(Y) ⇒ ¬likes(Mary, Y)) ∧ pasta(spaghetti)
  - Notice that we had to change some Xs to Ys
  - The scope of a variable is the single fact in which it occurs

Clause form

- A clause is a disjunction ("**or**") of zero or more literals, some or all of which may be negated
- Example:
  sinks(X) ∨ dissolves(X, water) ∨ ¬denser(X, water)
- Notice that clauses use only “or” and “not”—they do not use “and,” “implies,” or either of the quantifiers “for all” or “there exists”
- The impressive part is that *any* predicate calculus expression can be put into clause form
  - Existential quantifiers, ∃ , are the trickiest ones

Unification

- From the pair of facts (not yet clauses, just facts):
  - seafood(X) ⇒ likes(John, X)   (where X is a variable)
  - seafood(shrimp)
- We ought to be able to conclude
  - likes(John, shrimp)
- We can do this by unifying the variable X with the constant shrimp
  - This is the *same* "unification" as is done in Prolog
- This unification turns seafood(X) ⇒ likes(John, X) into seafood(shrimp) ⇒ likes(John, shrimp)
- Together with the given fact seafood(shrimp), the final deductive step is easy

The resolution principle

- Here it is:
  ```
  • From        X  ∨ someLiterals
    and        ¬X  ∨ someOtherLiterals
               ----------------------------------------------
    conclude:  someLiterals  ∨ someOtherLiterals
  ```
- That's all there is to it!
- Example:
  ```
  • broke(Bob) ∨ well-fed(Bob)
    ¬broke(Bob) ∨ ¬hungry(Bob)
    ----------------------------------------
    well-fed(Bob) ∨ ¬hungry(Bob)
  ```

A common error

- You can only do *one* resolution at a time
- Example:
  - broke(Bob) ⋁ well-fed(Bob) ⋁ happy(Bob)
    ¬broke(Bob) ⋁ ¬hungry(Bob) ⋁ ¬happy(Bob)
- You can resolve on broke to get:
  - well-fed(Bob) ⋁ happy(Bob) ⋁ ¬hungry(Bob) ⋁ ¬happy(Bob) ≡ T
- Or you can resolve on happy to get:
  - broke(Bob) ⋁ well-fed(Bob) ⋁ ¬broke(Bob) ⋁ ¬hungry(Bob) ≡ T
- Note that both legal resolutions yield a tautology (a trivially true statement, containing X ⋁ ¬X), which is correct but useless
- But you *cannot* resolve on both at once to get:
  - well-fed(Bob) ⋁ ¬hungry(Bob)

Contradiction

- A special case occurs when the result of a resolution (the resolvent) is empty, or "NIL"
- Example:
  - hungry(Bob)
    ¬hungry(Bob)

    NIL
- In this case, the fact base is inconsistent
- This will turn out to be a very useful observation in doing resolution theorem proving

A first example

- "Everywhere that John goes, Rover goes. John is at school."
  - at(John, X) ⇒ at(Rover, X)   (not yet in clause form)
  - at(John, school)   (already in clause form)
- We use implication elimination to change the first of these into clause form:
  - ¬at(John, X) ∨ at(Rover, X)
  - at(John, school)
- We can resolve these on at(-, -), but to do so we have to unify X with school; this gives:
  - at(Rover, school)

Refutation resolution

- The previous example was easy because it had very few clauses
- When we have a lot of clauses, we want to *focus* our search on the thing we would like to prove
- We can do this as follows:
  - Assume that our fact base is consistent (we can’t derive NIL)
  - Add the *negation* of the thing we want to prove to the fact base
  - Show that the fact base is now inconsistent
  - Conclude the thing we want to prove

Example of refutation resolution

- "Everywhere that John goes, Rover goes. John is at school. **Prove that Rover is at school.**"
  1. ¬at(John, X) ∨ at(Rover, X)
  2. at(John, school)
  3. ¬at(Rover, school) (this is the added clause)
- Resolve #1 and #3:
  4. ¬at(John, X)
- Resolve #2 and #4:
  4. NIL
- Conclude the negation of the added clause: at(Rover, school)
- This seems a roundabout approach for such a simple example, but it works well for larger problems

A second example

- Start with:
  - it_is_raining ∨ it_is_sunny
  - it_is_sunny ⇒ I_stay_dry
  - it_is_raining ⇒ I_take_umbrella
  - I_take_umbrella ⇒ I_stay_dry
- Convert to clause form:
  1. it_is_raining ∨ it_is_sunny
  2. ¬it_is_sunny ∨ I_stay_dry
  3. ¬it_is_raining ∨ I_take_umbrella
  4. ¬I_take_umbrella ∨ I_stay_dry
- Prove that I stay dry:
  5. ¬I_stay_dry

- Proof:
  6. (5, 2) ¬it_is_sunny
  7. (6, 1) it_is_raining
  8. (5, 4) ¬I_take_umbrella
  9. (8, 3) ¬it_is_raining
  10. (9, 7) NIL
- Therefore, ¬(¬I_stay_dry)
  - I_stay_dry

Converting sentences to CNF

1. Eliminate all $\leftrightarrow$ connectives

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

2. Eliminate all $\rightarrow$ connectives

   $(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

   $\neg\neg P \Rightarrow P$

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

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

   $\neg(\forall x)P \Rightarrow (\exists x)\neg P$

   $\neg(\exists x)P \Rightarrow (\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Converting sentences to clausal form Skolem constants and functions

5. Eliminate existential quantification by introducing Skolem constants/functions

$(\exists x)P(x) \Rightarrow P(c)$

c is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

$(\forall x)(\exists y)P(x,y) \Rightarrow (\forall x)P(x, f(x))$

since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function f** to construct a new value that **depends on** the universally quantified variable

f must be a brand-new function name not occurring in any other sentence in the KB.

E.g., $(\forall x)(\exists y)loves(x,y) \Rightarrow (\forall x)loves(x,f(x))$

In this case, f(x) specifies the person that x loves

Converting sentences to clausal form

6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the "prefix" part

   Ex: $(\forall x)P(x) \Rightarrow P(x)$

7. Put into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws

   $(P \land Q) \lor R \Rightarrow (P \lor R) \land (Q \lor R)$

   $(P \lor Q) \lor R \Rightarrow (P \lor Q \lor R)$

8. Split conjuncts into separate clauses

9. Standardize variables so each clause contains only variable names that do not occur in any other clause

An example

$$\mathbf{(\forall x)(P(x) \rightarrow ((\forall y)(P(y) \rightarrow P(f(x,y))) \land \neg(\forall y)(Q(x,y) \rightarrow P(y))))}$$

2. Eliminate $\rightarrow$

$$(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land \neg(\forall y)(\neg Q(x,y) \lor P(y))))$$

3. Reduce scope of negation

$$(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (\exists y)(Q(x,y) \land \neg P(y))))$$

4. Standardize variables

$$(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (\exists z)(Q(x,z) \land \neg P(z))))$$

5. Eliminate existential quantification

$$(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (Q(x,g(x)) \land \neg P(g(x)))))$$

6. Drop universal quantification symbols

$$(\neg P(x) \lor ((\neg P(y) \lor P(f(x,y))) \land (Q(x,g(x)) \land \neg P(g(x)))))$$

Example

7. Convert to conjunction of disjunctions

$(\neg P(x) \lor \neg P(y) \lor P(f(x,y))) \land (\neg P(x) \lor Q(x,g(x))) \land$
$(\neg P(x) \lor \neg P(g(x)))$

