Module 3: Knowledge & Reasoning

From search to logic — giving agents the power to KNOW and REASON about their world.

PECST522 — Artificial Intelligence | B.Tech CSE S5 (2024 Scheme)

Module 3 — Roadmap

8 contact hours | Maps to CO3 | Difficulty: ☆☆

3.1	**KB Agents**	Knowledge bases, TELL/ASK loop, declarative approach
3.2	**Logic Basics**	Syntax, semantics, inference — the three pillars
3.3	**Propositional Logic**	Symbols, models, truth tables, resolution
3.4	**First-Order Logic**	Objects, relations, quantifiers, unification
3.5	**Inference in FOL**	Forward & backward chaining, resolution refutation

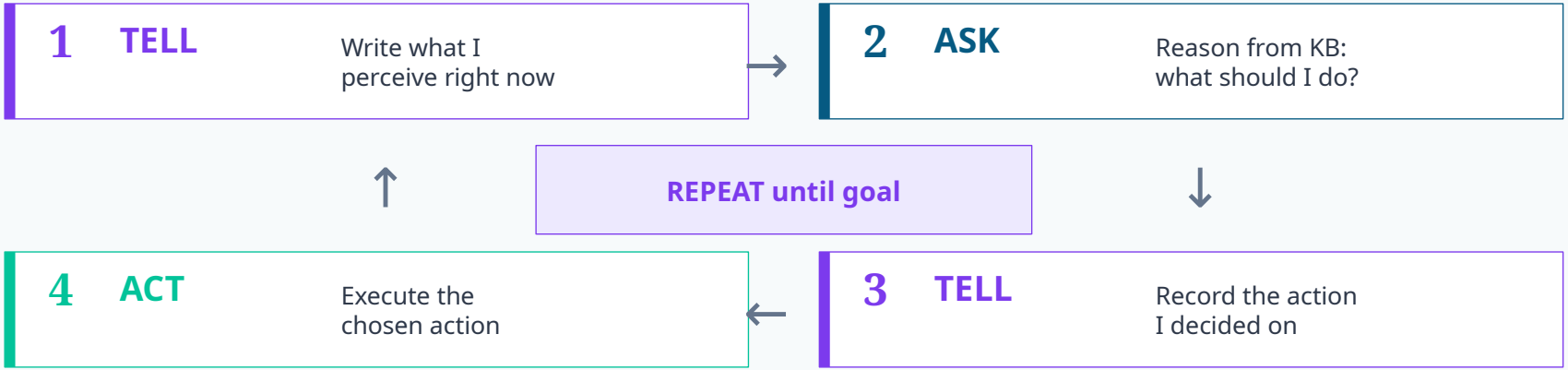
3.1

Knowledge-Based Agents

Agents that KNOW things and REASON about what to do

The Knowledge-Based Agent Loop

Think of the KB as the agent's "brain notebook" — write observations, read to decide



Declarative vs Procedural Knowledge:

	Declarative ✓	Procedural ✗
What it stores	WHAT is true ("Pit at [3,1]")	HOW to behave ("If breeze → turn left")
Flexibility	High — agent reasons with any KB content	Low — breaks if world changes
KB Agent uses?	✅ YES — the standard approach	❌ NO — too rigid for complex worlds

💡 One-liner: A KB agent stores what it knows, reasons to decide, and learns by adding new knowledge.

The Wumpus World

THE motivating example — why agents NEED logic to survive

[1,4]	[2,4] B	[3,4] P	[4,4]
[1,3] W	[2,3]	[3,3]	[4,3]
[1,2] S	[2,2]	[3,2] B	[4,2]
[1,1] S	[2,1] B	[3,1] P	[4,1] G

Legend:

- **W** — Wumpus (monster — kills you)
- **P** — Pit (bottomless — kills you)
- **G** — Gold (the goal!)
- **S** — Stench (adjacent to Wumpus)
- **B** — Breeze (adjacent to Pit)

Environment:

- Partially observable
- Deterministic
- Sequential
- Static
- Discrete
- Single-agent

 Agent starts at [1,1] — can only sense its CURRENT square

Percept Rules:

Percept	Meaning	Appears Where
Stench	Wumpus in adjacent square	Squares next to Wumpus
Breeze	Pit in adjacent square	Squares next to a pit
Glitter	Gold is HERE!	Same square as gold
Bump	Walked into a wall	When hitting boundary
Scream	Wumpus died (you shot it)	Anywhere in the world

Wumpus World — Reasoning Walkthrough

How the agent deduces danger from clues — logic in action!

1 **Agent at [1,1]** Percept: Nothing (no stench, no breeze)

Deduce: No Wumpus or Pit in adjacent squares [1,2] and [2,1].

→ Both adjacent squares are SAFE ✓

2 **Agent at [2,1]** Percept: Breeze

Deduce: A Pit is adjacent to [2,1]. Not [1,1] (we were there). Possible: [3,1] or [2,2].

→ Pit is in [3,1] or [2,2] — avoid both!

3 **Agent at [1,2]** Percept: Stench

Deduce: Wumpus is adjacent to [1,2]. Not [1,1] (safe). Possible: [1,3] or [2,2].

