

UNIT 3

Knowledge Representation and Reasoning

(15 Hours) — Artificial Intelligence (CACS410)

⚠ Please Note

These notes were prepared for quick revision and **may contain errors or omissions**. Please take care: cross-check important definitions, formulas, algorithms and worked examples against your prescribed textbook, class notes and teacher's instructions before relying on them (especially for exams).

3.1 Knowledge and Knowledge Representation

Definition and Importance of Knowledge

Data are raw facts (e.g., 38), **information** is data with meaning (body temperature is 38 °C), and **knowledge** is organized information together with rules, relationships and experience that can be used to reason and make decisions (a temperature of 38 °C with cough suggests fever or flu). **Wisdom** is using knowledge well.

- An intelligent system cannot reason, plan, understand language or learn without knowledge about its world.
- Knowledge guides search (heuristics) and reduces the number of possibilities to examine.
- Separating knowledge (the knowledge base) from the reasoning mechanism makes systems easier to build, explain and update.

Issues in Knowledge Representation

- **Important attributes:** are there attributes (such as *is-a* and *instance-of*) that apply to every domain?
- **Relationships among attributes:** inverses, existence of an is-a hierarchy, single-valued attributes, techniques for reasoning about values.
- **Granularity:** at what level of detail should knowledge be represented?
- **Representing sets of objects and finding the right structure** when it is needed (indexing and retrieval).
- Handling **incomplete or uncertain knowledge, exceptions** (non-monotonic reasoning) and **change** (the frame problem).

- **Efficiency:** storing large amounts of knowledge and accessing it quickly.

Knowledge Representation Systems and Their Properties

A **knowledge representation (KR) system** consists of a *language* to express knowledge (the knowledge base) and an *inference mechanism* to derive new knowledge from it. A good KR system should have four properties:

Property	Meaning
Representational adequacy	Ability to represent all kinds of knowledge needed in the domain
Inferential adequacy	Ability to manipulate the structures to derive new knowledge from old
Inferential efficiency	Ability to add heuristic guidance so inference is fast
Acquisitional efficiency	Ability to acquire new knowledge easily, ideally automatically

Types of Knowledge

Type	Description	Example
Declarative	Facts about objects and events (“what”)	Kathmandu is the capital of Nepal
Procedural	How to do something; rules and procedures (“how”)	Steps to solve a quadratic equation
Meta-knowledge	Knowledge about knowledge	Which rule set is more reliable for diagnosis
Heuristic	Rules of thumb from experience	If the car won't start, check the battery first
Structural	Relations between concepts	A car has an engine, wheels and seats
Domain-specific vs. general	Knowledge of one area vs. common-sense knowledge	Medical knowledge vs. “water is wet”

The Role of Knowledge

Knowledge is the foundation of an AI system: it supplies the facts and rules for **reasoning**, lets the system **generalize** from experience, supports **understanding** of language and perception, helps **explain** conclusions and makes **search efficient**. As Feigenbaum put it, the power of a system comes largely from the knowledge it holds, not only from its reasoning method.

3.2 Knowledge Representation Techniques

Rule-Based Representation (Production Systems)

Knowledge is stored as **IF–THEN rules**: IF *condition* THEN *action/conclusion*. A production system has three parts: a **rule base** (long-term memory), a **working memory** (current facts) and an **inference engine** that repeats the *recognize–act cycle* — **match** rules against facts, **resolve conflicts** if several rules fire, **execute** the chosen rule.

Example — Rules

R1: IF the animal has feathers THEN it is a bird.

R2: IF the animal is a bird AND it cannot fly AND it swims THEN it is a penguin.

Facts: feathers, cannot fly, swims. Firing R1 adds “bird”; then R2 adds “penguin”.

Advantages: modular, natural, easy to add or edit rules, can explain reasoning. **Disadvantages:** large rule bases become slow and hard to maintain, rule interactions are unclear, poor at representing structured objects.

Semantic Networks (Semantic Nets)

A semantic net is a **graph** in which **nodes** represent objects or concepts and labelled **arcs** represent relations between them (*is-a*, *instance-of*, *has-part*, *can*, ...). Properties are **inherited** along *is-a* links; a lower-level node may override an inherited value (an exception).