8. Create separate clauses

$\neg P(x) \lor \neg P(y) \lor P(f(x,y))$

$\neg P(x) \lor Q(x,g(x))$

$\neg P(x) \lor \neg P(g(x))$

9. Standardize variables

$\neg P(x) \lor \neg P(y) \lor P(f(x,y))$

$\neg P(z) \lor Q(z,g(z))$

$\neg P(w) \lor \neg P(g(w))$

Resolution

- Resolution is a **sound** and **complete** inference procedure for FOL
- Reminder: Resolution rule for propositional logic:
  - $P_1 \lor P_2 \lor \dots \lor P_n$
  - $\neg P_1 \lor Q_2 \lor \dots \lor Q_m$
  - Resolvent: $P_2 \lor \dots \lor P_n \lor Q_2 \lor \dots \lor Q_m$
- Examples
  - P and $\neg P \lor Q$: derive Q (Modus Ponens)
  - $(\neg P \lor Q)$ and $(\neg Q \lor R)$: derive $\neg P \lor R$
  - P and $\neg$ P : derive False [contradiction!]
  - $(P \lor Q)$ and $(\neg P \lor \neg Q)$: derive True

Resolution in first-order logic

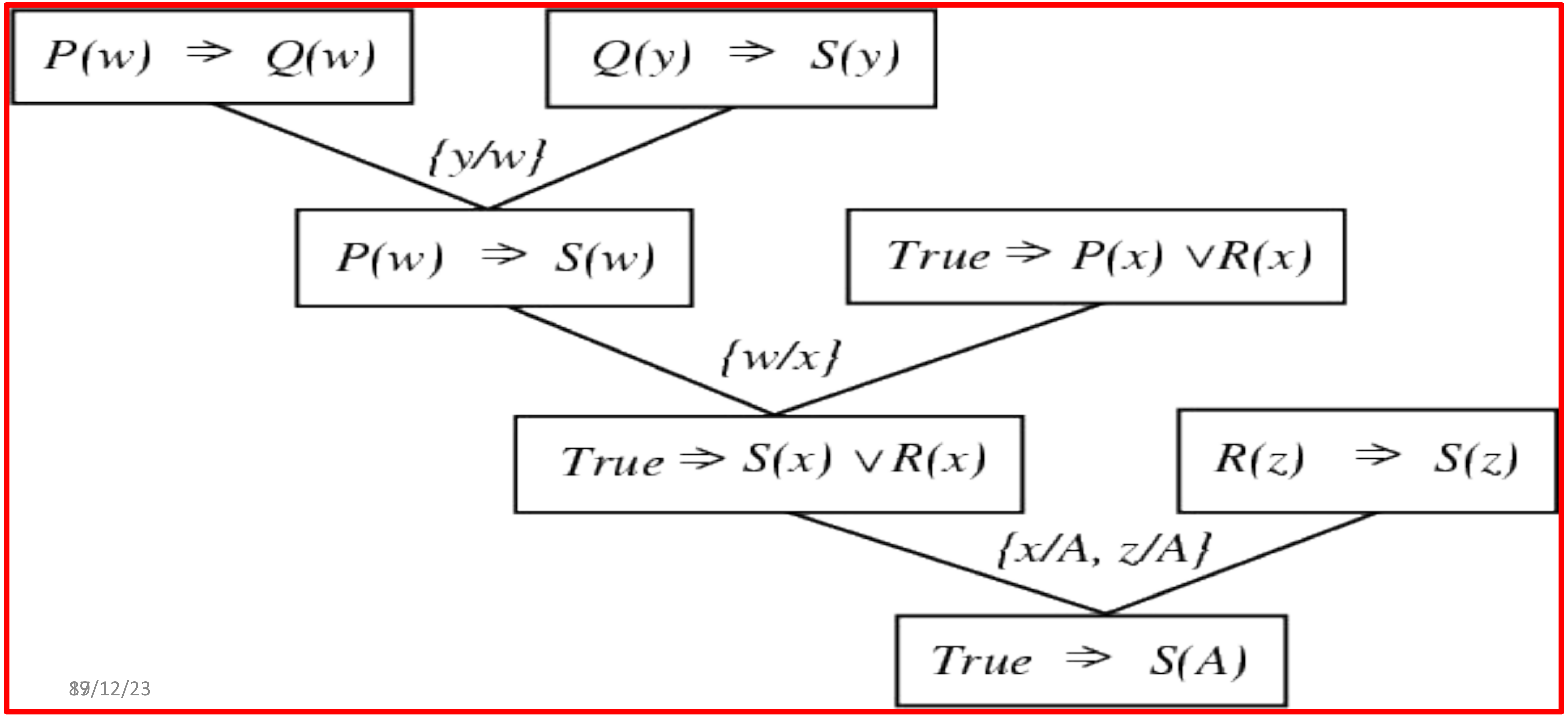
- Given sentences

  $P_1 \lor \ldots \lor P_n$

  $Q_1 \lor \ldots \lor Q_m$
- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ **unify** with substitution list θ, then derive the resolvent sentence:

  $\text{subst}(\theta, P_1 \lor \ldots \lor P_{j-1} \lor P_{j+1} \ldots P_n \lor Q_1 \lor \ldots Q_{k-1} \lor Q_{k+1} \lor \ldots \lor Q_m)$
- Example
  - from clause **$P(x, f(a)) \lor P(x, f(y)) \lor Q(y)$**
  - and clause **$\neg P(z, f(a)) \lor \neg Q(z)$**
  - derive resolvent **$P(z, f(y)) \lor Q(y) \lor \neg Q(z)$**
  - using **$\theta = \{x/z\}$**