⚡ EXAM ALERT: "Explain Wumpus world reasoning" is a recurring 9-mark question. Walk through step-by-step with deductions.

3.2

Logic — The Formal Framework

Syntax, semantics, and inference — the three pillars

Logic — The Three Pillars

How to write facts, what they mean, and how to derive new ones

Pillar	What It Does	Analogy
Syntax	Rules for forming valid sentences	*Grammar of a language — "S + V + O" is valid, "V S O" is not*
Semantics	What sentences MEAN — truth in a world (model)	*Dictionary — "It is raining" is true if water falls from sky*
Inference	Deriving NEW sentences from existing ones	*Detective work — "All men mortal + Socrates is a man → Socrates mortal"*

Key Definitions

Model	A possible world (assignment of truth values). E.g. {P=True, Q=False} is one model.
Entailment (KB ⊨ α)	α is true in EVERY model where KB is true. KB "logically implies" α.
Soundness	Inference derives ONLY true conclusions. Nothing false is ever produced. (No false positives)
Completeness	Inference can derive ALL true conclusions. Nothing true is ever missed. (No false negatives)

⚡ **EXAM ALERT: "Define entailment, soundness, completeness" — recurring 3-mark Part A question. Memorise these four definitions.**

3.3

Propositional Logic

The simplest logic — TRUE or FALSE, nothing in between

Propositional Logic — The 5 Connectives

How to combine propositions (P, Q, R...) into complex sentences

Propositions are statements that are either TRUE or FALSE. Connectives combine them into complex sentences.

Symbol	Name	English	Example
¬	**NOT (negation)**	"it is NOT the case that"	¬P = "It is NOT raining"
∧	**AND (conjunction)**	"and"	P ∧ Q = "Raining AND ground is wet"
∨	**OR (disjunction)**	"or" (inclusive — both OK)	P ∨ Q = "Raining OR ground is wet (or both)"
⇒	**IMPLIES (implication)**	"if ... then"	P ⇒ Q = "IF it rains THEN ground is wet"
⇔	**IFF (biconditional)**	"if and only if"	P ⇔ Q = "It rains IFF ground is wet"

⚠ Important: ∨ is INCLUSIVE OR — "P ∨ Q" is true when BOTH P and Q are true. This is NOT the everyday "either/or" — it's "at least one."

Complete Truth Tables

Every connective's behaviour for all possible inputs

NOT (¬P)

P	¬P
T	**F**
F	**T**

AND (P ∧ Q)

P	Q	P∧Q
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

OR (P ∨ Q)

P	Q	P∨Q
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

⚡ IMPLIES (P ⇒ Q) — THE TRICKY ONE

P	Q	P⇒Q
T	T	**T**
T	F	**F**
F	T	**T**
F	F	**T**

Only FALSE when T ⇒ F (True premise, false conclusion = broken promise)

BICONDITIONAL (P ⇔ Q)

P	Q	P⇔Q
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**T**

 Why is "False ⇒ anything" TRUE? Think of a promise: "If it rains, I'll bring an umbrella." If it DOESN'T rain (P is false), you haven't broken your promise regardless of what you do. The promise is ONLY broken when it DOES rain and you DON'T bring the umbrella (T ⇒ F = the only FALSE row).

Propositional Logic — Sentence Types

Three categories based on truth across all possible models

VALID (Tautology)

True in ALL models

$P \vee \neg P$

"It rains or it doesn't rain" — always true, no matter what.

✓ ALWAYS TRUE

SATISFIABLE

True in AT LEAST ONE model

$P \wedge Q$

"Raining AND ground wet" — true when both P and Q are true.

◐ TRUE SOMETIMES

UNSATISFIABLE

True in NO model (contradiction)

$P \wedge \neg P$

"Raining AND not raining" — impossible, always false.

✗ NEVER TRUE

Relationship: Every valid sentence is also satisfiable (true in all models ⊃ true in at least one). Unsatisfiable = negation is valid. **$KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is valid iff $(KB \wedge \neg\alpha)$ is unsatisfiable.**

Inference Rules & Key Equivalences

How to derive new truths from existing ones

Inference Rules

⚡ EXAM ALERT

Modus Ponens *(THE workhorse rule)*

From: A ⇒ B and A → Infer: B

"If rain⇒wet, AND it IS raining → ground is wet"

And-Elimination From: A ∧ B → Infer: A (and also B)

"Raining AND cold → therefore: it's raining"

Resolution From: (A ∨ B), (¬B ∨ C) → (A ∨ C)

Cancel B and ¬B, combine the rest. COMPLETE rule.

Key Equivalences (for CNF conversion)

Name	Equivalence
Implication elim.	$A \Rightarrow B \equiv \neg A \vee B$
Biconditional elim.	$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
De Morgan's (×2)	$\neg(A \wedge B) \equiv \neg A \vee \neg B$ | $\neg(A \vee B) \equiv \neg A \wedge \neg B$
Double negation	$\neg\neg A \equiv A$
Contraposition	$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$
Distributivity	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

⚠ **Memorise De Morgan's and Implication Elimination — you'll need them for EVERY CNF conversion problem.**

When to use what? Modus Ponens = direct forward chaining from implications. And-Elim = extract parts of a conjunction. Resolution = the universal proof method (works for ANY propositional proof if you have CNF).

