Artificial Intelligence

B.Tech(AIDS)_V Semester

Academic Year: 2024-2025

Dr D. L. Sreenivasa Reddy

Email: dlsrinivasareddy_aids@cbit.ac.in

Unit-III-Syllabus

Knowledge & Reasoning:

Knowledge-Based Logic Agents

Logic

First-Order Logic

Syntax-Semantics in FOL

Simple usage, Inference Procedure

Inference in FOL

Reduction

Inference Rules

Forward Chaining

Backward Chaining

Resolution.

Logical Agents(Knowledge Based Agents)

"Logical AI:
The idea is that an agent can represent knowledge of its world, its goals and the current situation by sentences in logic and decide what to do by inferring that a certain action or course of action is appropriate to achieve its goals."

Knowledge-based agents

Main Components of Knowledge base system

- **Knowledge base**: A knowledge base is a centralized repository of data specific Models to a given field.
- A knowledge base is a set of **sentences**
- Sentence is not identical to the sentences of English and other natural languages.
- **Knowledge representation:** Knowledge representation language and represents some assertion about the world.
- Sometimes we dignify a sentence with the name axiom, when the sentence is taken as given without being derived from other sentences.
- **Inference:** Deriving new sentences from old.
- **Inference must obey** the requirement that when one **ASKs** a question of the knowledge base, the answer should follow from what has been **told (or TELLed**) to the knowledge base previously
- **Inference engine**: Is set of Logic rules to retrieve conclusions from the combination of Models to execute inputs with less complexity

Knowledge-based agents

- Intelligent agents need **knowledge** about the world to choose good actions/decisions.
- Knowledge = {**sentences**} in a knowledge representation language (formal language).
- A sentence is an assertion about the world.
- A knowledge-based agent is composed of:
  1. Knowledge base: **domain-specific** content.
  2. Inference mechanism: **domain-independent** algorithms.

Knowledge based Agent

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

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

Representation language are hidden inside three functions that implement the interface between the sensors and actuators on one side and the core representation and reasoning system on the other

1. **MAKE-PERCEPT-SENTENCE** constructs a sentence asserting that the agent perceived the given percept at the given time
2. **MAKE-ACTION-QUERY** constructs a sentence that asks what action should be done at the current time
3. **MAKE-ACTION-SENTENCE** constructs a sentence asserting that the chosen action was executed

The details of the inference mechanisms are hidden inside **TELL and ASK**

Logical Concepts

- Mathematical foundations=**symbolic** representation to help statements and arguments through the study of formal systems and inferences.
- **Formal logic** deals with symbolic abstractions.
  - **A) Propositional logic**: Declarative statement(either true/false, not both)
  - **B)Predicate Logic(First Order Logic):** (Getting proposition from propositions using logic)(consistency/inconsistency/validity)

Propositional Calculus:

1. Is a language of propositions
2. It uses a set of rules and connectors called operators

   (~(Not), v(OR), ^(And), ->(Implies), <->(equivalence)

Logic

- **Knowledge base**: a set of sentences in a formal representation, logic
- **Logics**: are formal languages for representing knowledge to extract conclusions
  - **Syntax**: defines well-formed sentences in the language
  - **Semantic**: defines the truth or meaning of sentences in a world
- **Inference**: a procedure to derive a new sentence from other ones.
- **Logical entailment**: is a relationship between sentences. It means that a sentence follows logically from other sentences

$$KB \models \alpha$$

Propositional logic

- Propositional logic (PL) is the simplest logic.
- **Syntax of PL**: defines the allowable sentences or propositions.
- **Definition (Proposition)**: A proposition is a declarative statement that's either True or False.
- **Atomic proposition**: single proposition symbol. Each symbol is a proposition. Notation: upper case letters and may contain subscripts.
- **Compound proposition**: constructed from from atomic propositions using parentheses and logical connectives.

Atomic proposition

Examples of atomic propositions:

- 2+2=4 is a true proposition
- $W_{1,3}$ is a proposition. It is true if there is a Wumpus in [1,3]
- "If there is a stench in [1,2] then there is a Wumpus in [1,3]" is a proposition
- "How are you?" or "Hello!" are not propositions. In general, statement that are questions, commands, or opinions are not propositions.