A resolution proof tree

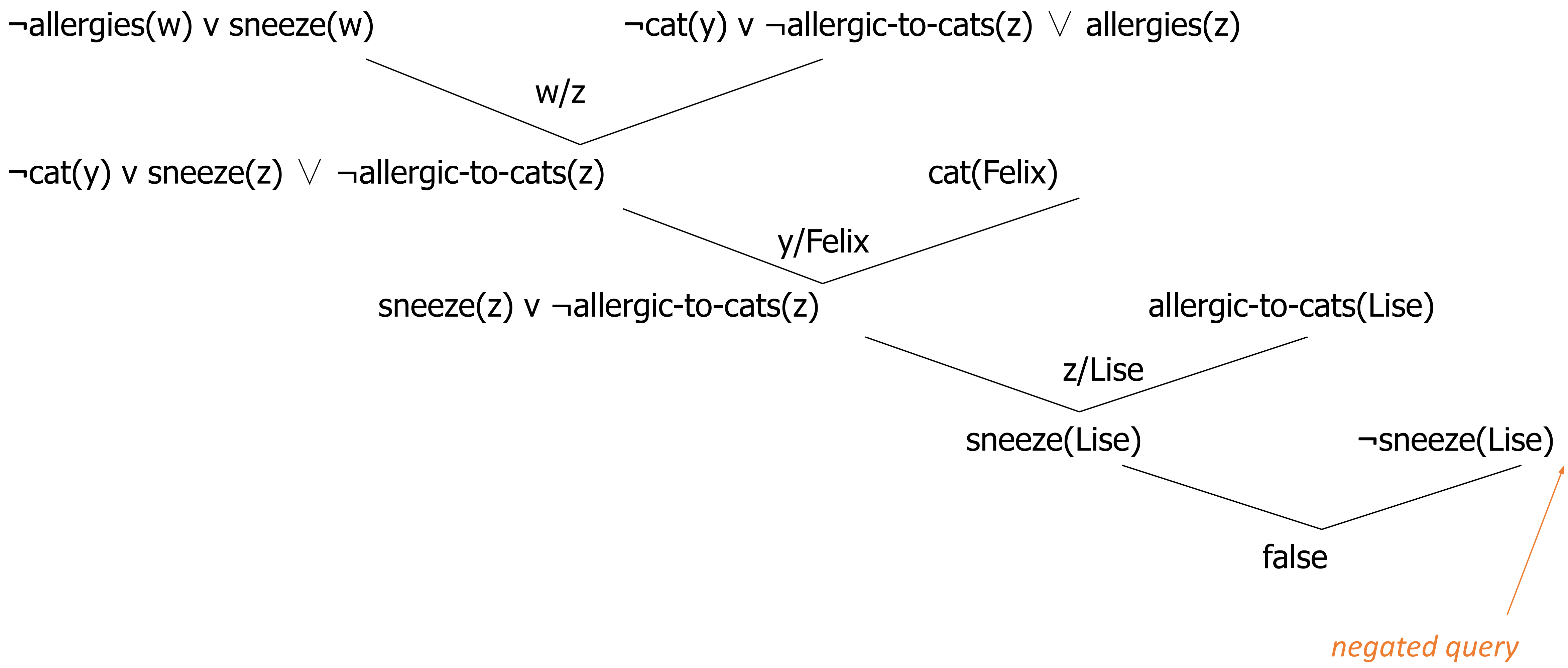


Resolution refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- **Proof by contradiction:** Add ¬Q to KB and try to prove false.

  i.e., (KB |- Q) ↔ (KB ⋁ ¬Q |- False)
- Resolution is **refutation complete**: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is **not entailed** by KB.
- Resolution **won't always give an answer** since entailment is only semidecidable
  - And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Refutation resolution proof tree


¬allergies(w) v sneeze(w)
¬cat(y) v ¬allergic-to-cats(z) ∨ allergies(z)
w/z
¬cat(y) v sneeze(z) ∨ ¬allergic-to-cats(z)
cat(Felix)
y/Felix
sneeze(z) v ¬allergic-to-cats(z)
allergic-to-cats(Lise)
z/Lise
sneeze(Lise)
¬sneeze(Lise)
false
negated query

We need answers to the following questions

- How to convert FOL sentences to conjunctive normal form (a.k.a. CNF, clause form): **normalization and skolemization**
- How to unify two argument lists, i.e., how to find their most general unifier (**mgu**) θ: **unification**
- How to determine which two clauses in KB should be resolved next (among all resolvable pairs of clauses) : **resolution (search) strategy**

Unification

- Unification is a **"pattern-matching"** procedure
  - Takes two atomic sentences, called literals, as input
  - Returns "Failure" if they do not match and a substitution list, θ, if they do
- That is, *unify(p,q) = θ* means *subst(θ, p) = subst(θ, q)* for two atomic sentences, *p* and *q*
- **θ** is called the **most general unifier** (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally quantified) variables by terms

Unification algorithm

```
procedure unify(p, q, $\theta$)
    Scan p and q left-to-right and find the first corresponding
      terms where p and q “disagree” (i.e., p and q not equal)
    If there is no disagreement, return $\theta$  (success!)
    Let r and s be the terms in p and q, respectively,
      where disagreement first occurs
    If variable(r) then {
      Let $\theta$ = union($\theta$, {r/s})
      Return unify(subst($\theta$, p), subst($\theta$, q), $\theta$)
    } else if variable(s) then {
      Let $\theta$ = union($\theta$, {s/r})
      Return unify(subst($\theta$, p), subst($\theta$, q), $\theta$)
    } else return “Failure”
  end
```

Unification: Remarks

- *Unify* is a linear-time algorithm that returns the most general unifier (mgu), i.e., the shortest-length substitution list that makes the two literals match.
- In general, there is not a **unique** minimum-length substitution list, but unify returns one of minimum length
- A variable can never be replaced by a term containing that variable

  Example: x/f(x) is illegal.
- This "occurs check" should be done in the above pseudo-code before making the recursive calls

Unification examples

- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(Bill), y)
  - {x/Bill, y/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(y), z)
  - {x/Bill, y/Bill, z/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Jane))
  - parents(Bill, father(y), mother(y))
  - Failure

Resolution example

Practice example : *Did Curiosity kill the cat*

- Jack owns a dog. Every dog owner is an animal lover. No animal lover kills an animal. Either Jack or Curiosity killed the cat, who is named Tuna. Did Curiosity kill the cat?
- These can be represented as follows:

A. $(\exists x)\ Dog(x) \land Owns(Jack,x)$

B. $(\forall x)\ ((\exists y)\ Dog(y) \land Owns(x, y)) \rightarrow AnimalLover(x)$

C. $(\forall x)\ AnimalLover(x) \rightarrow ((\forall y)\ Animal(y) \rightarrow \neg Kills(x,y))$

D. $Kills(Jack,Tuna) \lor Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $(\forall x)\ Cat(x) \rightarrow Animal(x)$ $\longleftarrow$ GOAL

G. $Kills(Curiosity, Tuna)$

- **Convert to clause form**
  - A1. (Dog(D)) ← D is a skolem constant
  - A2. (Owns(Jack,D))
  - B. (¬Dog(y), ¬Owns(x, y), AnimalLover(x))
  - C. (¬AnimalLover(a), ¬Animal(b), ¬Kills(a,b))
  - D. (Kills(Jack,Tuna), Kills(Curiosity,Tuna))
  - E. Cat(Tuna)
  - F. (¬Cat(z), Animal(z))
- **Add the negation of query:**
  - ¬G: (¬Kills(Curiosity, Tuna))

- **The resolution refutation proof**

R1: ¬G, D, {}   (Kills(Jack, Tuna))

R2: R1, C, {a/Jack, b/Tuna}   (~AnimalLover(Jack), ~Animal(Tuna))

R3: R2, B, {x/Jack}   (~Dog(y), ~Owns(Jack, y), ~Animal(Tuna))