CNF — What Is It and Why Do We Need It?

Conjunctive Normal Form: the standard input for resolution

CNF = AND of ORs. Each "clause" is an OR of literals (a literal = variable or its negation). The whole sentence is an AND of these clauses.

Example: (A ∨ B ∨ ¬C) ∧ (¬A ∨ D) ∧ (B ∨ C)

Why CNF?

Resolution needs CNF because each clause becomes a simple set of literals. Two clauses with a complementary pair (e.g. B and ¬B) can be resolved → the single most powerful proof method in AI.

⚡ EXAM ALERT — The 5-Step CNF Conversion Recipe

1	**Eliminate ⇔**	Replace A ⇔ B with (A ⇒ B) ∧ (B ⇒ A)
2	**Eliminate ⇒**	Replace A ⇒ B with ¬A ∨ B
3	**Move ¬ inward**	Use De Morgan's laws + double negation to push ¬ down to literals
4	**Distribute ∨ over ∧**	A ∨ (B ∧ C) → (A ∨ B) ∧ (A ∨ C)
5	**Result: AND of ORs**	Each clause is a disjunction; the whole is a conjunction ✓

⚠ **Steps MUST be done in order. Skipping step 3 (¬ inward) before distributing is the #1 student mistake.**

CNF Conversion — Worked Example

Convert (A ⇒ B) ⇒ C to Conjunctive Normal Form

Starting expression: (A ⇒ B) ⇒ C

Step 1: Eliminate ⇔

No ⇔ present — skip.

Step 2: Eliminate ⇒

Outer ⇒ first: (A ⇒ B) ⇒ C = ¬(A ⇒ B) ∨ C

Then inner ⇒: A ⇒ B = ¬A ∨ B. Substitute: ¬(¬A ∨ B) ∨ C

→ **¬(¬A ∨ B) ∨ C**

Step 3: Move ¬ inward (De Morgan's)

¬(¬A ∨ B) = (¬¬A ∧ ¬B) = (A ∧ ¬B) [De Morgan's + double negation]

Substitute back: (A ∧ ¬B) ∨ C

→ **(A ∧ ¬B) ∨ C**

Step 4: Distribute ∨ over ∧

(A ∧ ¬B) ∨ C = (A ∨ C) ∧ (¬B ∨ C) [distribute C into each conjunct]

→ **(A ∨ C) ∧ (¬B ∨ C)**

CNF RESULT: (A ∨ C) ∧ (¬B ∨ C) ✓

✓ **Verify:** Two clauses: (A ∨ C) and (¬B ∨ C). Each clause is an OR of literals. The whole is an AND of clauses. This IS CNF. ✓

⚡ **EXAM ALERT: CNF conversion is a guaranteed Part B question. Show EVERY step — marks are for the process.**

Resolution — The Proof Method

A single rule that can prove ANY propositional entailment

Why resolution? It's a SINGLE inference rule that is COMPLETE — if you can only use resolution, you can still prove anything that's provable. Perfect for computers.

⚡ EXAM ALERT — TOP Part B question

1. Add ¬α to the KB (assume the OPPOSITE of what you want to prove)
2. Convert everything to CNF (use the 5-step recipe)
3. Apply resolution repeatedly: pick two clauses with complementary literals, resolve them
4. **Empty clause □ derived → CONTRADICTION → α is PROVED!**

Key Insight — Proof by Contradiction: We assume ¬α is true. If this leads to a contradiction (empty clause), our assumption MUST be wrong. Therefore α is true. This is the same logic as "proof by contradiction" in mathematics.

Resolution Rule:

$$\frac{(A \vee B) \text{ and } (\neg B \vee C)}{(A \vee C)}$$

What if no empty clause? If all possible resolutions are exhausted and no empty clause appears → α is NOT entailed by the KB.

Resolution — Complete Worked Example

Given KB: {A ⇒ B, B ⇒ C, A}. Prove: C

Step 1: Add ¬C and convert to CNF

Clause 1:	**¬A ∨ B**	*A ⇒ B (eliminate ⇒)*
Clause 2:	**¬B ∨ C**	*B ⇒ C (eliminate ⇒)*
Clause 3:	**A**	*given*
Clause 4:	**¬C**	*negated goal (¬α)*

Step 2: Resolve

Resolve: Clause 1 + Clause 3
(¬A ∨ B) + (A) → resolve on A/¬A
→ **Clause 5: B**

↓

Resolve: Clause 2 + Clause 5
(¬B ∨ C) + (B) → resolve on B/¬B
→ **Clause 6: C**

↓

Resolve: Clause 4 + Clause 6
(¬C) + (C) → resolve on C/¬C
→ **Clause 7: □ EMPTY CLAUSE!**

✓ **CONTRADICTION DERIVED — □ (empty clause)** Our assumption ¬C led to a contradiction. Therefore C MUST be true. KB ⊨ C ∎

Student FAQ

Q: Why add ¬α? A: It's proof by contradiction. If assuming "α is false" leads to impossibility, then α must be true.