Compound proposition

Examples of compound/complex propositions:

Let p, p_1, and p_2 be propositions

- **Negation** $\neg p$ is also a proposition. We call a **literal** either an atomic proposition or its negation. E.g., $W_{1,3}$ is a positive literal, and $\neg W_{1,3}$ is a negative literal.
- **Conjunction** $p_1 \wedge p_2$. E.g., $W_{1,3} \wedge P_{3,1}$
- **Disjunction** $p_1 \vee p_2$ E.g., $W_{1,3} \vee P_{3,1}$
- **Implication** $p_1 \rightarrow p_2$. E.g., $W_{1,3} \wedge P_{3,1} \rightarrow \neg W_{2,2}$
- **If and only if** $p_1 \leftrightarrow p_2$. E.g., $W_{1,3} \leftrightarrow \neg W_{2,2}$

Truth tables

- The semantics define the rules to determine the truth of a sentence.
- Semantics can be specified by truth tables.
- Boolean values domain: T,F
- n-tuple: $(x_1, x_2, ..., x_n)$
- Operator on n-tuples : $g(x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$
- Definition: A truth table defines an operator g on n- tuples by specifying a boolean value for each tuple
- Number of rows in a truth table? $R = 2^n$

Building Truth Tables

Negation:

p	$\neg p$
T	F
F	T

Conjunction:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Interpretation →

Exclusive or:

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Implication:

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional or If and only if (IFF):

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

1. Formula/model=Compound Statement/Proposition=ά
2. If a formula contains n atoms, it requires 2 power n interpretations

Truth Tables for connectives

Summary:

$\rightarrow$ IFF

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

$2^2 = 4$

$P \longrightarrow q$ implication

LHS premisse

RHS Conclusion

Building Truth tables

Precedence of operators

- Just like arithmetic operators, there is an operator precedence when evaluating logical operators as follows:
  1. Expressions in parentheses are processed (inside to outside)
  2. Negation
  3. AND
  4. OR
  5. Implication
  6. Biconditional
  7. Left to right
- Use parentheses whenever you have any doubt!

Building Truth tables

Building propositions

p	q	r	¬r	p ∨ q	p ∨ q → ¬r
T	T	T	F	T	F
T	T	F	T	T	T
T	F	T	F	T	F
T	F	F	T	T	T
F	T	T	F	T	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	F	T

Logical equivalence

- Two propositions p and q are logically equivalent if and only if the columns in the truth table giving their truth values agree.
- We write this as $p \Leftrightarrow q$ or $p \equiv q$.

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

show $p \wedge (\sim q \vee p) \equiv p$

Properties of operators

- Commutativity: $p \wedge q = q \wedge p \qquad p \vee q = q \vee p$
- Associativity: $(p \wedge q) \wedge r = p \wedge (q \wedge r) \qquad (p \vee q) \vee r = p \vee (q \vee r)$
- Identity element: $p \wedge True = p \qquad p \vee True = True$
- $\neg(\neg p) = p$
- $p \wedge p = p \qquad p \vee p = p$
- Distributivity:
  $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$
  $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
- $p \wedge (\neg p) = False$ and $p \vee (\neg p) = True$
- DeMorgan's laws:
  $\neg(p \wedge q) = (\neg p) \vee (\neg q)$
  $\neg(p \vee q) = (\neg p) \wedge (\neg q)$

Properties of Operators

TABLE 6 Logical Equivalences.

Equivalence	*Name*
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

TABLE 7 Logical Equivalences Involving Conditional Statements.

$p \to q \equiv \neg p \vee q$

$p \to q \equiv \neg q \to \neg p$

$p \vee q \equiv \neg p \to q$

$p \wedge q \equiv \neg(p \to \neg q)$

$\neg(p \to q) \equiv p \wedge \neg q$

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$

$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$

$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$

$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

TABLE 8 Logical Equivalences Involving Biconditional Statements.

$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$

$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Inference (Modus Ponens)

$$\frac{p \qquad p \to q}{q}$$

$$\frac{warm \qquad warm \to sunny}{sunny}$$

Inference (Modus Ponens)

Horn clauses: a proposition of the form:

$$p_1 \wedge \ldots \wedge p_n \to q$$

Modus Ponens deals with **Horn clauses**:

$$\frac{p_1, \ldots, p_n \qquad (p_1 \wedge \ldots \wedge p_n) \to q}{q}$$

Inference (Modus Tollens)

$$\frac{\neg q \qquad p \to q}{\neg p}$$

$\neg beach \to \neg hot$

$$\frac{\neg beach \qquad hot \to beach}{\neg hot}$$

$$\frac{hot \qquad hot \to beach}{beach.}$$

Common Rules

- Addition: $\dfrac{p}{p \vee q}$
- Simplification: $\dfrac{p \wedge q}{q}$
- Disjunctive-syllogism: $\dfrac{\begin{array}{c} p \vee q \\ \neg p \end{array}}{q}$
- Hypothetical-syllogism: $\dfrac{\begin{array}{c} p \to q \\ q \to r \end{array}}{p \to r}$

Tautology and contradiction

- **Tautology** is a proposition which is always true
- **Contradiction** is a proposition which is always false
- **Contingency** is a proposition which is neither a tautology or a contradiction

P	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Contrapositive, inverse, etc.

