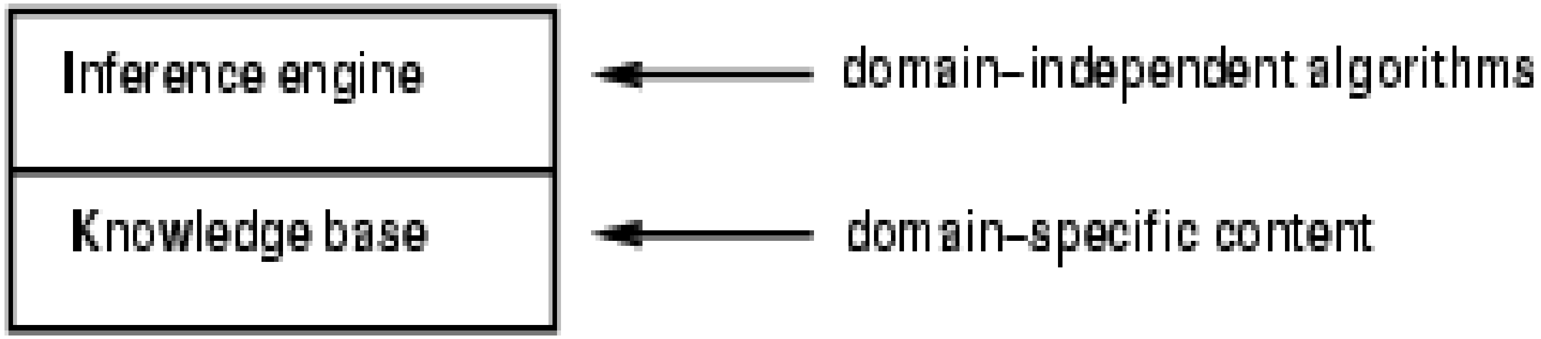
CHAPTER FOUR

KNOWLEDGE & REASONING

Knowledge Base Agent

- Knowledge base agent is an agent that perform action using the knowledge it has and reason about their action using its inference procedure.



- Knowledge base is a set of representation of facts and their relationships called **rules** about the world.
- The central component of a knowledge-based agent is its **knowledge base**, or KB.
- A knowledge base is a set of **sentences**

Cont…

- There must mechanisms to derive new sentences from old ones.
- This process is known as **inferencing or reasoning**.
- Inference must obey the primary requirement that the new sentences should follow logically from the previous ones.
- *Logic* is the primary vehicle for representing and reasoning about knowledge

➢ **Declarative approach to building an agent (or other system):**
  - ➢ Tell it what it needs to know (Knowledge base)
  - ➢ Ask what it knows

➢ Answers should follow from the KB

Knowledge-based agent program

- It takes a percept as input, returns an action and maintains a kb (i.e., initial background knowledge)
- Each time the agent program is called, it does three things:
  - It TELLs the kb what it perceives
  - It ASKs the kb what action it should perform
  - The agent program TELLs the kb which action was chosen, and the agent executes the action

Cont…

- A generic knowledge-based agent
- **function** KB-AGENT(*percept*) **returns** an *action*

```
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
```

Knowledge Bases Agent

- The agent must be able to:
  - Represent states of the world, actions, etc.
  - Incorporate new percepts (facts and rules)
  - Deduce hidden properties of the world
  - Deduce appropriate actions
  - Update internal representations of the world

Knowledge Bases Agent

- Representation of state of the world using KB
  - **Example of KB written in that represent a state of a world in which only Azieb is living (PROLOG representation)**
  - **Moreover, the required information are gender and age given name**
  - FACTS
    1. female(azieb).
    2. age(azieb, 18).

Example

Agent should incorporate new percept

For example,

Azieb get maried with Melaku, they have two kids selam and solomon

This can be reflected by incorporating the new percepts

- FACTS
  1. female(azieb).
  2. Age(azieb, 18).
  3. Married(azieb, melaku).
  4. male(melaku).
  5. female(selam).
  6. male(solomon).
  7. parent(melaku,selam).
  8. parent(azieb,selam).
  9. parent(melaku,solomon).
  10. parent(azieb,solomon).
  11. age(melaku,26).
  12. age(selam, 5).
  13. age(solomon, 1).