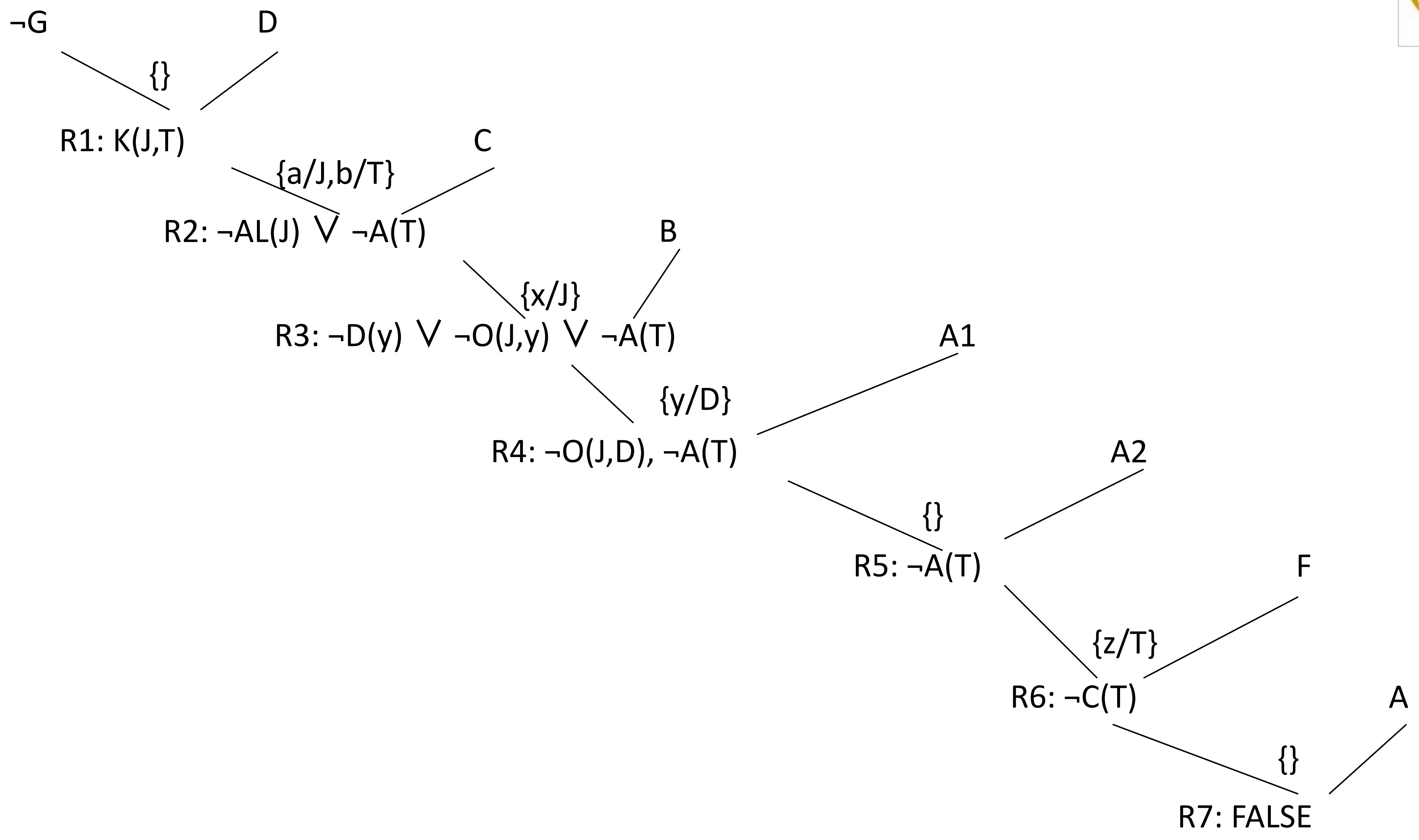
R4: R3, A1, {y/D}   (~Owns(Jack, D), ~Animal(Tuna))

R5: R4, A2, {}   (~Animal(Tuna))

R6: R5, F, {z/Tuna}   (~Cat(Tuna))

R7: R6, E, {}   FALSE

- **The proof tree**



Knowledge and Reasoning
Table of Contents

Production Rules

- Condition-Action Pairs
  - IF this condition (or premise or antecedent) occurs, THEN some action (or result, or conclusion, or consequence) will (or should) occur
  - IF the traffic light is red AND you have stopped, THEN a right turn is OK

Production Rules

- Each production rule in a knowledge base represents an autonomous chunk of expertise
- When combined and fed to the inference engine, the set of rules behaves synergistically
- Rules can be viewed as a simulation of the cognitive behaviour of human experts
- Rules represent a model of actual human behaviour
- Predominant technique used in expert systems, often in conjunction with frames

Forms of Rules

- IF premise, THEN conclusion
  - IF your income is high, THEN your chance of being audited by the Inland Revenue is high
- Conclusion, IF premise
  - Your chance of being audited is high, IF your income is high

Forms of Rules

- Inclusion of ELSE
  - IF your income is high, OR your deductions are unusual, THEN your chance of being audited is high, OR ELSE your chance of being audited is low
- More complex rules
  - IF credit rating is high AND salary is more than £30,000, OR assets are more than £75,000, AND pay history is not "poor," THEN approve a loan up to £10,000, and list the loan in category "B."”
- Action part may have more information: THEN "approve the loan" and "refer to an agent"

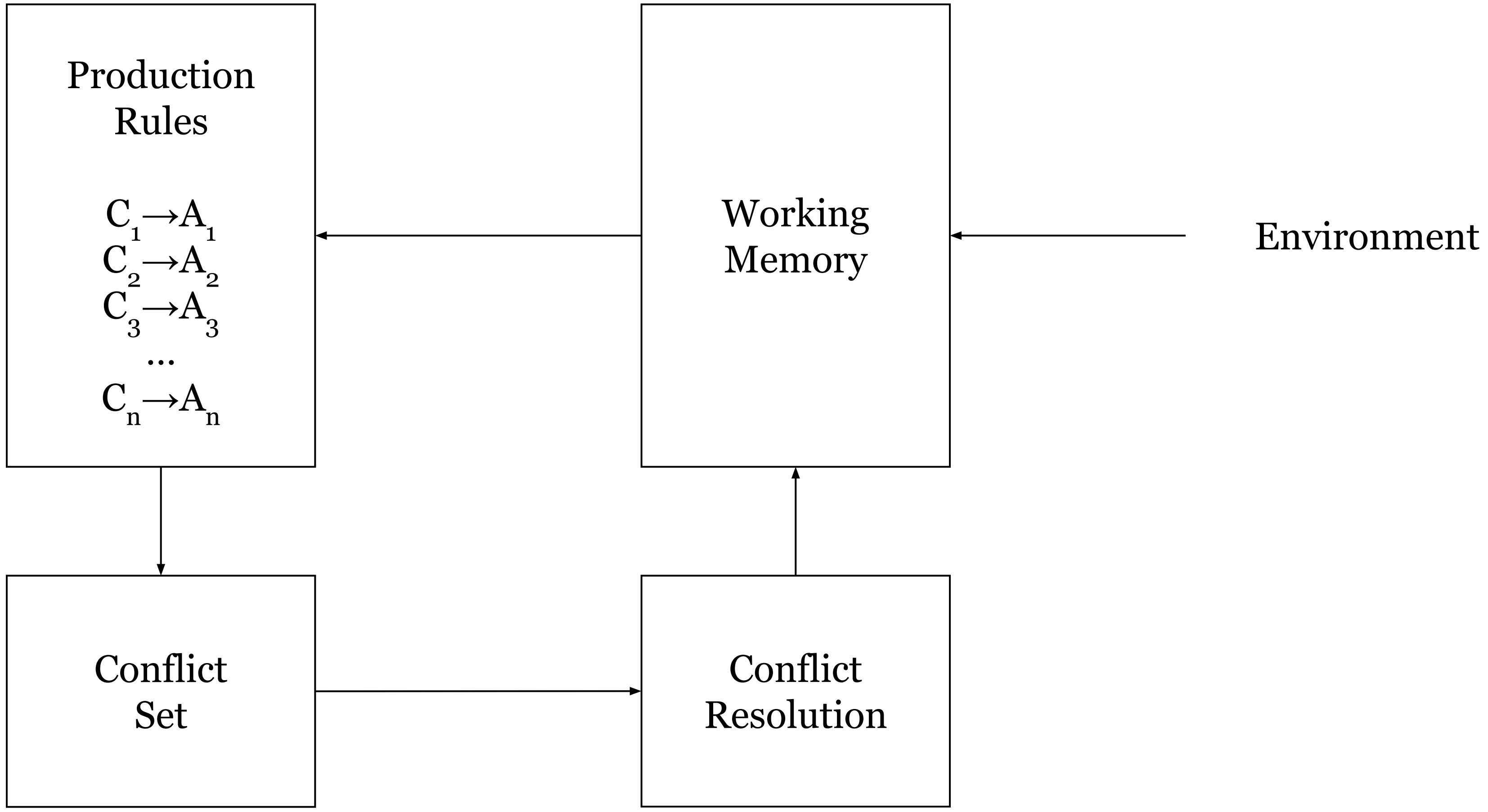
Characteristics of Rules

	First Part	**Second Part**
Names	Premise Antecedent Situation IF	Conclusion Consequence Action THEN
Nature	Conditions, similar to declarative knowledge	Resolutions, similar to procedural knowledge
Size	Can have many IFs	Usually only one conclusion
Statement	AND statements	All conditions must be true for a conclusion to be true
	OR statements	If any condition is true, the conclusion is true