- Given an **implication** $p \to q$
- The **converse** is: $q \to p$
- The **contrapositive** is: $\neg q \to \neg p$
- The **inverse** is: $\neg p \to \neg q$

Example: Hot is a sufficient condition for my going to the beach.

- The **implication** is:
- The **converse** is:
- The **contrapositive** is:
- The **inverse** is:

Entailment and Inference

- **Semantics**: Determine entailment by **Model Checking**, that is enumerate all models and show that the sentence α must hold in all models.

$$KB \models \alpha$$

- **Syntax**: Determine entailment by **Theorem Proving**, that is apply rules of inference to KB to build a proof of α *without enumerating and checking all models*.

$$KB \vdash \alpha$$

- But how are entailment and inference related?

Soundness & Completeness

- We want an inference algorithm that is:
  1. **Sound**: does not infer false formulas, that is, derives only entailed sentences.

  $$\{\alpha | KB \vdash \alpha\} \subseteq \{KB \models \alpha\}$$

  2. **Complete**: derives ALL entailed sentences.

  $$\{\alpha | KB \vdash \alpha\} \supseteq \{KB \models \alpha\}$$

Validity & satisfiability

- A sentence is valid (aka tautology) if it is true in all models, e.g., $True$, $p \vee p$, $p \Rightarrow p$, $(p \wedge (p \Rightarrow q)) \Rightarrow q$

- Validity is connected to inference via the Deduction Theorem: $KB \models \alpha$ IFF $(KB \Rightarrow \alpha)$ is valid

- A sentence is satisfiable if it is true in some model e.g., $p \vee q, r$

- A sentence is unsatisfiable if it is true in no models e.g., $p \wedge \neg p$

- Satisfiability is connected to inference via the following: $KB \models \alpha$ IFF $(KB \wedge \neg\alpha)$ is unsatisfiable i.e., prove α by contradiction

Determining entailment

- Given a Knowledge Base (KB) (set of sentences in PL), given a query α, output whether KB entails α, noted: $KB \models \alpha$
- We will see two ways of doing proofs in PL:
  - **Model checking** enumerate the models (truth table enumeration, exponential).
  - **Application of inference rules** (proof checking/theorem proving): Syntactic derivations with rules like Modus Ponens (Backward chaining and forward chaining). A proof is a sequence of inference rule applications.

Validity of Formula

- **Truth table method is a time waste and tedious process,**
- **Instead can check the same by the below methods**
  - 1. Natural Deduction Systems
  - 2. Axiomatic System
  - 3. Semantic tableau method
  - 4. Resolution refutation method

First Order Logic

- Alternative to PL: Another more powerful language, **First Order Logic (FOL)**.
- Syntax of FOL:
  - Terms are either:
    * Constants symbols (e.g., A, 10, Columbia),
    * Variables (e.g., x, y)
    * Functions of terms, e.g., sqrt(x), sum(1,2).
  - Atomic formulas: predicates applied to terms, e.g., brother(x,y), above(A,B)
  - Connectives: $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$
  - Equality: $=$
  - Quantifiers: $\forall$ $\exists$
  - Connectives, equality, quantifiers can be applied to atomic formulas to create sentences in FOL.

First Order Logic

All squares are clean:

$$\forall\ x\ Square(x)\ \Rightarrow\ Clean(x)$$

There exists some dirty squares:

$$\exists\ x\ Square(x)\ \wedge\ \neg Clean(x)$$

Question: Now, can we express that some squares have chairs on top?

Note:

- $\forall x\ P(x)$ is like $P(A) \wedge P(B) \wedge \ldots$
- $\exists x\ P(x)$ is like $P(A) \vee P(B) \vee \ldots$
- $\neg\forall x\ P(x)$ is like $\exists\ x\ \neg P(x)$
- $\forall x\ \exists\ y\ likes(x, y)$ is NOT like $\exists y\ \forall x\ likes(x, y)$

First Order Logic

- All birds fly:

$$\forall\ x\ bird(x)\ \Rightarrow\ Fly(x)$$

- All birds except penguins fly:

$$\forall\ x\ bird(x)\ \wedge \neg penguin(x) \Rightarrow\ Fly(x)$$

- Every kid likes candy:

$$\forall\ x\ Kid(x)\ \Rightarrow\ Likes(x, candy)$$

- Some kids like candy:

$$\exists\ x\ Kid(x)\ \wedge\ Likes(x, candy)$$

- Brothers are sibling:

$$\forall x, y\ Brothers(x, y)\ \Rightarrow\ Sibling(x, y)$$

- One's mother is one's female parent:

$$\forall x, y\ Mother(x, y) \Leftrightarrow Female(x) \wedge Parent(x, y)$$

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - **Syntax**: formal structure of sentences
  - **Semantics**: truth of sentences wrt models
  - **Entailment**: necessary truth of one sentence given another
  - **Inference**: deriving sentences from other sentences
  - **Soundness**: derivations produce only entailed sentences
  - **Completeness**: derivations can produce all entailed sentences
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Forward, backward chaining are linear in time, complete for Horn clauses Resolution is complete for propositional logic.

