## Propositional Logic Language

Language { available symbols. / formation rules

### Propositional Logic Syntax

Symbols { ¬ , ∨ , ∧ , ⊃ (if...then...) , ≡ (Equivalence) (Logical Symbols)
p1 , p2 (Non-logical Symbols : proposition Symbols)

Formation Rules

① A proposition symbol is a sentence

② if α , β is sentence , ¬α , α∧β , α∨β , α⊃β , α≡β are sentences

③ no other expressions are sentences

### Truth Conditions

Truth assignment (interpretation) : Set of propositional symbols ⟶ {T, F}.

Truth Table :

| $p$ | $q$ | $\neg p$ | $p \wedge q$ | $p \vee q$ | $p \supset q$ | $p \equiv q$ |
|---|---|---|---|---|---|---|
| $T$ | $T$ | $F$ | $T$ | $T$ | $T$ | $T$ |
| $T$ | $F$ | $F$ | $F$ | $T$ | $F$ | $F$ |
| $F$ | $T$ | $T$ | $F$ | $T$ | $T$ | $F$ |
| $F$ | $F$ | $T$ | $F$ | $F$ | $T$ | $T$ |

### Entailment and tautology

① v satisfies a sentence α iff v(α) = T

② a Set of Sentence Σ = {a1 ... an} entails α iff (∀ ai ∈ Σ , v(ai) = T) ⇒ v(α) = T

written Σ ⊨ α

③ α is a tautology iff ∅ ⊨ α , written ⊨ α

Deduction Theorem.

{α1 ... an} ⊨ α iff ⊨ a1 ∧ a2 ... ∧ an ⊃ α

### Tautology Example

- De Morgan's laws:
  $\neg(p \wedge q) \equiv \neg p \vee \neg q$ and $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- Distributive laws:
  $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ and $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- Excluded middle: $p \vee \neg p$
- Contradiction: $\neg(p \wedge \neg p)$
- Contraposition: $p \supset q \equiv \neg q \supset \neg p$
- Exportation: $p \wedge q \supset r \equiv p \supset (q \supset r)$
- Tautologies that use ¬ and ∨ to define all other connectives:
  $p \wedge q \equiv \neg(\neg p \vee \neg q)$ $\quad$ $p \supset q \equiv \neg p \vee q$
  $(p \equiv q) \equiv (p \supset q) \wedge (q \supset p)$

### Resolution and Refutation

Conjunctive Normal Form (CNF)

α ⟶ CNF: α' (⊨ α ≡ α')

① α≡β ⇒ (α⊃β) ∧ (β∧α) , α⊃β ≡ ¬α∨β

② ¬(α∧β) ⇒ (¬α∨¬β) , ¬(α∨β) ⇒ (¬α∧¬β)

③ (α∧β)∨γ ⇒ (α∨γ)∧(β∨γ)

④ α∨α ⇒ α , ¬(¬α) ⇒ α.

⑤ α∨¬α ⇒ null

e.x. ((p⊃q)⊃p)⊃q = ¬(¬(¬p∨q)∨p)∨q = ((¬p∨q)∧¬p)∨q
= (¬p∨(q∧¬p))∨q = (¬p∨q)∨(¬p∧q)
= (¬p∨q)

Resolvent $\quad p \vee C_1 \;,\; p \vee C_2 \Rightarrow C_1 \vee C_2$ (Resolvent)

Derivation of c :

{ C1 , C2 ..... , Cn = C
Ci ∈ S or Ci is a resolvent of two earlier clauses in derivation

Denote : S ⟶ C.

if S → c then S ⊨ c.

if S → [] then S ⊨ false (i.e. S is unsatisfiable)

Theorem: S is unsatisfiable iff S → []

Proof by refutation:

Σ ⊨ α iff Σ ∪ {¬α} is unsatisfiable

⇒ Σ ∪ {¬α} ⟶ []

- Check if $\mathcal{S} \rightarrow []$:
  1. Check if [] is in $\mathcal{S}$. If yes, then return "unsatisfiable".
  2. Check if there are two clauses $c_1$ and $c_2$ in $\mathcal{S}$ such that they resolve to produce a $c_3$ not already in $\mathcal{S}$. If no such $c_3$ can be generated, then return "satisfiable".
  3. Add $c_3$ to $\mathcal{S}$ and go back to the first step.

## First-Order Logic Language

① Objects , ② Facts : properties about or relations among these objects

### Alphabet :

Logical Symbols: { punctuations "," parentheses "( )"
¬, ∨, ∧, ⊃, ≡
quantifiers: ∀ , ∃
equality : =
variables : x, y, z ...