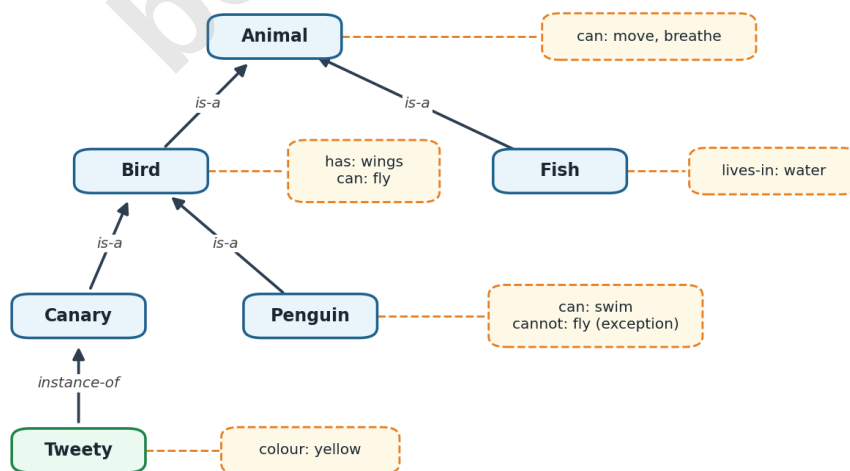


Figure 3.1 — A semantic network with inheritance and an exception

Example — Inference by inheritance

Query: *Can Tweety fly?* Tweety is an instance of Canary, Canary is-a Bird, and Bird has “can: fly”, so Tweety can fly. Query: *Can a penguin fly?* Penguin is-a Bird, but Penguin has the local value “cannot fly”, which overrides the inherited one.

Advantages: intuitive, shows relationships clearly, efficient inheritance. **Disadvantages:** no standard meaning for links, difficult to represent negation, quantifiers (“all”, “some”) and complex rules, searching a large net is slow.

Frames

Proposed by **Marvin Minsky (1974)**. A **frame** is a data structure describing a stereotyped object or situation. It has **slots** (attributes) filled with values, defaults or procedures, and frames are linked in a hierarchy so slots are inherited.

Frame: Car (class)	Slot value
is-a	Vehicle
wheels	4 (default)
engine	Petrol / diesel (default: petrol)
fuel-left	if-needed: read fuel gauge

Frame: MyCar (instance)	Slot value
instance-of	Car
owner	Ram
colour	Red
wheels	inherited: 4

- **Facets** of a slot: *value*, *default*, *range* (allowed values), and attached procedures — *if-needed* (compute when required) and *if-added* (run when a value is stored, also called demons).
- **Advantages:** organizes related knowledge together, supports default reasoning and inheritance. **Disadvantages:** hard to design well, weak inference, no clear formal semantics.

Logic-Based Representation

Knowledge is written as sentences in a formal logic — **propositional logic** or **first-order predicate logic** — with precise syntax and semantics, and new knowledge is derived by sound **inference rules**. It is expressive, unambiguous and supports provable reasoning. Its drawbacks are that theorem proving can be computationally expensive and that ordinary logic is monotonic (it cannot easily handle exceptions or uncertainty).

Technique	Strength	Weakness	Best for
Rules	Modular, easy to explain	Hard to maintain when large	Expert systems
Semantic nets	Visual, inheritance	No formal semantics	Concept hierarchies
Frames	Structured, defaults	Weak inference	Objects, situations
Logic	Precise, sound reasoning	Expensive, rigid	Formal reasoning, theorem proving

3.3 Propositional Logic

Syntax

A **proposition** is a declarative sentence that is either true or false. **Atomic sentences** are single proposition symbols ($P, Q, R, \dots$ or True/False). **Complex sentences** are built from them with **connectives**:

Connective	Symbol	Name	Reading
Not	$\neg P$	Negation	It is not the case that P
And	$P \wedge Q$	Conjunction	P and Q
Or	$P \vee Q$	Disjunction	P or Q (or both)
Implies	$P \rightarrow Q$	Implication	If P then Q
If and only if	$P \leftrightarrow Q$	Biconditional	P and Q have the same truth value

Precedence from highest to lowest: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ (use brackets to avoid doubt).

Semantics

A **model** assigns true or false to every proposition symbol. The truth value of a complex sentence is computed with **truth tables**.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

*Note: $P \rightarrow Q$ is false **only** when P is true and Q is false. If P is false the implication is true regardless of Q .*

Important Concepts and Equivalences

- **Entailment** $KB \models \alpha$: α is true in every model in which KB is true.
- **Valid** (tautology): true in all models. **Satisfiable**: true in at least one model. **Unsatisfiable** (contradiction): true in none.
- **Soundness**: an inference procedure derives only entailed sentences. **Completeness**: it can derive every entailed sentence.