Natural Deduction System

- ND is based on the set of few deductive inference rules.
- The name natural deductive system is given because it mimics the pattern of natural reasoning.
- It has about 10 deductive inference rules.

Conventions:

- E for Elimination, I for Introducing.
- P, P_k , $(1 \leq k \leq n)$ are atoms.
- α_k, $(1 \leq k \leq n)$ and β are formulae.

ND RULES:

Rule 1: I-$\wedge$ (Introducing $\wedge$)

I-$\wedge$: If $P_1, P_2, \ldots, P_n$ then $P_1 \wedge P_2 \wedge \ldots \wedge P_n$

Interpretation: If we have hypothesized or proved P_1, P_2, … and P_n, then their conjunction $P_1 \wedge P_2 \wedge \ldots \wedge P_n$ is also proved or derived.

Rule 2: E-$\wedge$ (Eliminating $\wedge$)

E-$\wedge$: If P1 $\wedge$ P2 $\wedge$ …$\wedge$ Pn then Pi ($1 \leq i \leq n$)

Interpretation: If we have proved P1 $\wedge$ P2 $\wedge$ … $\wedge$ Pn , then any Pi is also proved or derived. This rule shows that $\wedge$ can be eliminated to yield one of its conjuncts.

Rule 3: I-V (Introducing V)

I-V : If P_i ($1 \leq i \leq n$) then $P_1 \vee P_2 \vee \ldots \vee P_n$

Interpretation: If any Pi ($1 \leq i \leq n$) is proved, then $P_1 \vee \ldots \vee P_n$ is also proved.

ND RULES...

Rule 4: E-V (Eliminating V)

E-V : If $P_1 \vee \ldots \vee P_n$, $P_1 \rightarrow P$, … , $P_n \rightarrow P$ then P

Interpretation: If $P_1 \vee \ldots \vee P_n$, $P_1 \rightarrow P$, … , and $P_n \rightarrow P$ are proved, then P is proved.

Rule 5: I- $\rightarrow$ (Introducing $\rightarrow$)

I- $\rightarrow$: If from $\alpha_1, \ldots, \alpha_n$ infer β is proved then $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \beta$ is proved

Interpretation: If given $\alpha_1, \alpha_2, \ldots$ and α_n to be proved and from these we deduce β then $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta$ is also proved.

Rule 6: E- $\rightarrow$ (Eliminating $\rightarrow$) - *Modus Ponen*

E- $\rightarrow$: If $P_1 \rightarrow P$, P_1 then P

ND RULES…

Rule 7: I- $\leftrightarrow$ (Introducing $\leftrightarrow$)

I- $\leftrightarrow$: If $P_1 \rightarrow P_2$, $P_2 \rightarrow P_1$ then $P_1 \leftrightarrow P_2$

Rule 8: E- $\leftrightarrow$ (Elimination $\leftrightarrow$)

E- $\leftrightarrow$: If $P_1 \leftrightarrow P_2$ then $P_1 \rightarrow P_2$, $P_2 \rightarrow P_1$

Rule 9: I- ~ (Introducing ~)

I- ~ : If from P infer $P_1 \wedge \sim P_1$ is proved then ~P is proved

Rule 10: E- ~ (Eliminating ~)

E- ~ : If from ~ P infer $P_1 \wedge \sim P_1$ is proved then P is proved

- If a formula β is derived / proved from a set of premises / hypotheses $\{ \alpha_1,\ldots, \alpha_n \}$,
  - then one can write it as **from** α_1**,** **…,** α_n **infer** β.
- In natural deductive system,
  - a theorem to be proved should have a form **from** α**1, …,** α**n** **infer** β.
- Theorem **infer** β means that
  - there are no premises and β is true under all interpretations i.e., β is a tautology or valid.

- If we assume that $\alpha \rightarrow \beta$ is a premise, then we conclude that β is proved if α is given i.e.,
  - if 'from α infer β' is a theorem then $\alpha \rightarrow \beta$ is concluded.
  - The converse of this is also true.

Deduction Theorem: Infer $(\alpha_1 \Lambda \alpha_2 \Lambda \ldots \Lambda \alpha_n \rightarrow \beta)$ is a theorem of natural deductive system if and only if

from $\alpha_1, \alpha_2, \ldots, \alpha_n$ infer β is a theorem.

Useful tips: To prove a formula $\alpha_1 \Lambda \alpha_2 \Lambda \ldots \Lambda \alpha_n \rightarrow \beta$, it is sufficient to prove a theorem **from $\alpha_1, \alpha_2, \ldots, \alpha_n$ infer** β.

