Propositional Logic

Syntactic Elements
- An atomic sentence consists of a single propositional symbol, representing a proposition that can be true or false.
- A literal is a propositional symbol or its negation.
- Complex sentences are constructed from simpler sentences using logical connectives: ¬ (not), ∧ (and), ∨ (or), → (implies) [I prefer → to ⇒], and ↔ (iff).

Syntax
- A proposition or propositional sentence can be formed as follows:
  - Every propositional symbol is a sentence.
  - If \( A \) is a sentence, then \( \neg A \) is a sentence.
  - If \( A_1 \) and \( A_2 \) are sentences, then so are:
    - \( A_1 \wedge A_2 \) (and, conjunction)
    - \( A_1 \vee A_2 \) (or, disjunction)
    - \( A_1 \rightarrow A_2 \) (implies, implication)
    - \( A_1 \leftrightarrow A_2 \) (iff, biconditional)

Semantics
- A world or domain is the set of facts we want to represent.
- An interpretation maps each propositional symbol to the world.
- A sentence is true if its interpretation in the world is true.
- A knowledge base is a set of sentences.
- A world is a model of a KB if the KB is true for that world.
- For convenience, a model can be thought of as a truth assignment to the symbols.
Deduction

- A sentence $S$ is **satisfiable** within a KB if $S$ is true in some model of the KB, i.e., some truth assignment makes $S$ and the KB true.
- A sentence $S$ is **entailed** by a KB if $S$ is true in all models of the KB (denoted as $KB \models S$), i.e., every truth assignment that makes KB true also makes $S$ true.
- **Logical inference or deduction** is concerned with producing entailed sentences from KBs.

Entailment

**Sentences (KB)** $\xrightarrow{\text{Entails}}$ **Sentence** $\xrightarrow{\text{True in all Possible Truth Assignments}}$ **True in all Possible Truth Assignments**

The Wumpus World Example

![The Wumpus World Example Diagram](image)

Inference Rules

- **Modus Ponens** From $A \rightarrow B$ and $A$ Infer $B$
- **Unit Resolution** From $A \lor B$ and $\neg B$ Infer $A$
- **Resolution** From $A \lor B$ and $\neg B \lor C$ Infer $A \lor C$
Inference Procedures

- An inference procedure uses inference rules to produce proofs. Denoted as $KB \vdash S$.
- An inference procedure is sound if it can only prove entailed sentences, i.e., every sentence it proves is entailed.
- An inference procedure is complete if it can prove any entailed sentence, i.e., every entailed sentence can be proved.

Resolution Inference Procedure

- Resolution applies to conjunctive normal form.
  - A KB is a conjunction of sentences.
  - Each sentence is a disjunction of literals.
- Resolution is refutation complete. If a KB is in CNF, and if the KB is inconsistent, then resolution will infer inconsistent literals.
- To deduce a sentence $S$, first temporarily add $\neg S$ to the KB. Then, if resolution infers inconsistent literals, then $\neg S$ is not satisfiable within the KB, which means that $S$ is entailed.

Useful Equivalences for Converting to CNF

Several logical equivalences are handy for converting sentences to CNF:

- $(p \rightarrow q) \equiv (\neg p \lor q)$
- $((p \land r) \rightarrow (q \lor s)) \equiv (\neg p \lor \neg r \lor q \lor s)$
- $(p \rightarrow (q \land r)) \equiv ((p \rightarrow q) \land (p \rightarrow r))$
- $(p \rightarrow (q \land r)) \equiv ((\neg p \lor q) \land (\neg p \lor r))$
- $((p \lor q) \rightarrow r) \equiv ((p \rightarrow r) \land (q \rightarrow r))$
- $((p \lor q) \rightarrow r) \equiv ((\neg p \lor r) \land (\neg q \lor r))$

First-Order Logic

Syntactic Elements

- A symbol in first-order logic can be a predicate, a function, a constant term, or a variable term.
- First-order logic uses the connectives $\neg$ (not), $\land$ (and), $\lor$ (or), $\rightarrow$ (implies), and $\leftrightarrow$ (iff).
- First-order logic also uses the quantifiers $\forall$ (for all) and $\exists$ (there exists).

Syntax

- A term is a constant or variable, or a function applied to a sequence of terms.
- A ground term is a term with no variables.
- An atomic sentence or atom is a predicate applied to a sequence of terms.
- A ground atom is an atom with no variables.
- Connectives can be used to construct more complex sentences in the usual way.
- If $A$ is a sentence and $x$ is a variable, then:
  - $\forall x \ A$ is a sentence (universal quantifier).
  - $\exists x \ A$ is a sentence (existential quantifier).
- A well-formed formula is a sentence in which all the variables are quantified.