- RULE
  1. married(X,Y):-married(Y,X).
  2. father(X,Y):-male(X),parent(X,Y).
  3. mother(X,Y):-female(X),parent(X,Y).
  4. wife(X,Y):-parent(X,Z),parent(Y,Z).
  5. brother(X,Y):-male(X), parent(Z,X),parent(Z,Y).
  6. sister(X,Y):-female(X), parent(Z,X),parent(Z,Y).

Example

- The agent should deduce hidden portion of the world like Azieb and Selam are beautiful.
  - If one asks questions like
    - ?brother(aster,belay).
    - Married(melaku, azieb).
  - The system must respond correctly
- Deduce appropriate action to query.
  - If one asks questions like
    - who is the sister of whom,
    - who is the father of belay,
    - who is the mother of Selam, etc
  - The system must respond correctly
- Agent should update internal representation of the world.
  - For example, if died(melaku) is given we should modify any fact that tells us about live history about melaku.

Knowledge representation

- Knowledge representation refers to the technique how to express the available facts and rules inside a computer so that agent will use it to perform well.
- Knowledge representation consists of:
  - **Syntax (grammar)**: possible physical configuration that constitute a sentence (fact or rule) inside the agent architecture.
    - For example one possible syntax rule may be every sentence must end with full stop.
  - **Semantics (concept)**: determine the facts in the world to which the sentence refers
    - Without semantics a sentence is just a sequence of characters or binary sequences
    - Semantic defines the meaning of the sentence

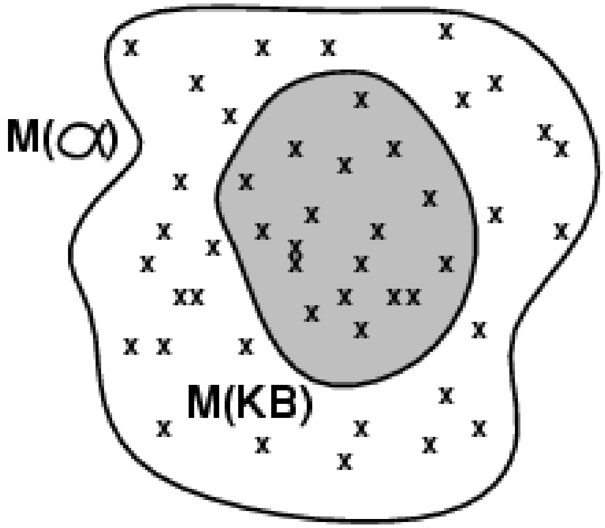
Knowledge representation

- E.g., In the language of arithmetic
  - $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence
  - $x+2 \geq y$ is true iff the number $x+2$ is not less than the number y
  - $x+2 \geq y$ is true in a world where $x = 7, y = 1$
  - $x+2 \geq y$ is false in a world where $x = 0, y = 6$

- Given clear definition of semantics and syntax of a language, we call that language **logical language**.
- Knowledge representation is used to represent part of the world (facts and their association) into ideal computer system
- KB for agent program can be represented using programming language designed for this purpose like LISP and PROLOG.

Model

- Any world in which a sentence is true under a particular interpretation is called model of that sentence under that interpretation
- We say *m* is a model of a sentence α if α is true in *m*
- *M(α)* is the set of all models of α
- Then KB $\models$ α iff *M(KB)* $\subseteq$ *M*(α)
  - E.g. *KB* = Giants won and Reds won α = Giants won



Logic

- **Logic** is the study of the principles of reasoning and arguments towards the truth of a given conclusion given premises.

- **Logic** in AI is the key idea for KB design, KB representation and inferencing (reasoning)

- **Logic** is formal languages use for representing information so that conclusions can be drawn

- Logic is the systematic study of the general conditions of valid inferences

Types of Logic

- In mathematics there are different kinds of logics.
- Some of these according to order of their generality are
  - Propositional logic
  - First order logic
  - Second order logic and more
- First order logic can be used to design, represent or infer for any environment in the real world.