Q: What if no empty clause? A: Then α is NOT entailed by the KB. The KB doesn't have enough info to prove α.

Q: Why not use truth tables? A: Works for small problems, but with n variables you need 2^n rows. Resolution is often much faster.

3.4

First-Order Logic

Objects, relationships, and quantifiers — logic that scales

Why First-Order Logic?

Propositional Logic — The Problem

"Every room adjacent to a pit is breezy"
In PL you need ONE sentence per square:

$Breezy_{11} \Rightarrow Adjacent(1_1, Pit_{12}) \vee \ldots$
$Breezy_{12} \Rightarrow Adjacent(1_2, Pit_{11}) \vee \ldots$
$Breezy_{13} \Rightarrow \ldots$ ← infinite!

First-Order Logic — One Rule

$$\forall r\ \forall p\ \ Adjacent(r, p) \wedge Pit(p) \Rightarrow Breezy(r)$$

ONE sentence covers ALL rooms, ALL pits.
Quantifiers (∀, ∃) let us talk about every object or some object without listing them individually.

Analogy — Driving Directions

PL = Turn-by-turn GPS
"At junction 1 go left, at junction 2 go right, at junction 3 go left..."

FOL = General road rule
"Always follow the road with the green sign" — one rule, every junction

PL vs FOL — Quick Comparison

Feature	Propositional Logic	First-Order Logic
Talks about objects?	✗ No — only facts (P, Q, R)	✓ Yes — John, $Square_{11}$, NSSCE
General rules?	✗ One sentence per case	✓ One rule with ∀ covers all cases
Relationships?	✗ Must encode as symbols	✓ Adjacent(r, p), Brother(x, y)
Scalability	✗ Explodes with domain size	✓ Compact even for infinite domains

FOL Building Blocks

Six elements that make FOL more powerful than PL

Element	What it is	Examples
Constants	Names for SPECIFIC objects	`John, NSSCE, Square₁₁, M1`
Variables	Placeholder for ANY object	`x, y, z  (like algebra)`
Predicates	Properties / relations → True or False	`King(John), Adjacent(r, p), Rich(x)`
Functions	Map objects → objects (NOT true/false)	`FatherOf(John), LeftOf(x), x + 1`
Quantifiers	"For all" (∀) and "There exists" (∃)	`∀x King(x)⇒Person(x),  ∃x Crown(x)`
Connectives	Same as PL — join formulas	`¬  ∧  ∨  ⇒  ⇔`

Key Distinction — Predicate vs Function
Predicate: King(John) → TRUE or FALSE (it's a claim)
Function: FatherOf(John) → an OBJECT (a person), not true/false

A FOL formula = **Quantifier** + **Predicate(function(constants, variables))** + **Connectives**
`Example: ∀x (King(x) ⇒ ∃y Crown(y) ∧ Wears(x, y))`

Quantifiers — ∀ (For All) and ∃ (There Exists)

∀ Universal — "For EVERY x ..."

Pairs with ⇒ (implication)

`∀x King(x) ⇒ Person(x)`
"If x is a king, then x is a person"
= "Every king is a person"

∃ Existential — "There is SOME x ..."

Pairs with ∧ (conjunction)

`∃x King(x) ∧ Rich(x)`
"There is an x that is king AND rich"
= "There exists a rich king"

⚠ Common Mistakes (Exam Trap!)

✗ **WRONG:** `∀x King(x) ∧ Rich(x)`
This says: "EVERYTHING is a king AND everything is rich!"
✓ **RIGHT:** `∀x King(x) ⇒ Rich(x)`

✗ **WRONG:** `∃x King(x) ⇒ Rich(x)`
Vacuously true! If anything is NOT a king, the ⇒ is true → ∃ satisfied.
✓ **RIGHT:** `∃x King(x) ∧ Rich(x)`

Memory Aid — Which connective with which quantifier?

`∀ goes with ⇒` ("For ALL x, **IF** it's a king, THEN it's rich")
`∃ goes with ∧` ("There EXISTS an x that is king **AND** rich — both must hold")

⚡ **EXAM ALERT — "State the correct quantifier-connective pairing with justification" appears in almost every paper.**

English → FOL Translation

⚡ EXAM ALERT — 5+ marks in almost every paper

#	English Sentence	FOL Translation
1	**Every student is smart**	`∀x Student(x) ⇒ Smart(x)`
2	**Some student is smart**	`∃x Student(x) ∧ Smart(x)`
3	**Everyone loves someone**	`∀x ∃y Loves(x, y)`
4	**Someone is loved by everyone**	`∃y ∀x Loves(x, y)`
5	**Not all birds fly**	`¬(∀x Bird(x) ⇒ Fly(x))  ≡  ∃x Bird(x) ∧ ¬Fly(x)`
6	**Every room adjacent to pit is breezy**	`∀r∀p (Adjacent(r,p) ∧ Pit(p)) ⇒ Breezy(r)`
7	**John's father is rich**	`Rich(FatherOf(John))`
8	**All students who study pass**	`∀x (Student(x) ∧ Studies(x)) ⇒ Pass(x)`
9	**No student likes Monday**	`∀x Student(x) ⇒ ¬Likes(x, Monday)`
10	**There is a crown on John's head**	`∃c Crown(c) ∧ On(c, HeadOf(John))`

Translation Patterns — Memorize These!

```
"Every/All X is Y"      →  ∀x X(x) ⇒ Y(x)          (∀ with ⇒)
"Some/There is X that Y"  →  ∃x X(x) ∧ Y(x)        (∃ with ∧)
"No X is Y"             →  ∀x X(x) ⇒ ¬Y(x)         OR  ¬∃x X(x) ∧ Y(x)
```

⚡ EXAM TIP — English → FOL: spot "every/some/no" → pick ∀/∃ → pair with ⇒/∧.