Law	Equivalence
Double negation	$\neg(\neg P) \equiv P$
Implication elimination	$P \rightarrow Q \equiv \neg P \vee Q$
Contraposition	$P \rightarrow Q \equiv \neg Q \rightarrow \neg P$
Biconditional elimination	$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$
De Morgan's laws	$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$; $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
Distributivity	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$; $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
Commutativity / associativity	$P \wedge Q \equiv Q \wedge P$; $P \vee Q \equiv Q \vee P$; $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$

Rules of Inference

Rule	Premises	Conclusion
Modus Ponens	$P \rightarrow Q, P$	Q
Modus Tollens	$P \rightarrow Q, \neg Q$	$\neg P$
And-elimination	$P \wedge Q$	P (or Q)
And-introduction	P, Q	$P \wedge Q$
Or-introduction	P	$P \vee Q$

Rule	Premises	Conclusion
Hypothetical syllogism	$P \rightarrow Q, Q \rightarrow R$	$P \rightarrow R$
Disjunctive syllogism	$P \vee Q, \neg P$	Q
Resolution	$P \vee Q, \neg P \vee R$	$Q \vee R$

Conjunctive Normal Form (CNF)

A sentence is in **CNF** if it is a **conjunction of clauses**, where each **clause** is a disjunction of literals (a literal is a symbol or its negation), e.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee D)$. Every propositional sentence can be converted to CNF:

- **Step 1:** eliminate $\leftrightarrow$ using $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$.
- **Step 2:** eliminate $\rightarrow$ using $\alpha \rightarrow \beta \equiv \neg \alpha \vee \beta$.
- **Step 3:** move $\neg$ inward using De Morgan's laws and double negation.
- **Step 4:** distribute $\vee$ over $\wedge$.

Example — Convert $A \leftrightarrow (B \vee C)$ to CNF

Step 1: $(A \rightarrow (B \vee C)) \wedge ((B \vee C) \rightarrow A)$

Step 2: $(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$

Step 3: $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$

Step 4: $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

Proof by Resolution and the Resolution Algorithm

Resolution rule: from clauses $(l_1 \vee \dots \vee l_k)$ and $(m_1 \vee \dots \vee m_n)$ where l_i and m_j are complementary literals (one is the negation of the other), infer a new clause containing all the remaining literals of both. Resolving P with $\neg P$ gives the **empty clause** ($\square$), which represents a contradiction.

Resolution works by **refutation (proof by contradiction)**: to prove $KB \models \alpha$, show that $KB \wedge \neg \alpha$ is **unsatisfiable**.

```

function PL-RESOLUTION(KB, alpha)
  clauses <- CNF clauses of (KB and not alpha)
  new <- {}
  loop:
    for each pair of clauses Ci, Cj in clauses:
      resolvents <- PL-RESOLVE(Ci, Cj)
      if resolvents contains the empty clause: return TRUE      # KB entails alpha
      new <- new U resolvents
    if new is a subset of clauses: return FALSE      # nothing new: not entailed
    clauses <- clauses U new

```

Example 1 — Prove R from: $P \rightarrow Q, Q \rightarrow R, P$

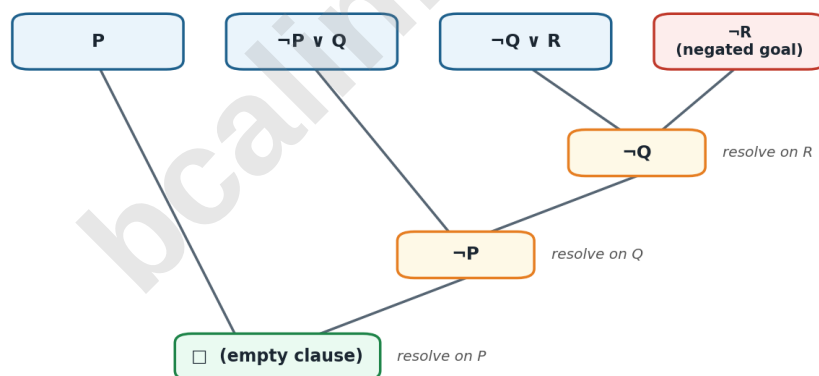
In English: if I study (P) I pass (Q); if I pass, my parents are happy (R); I study. Prove: parents are happy.

CNF clauses: (1) $\neg P \vee Q$ (2) $\neg Q \vee R$ (3) P and the negated goal (4) $\neg R$.

Solution

(2) and (4) resolve on R $\rightarrow \neg Q$. (1) and $\neg Q$ resolve on Q $\rightarrow \neg P$. (3) P and $\neg P$ resolve $\rightarrow \square$ (empty clause).

A contradiction is reached, so $KB \wedge \neg R$ is unsatisfiable and therefore **$KB \models R$** .



Empty clause derived $\Rightarrow KB \wedge \neg R$ is unsatisfiable $\Rightarrow KB \models R$

Figure 3.2 — Resolution refutation tree for Example 1

Example 2 — Prove C from: $A \vee B, \neg A \vee C, \neg B \vee C$ (proof by cases)

Negated goal: $\neg C$. Clauses: (1) $A \vee B$ (2) $\neg A \vee C$ (3) $\neg B \vee C$ (4) $\neg C$.

Solution

(2) + (4) $\rightarrow \neg A$. (3) + (4) $\rightarrow \neg B$. (1) + $\neg A \rightarrow B$. $B + \neg B \rightarrow \square$. Hence C is entailed.