Rule-based Inference

- Production rules are typically used as part of a *production system*
- Production systems provide pattern-directed control of the reasoning process
- Production systems have:
  - Productions: set of production rules
  - Working Memory (WM): description of current state of the world
  - Recognise-act cycle

Production Systems



Recognise-Act Cycle

- Patterns in WM matched against production rule conditions
- Matching (activated) rules form the **conflict set**
- One of the matching rules is selected (conflict resolution) and **fired**
    - Action of rule is performed
    - Contents of WM updated
- Cycle repeats with updated WM

Conflict Resolution

- Reasoning in a production system can be viewed as a type of search
  - Selection strategy for rules from the conflict set controls search
- Production system maintains the conflict set as an **agenda**
  - Ordered list of **activated rules** (those with their conditions satisfied) which have not yet been executed
  - Conflict resolution strategy determines where a newly-activated rule is inserted

Salience

- Rules may be given a precedence order by assigning a **salience value**
- Newly activated rules are placed in the agenda above all rules of lower salience, and below all rules with higher salience
  - Rule with higher salience are executed first
- Conflict resolution strategy applies between rules of the same salience
- If salience and the conflict resolution strategy can't determine which rule is to be executed next, a rule is chosen at random from the most highly ranked rules

Conflict Resolution Strategies

- Depth-first: newly activated rules placed above other rules in the agenda
- Breadth-first: newly activated rules placed below other rules
- Specificity: rules ordered by the number of conditions in the LHS (simple-first or complex-first)
- Least recently fired: fire the rule that was last fired the longest time ago
- Refraction: don’t fire a rule unless the WM patterns that match its conditions have been modified
- Recency: rules ordered by the timestamps on the facts that match their conditions

Salience

- Salience facilitates the modularization of expert systems in which modules work at different levels of abstraction
- Over-use of salience can complicate a system
  - Explicit ordering to rule execution
  - Makes behaviour of modified systems less predictable
- Rule of thumb: if two rules have the same salience, are in the same module, and are activated concurrently, then the order in which they are executed should not matter

Common Types of Rules

- Knowledge rules, or declarative rules, state all the facts and relationships about a problem
- Inference rules, or procedural rules, advise on how to solve a problem, given that certain facts are known
- Inference rules contain rules about rules (metarules)
- Knowledge rules are stored in the knowledge base
- Inference rules become part of the inference engine

Major Advantages of Rules

- Easy to understand (natural form of knowledge)
- Easy to derive inference and explanations
- Easy to modify and maintain
- Easy to combine with uncertainty
- Rules are frequently independent

Major Limitations of Rules

- Complex knowledge requires many rules
- Search limitations in systems with many rules

Knowledge and Reasoning
Table of Contents

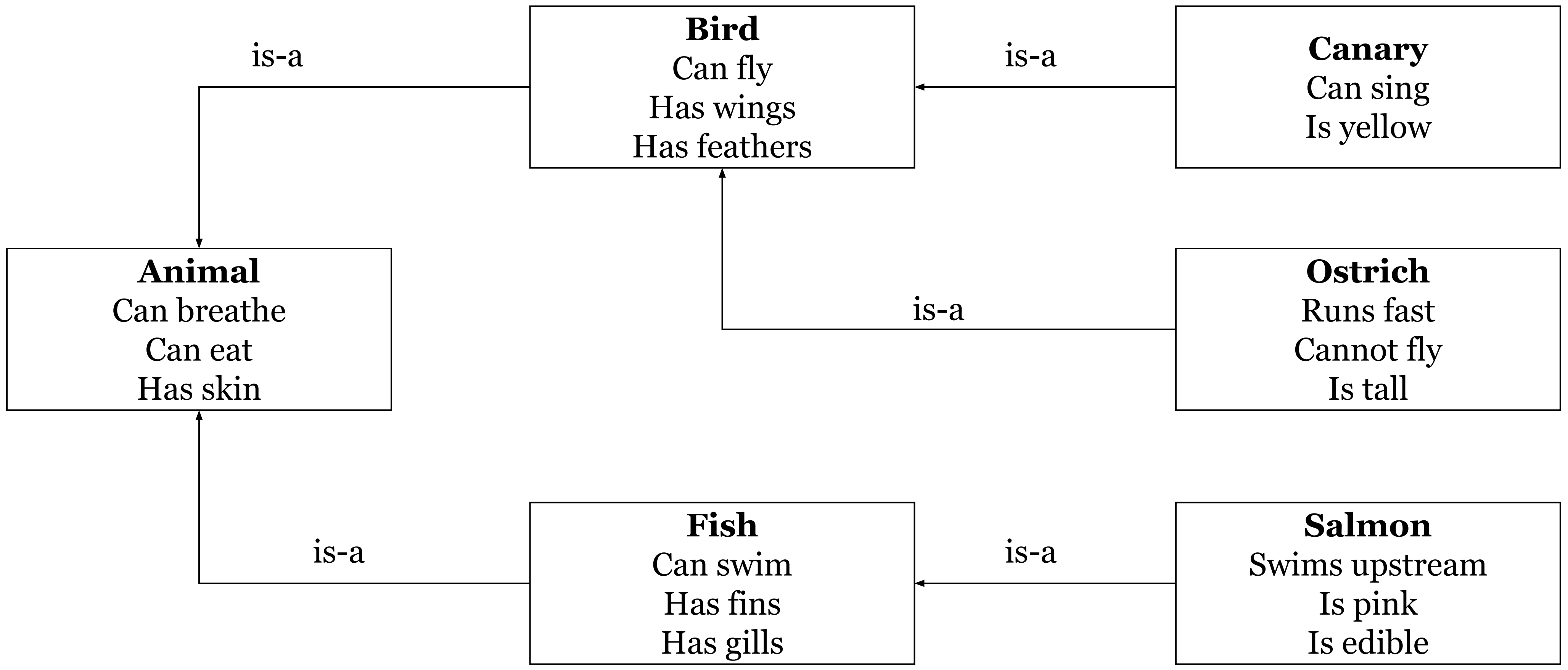
Semantic Networks

- A semantic network is a structure for representing knowledge as a pattern of interconnected nodes and arcs
- Nodes in the net represent concepts of entities, attributes, events, values
- Arcs in the network represent relationships that hold between the concepts