Quantifier Order Matters!

Swapping ∀ and ∃ changes the meaning completely

∀x ∃y Loves(x, y)

"Everyone loves SOMEONE"

For EACH person x, there exists some y (possibly different) that x loves.

```
John → loves Mary
Jane → loves Bob
Raj  → loves Priya
```

(each person has their own "someone")

Weaker claim — easier to satisfy

∃y ∀x Loves(x, y)

"ONE person EVERYONE loves"

There is ONE specific y such that EVERY person x loves that same y.

```
∃ y = Mother Teresa
John loves her ✓
Jane loves her ✓
Raj  loves her ✓
```

(everyone loves the SAME person)

Stronger claim — harder to satisfy

Visual Intuition:

⚠ **Exam Trap — "Does ∀x∃y P(x,y) imply ∃y∀x P(x,y)?" NO! The weaker does NOT imply the stronger. But the reverse IS true.**

⚡ **EXAM ALERT — "Explain why quantifier order matters with an example" — direct 4-mark question.**

⚡ EXAM ALERT — 3 Classic FOL Problems

These EXACT problems appear in 6 out of 7 KTU papers

Problem 1 — "John likes food" (Prove: John likes peanuts)

FOL Translation:

```
∀x Food(x) ⇒ Likes(John, x)          "John likes all food"
Food(Peanuts)                         "Peanuts is food"
Goal: Likes(John, Peanuts)  →  Unify {x/Peanuts} → Likes(John, Peanuts) ✓
```

Problem 2 — "Curiosity killed the cat"

FOL Translation:

```
∀x ∀y Cat(x) ∧ Curious(y) ⇒ Kill(y, x)    "Any cat is killed by curiosity"
Cat(Tuna)   ∧   Curious(Jack)              "Tuna is a cat, Jack is curious"
Goal: Kill(Jack, Tuna)  →  Unify {x/Tuna, y/Jack} → Kill(Jack, Tuna) ✓
```

Problem 3 — "West is criminal" (Weapons problem)

FOL Translation:

```
∀x American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
American(West)  ∧  Weapon(M1)  ∧  Sells(West,M1,Nono)  ∧  Hostile(Nono)
Goal: Criminal(West)  →  Unify {x/West, y/M1, z/Nono} → Criminal(West) ✓
```

Exam Strategy for ALL three:
① Write FOL sentences ② Identify goal ③ Show unification substitution ④ Apply Generalized Modus Ponens → derive goal

⚡ Unification & Most General Unifier (MGU)

EXAM ALERT — 5 out of 7 papers ask this (4-6 marks)

Unification = finding a substitution θ that makes two expressions IDENTICAL
Most General Unifier (MGU) = the simplest θ (doesn't over-commit). If θ_1 and θ_2 both work, MGU is more general.

Example 1 — Simple

UNIFY(**Knows(John, x)** , **Knows(John, Jane)**)

θ = { x / Jane }

Knows(John, Jane) = Knows(John, Jane) ✓

Example 2 — Variable ↔ Function

UNIFY(**Knows(John, x)** , **Knows(y, Mother(y))**)

θ = { y / John, x / Mother(John) }

Knows(John, Mother(John)) = Knows(John, Mother(John)) ✓

Example 3 — Nested Function

UNIFY(**P(x, f(x))** , **P(A, y)**)

θ = { x / A, y / f(A) }

P(A, f(A)) = P(A, f(A)) ✓

Example 4 — FAILS!

UNIFY(**Knows(John, x)** , **Knows(x, Jane)**)

CANNOT UNIFY ✗

x/John → Knows(John,John) vs Knows(John,Jane) → John≠Jane

⚠ Occur Check — Why Example 4 fails:
x appears inside Knows(x, Jane). If we substitute x/John, then the second arg becomes John, but first arg wants x=John → Knows(John, John) ≠ Knows(John, Jane). Same variable can't map to two different constants.

⚡ EXAM — "Find the MGU or show unification fails" — always 1 simple, 1 with function, 1 that fails. Practice all 3 types.

Generalized Modus Ponens — "Lifting" to FOL

How PL inference rules work with variables via unification

PL — Modus Ponens

```
Rule:   P ⇒ Q
Fact:   P
─────────────
Result: Q   ✓
```

FOL — Generalized Modus Ponens

```
Rule:   ∀x P(x) ⇒ Q(x)
Fact:   P(John)
Unify:  θ = {x/John}
Result: Q(John)   ✓
```

What does "Lifting" mean?