Forward and Backward Chaining

For knowledge bases of **definite (Horn) clauses** such as $(A \wedge B \rightarrow C)$, two simple and efficient inference methods exist.

- **Forward chaining (data-driven):** start with the known facts; fire any rule whose premises are all known and add its conclusion to the facts; repeat until the query is derived or no new facts appear.
- **Backward chaining (goal-driven):** start from the query; find rules that conclude it; make their premises new sub-goals and prove them recursively, stopping at known facts.

Example — KB with rules and facts

Rules — R1: $P \rightarrow Q$; R2: $L \wedge M \rightarrow P$; R3: $B \wedge L \rightarrow M$; R4: $A \wedge P \rightarrow L$; R5: $A \wedge B \rightarrow L$. Facts: A, B. Query: Q?

Forward chaining

Known: A, B. R5 ($A \wedge B$) fires $\rightarrow$ L. R3 ($B \wedge L$) fires $\rightarrow$ M. R2 ($L \wedge M$) fires $\rightarrow$ P. R1 (P) fires $\rightarrow$ Q. Query proved.

(R4 adds only L, which is already known.)

Backward chaining

Goal $Q \leftarrow$ need P (R1). P $\leftarrow$ need L and M (R2). L $\leftarrow$ R5: A and B are facts ✓ (R4 would need P, which is already being proved, so it is skipped to avoid a loop).

M $\leftarrow$ R3: needs B ✓ and L ✓. So M, L, hence P, hence Q are all proved.

Point	Forward chaining	Backward chaining
Direction	Facts $\rightarrow$ conclusions (data-driven)	Goal $\rightarrow$ facts (goal-driven)
Starts with	Known facts	The query / hypothesis
May derive	Many irrelevant facts	Only what is relevant to the goal
Suited for	Monitoring, planning, design, configuration	Diagnosis, question answering, Prolog

Limitations of Propositional Logic

- **Low expressive power:** it cannot talk about objects and their properties or relations. “All students are clever” would need a separate proposition for every student.
- No **variables or quantifiers** (“for all”, “there exists”) and no **functions**.
- It **scales poorly:** the number of sentences grows rapidly with the number of objects.

- It cannot express general rules such as “every man is mortal” and then apply them to a particular man.
- It is monotonic and cannot handle uncertainty. These limits lead to **first-order predicate logic**.

3.4 Predicate Logic (First-Order Predicate Logic, FOPL)

Syntax

FOPL models the world with **objects**, **properties** and **relations**.

Element	Meaning	Examples
Constants	Names of specific objects	Ram, 2, Kathmandu
Variables	Stand for any object	x, y, z
Predicates	Properties of / relations between objects; true or false	Student(x), Loves(x, y), Greater(x, y)
Functions	Map objects to objects; denote an object	FatherOf(x), Plus(x, y)
Connectives	$\neg \wedge \vee \rightarrow \leftrightarrow$	Student(x) $\wedge$ Smart(x)
Quantifiers	$\forall$ (for all), $\exists$ (there exists)	$\forall x$ Human(x) $\rightarrow$ Mortal(x)
Equality	Two terms refer to the same object	FatherOf(Ram) = Hari

A **term** is a constant, variable or function applied to terms. An **atomic sentence** is a predicate applied to terms. **Complex sentences** combine atomic sentences with connectives and quantifiers. A **ground** sentence contains no variables.

Semantics

A model contains a **domain** of objects. An **interpretation** maps each constant to an object, each predicate to a relation on objects and each function to a function on objects. An atomic sentence is true if the objects named stand in the relation named.

Quantification

- **Universal $\forall x P(x)$** : P holds for *every* object x in the domain. Normally used with $\rightarrow$: $\forall x$ (Human(x) $\rightarrow$ Mortal(x)).
- **Existential $\exists x P(x)$** : P holds for *at least one* object. Normally used with $\wedge$: $\exists x$ (Student(x) $\wedge$ Clever(x)).

- **Common mistakes:** $\forall x (\text{Student}(x) \wedge \text{Smart}(x))$ says *everyone* is a smart student; $\exists x (\text{Student}(x) \rightarrow \text{Smart}(x))$ is trivially true if anything is not a student.
- **Duality:** $\neg \forall x P(x) \equiv \exists x \neg P(x)$; $\neg \exists x P(x) \equiv \forall x \neg P(x)$.
- **Nested quantifiers:** $\forall x \exists y \text{Loves}(x, y)$ means everyone loves someone; $\exists y \forall x \text{Loves}(x, y)$ means there is someone loved by everyone. The order matters.