Semantics

- The connectives $\land$, $\lor$, $\neg$, $\rightarrow$, and $\leftrightarrow$ are evaluated in the usual way.
- $\forall x \ A$ is true if every substitution for $x$ makes $A$ true.
- $\exists x \ A$ is true if at least one substitution for $x$ makes $A$ true.
- An interpretation consists of objects in the world and mappings for terms and predicates.
- Each ground term is mapped to an object.
- Each predicate is mapped to a relation (think relational database).
- A ground atom is true if the predicate’s relation holds between the terms’ objects.
Inference Rules

- **Universal From** $\forall x \ A(x)$
  - Elimination Infer $A(t)$ where $t$ is any term
- **Existential From** $\exists x \ A(x)$
  - Elimination Infer $A(c)$ where $c$ is a new constant
- **Resolution From** $\forall x, y \ B(y) \lor A(x)$ (disjunctive) and $\forall x, z \ \neg A(x) \lor C(z)$
  - Infer $\forall y, z \ B(y) \lor C(z)$
- **Resolution From** $\forall x, y \ B(y) \rightarrow A(x)$ (implicative) and $\forall x, z \ A(x) \rightarrow C(z)$
  - Infer $\forall y, z \ B(y) \rightarrow C(z)$

Example: Setup

- **Knowledge Base**
  - $\neg \text{parent}(x,y) \mid \neg \text{ancestor}(y,z) \mid \text{ancestor}(x,z)$
  - $\neg \text{parent}(x,y) \mid \text{ancestor}(x,y)$
  - $\neg \text{mother}(x,y) \mid \text{parent}(x,y)$
  - $\neg \text{father}(x,y) \mid \text{parent}(x,y)$
  - $\text{mother}($Liz,Charley$)$
  - $\text{father}($Charley,Billy$)$

- **To prove** $\text{ancestor}(Liz,Billy)$
  - Refute $\neg \text{ancestor}(Liz,Billy)$

Example, Proof, Part 1

- $\neg \text{parent}(x,y) \mid \neg \text{ancestor}(y,z) \mid \text{ancestor}(x,z)$
  - $\neg \text{ancestor}(Liz,Billy)$
  - $\neg \text{parent}(Liz,y) \mid \neg \text{ancestor}(y,Billy)$
  - $\text{mother}(Liz,y)$
  - $\text{mother}(Liz,y)$
  - $\text{mother}(Liz,Charley)$
  - $\text{father}(Charley,Billy)$
  - $\text{father}(Charley,Billy)$
  - $\text{father}(Charley,Billy)$

Example, Proof, Part 2

- $\neg \text{parent}(x,y) \mid \text{ancestor}(x,y)$
  - $\text{ancestor}(Charley,Billy)$
  - $\text{parent}(Charley,Billy)$
  - $\text{father}(Charley,Billy)$
  - $\text{father}(Charley,Billy)$

Unification

- To use the resolution inference rule, we need to be able to match atoms in sentences. This is called **unification**.
- If successful, unification returns a **substitution**.
- A substitution specifies values for variables that would make the two atoms identical.
Unify-Lists Procedure

function Unify-Lists(A, B, θ)
    if A and B have different predicates/functions
        or have different numbers of arguments
        then return failure
    for i ← 1 to number of arguments do
        termA ← i-th argument of A
        termB ← i-th argument of B
        θ ← Unify-Terms(termA, termB, θ)
        if θ = failure then return failure
    return θ

Unify-Terms Procedure

function Unify-Terms(A, B, θ)
    while A is a variable and θ[A] exists
        do A ← θ[A]
    while B is a variable and θ[B] exists
        do B ← θ[B]
    if A = B then do nothing
    else if A is a variable then θ[A] ← B
    else if B is a variable then θ[B] ← A
    else if A or B is a constant then θ ← failure
    else θ ← Unify-Lists(A, B, θ)
    return θ

Resolve Procedure Preliminaries

- The Resolve procedure assumes that sentences A and B are disjunctions represented as sets of literals. A literal is an atom or its negation.
- Resolve assumes that sentences A and B have no variables with the same names.
- Resolve returns all the sentences it infers.

Resolve Procedure

function Resolve(A, B)
    for each litA in A do
        for each litB in B do
            if one of litA and litB is negated, not both then
                θ ← Unify-Lists(litA, litB, empty substitution)
            if θ ≠ failure and has no recursive subs.
                then
                    A′ ← apply substitution θ to A
                    litA′ ← apply substitution θ to litA
                    B′ ← apply substitution θ to B
                    litB′ ← apply substitution θ to litB
                    C ← (A′ − {litA′}) ∪ (B′ − {litB′})
                    add C to formulas inferred
    return formulas inferred