PL rules work on GROUND (specific) propositions. FOL rules work on GENERAL formulas with variables.
We "lift" PL → FOL by adding a **unification step** to match variables before applying the rule.

Worked Example — Step by Step

Step 1	**Identify rule**	`∀x King(x) ⇒ Mortal(x)`
Step 2	**Identify fact**	`King(John)`
Step 3	**Unify**	`UNIFY(King(x), King(John)) → θ = {x/John}`
Step 4	**Apply θ to conclusion**	`Mortal(x) with {x/John} → Mortal(John) ✓`

This is the engine behind Forward Chaining and Backward Chaining (next slides) — they repeatedly apply Generalized Modus Ponens.

⚡ FOL → CNF Conversion Recipe

EXAM ALERT — 6-8 mark question: convert + show each step

1	**Eliminate ⇒ and ⇔**	`A ⇒ B → ¬A ∨ B` `A ⇔ B → (¬A∨B) ∧ (A∨¬B)`
2	**Move ¬ inward (De Morgan's)**	`¬(A∧B) → ¬A∨¬B` `¬(A∨B) → ¬A∧¬B` `¬∀xP → ∃x¬P` `¬∃xP → ∀x¬P`
3	**Standardize variables apart**	Rename so each quantifier uses a DIFFERENT variable: `∀xP(x) ∧ ∀xQ(x) → ∀xP(x) ∧ ∀yQ(y)`
4	**Skolemize ∃ (remove existential)**	`∃x P(x) → P(c)` where c is a NEW Skolem constant. `∃x∀y Loves(x,y) → ∀y Loves(c,y)`
5	**Drop ∀ (all remaining are universal)**	After Skolemization, every variable is implicitly ∀ — just drop the quantifier symbols.
6	**Distribute ∨ over ∧**	`(A ∨ (B ∧ C)) → (A∨B) ∧ (A∨C)`. Result: conjunction of disjunctions = CNF!

Quick Example: ∀x (King(x) ⇒ ∃y Crown(y) ∧ Wears(x, y))

① **Eliminate ⇒:** ∀x (¬King(x) ∨ ∃y Crown(y) ∧ Wears(x,y))
② **¬ already inward** ✓ ③ Vars already unique ✓
④ **Skolemize ∃y:** ∀x (¬King(x) ∨ Crown(f(x)) ∧ Wears(x, f(x))) [y→f(x) Skolem function]
⑤ **Drop ∀:** ¬King(x) ∨ Crown(f(x)) ∧ Wears(x, f(x))
⑥ **Distribute ∨:** **(¬King(x) ∨ Crown(f(x))) ∧ (¬King(x) ∨ Wears(x, f(x))) ← CNF ✓**

⚡ **EXAM — Show ALL 6 steps even if trivial (write "no change needed"). Marks are for the process.**

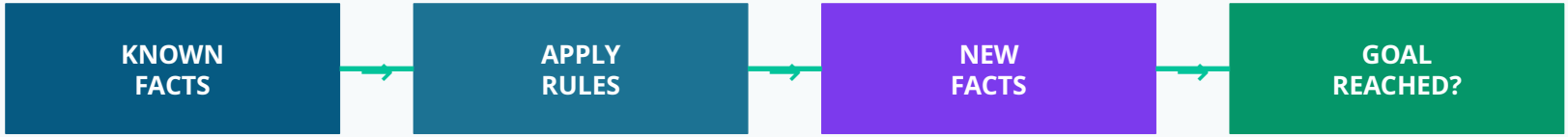
3.5

Inference in First-Order Logic

Forward and backward chaining — two ways to reason

Forward Chaining — Data-Driven Reasoning

Start with facts, apply rules, derive new facts until goal is reached



Direction: Facts → Rules → New Facts → ... → Goal

Cooking Analogy

Open the fridge → see what ingredients you have → check recipes → figure out what you can cook.

How Forward Chaining Works

1. Start with all known facts in the KB
2. For each rule, check if its premises match current facts (unification)
3. If ALL premises match → fire the rule → add conclusion as new fact
4. Repeat until goal is found OR no new facts can be derived

⚡ **EXAM ALERT — Forward chaining is a frequent Part B question. Know the algorithm + worked example.**

Forward Chaining — Worked Example: "West is Criminal"

THE classic exam problem — appeared in 6/7 papers

Knowledge Base

Rule: American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Facts:

F1: American(West)

F2: Weapon(M1)

F3: Sells(West, M1, Nono)

F4: Hostile(Nono)

⚡ Goal: Criminal(West)?

Step 1: Match Rule premises against facts

✓	American(x)	← F1: American(West)
✓	Weapon(y)	← F2: Weapon(M1)
✓	Sells(x,y,z)	← F3: Sells(West,M1,Nono)
✓	Hostile(z)	← F4: Hostile(Nono)

Substitution: { x/West, y/M1, z/Nono } — ALL premises match!

Apply rule → Derive Criminal(West) ✓ GOAL FOUND!

Key Insight

Forward chaining is data-driven: we didn't "aim" for Criminal(West). We just fired every rule we could, and the goal happened to appear. Like opening the fridge and discovering you can make pasta!