English sentence	FOPL
All humans are mortal	$\forall x (\text{Human}(x) \rightarrow \text{Mortal}(x))$
Some students are clever	$\exists x (\text{Student}(x) \wedge \text{Clever}(x))$
No student failed	$\neg \exists x (\text{Student}(x) \wedge \text{Failed}(x)) \equiv \forall x (\text{Student}(x) \rightarrow \neg \text{Failed}(x))$
Everyone loves someone	$\forall x \exists y \text{Loves}(x, y)$
Ram likes all food	$\forall x (\text{Food}(x) \rightarrow \text{Likes}(\text{Ram}, x))$
Not all birds can fly	$\neg \forall x (\text{Bird}(x) \rightarrow \text{Fly}(x)) \equiv \exists x (\text{Bird}(x) \wedge \neg \text{Fly}(x))$
Socrates is a man	$\text{Man}(\text{Socrates})$

Horn Clauses

A **Horn clause** is a disjunction of literals with **at most one positive literal**. A **definite clause** has exactly one positive literal: $\neg P_1 \vee \dots \vee \neg P_n \vee Q$, which is equivalent to the rule $P_1 \wedge \dots \wedge P_n \rightarrow Q$. A fact is a definite clause with no negative literals; a **goal clause** has no positive literal. Inference with Horn clauses (forward/backward chaining) is efficient, sound and complete — the basis of **Prolog**.

Inference in FOPL — 1. By Converting to Propositional Logic

- **Universal Instantiation (UI):** from $\forall v \alpha$ infer $\text{SUBST}(\{v/g\}, \alpha)$ for *any* ground term g .
Example: from $\forall x (\text{King}(x) \wedge \text{Greedy}(x) \rightarrow \text{Evil}(x))$ infer $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \rightarrow \text{Evil}(\text{John})$.
- **Existential Instantiation (EI):** from $\exists v \alpha$ infer $\text{SUBST}(\{v/k\}, \alpha)$ where k is a **new constant** (Skolem constant) not used elsewhere. Example: from $\exists x (\text{Crown}(x) \wedge \text{OnHead}(x, \text{John}))$ infer $\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$.
- After instantiating all quantifiers with all ground terms, the KB becomes propositional and a propositional inference method can be used. **Drawback:** with function symbols there are infinitely many ground terms, and many generated sentences are irrelevant, so this is very inefficient.

2. Rules of Inference for FOPL

Rule	Meaning
Universal instantiation	$\forall x P(x) \vdash P(c)$ for any constant c
Existential instantiation	$\exists x P(x) \vdash P(k)$ for a new constant k
Existential generalization	$P(c) \vdash \exists x P(x)$
Modus Ponens (propositional rules)	$P, P \rightarrow Q \vdash Q$
Generalized Modus Ponens	A single step that unifies and applies a rule (see lifting below)

3. Unification and Lifting

Unification finds a substitution θ that makes two logical expressions identical: $\text{UNIFY}(p, q) = \theta$ where $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$. The **most general unifier (MGU)** imposes the fewest restrictions on the variables.

p	q	θ (unifier)
Knows(John, x)	Knows(John, Jane)	{x/Jane}
Knows(John, x)	Knows(y, Bill)	{x/Bill, y/John}
Knows(John, x)	Knows(y, Mother(y))	{y/John, x/Mother(John)}
Knows(John, x)	Knows(x, Elizabeth)	fail — both use x; <i>standardize apart</i> (rename x to z) and it becomes {x/Elizabeth, z/John}

The algorithm compares the two expressions term by term: identical terms succeed; a variable is bound to the other term (after an **occur check** that the variable does not appear inside that term); different constants or predicates cause failure.

Lifting: instead of instantiating with all constants, inference rules are “lifted” to work directly with variables. **Generalized Modus Ponens:** for atomic sentences p_i' , p_i and q with $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$ for all i : from $p_1', \dots, p_n'$ and $(p_1 \wedge \dots \wedge p_n \rightarrow q)$ infer $\text{SUBST}(\theta, q)$.

Example — Generalized Modus Ponens

Rule: $\text{King}(x) \wedge \text{Greedy}(x) \rightarrow \text{Evil}(x)$. Facts: $\text{King}(\text{John})$, $\text{Greedy}(y)$.

Unify: $\theta = \{x/\text{John}, y/\text{John}\}$. Conclusion: **Evil(John)**.

4. Conjunctive Normal Form for FOPL

- 1. Eliminate implications and biconditionals.
- 2. Move $\neg$ inward ($\neg\forall x P \equiv \exists x \neg P$; $\neg\exists x P \equiv \forall x \neg P$; De Morgan).
- 3. Standardize variables apart (each quantifier uses a unique variable name).
- 4. **Skolemize**: remove $\exists$ by replacing the variable with a Skolem constant, or a Skolem function of the enclosing universally quantified variables.
- 5. Drop the universal quantifiers (all remaining variables are universal).
- 6. Distribute $\vee$ over $\wedge$ to obtain a conjunction of clauses.