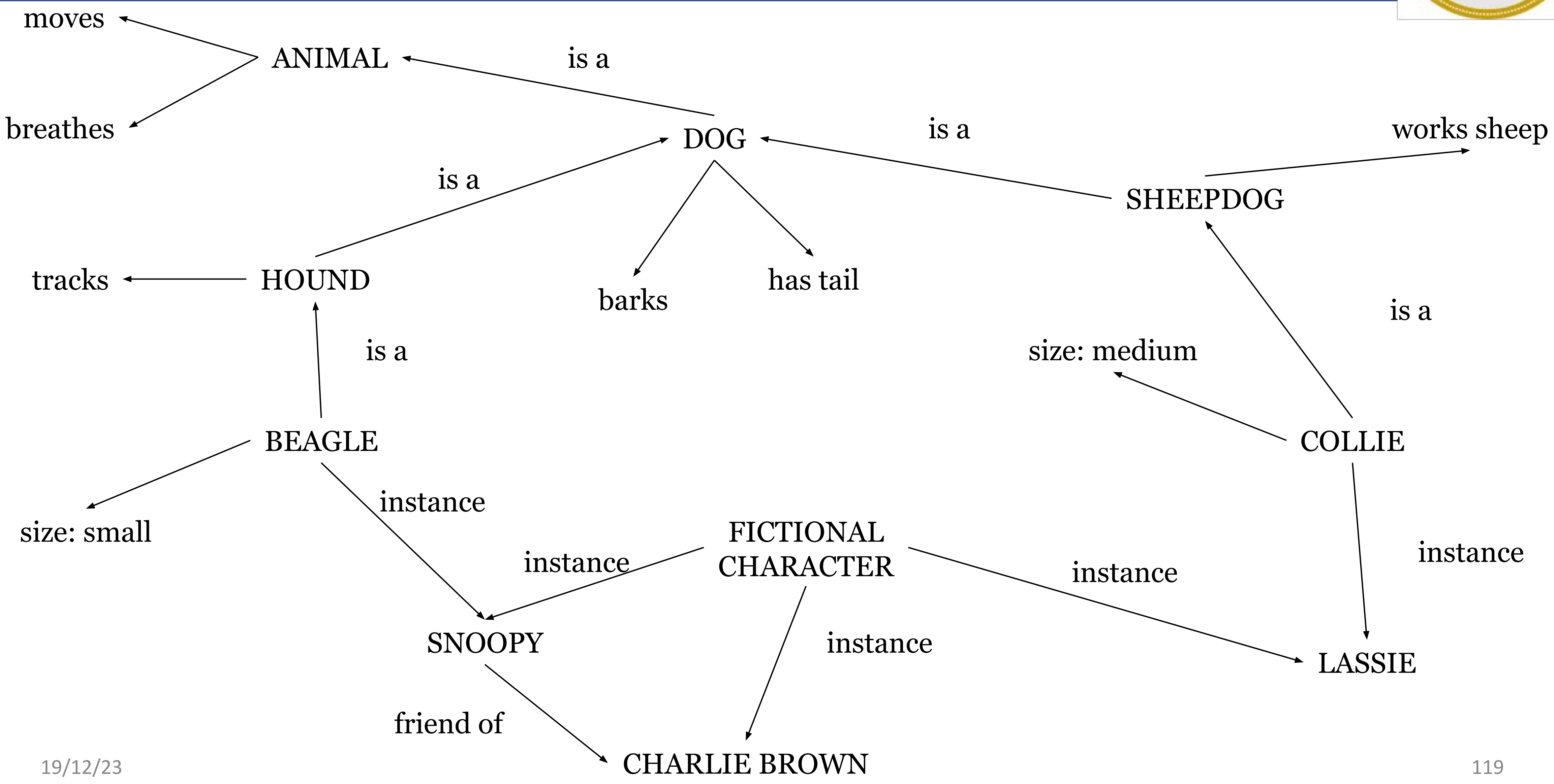
Semantic Networks

- Semantic networks can show inheritance
  - Relationship types – is-a, has-a
- Semantic Nets - visual representation of relationships
- Can be combined with other representation methods

Semantic Networks



Semantic Networks



Semantic Networks

What does or should a node represent?

- A class of objects?
- An instance of an class?
- The canonical instance of a class?
- The set of all instances of a class?

Semantic Networks

- Semantics of links that define new objects and links that relate existing objects, particularly those dealing with ‘intrinsic’ characteristics of a given object
- How does one deal with the problems of comparison between objects (or classes of objects) through their attributes?
  - Essentially the problem of comparing object instances
- What mechanisms are there are to handle quantification in semantic network formalisms?

Transitive inference, but…

- Clyde is an elephant, an elephant is a mammal: Clyde is a mammal.

- The US President is elected every 4 years, Bush is US President: Bush is elected every 4 years
- My car is a Ford, Ford is a car company: my car is a car company

Knowledge and Reasoning
Table of Contents

Frames

- A *frame* is a knowledge representation formalism based on the idea of a frame of reference.
- A frame is a data structure that includes all the knowledge about a particular object
- Frames organised in a hierarchy Form of object-oriented programming for AI and ES.
- Each frame describes one object
- Special terminology

M. Minsky (1974) A Framework for Representing Knowledge, *MIT-AI Laboratory Memo 306*

Frames

- There are two types of frame:
  - Class Frame
  - Individual or Instance Frame
- A frame carries with it a set of *slots* that can represent objects that are normally associated with a subject of the frame.

Frames

- The slots can then point to other slots or frames. That gives frame systems the ability to carry out inheritance and simple kinds of data manipulation.
- The use of procedures - also called *demons* in the literature - helps in the incorporation of substantial amounts of procedural knowledge into a particular frame-oriented knowledge base

Frame-based model of semantic memory

- Knowledge is **organised** in a data structure
- Slots in structure are instantiated with particular values for a given instance of data
- ...translation to OO terminology:
  - frames == classes or objects
  - slots == variables/methods

General Knowledge as Frames

DOG

Fixed
- legs: 4

Default
- diet: carnivorous
- sound: bark

Variable
- size:
- colour:

COLLIE

Fixed
- breed of: DOG
- type: sheepdog

Default
- size: 65cm

Variable
- colour:

General Knowledge as Frames

```
MAMMAL:
    subclass:    ANIMAL
    has_part:    head

ELEPHANT
    subclass:    MAMMAL
    colour:      grey
    size:            large

Nellie
    instance:    ELEPHANT
    likes:       apples
```

Logic underlies Frames

- $\forall x$ mammal(x) $\Rightarrow$ has_part(x, head)
- $\forall x$ elephant(x) $\Rightarrow$ mammal(x)

- elephant(clyde)
  $\therefore$
  mammal(clyde)
  has_part(clyde, head)

Logic underlies Frames

```
MAMMAL:
        subclass:       ANIMAL
        has_part:       head
        *furry:                 yes

ELEPHANT
        subclass:       MAMMAL
        has_trunk:      yes
        *colour:  grey
        *size:          large
        *furry:         no

Clyde
        instance:       ELEPHANT
        colour:         pink
        owner:          Fred

Nellie
        instance:       ELEPHANT
        size:                   small
```

Frames (Contd.)

- Can represent subclass and instance relationships (both sometimes called ISA or “is a”)
- Properties (e.g. colour and size) can be referred to as slots and slot values (e.g. grey, large) as slot fillers
- Objects can inherit all properties of parent class (therefore Nellie is grey and large)
- But can inherit properties which are only typical (usually called default, here starred), and can be overridden
- For example, mammal is typically furry, but this is not so for an elephant

Frames (Contd.)