⚡ EXAM ALERT — "West is Criminal" forward chaining appeared in 6 out of 7 KTU papers.

Memorise the KB + show each premise match with substitution. Full marks = show the {x/West, y/M1, z/Nono} step.

Backward Chaining — Goal-Driven Reasoning

Start with the goal, find rules that prove it, recursively prove their premises

Direction: Goal ← Rules ← Premises ← Facts (work BACKWARDS!)

Biryani Analogy

"I want to make biryani!" → Do I have rice? ✓ → Spices? ✓ → Chicken? ✓ → I can make it!

How Backward Chaining Works

1. Start with the GOAL query (what you want to prove)
2. Find rules whose CONCLUSION matches the goal (unification)
3. For each matching rule, its premises become SUBGOALS to prove
4. Recursively prove each subgoal (may trigger more rules or match facts)
5. If ALL subgoals proved → goal is TRUE. If any fails → try another rule.

Best when: You have a SPECIFIC question. Don't compute everything — just check what's needed.

Backward Chaining — Worked Example: "West is Criminal"

Same KB as forward chaining, but we start from the GOAL

⚡ Goal: Criminal(West)?

↓

Step 1: Rule 1 concludes Criminal(x). Unify Criminal(West) with Criminal(x) → {x/West}

Step 2: Prove ALL premises (x = West):

✓	**Subgoal 1: American(West)?**	→ Matches Fact F1
✓	**Subgoal 2: Weapon(y)?**	→ Try y=M1, matches F2
✓	**Subgoal 3: Sells(West, M1, z)?**	→ Try z=Nono, matches F3
✓	**Subgoal 4: Hostile(Nono)?**	→ Matches Fact F4

All 4 subgoals proved → Criminal(West) is TRUE ✓

Recursive Structure

Each subgoal may itself need a rule → triggers more subgoals → like a proof tree.
Backward chaining = DFS through this tree. Prolog uses exactly this strategy!

Forward vs Backward Chaining — Comparison

Guaranteed exam question — memorise this table!

⚡ **EXAM ALERT — This comparison is a guaranteed Part A (3 marks) or Part B question.**

Aspect	Forward Chaining	Backward Chaining
Direction	Facts → Rules → New Facts → Goal	Goal → Rules → Premises → Facts
Driven by	Available DATA (data-driven)	Specific GOAL (goal-driven)
Good when	Lots of data, want ALL conclusions	Specific question to answer
Computes	Everything derivable (exhaustive)	Only what's needed for the goal
Analogy	"Open fridge → what can I cook?"	"Want biryani → do I have ingredients?"
Used in	Expert systems, monitoring, production rules	Prolog, query answering, diagnosis

Key Takeaway

Forward = bottom-up (data → conclusions). Backward = top-down (goal → facts).
Both use UNIFICATION to match rules. Forward is like BFS; backward is like DFS through the proof tree.

Resolution in FOL — "Curiosity Killed the Cat"

Full worked example — appeared in 3/7 papers, worth 8–10 marks

⚡ **EXAM ALERT — FOL resolution (CNF + Skolemization + refutation) = 8–10 mark Part B question.**

Step 1: Knowledge in FOL

(i) ∀x [LovesAllAnimals(x) ⇒ ∃y Loves(y,x)]
(ii) ∀x [Animal(x) ⇒ Loves(Jack, x)]
(iii) ∀x∀z [Kills(x,z) ∧ Animal(z) ∧ ¬∃y Loves(y,z) ⇒ Kills(x,z)]
(iv) Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
(v) Cat(Tuna) → Animal(Tuna)

Step 2: Convert to CNF

CNF clauses:
C1: ¬LovesAllAnimals(x) ∨ Loves(f(x), x)
(Skolem function f(x) for ∃y)
C2: ¬Animal(x) ∨ Loves(Jack, x)
C3: ¬Kills(x,z) ∨ ¬Animal(z) ∨ Loves(g(z),z)
(Skolem: no one loves it → ¬∃y = ∀y ¬Loves)
C4: Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
C5: Animal(Tuna)

Step 3: Negate goal & resolve (Refutation Proof)

Goal: Kills(Curiosity, Tuna). Negate: ¬Kills(Curiosity, Tuna) → Clause C6

Resolve C6 with C4: ¬Kills(Curiosity,Tuna) + Kills(Jack,Tuna) ∨ Kills(Curiosity,Tuna)
→ C7: Kills(Jack, Tuna)

Resolve C7 with C3 {x/Jack, z/Tuna}: Kills(Jack,Tuna) + ¬Kills(Jack,Tuna) ∨ ¬Animal(Tuna) ∨ Loves(g(Tuna),Tuna)
→ C8: ¬Animal(Tuna) ∨ Loves(g(Tuna), Tuna)

Resolve C8 with C5 (Animal(Tuna)):
→ C9: Loves(g(Tuna), Tuna)

Resolve C9 with C2 {x/Tuna}: Loves(g(Tuna),Tuna) + ¬Animal(Tuna) ∨ Loves(Jack,Tuna)
→ EMPTY CLAUSE □ CONTRADICTION! ∴ Curiosity killed Tuna ✓