Example — Convert: $\forall x [\forall y \text{ Animal}(y) \rightarrow \text{Loves}(x, y)] \rightarrow [\exists y \text{ Loves}(y, x)]$ (“everyone who loves all animals is loved by someone”)

1. $\forall x [\neg\forall y (\neg\text{Animal}(y) \vee \text{Loves}(x, y))] \vee [\exists y \text{ Loves}(y, x)]$
2. $\forall x [\exists y (\text{Animal}(y) \wedge \neg\text{Loves}(x, y))] \vee [\exists y \text{ Loves}(y, x)]$
3. $\forall x [\exists y (\text{Animal}(y) \wedge \neg\text{Loves}(x, y))] \vee [\exists z \text{ Loves}(z, x)]$
4. Skolemize ($y \rightarrow F(x)$, $z \rightarrow G(x)$): $\forall x [\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$
- 5–6. $[\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)] \wedge [\neg\text{Loves}(x, F(x)) \vee \text{Loves}(G(x), x)]$

5. Inference using Resolution and the Resolution Refutation System (RRS)

Lifted resolution: for clauses with literals l_i and m_j such that $\text{UNIFY}(l_i, \neg m_j) = \theta$, the resolvent is $\text{SUBST}(\theta, \text{all other literals of both clauses})$. The **Resolution Refutation System** proves a goal α by (1) converting the KB to CNF, (2) adding the negated goal $\neg\alpha$ in CNF, (3) repeatedly resolving pairs of clauses (applying unifiers) until the **empty clause** is derived. It is **refutation-complete**: if the KB entails α , an empty clause will eventually be found. Strategies to keep the search small: *set of support*, *unit preference*, *input/linear resolution* and *subsumption*.

Example 1 — Socrates

KB: (1) $\forall x \text{ Human}(x) \rightarrow \text{Mortal}(x)$ (2) $\text{Human}(\text{Socrates})$. Prove: $\text{Mortal}(\text{Socrates})$.

Clauses: C1: $\neg\text{Human}(x) \vee \text{Mortal}(x)$; C2: $\text{Human}(\text{Socrates})$; negated goal C3: $\neg\text{Mortal}(\text{Socrates})$.

Solution

C3 and C1 with $\{x/\text{Socrates}\} \rightarrow \neg\text{Human}(\text{Socrates})$. With C2 $\rightarrow \square$. Hence $\text{Mortal}(\text{Socrates})$ is proved.

Example 2 — “John likes peanuts”

1. John likes all kinds of food. 2. Apples and chicken are food. 3. Anything anyone eats and is not killed by is food.

4. Bill eats peanuts and is still alive. 5. Sue eats everything Bill eats. **Prove: John likes peanuts.**

FOPL: (a) $\forall x \text{ Food}(x) \rightarrow \text{Likes}(\text{John}, x)$ (b) $\text{Food}(\text{Apple}) \wedge \text{Food}(\text{Chicken})$ (c) $\forall x \forall y (\text{Eats}(x, y) \wedge \neg \text{Killed}(x)) \rightarrow \text{Food}(y)$

(d) $\text{Eats}(\text{Bill}, \text{Peanuts}) \wedge \text{Alive}(\text{Bill})$ (e) $\forall x \text{ Eats}(\text{Bill}, x) \rightarrow \text{Eats}(\text{Sue}, x)$ (f) $\forall x \text{ Alive}(x) \rightarrow \neg \text{Killed}(x)$

Clauses and Resolution Steps

CNF: 1. $\neg \text{Food}(x) \vee \text{Likes}(\text{John}, x)$ 2. $\text{Food}(\text{Apple})$ 3. $\text{Food}(\text{Chicken})$ 4. $\neg \text{Eats}(x, y) \vee \text{Killed}(x) \vee \text{Food}(y)$ 5a. $\text{Eats}(\text{Bill}, \text{Peanuts})$ 5b. $\text{Alive}(\text{Bill})$ 6. $\neg \text{Eats}(\text{Bill}, x) \vee \text{Eats}(\text{Sue}, x)$ 7. $\neg \text{Alive}(x) \vee \neg \text{Killed}(x)$ 8. **$\neg \text{Likes}(\text{John}, \text{Peanuts})$** (negated goal)

- $8 + 1 \{x/\text{Peanuts}\} \rightarrow \neg \text{Food}(\text{Peanuts})$
- $\neg \text{Food}(\text{Peanuts}) + 4 \{y/\text{Peanuts}\} \rightarrow \neg \text{Eats}(x, \text{Peanuts}) \vee \text{Killed}(x)$
- $+ 5a \{x/\text{Bill}\} \rightarrow \text{Killed}(\text{Bill})$
- $\text{Killed}(\text{Bill}) + 7 \{x/\text{Bill}\} \rightarrow \neg \text{Alive}(\text{Bill})$
- $\neg \text{Alive}(\text{Bill}) + 5b \rightarrow \square$. Contradiction $\Rightarrow$ **John likes peanuts.**

3.5 Handling Uncertain Knowledge

Real agents rarely have complete knowledge. Uncertainty arises from **partial observability**, **noisy sensors**, **laziness** (too many exceptions to list) and **ignorance** of the domain. Pure logic fails: “Toothache $\rightarrow$ Cavity” is wrong, as toothache has other causes. **Probability theory** summarizes the uncertainty as a **degree of belief** between 0 and 1.