Example

Example1: Prove that $P \wedge (Q \vee R)$ follows from $P \wedge Q$

Solution: This problem is restated in natural deductive system as "**from $P \wedge Q$ infer $P \wedge (Q \vee R)$**". The formal proof is given as follows:

{Theorem}	**from $P \wedge Q$ infer $P \wedge (Q \vee R)$**	
{ premise}	$P \wedge Q$	(1)
{ E-$\wedge$, (1)}	P	(2)
{ E-$\wedge$, (1)}	Q	(3)
{ I-$\vee$, (3) }	$Q \vee R$	(4)
{ I-$\wedge$, (2, 4)}	**$P \wedge (Q \vee R)$**	**Conclusion**

Axiomatic System

- AS is based on the set of only three axioms and one rule of deduction.
  - It is minimal in structure but as powerful as the truth table and natural deduction approaches.
  - The proofs of the theorems are often difficult and require a guess in selection of appropriate axiom(s) and rules.
  - These methods basically require forward chaining strategy where we start with the given hypotheses and prove the goal.
  - Only two logical operators not(~) and implies (->) are allowed.

  (V , Λ , <-> can be converted into the above operators).

  Example:

  A Λ B = ~(A–>~B)

  A V B = ~A->B

  A<-> B = (A->B) Λ (B–>A) = ~[((A->B) -> ~((B–>A)]

- Three axioms and one rule of deduction.

Axiom1 (A1): $\alpha \rightarrow (\beta \rightarrow \alpha)$

Axiom2 (A2): $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$

Axiom3 (A3): $(\sim \alpha \rightarrow \sim \beta) \rightarrow (\beta \rightarrow \alpha)$

Modus Ponen (MP) defined as follows:

Hypotheses: $\alpha \rightarrow \beta$ and α ***Consequent:*** β

Definition: A *deduction* of a formula in Axiomatic System for Propositional Logic is a sequence of well-formed formulae $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that for each i, ($1 \leq i \leq n$), either

- Either α_i is an *axiom* or α_i is a *hypothesis* (given to be true)
- Or α_i is derived from α_j and α_k where j, k < i using modus ponen inference rule.

- We call α_i to be a *deductive consequence* of $\{\alpha_1, \ldots, \alpha_{i-1}\}$.
- It is denoted by $\{\alpha_1, \ldots, \alpha_{i-1}\} \vdash \alpha_i$. More formally, deductive consequence is defined on next slide.

Examples

Establish the following:

Ex1:

$\{Q\} \mid\!- (P \to Q)$ i.e., $P \to Q$ is a deductive consequence of $\{Q\}$.

{Hypothesis}	Q	(1)
{Axiom A1}	$Q \to (P \to Q)$	(2)
{MP, (1,2)}	$\mathbf{P \to Q}$	**proved**

Ex2:

$\{ P \to Q,\ Q \to R \} \vdash (P \to R)$ i.e., $P \to R$ is a deductive consequence of $\{ P \to Q,\ Q \to R \}$.

{Hypothesis}	$P \to Q$	(1)
{Hypothesis}	$Q \to R$	(2)
{Axiom A1}	$(Q \to R) \to (P \to (Q \to R))$	(3)
{MP, (2, 3)}	$P \to (Q \to R)$	(4)
{Axiom A2}	$(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$	(5)
{MP , (4, 5)}	$(P \to Q) \to (P \to R)$	(6)
{MP, (1, 6)}	$\mathbf{P \to R}$	**proved**

Semantic Tableau System

- Earlier approaches require
  - construction of proof of a formula from given set of formulae and are called direct methods.
- In **semantic tableaux**,
  - the set of rules are applied systematically on a formula or set of formulae to establish its consistency or inconsistency.
- *Semantic tableau*
  - binary tree constructed by using semantic rules formula as a root
- Assume α and β be any two formulae.

- RULES
- Let α and β be any two formulae.

Rule 1: A tableau for a formula ($\alpha \wedge \beta$) is constructed by adding both α and β to the same path (branch). This can be represented as follows:

$$\begin{array}{l} \alpha \wedge \beta \\ \alpha \\ \beta \end{array}$$

- *Semantic tableau*
  - binary tree constructed by using semantic rules formula as a root
- Assume α and β be any two formulae.