Non-logical Symbols

{ predicate symbols: Dog(fido) : fido is a dog
function symbols: Sum(x, y) : Sum of x, y

Logical Expression : ( term )

{ a constant , a variable is term
if t1 ... tn are terms. ⇒ f(t1, t2 ... tn) is a term

A *sentence* is a logical expression that represents a fact:

- if $t_1, \cdots, t_n$ are terms, and $P$ is an $n$-ary predicate (relation) symbol, then $P(t_1, ..., t_n)$ is a sentence (*atomic sentence*).
- if $t_1$ and $t_2$ are terms, then $t_1 = t_2$ is a sentence (*atomic sentence*).
- if $\alpha$ and $\beta$ are sentences, then $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, $(\alpha \equiv \beta)$ are also sentences (follow the same precedence order as propositional logic).
- if $\alpha$ is a sentence, and $v$ is a variable, then $(\forall v\ \alpha)$, $(\exists v\ \alpha)$ are also sentences.

- All purple mushrooms are poisonous:

$$\forall x\ \textit{Mushroom}(x) \wedge \textit{Purple}(x) \supset \textit{Poisonous}(x)$$

- No purple mushroom is poisonous:

$$\forall x\ \textit{Mushroom}(x) \wedge \textit{Purple}(x) \supset \neg\textit{Poisonous}(x)$$

- All mushrooms are either purple or poisonous:

$$\forall x\ \textit{Mushroom}(x) \supset \textit{Purple}(x) \vee \textit{Poisonous}(x)$$

- All mushrooms are either purple or poisonous but not both:

$$\begin{aligned} \forall x \quad & \textit{Mushroom}(x) \supset \\ & (\textit{Purple}(x) \wedge \neg\textit{Poisonous}(x)) \vee \\ & (\neg\textit{Purple}(x) \wedge \textit{Poisonous}(x)) \end{aligned}$$

### Resolution and Refutation

Interpretation : "Truth Assignment"

{ if α is true in I. then I is a model of sentence α.
Σ ⊨ α iff for any I. I is model of Σ ⇒ I is model of α
α is tautology if ∅ ⊨ α.

### Proof By Refutation

to prove KB ⊨ α, we can alternatively prove KB ∪ {¬α} is a Contradictory
⇒ KB ∪ {¬α} ⟶ []

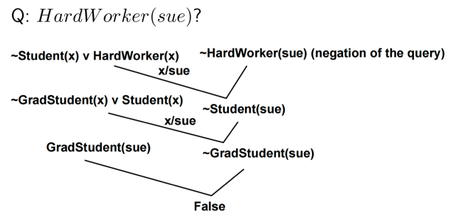
To determine if $KB \models \alpha$:

- Put $KB$ and $\neg\alpha$ into CNF to get $\mathcal{S}$.
- Check if $\mathcal{S} \rightarrow []$:
  1. Check if [] is in $\mathcal{S}$. If yes, then return "unsatisfiable".
  2. Check if there are two clauses $c_1$ and $c_2$ in $\mathcal{S}$ such that they resolve to produce a $c_3$ not already in $\mathcal{S}$. If no such $c_3$ can be generated, then return "satisfiable".
  3. Add $c_3$ to $\mathcal{S}$ and go back to the first step.

$KB$:

$$\forall x\ \textit{GradStudent}(x) \supset \textit{Student}(x),$$
$$\forall x\ \textit{Student}(x) \supset \textit{HardWorker}(x),$$
$$\textit{GradStudent}(sue)$$

Q: $\textit{HardWorker}(sue)$?




## GSCA

**Generic Separate-Conquer Algorithm**: Given an atom $\gamma$ that we want to learn rules about, and a training set $\Sigma$, $GSCA$ works as follows:

Initialize $\Sigma_{cur} = \Sigma$.
Initialize $\pi = \{\}$, an empty set of rules.
**repeat**
  Initialize $\Sigma_{temp} = \Sigma_{cur}$.
  Initialize $\Gamma = true$.
  Initialize $\tau = \Gamma \supset \gamma$.
  **repeat**
    Select an atom $\alpha$ from feature set. This is a nondeterministic step, and a backtracking point, needs a *heuristic* to select $\alpha$.
    Let $\Gamma = \Gamma \wedge \alpha$, $\tau = \Gamma \supset \gamma$, $\Sigma_{temp} = \{$the instances in $\Sigma_{temp}$ with $\alpha = true\}$.
  **until** $\Sigma_{temp}$ covers only positive instances (instances with $\gamma = true$).
  Let $\pi = \pi \cup \tau$ (add the newly learned rule).
  Let $\Sigma_{cur} = \Sigma_{cur} - \Sigma_{temp}$ (exclude positive instances already covered by $\pi$).
**until** $\pi$ covers all (or most) of the positive instances in $\Sigma$.