Random Variables and Basic Probability

A **random variable** takes values from a domain (Boolean: $\text{Cavity} \in \{\text{true}, \text{false}\}$; discrete: $\text{Weather} \in \{\text{sunny}, \text{rain}, \text{cloudy}\}$; or continuous). Axioms:

- $0 \leq P(a) \leq 1$ for every proposition a ; $P(\text{true}) = 1$, $P(\text{false}) = 0$.
- $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$ (and $P(\neg a) = 1 - P(a)$).

Prior and Posterior Probability

- **Prior (unconditional) probability** $P(a)$: degree of belief in a *before* any evidence, e.g., $P(\text{cavity}) = 0.2$.

- **Posterior (conditional) probability** $P(a | b)$: belief in a *after* observing evidence b .
Defined as $P(a | b) = P(a \wedge b) / P(b)$, for $P(b) > 0$.
- **Product rule:** $P(a \wedge b) = P(a | b) \cdot P(b)$. **Independence:** a and b are independent if $P(a \wedge b) = P(a) \cdot P(b)$.

Inference using the Full Joint Distribution

The **full joint probability distribution** lists the probability of every combination of values of all variables. Any query can be answered by summing the matching entries.

	toothache, catch	toothache, $\neg$ catch	$\neg$ toothache, catch	$\neg$ toothache, $\neg$ catch
cavity	0.108	0.012	0.072	0.008
$\neg$ cavity	0.016	0.064	0.144	0.576

Example — Queries on the joint distribution (the entries sum to 1)

$P(\text{toothache}) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$

$P(\text{cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$

$P(\text{cavity} | \text{toothache}) = P(\text{cavity} \wedge \text{toothache}) / P(\text{toothache}) = (0.108 + 0.012) / 0.2 = 0.6$;

$P(\neg \text{cavity} | \text{toothache}) = 0.08 / 0.2 = 0.4$

Normalization: $P(\text{Cavity} | \text{toothache}) = \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle$, where $\alpha = 1/P(\text{toothache})$.

Note: With n Boolean variables the table has 2^n entries, so the full joint distribution is impractical for large problems. Independence and Bayesian networks solve this.

Bayes' Rule and Its Use

From the product rule, $P(b | a) = P(a | b) \cdot P(b) / P(a)$. It lets us compute a **diagnostic** probability (cause given effect) from a **causal** one (effect given cause), which is usually easier to obtain: $P(\text{cause} | \text{effect}) = P(\text{effect} | \text{cause}) \cdot P(\text{cause}) / P(\text{effect})$.

Example 1 — Meningitis

A doctor knows: $P(\text{stiff neck} | \text{meningitis}) = 0.7$, $P(\text{meningitis}) = 1/50000$, $P(\text{stiff neck}) = 0.01$.

Solution

$P(m | s) = 0.7 \times (1/50000) / 0.01 = 0.0014$. Even with a stiff neck, meningitis is unlikely (0.14%) because it is so rare.

Example 2 — Medical test

1% of people have a disease D. A test is positive for 90% of people with D and for 5% of people without D. A person tests positive; what is $P(D \mid \text{positive})$?

Solution

$P(+) = 0.9 \times 0.01 + 0.05 \times 0.99 = 0.009 + 0.0495 = 0.0585$. $P(D \mid +) = 0.009 / 0.0585 \approx \mathbf{0.154}$ (about 15%).

A positive result is far from conclusive because the disease is rare.

Naïve Bayes: if the effects $e_1 \dots e_n$ are conditionally independent given the cause, $P(\text{Cause} \mid e_1 \dots e_n) = \alpha \cdot P(\text{Cause}) \cdot \prod P(e_i \mid \text{Cause})$. It is simple and widely used (spam filters, text classification).