Interpretation: $\alpha \wedge \beta$ is true if both α and β are true

Rule 2: A tableau for a formula $\sim(\alpha \wedge \beta)$ is constructed by adding two alternative paths one containing $\sim\alpha$ and other containing $\sim\beta$

$$\begin{array}{ccc} & \sim(\alpha \wedge \beta) & \\ \swarrow & & \searrow \\ \sim\alpha & & \sim\beta \end{array}$$

Interpretation: $\sim(\alpha \wedge \beta)$ is true if either $\sim\alpha$ or $\sim\beta$ is true

Rule 3: A tableau for a formula ($\alpha \vee \beta$) is constructed by adding two new paths one containing α and other containing β.

$$\alpha \vee \beta$$

$$\alpha \qquad\qquad \beta$$

Interpretation: $\alpha \vee \beta$ is true if either α or β is true

Rule 4: A tableau for a formula ~ ($\alpha \vee \beta$) is constructed by adding both ~ α and ~ β to the same path. This can be expressed as follows:

$$\sim(\alpha \vee \beta)$$
$$\sim\alpha$$
$$\sim\beta$$

Rule 5: Semantic tableau for ~~ α

$$\sim\sim\alpha$$
$$\alpha$$

Rule 6: Semantic tableau for $\alpha \rightarrow \beta$

$$\alpha \rightarrow \beta$$

$$\sim\alpha \qquad\qquad \beta$$

Rule 7: Semantic tableau for ~ ($\alpha \rightarrow \beta$)

$$\sim(\alpha \rightarrow \beta)$$
$$\alpha$$
$$\sim\beta$$

Rule 8: Semantic tableau for $\alpha \leftrightarrow \beta$ $\alpha \leftrightarrow \beta \cong (\alpha \wedge \beta) \vee (\sim\alpha \wedge \sim\beta)$

$$\alpha \leftrightarrow \beta$$

$$\alpha \wedge \beta \qquad\qquad \sim\alpha \wedge \sim\beta$$

Rule 9: Semantic tableau for ~ ($\alpha \leftrightarrow \beta$)

$$\sim(\alpha \leftrightarrow \beta) \cong (\alpha \wedge \sim\beta) \vee (\sim\alpha \wedge \beta)$$

$$\sim(\alpha \leftrightarrow \beta)$$

$$\alpha \wedge \sim\beta \qquad\qquad \sim\alpha \wedge \beta$$

Consistency and Inconsistency: Satisfiability and Unsatisfiability

- If an atom P and ~ P appear on a same path of a semantic tableau,

  - then inconsistency is indicated and such path is said to be **contradictory** or **closed** (finished) path.
  - Even if one path remains **non contradictory** or **unclosed** (open), then the formula α at the root of a tableau is **consistent**.

Valuation

- A valuation ν is said to be a **model** of α (or ν **satisfies** α) iff $\nu(\alpha) = T$.
- In tableaux approach, model for a consistent formula α is constructed as follows:
  - On an open path, assign truth values to atoms (positive or negative) of α which is at the root of a tableau such that α is made to be true.
  - It is achieved by assigning truth value T to each atomic formula (positive or negative) on that path.

Example: Show that

α : $(P \wedge Q \rightarrow R) \wedge (\sim P \rightarrow S) \wedge Q \wedge \sim R \wedge \sim S$ is inconsistent(Unsatisfiable) using tableaux method.

{T-root} $(P \wedge Q \rightarrow R) \wedge (\sim P \rightarrow S) \wedge Q \wedge \sim R \wedge \sim S$ (1)

{Apply rule 1 to 1}

$P \wedge Q \rightarrow R$ (2)

$\sim P \rightarrow S$ (3)

Q

$\sim R$

$\sim S$

{Apply rule 6 to 3} P S

Closed: $\{S, \sim S\}$ on the path

{Apply rule 6 to 2)} $\sim (P \wedge Q)$ R

Closed $\{R, \sim R\}$

$\sim P$ $\sim Q$

Closed $\{P, \sim P\}$ **Closed** $\{\sim Q, Q\}$

Problem: Show that α: $(Q \wedge \sim R) \wedge (R \rightarrow P)$ is consistent(satisfiable) and find its model.

Solution:

{T-root}		$(Q \wedge \sim R) \wedge (R \rightarrow P)$	(1)
{Apply rule 1 to 1}		$(Q \wedge \sim R)$	(2)
		$(R \rightarrow P)$	(3)
{Apply rule 1 to 2}		Q	
{Apply rule 6 to 3}		$\sim R$	

$\sim R$ P

open   **open**

- Since tableau for α has open paths, we conclude that α is consistent.
- The models are constructed by assigning T to all atomic formulae appearing on open paths.
  - Assign Q = T and ~R = T i.e., R = F.
    - **So { Q = T, R = F } is a model of α.**
  - Assign Q = T and ~R = T and P = T.
    - **So { P = T , Q = T, R = F } is another model.**
- **Useful Tip:**
  - Thumb rule for constructing a tableau is to apply non branching rules before the branching rules in any order

Soundness and Completeness

- **Theorem:** (Soundness)

If α is tableau provable ($\vdash \alpha$), then α is valid ($\models \alpha$) i.e., $\vdash \alpha \Rightarrow \models \alpha$.