Module 3 — Summary: All Key Concepts

One-slide master reference for revision

(1) PL vs FOL

Aspect	Prop. Logic	First-Order Logic
What	True/False facts	Objects + relations
Variables	None	Variables, constants
Quantifiers	None	∀ (for all), ∃ (exists)
Expressiveness	Limited	Very expressive

(2) Inference Methods

Method	Key Idea
Modus Ponens	P ⇒ Q, P ⊢ Q
Resolution	Resolve complementary literals
Forward Chain	Facts → Rules → New Facts
Backward Chain	Goal → Rules → Prove premises

(3) ⚡ Must-Memorise Formulas & Rules

Quantifiers: ∀x P(x) ⇒ Q(x) | ∃x P(x) ∧ Q(x) — ∀ uses ⇒, ∃ uses ∧

Resolution: (A ∨ B) ∧ (¬B ∨ C) → (A ∨ C) — resolve on complementary literal B

CNF Recipe: Eliminate ⇒ → Move ¬ inward → Standardise vars → Skolemise → Drop ∀ → Distribute ∨ over ∧

MGU: UNIFY(P(x,f(y)), P(A,f(B))) = {x/A, y/B} — simplest substitution making both identical

De Morgan: ¬(A ∧ B) = ¬A ∨ ¬B | ¬(A ∨ B) = ¬A ∧ ¬B | ¬∀x P = ∃x ¬P | ¬∃x P = ∀x ¬P

(4) Key Definitions to Remember

- Entailment (α ⊨ β): β is true in EVERY model where α is true
- Soundness: Inference derives ONLY entailed sentences (no false positives)
- Completeness: Inference can derive EVERY entailed sentence (no misses)
- Unification: Finding substitution θ that makes two expressions identical: UNIFY(p,q) = θ

Self-Check Questions

1. Describe the TELL/ASK loop of a KB agent. What's the advantage of declarative knowledge?
2. **In the Wumpus World, if you feel a breeze at [1,2] and no breeze at [1,1], what can you deduce?** ⚡
3. Define: model, entailment, soundness, completeness. Give an example of each.
4. **Write the truth table for A ⇒ B. When is it false? Why is F ⇒ T true?** ⚡
5. State Modus Ponens and the Resolution rule with examples.
6. **Convert (P ∧ Q) ⇒ R to CNF using the step-by-step recipe.** ⚡
7. **Prove by resolution: Given {A ⇒ B, B ⇒ C, A}, prove C.** ⚡
8. What does FOL add over propositional logic? Define constant, variable, predicate, function.
9. **Translate to FOL: "Every student who studies hard passes"; "Some teacher likes all students."** ⚡
10. **Find the MGU: UNIFY(P(x, f(y)), P(A, f(B))).** ⚡
11. **Explain forward and backward chaining with an example. When do you use each?** ⚡

⚡ = *Exam hot topics (Part A: 3 marks | Part B: 9 marks)*

Exam Cheat Sheet — Module 3 Quick Revision

Compact reference for last-minute revision before the exam

(1) 3 Recurring Problem Families (from past papers)

▸ **"John likes food"**	Convert FOL → CNF → Resolution proof	**8–10 marks**
▸ **"Curiosity killed the cat"**	FOL → Skolemization → CNF → Refutation	**8–10 marks**
▸ **"West is Criminal"**	Forward chaining OR backward chaining with full KB	**6–9 marks**

(2) CNF Conversion Recipe

1. Eliminate ⇒ : $A \Rightarrow B \rightarrow \neg A \vee B$
2. Move ¬ inward: $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$; $\neg\forall x \rightarrow \exists x \neg$
3. Standardise variables apart
4. Skolemise: $\exists x$ → replace with Skolem constant/function
5. Drop ∀ (all remaining vars are universal)
6. Distribute ∨ over ∧: $A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$

(3) Resolution Proof Method

1. Convert ALL premises to CNF (clauses)
2. NEGATE the goal → add as a clause
3. Find two clauses with complementary literals
   (P in one clause, ¬P in another)
4. Resolve: produce new clause (drop P and ¬P)
5. Add resolvent to clause set
6. Repeat until EMPTY CLAUSE □ (contradiction!)
7. □ found → goal is PROVED (by refutation)

(4) Forward vs Backward Chaining — Keywords for Exam

Forward Chaining:
- Data-driven • Bottom-up • Starts from facts
- Fires rules when premises match • Derives ALL conclusions
- Used in: expert systems, production rules, monitoring
- Analogy: "What can I cook with these ingredients?"

Backward Chaining:
- Goal-driven • Top-down • Starts from the query
- Finds rules → proves premises as subgoals • Only what's needed
- Used in: Prolog, query answering, medical diagnosis
- Analogy: "I want biryani — do I have ingredients?"

Exam Tip:

For ANY worked example: (1) Write the KB clearly, (2) Show each substitution/unification step, (3) State the rule used at each step.
Examiners award marks for PROCESS, not just the final answer. Even if the answer is wrong, steps get marks!

Module 3 Complete

Next: Module 4 — Reinforcement Learning (Learning from experience, not knowledge)