Bayesian Networks

A **Bayesian (belief) network** is a **directed acyclic graph (DAG)** in which each node is a random variable and each edge $X \rightarrow Y$ means X directly influences Y . Every node has a **conditional probability table (CPT)** $P(X_i \mid \text{Parents}(X_i))$. The full joint distribution is the product of local distributions:

$$P(x_1, x_2, \dots, x_n) = \prod_i P(x_i \mid \text{parents}(X_i))$$

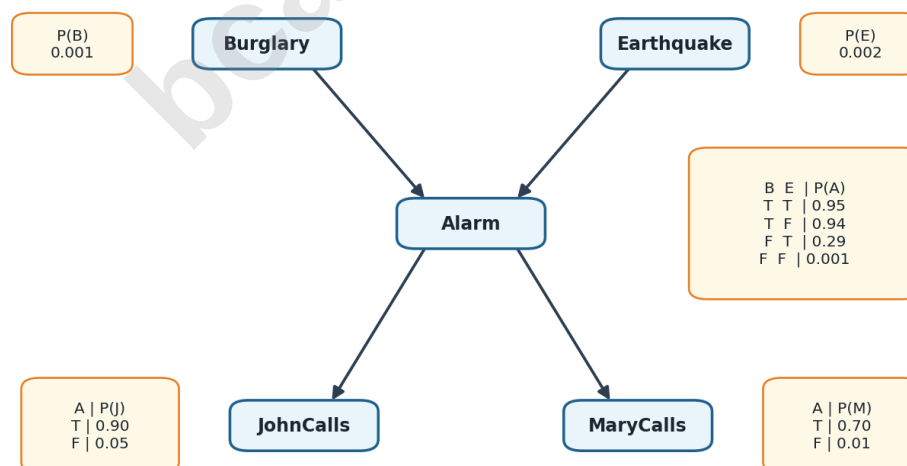


Figure 3.3 — The burglary-alarm Bayesian network with CPTs

Story: an alarm may be triggered by a burglary or an earthquake; neighbours John and Mary call if they hear the alarm (they are sometimes wrong).

Example — Probability of a specific event

Find the probability that the alarm sounded, both John and Mary called, but there was **no burglary and no earthquake**.

Solution

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j | a) \cdot P(m | a) \cdot P(a | \neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 \approx \mathbf{0.00063} \end{aligned}$$

Reasoning (Inference) in Bayesian Networks

- **Causal (top-down):** from causes to effects, e.g., $P(\text{JohnCalls} | \text{Burglary})$.
- **Diagnostic (bottom-up):** from effects to causes, e.g., $P(\text{Burglary} | \text{JohnCalls})$.
- **Intercausal (explaining away):** once an alarm is known, learning about an earthquake lowers the belief in a burglary.
- **Exact inference by enumeration:** a query $P(X | e) = \alpha \sum$ over hidden variables of the joint, written as a product of CPT entries. **Variable elimination** does the same more efficiently; **approximate** inference uses sampling.

Example — $P(\text{Burglary} | \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true})$

$P(B | j, m) = \alpha \sum_e \sum_a P(B) P(e) P(a | B, e) P(j | a) P(m | a)$. Summing over e and a for $B = \text{true}$ and $B = \text{false}$ gives the unnormalized values 0.000592 and 0.001492.

Result

Normalizing: $P(\text{Burglary} | j, m) = \langle 0.284, 0.716 \rangle$. Even though both neighbours called, the chance of a real burglary is only about **28%**, because burglaries are very rare.

Conditional independence: in a Bayesian network, each node is conditionally independent of its non-descendants given its parents. This is why the network needs far fewer probabilities than a full joint table (here 10 numbers instead of $2^5 - 1 = 31$).

Key Takeaways

- ✓ Knowledge = organized information; KR systems need representational/inferential adequacy and efficiency and good acquisition.
- ✓ Techniques: rules (IF-THEN), semantic nets (graph + inheritance), frames (slots, defaults), and logic.
- ✓ Propositional logic: connectives, truth tables, CNF, resolution refutation; forward (data-driven) vs backward (goal-driven) chaining.

- ✓ Propositional logic cannot express objects/relations; FOPL adds predicates, functions and quantifiers $\forall, \exists$.
- ✓ FOPL inference: UI/EI, unification (MGU), generalized Modus Ponens, CNF with Skolemization, and resolution refutation.
- ✓ Uncertainty: prior vs posterior probability, Bayes' rule, joint distributions, and Bayesian networks (joint = product of CPT entries).

Review Questions

- What is knowledge? Explain the issues in knowledge representation and the properties of a good KR system.
- Compare rule-based, semantic net, frame and logic-based representations.
- Convert $(P \leftrightarrow Q) \rightarrow R$ to CNF and prove a given conclusion using resolution.
- Differentiate forward and backward chaining with an example.
- Why is propositional logic insufficient? Translate five English sentences into FOPL.
- Explain unification with examples. Convert a FOPL sentence into CNF (with Skolemization).
- Prove “John likes peanuts” using resolution refutation.
- State Bayes' rule and solve a numeric example. Explain Bayesian networks and compute a joint probability.

— End of Unit 3 —

Reminder: these notes may contain errors — verify with your textbook and teacher.