- **Theorem:** (Completeness)

If α is valid, then α is tableau provable i.e., $\models \alpha \Rightarrow \vdash \alpha$.

- If S is a set of formulae. The formula ά is said to be tableau provable from S(S|- ά) if there is a contradictory tableau from S with ~ ά as root/
- A formula ά is said to be logical consequence of a set S if and only if ά is tableau provable from S
- If ά is tableau provable (|- ά) then it is also valid(|= ά) and vice versa
- **Clause**: it is used to denote a formula containing the Boolean operators ~ and V. can be Ck=(L1vL1VL3…..)
- A formula is said to be in ***normal form,*** *if it is constructed using only natural connectives{~, V, ^}*
- ***Literal Lij is a positive or negative atoms***

Example - Validity

Example: Show that $\alpha : P \rightarrow (Q \rightarrow P)$ is valid

Solution: In order to show that α is a valid, we will try to show that α is tableau provable i.e., $\sim \alpha$ is inconsistent.

{**T-root**}	$\sim(P \rightarrow (Q \rightarrow P))$	(1)
{Apply rule7 to 1}	P	
	$\sim(Q \rightarrow P)$	(2)
{Apply rule 7 to 2}	Q	
	$\sim P$	
	Closed $\{P, \sim P\}$	

Hence $P \rightarrow (Q \rightarrow P)$ is valid.

Resolution and Refutation

- **Resolution refutation** is another simple method to prove a formula by contradiction.
  - Here negation of goal to be proved is added to given set of clauses.
  - It is shown then that there is a refutation in new set using resolution principle.
- **Resolution:** During this process we need to identify
  - two clauses, one with positive atom (P) and other with negative atom (~P) for the application of resolution rule.
- Resolution is based on modus ponen inference rule.
  - This method is most favoured for developing computer based theorem provers.
  - Automatic theorem provers using resolution are simple and efficient systems .
- Resolution is performed on special types of formulae called **clauses**.
  - Clause is propositional formula expressed using
    - {V, ~ } operators.

Conjunctive and Disjunctive Normal Forms

- In **Disjunctive Normal Form** (DNF),
  - a formula is represented in the form
  - $(L_{11} \wedge \ldots\ldots \wedge L_{1n}) \vee \ldots\ldots \vee (L_{m1} \wedge \ldots\ldots \wedge L_{mk})$, where all L_{ij} are literals. It is a disjunction of conjunction.
- In **Conjunctive Normal Form** (CNF),
  - a formula is represented in the form
  - $(L_{11} \vee \ldots\ldots \vee L_{1n}) \wedge \ldots \ldots \wedge (L_{p1} \vee \ldots\ldots \vee L_{pm})$,

    where all L_{ij} are literals. It is a conjunction of disjunction.
- A **clause** is a special formula expressed as disjunction of literals. If a clause contains only one literal, then it is called unit clause.

Conversion of a Formula to its CNF

- Each formula in Propositional Logic can be easily transformed into its equivalent DNF or CNF representation using equivalence laws .
  - Eliminate $\rightarrow$ and $\leftrightarrow$ by using the following equivalence laws.

$$P \rightarrow Q \cong \sim P \vee Q$$

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

  - Eliminate double negation signs by using

$$\sim\sim P \cong P$$

- Use De Morgan's laws to push ~ (negation) immediately before atomic formula.

$$\sim(P \land Q) \cong \sim P \lor \sim Q$$

$$\sim(P \lor Q) \cong \sim P \land \sim Q$$

- Use distributive law to get CNF.

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

- We notice that CNF representation of a formula is of the form
  - $(C_1 \land \ldots\ldots \land C_n)$, where each C_k, $(1 \leq k \leq n)$ is a clause that is disjunction of literals.

Resolution of Clauses

- If two clauses C_1 and C_2 contain a complementary pair of literals {L, ~L}, then
  - these clauses can be resolved together by deleting L from C_1 and ~ L from C_2 and constructing a new clause by the disjunction of the remaining literals in C_1 and C_2.
- The new clause thus generated is called **resolvent** of C_1 and C_2 .
  - Here C_1 and C_2 are called **parents** of resolved clause.
  - If the resolvent contains one or more set of complementary pair of literals, then resolvent is always true.