- Provide a concise, structural representation of knowledge in a natural manner
- Frame encompasses complex objects, entire situations or a management problem as a single entity
- Frame knowledge is partitioned into slots
- Slot can describe declarative knowledge or procedural knowledge
- Hierarchy of Frames: Inheritance

Capabilities of Frames

- Ability to clearly document information about a domain model; for example, a plant's machines and their associated attributes
- Related ability to constrain allowable values of an attribute
- Modularity of information, permitting ease of system expansion and maintenance
- More readable and consistent syntax for referencing domain objects in the rules

Capabilities of Frames

- Platform for building graphic interface with object graphics
- Mechanism to restrict the scope of facts considered during forward or backward chaining
- Access to a mechanism that supports the inheritance of information down a class hierarchy
- Used as underlying model in standards for accessing KBs (Open Knowledge Base Connectivity - OKBC)

Summary

- Frames have been used in conjunction with other, less well-grounded, representation formalisms, like production systems, when used to build to pre-operational or operational expert systems
- Frames cannot be used efficiently to organise ‘a whole computation

Knowledge and Reasoning
Table of Contents

Types of Inference

- Deduction
- Induction
- Abduction

Deduction

- Deriving a conclusion from given axioms and facts
- Also called logical inference or truth preservation

Axiom – All kids are naughty

Fact/Premise – Riya is a kid

Conclusion – Riya is naughty

Induction

- Deriving general rule or axiom from background knowledge and observations
- Riya is a kid
- Riya is naughty

General axiom which is derived is:

Kids are naughty

Abduction

- A premise is derived from a known axiom and some observations
- All kids are naughty
- Riya is naughty

Inference

Riya is a kid

Knowledge and Reasoning Table of Contents

Uncertain knowledge and reasoning

- In real life, it is not always possible to determine the state of the environment as it might not be clear. Due to partially observable or non-deterministic environments, agents may need to handle uncertainty and deal with it.
- Uncertain data: Data that is missing, unreliable, inconsistent or noisy
- Uncertain knowledge: When the available knowledge has multiple causes leading to multiple effects or incomplete knowledge of causality in the domain
- Uncertain knowledge representation: The representations which provides a restricted model of the real system, or has limited expressiveness
- Inference: In case of incomplete or default reasoning methods, conclusions drawn might not be completely accurate. Let's understand this better with the help of an example.
- IF primary infection is bacteria cea
- AND site of infection is sterile
- AND entry point is gastrointestinal tract
- THEN organism is bacteriod (0.7).
- In such uncertain situations, the agent does not guarantee a solution but acts on its own assumptions and probabilities and gives some degree of belief that it will reach the required solution.

Uncertain knowledge and reasoning

- For example, In case of Medical diagnosis consider the rule Toothache = Cavity. This is not complete as not all patients having toothache have cavities. So we can write a more generalized rule Toothache = Cavity V Gum problems V Abscess… To make this rule complete, we will have to list all the possible causes of toothache. But this is not feasible due to the following rules:
- *Laziness- It will require a lot of effort to list the complete set of antecedents and consequents to make the rules complete.*
- *Theoretical ignorance- Medical science does not have complete theory for the domain*
- *Practical ignorance- It might not be practical that all tests have been or can be conducted for the patients.*
- Such uncertain situations can be dealt with using

Probability theory

Truth Maintenance systems

Fuzzy logic.

Uncertain knowledge and reasoning

Probability

- Probability is the degree of likeliness that an event will occur. It provides a certain degree of belief in case of uncertain situations. It is defined over a set of events U and assigns value P(e) i.e. probability of occurrence of event e in the range [0,1]. Here each sentence is labeled with a real number in the range of 0 to 1, 0 means the sentence is false and 1 means it is true.
- Conditional Probability or Posterior Probability is the probability of event A given that B has already occurred.
- $P(A|B) = (P(B|A) * P(A)) / P(B)$
- For example, P(It will rain tomorrow| It is raining today) represents conditional probability of it raining tomorrow as it is raining today.
- $P(A|B) + P(NOT(A)|B) = 1$
- Joint probability is the probability of 2 independent events happening simultaneously like rolling two dice or tossing two coins together. For example, Probability of getting 2 on one dice and 6 on the other is equal to 1/36. Joint probability has a wide use in various fields such as physics, astronomy, and comes into play when there are two independent events. The full joint probability distribution specifies the probability of each complete assignment of values to random variables.

Uncertain knowledge and reasoning

Bayes Theorem

- It is based on the principle that every pair of features being classified is independent of each other. It calculates probability P(A|B) where A is class of possible outcomes and B is given instance which has to be classified.
- P(A|B) = P(B|A) * P(A) / P(B)
- P(A|B) = Probability that A is happening, given that B has occurred (posterior probability)
- P(A) = prior probability of class
- P(B) = prior probability of predictor
- P(B|A) = likelihood

Uncertain knowledge and reasoning

Consider the following data. Depending on the weather (sunny, rainy or overcast), the children will play(Y) or not play(N).

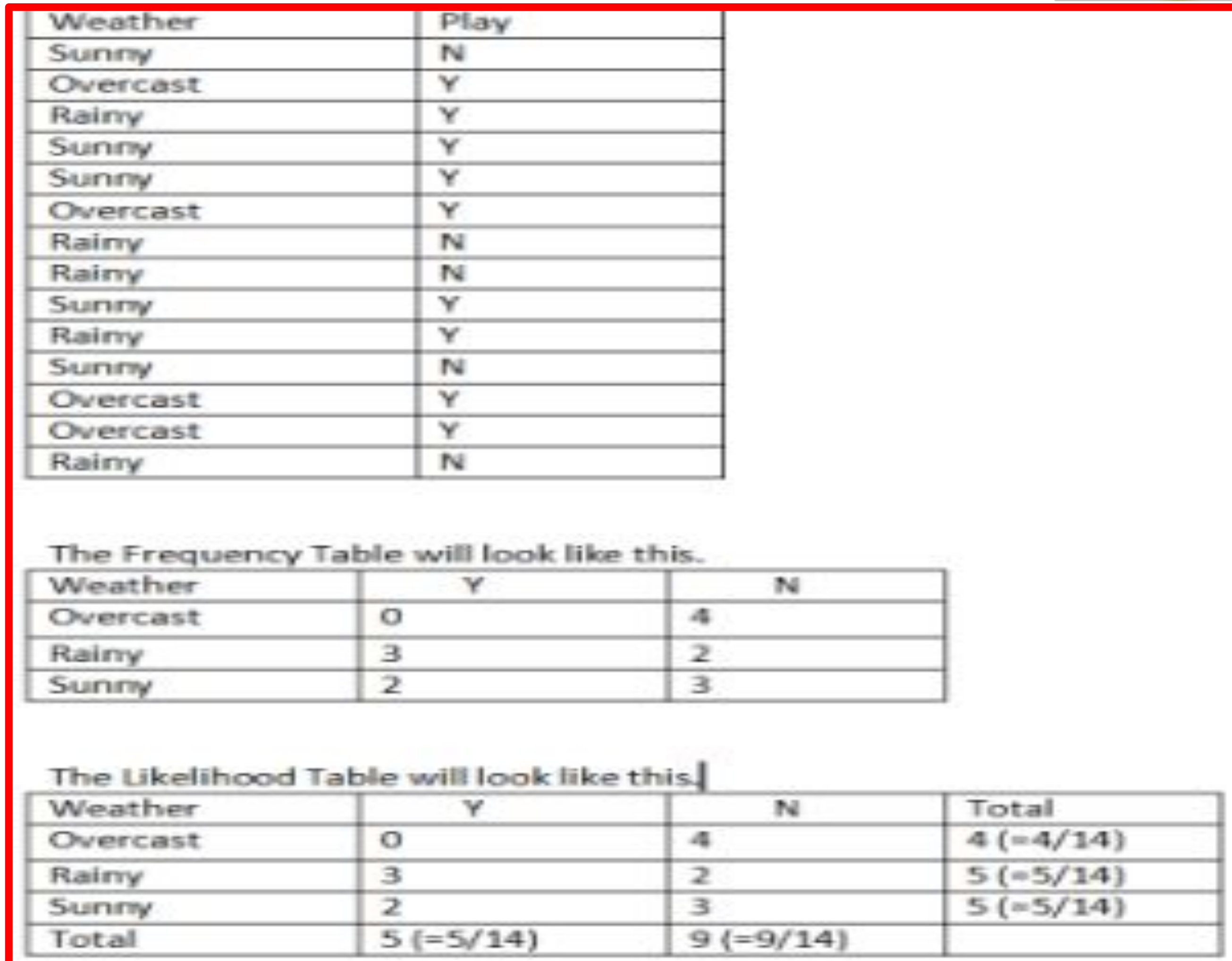
Here, the total number of observations = 14

Probability that children will play given that weather is sunny :

P(Yes| Sunny) = P(Sunny | Yes) * P(Yes) / P(Sunny)

= 0.33 * 0.64 / 0.36

= 0.59



Weather	Play
Sunny	N
Overcast	Y
Rainy	Y
Sunny	Y
Sunny	Y
Overcast	Y
Rainy	N
Rainy	N
Sunny	Y
Rainy	Y
Sunny	N
Overcast	Y
Overcast	Y
Rainy	N

The Frequency Table will look like this.