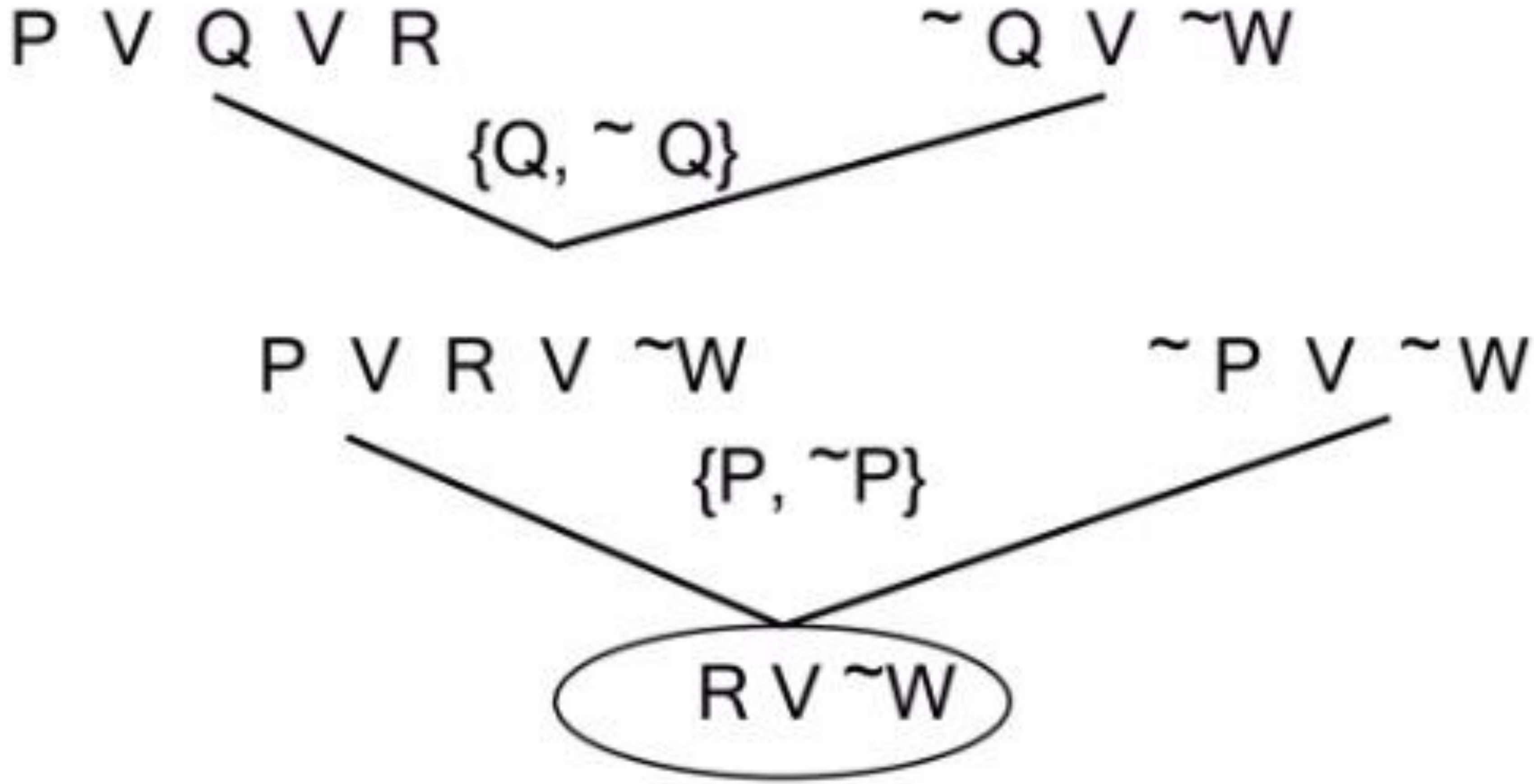
Resolution Tree

- Inverted binary tree is generated with the last node of the binary tree to be a resolvent.
- This also called resolution tree.

Example: Find resolvent of:

$$C_1 = P \vee Q \vee R$$
$$C_2 = \sim Q \vee \sim W$$
$$C_3 = \sim P \vee \sim W$$

Example- Resolution Tree



- Thus Resolvent(C_1,C_2, C_3) = R V ~W

Example: "Mary will get her degree if she registers as a student and pass her exam. She has registered herself as a student. She has passed her exam". Show that she will get a degree.

Solution: Symbolize above statements as follows: R: Mary is a registered student

P: Mary has passed her exam D: Mary gets her degree

- The formulae corresponding to above listed sentences are as follows:

- Mary will get her degree if she registers as a student and pass her exam.

  $$R \wedge P \rightarrow D \cong (\sim R \vee \sim P \vee D)$$

- She has registered herself as a student.

  R
- She has passed her exam.

  P
- Conclude “Mary will get a degree”.

  D

Example – Cont…

- Set of clauses are:
  - S = {~ R V ~ P V D , R, P }
- Add negation of "Mary gets her degree (= D)" to S.
- New set S' is:
  - S' = {~ R V ~ P V D , R, P, ~ D}
- We can easily see that empty clause is deduced from above set.
- Hence we can conclude that “Mary gets her degree”

Deriving Contradiction