Weather	Y	N
Overcast	0	4
Rainy	3	2
Sunny	2	3

The Likelihood Table will look like this.

Weather	Y	N	Total
Overcast	0	4	4 (=4/14)
Rainy	3	2	5 (=5/14)
Sunny	2	3	5 (=5/14)
Total	5 (=5/14)	9 (=9/14)	

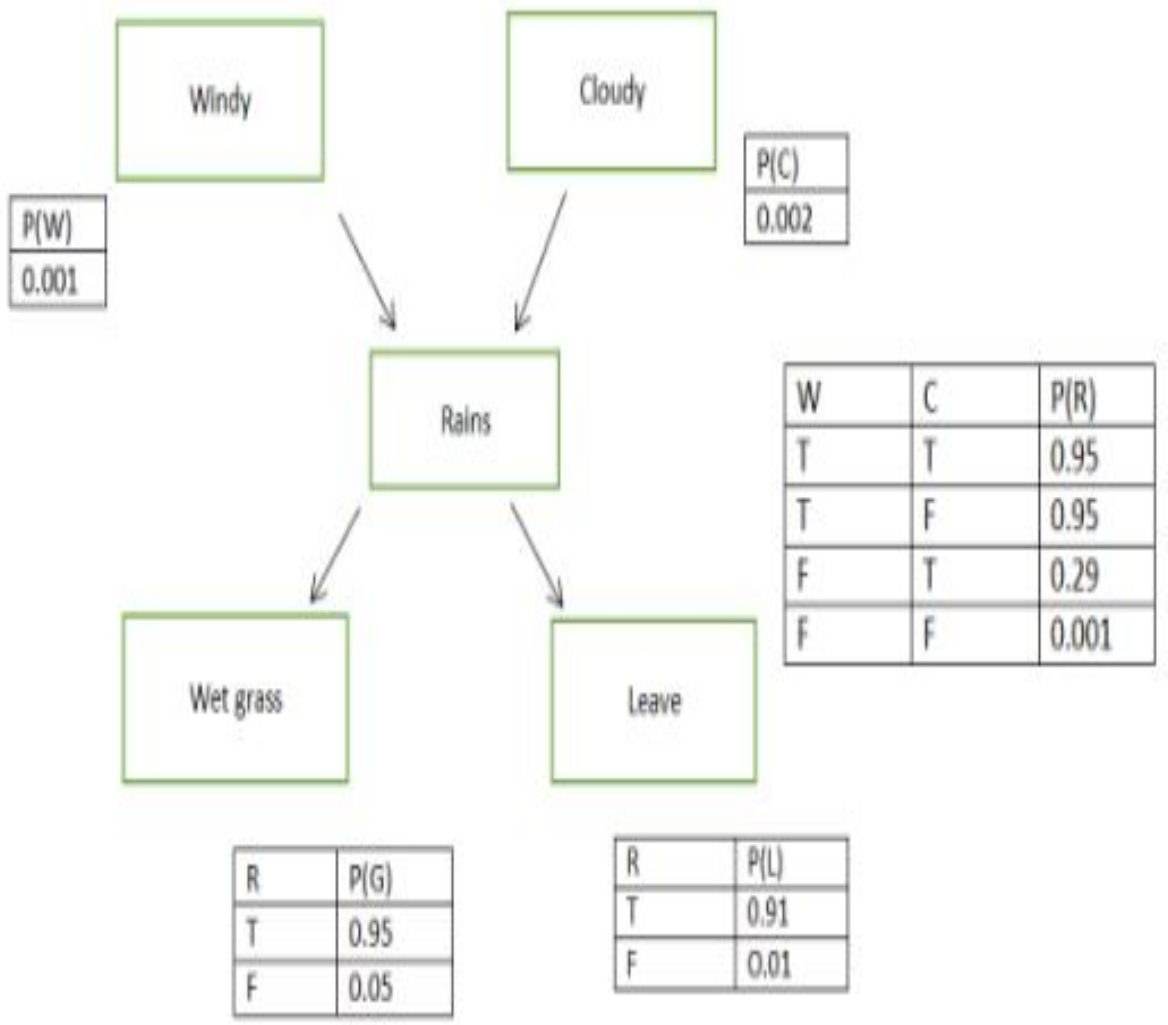
Uncertain knowledge and reasoning

It is a probabilistic graphical model for representing uncertain domain and to reason under uncertainty. It consists of nodes representing variables, arcs representing direct connections between them, called causal correlations. It represents conditional dependencies between random variables through a Directed Acyclic Graph (DAG). A belief network consist of:

1. A DAG with nodes labeled with variable names,

2. Domain for each random variable,

3. Set of conditional probability tables for each variable given its parents, including prior probability for nodes with no parents.



P(W)
0.001

P(C)
0.002

W	C	P(R)
T	T	0.95
T	F	0.95
F	T	0.29
F	F	0.001

R	P(G)
T	0.95
F	0.05

R	P(L)
T	0.91
F	0.01

Uncertain knowledge and reasoning

- Let’s have a look at the steps followed.

1. Identify nodes which are the random variables and the possible values they can have from the probability domain. The nodes can be boolean (True/ False), have ordered values or integral values.
2. Structure- It is used to represent causal relationships between the variables. Two nodes are connected if one affects or causes the other and the arc points towards the effect. For instance, if it is windy or cloudy, it rains. There is a direct link from Windy/Cloudy to Rains. Similarly, from Rains to Wet grass and Leave, i.e., if it rains, grass will be wet and leave is taken from work.

Uncertain knowledge and reasoning

3. Probability- Quantifying relationship between nodes. Conditional Probability:

- P(B|A) = P(A|B) * P(B) / P(A)
- Joint probability:

4. Markov property- Bayesian Belied Networks require assumption of Markov property, i.e., all direct dependencies are shown by using arcs. Here there is no direct connection between it being Cloudy and Taking a leave. But there is one via Rains. Belief Networks which have Markov property are also called independence maps.

Uncertain knowledge and reasoning

Inference in Belief Networks

- Bayesian Networks provide various types of representations of probability distribution over their variables. They can be conditioned over any subset of their variables, using any direction of reasoning.
- For example, one can perform diagnostic reasoning, i.e. when it Rains, one can update his belief about the grass being wet or if leave is taken from work. In this case reasoning occurs in the opposite direction to the network arcs. Or one can perform predictive reasoning, i.e., reasoning from new information about causes to new beliefs about effects, following direction of the arcs. For example, if the grass is already wet, then the user knows that it has rained and it might have been cloudy or windy. Another form of reasoning involves reasoning about mutual causes of a common effect. This is called inter causal reasoning.
- There are two possible causes of an effect, represented in the form of a 'V'. For example, the common effect 'Rains' can be caused by two reasons 'Windy' and 'Cloudy.' Initially, the two causes are independent of each other but if it rains, it will increase the probability of both the causes. Assume that we know it was windy. This information explains the reasons for the rainfall and lowers probability that it was cloudy.

Thank you