Propositional logic (PL)

- Proposition is statement which is either true or false but not both at any time.
- A statement is a sentence which is either true or false.
- PL uses declarative sentences only
- PL doesn’t involve quantifiers.
- Not all sentences are statement (interrogatives, imperatives and exclamatory)
- Proposition can be conditional or unconditional
- **Examples**
  - Socrates is mortal (unconditional)
  - If the winter is severe, students will not succeed. (conditional)
  - All are the same iff their color is black (conditional)
- In Propositional logic, symbols represent the whole Proposition.
- **Examples**:
  - M = Socrates is mortal
  - W = winter is sever
  - S = students will not succeed

Con’t…

- There are two types of Propositions:

i. **Atomic Propositions**

   - simple propositions.
   - It consists of a single proposition symbol.
   - These are the sentences which must be either true or false.

   Example:

   a. 2+2 is 4, it is an atomic proposition as it is a **true** fact.

   b. "The Sun is cold" is also a proposition as it is a **false** fact.

ii. **Compound propositions**

   - constructed by combining simpler or atomic propositions, using ***parenthesis*** and ***logical connectives***.

- **Example:** "It is raining today, and street is wet."

Propositional logic (PL)

- Proposition symbols can be combined using **Boolean connectives** to generate new Proposition with complex meaning
- Symbols involved in PL:
  - Logical constants (TRUE and FALSE)
  - Proposition symbols (also called atomic symbols) like M, W, S
  - Logical connectives

$\neg$	(negation),
$\wedge$	(conjunction),
$\vee$	(disjunction),
$\Leftrightarrow$	(bi-implication or equivalence),
$\Rightarrow$	(implication) and parenthesis

Propositional logic (PL)

- Rules
  - Logical constants and propositional symbols are sentence by them selves
  - Wrapping parenthesis around a sentence yield a sentence like (P V Q)
  - Literal are atomic symbols or negated atomic symbols
  - Complex sentence can be formed by combining simpler sentences with logical connectors

PL connector priority

- Priority of logical connectives from **highest** to **lowest**
  - Parenthesis
  - Negation
  - Conjunction
  - Disjunction
  - Implication
  - Bi-implication

Types of Sentence

- Given a sentence α, this sentence according to the world considered can be
  - Valid (tautology)
  - Invalid (contradiction)
  - Satisfiable (neither valid nor invalid)

Validity (tautology)

- A sentence is valid iff it is true under any interpretations in all possible world
  - Example: x>4 or x<=4;
  - Water boils at 100 degree centigrade
  - Human has two legs (may not be valid)
  - Books have page number (may not be valid)

Satisfiability

- A sentence is **satisfiable** iff there is some interpretation in some world for which it is true.
- Every valid sentence is satisfiable
  - Example: x+2 = 20
  - Every students of AI are in their class
- A sentence which is not satisfiable is unsatisfiable (contradiction or invalid).

Entailment

- Entailment means that one thing follows from another:
- It can be represented by $\models$ symbol (double turnstyle)
- KB $\models \alpha$ shows α can be entailed from KB
- Knowledge base, *KB* entails sentence α if and only if α is true in all worlds where *KB* is true
  - E.g., the KB containing “the Giants won” and “the Reds won” entails “Either the Giants won or the Reds won”
  - E.g., $x+y = 4$ entails $4 = x+y$
  - E.g., $x+y = 4$ can’t entails $x= 2$ and $y = 2$
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

Inference Procedure

- An inference procedure is a procedure used as reasoning engine.
- It can do:
  1. Given KB, generate new sentence α that can be entailed by KB and we call the inference procedure entail α
  2. Given KB and α, it will prove whether α is entailed by KB or not
- ***KB* $\vdash_i$ α** means sentence α can be derived from *KB* by procedure i (|- is called turnstyle or single turnstyle)
- **Inference Procedure property**
- **Soundness**: inference procedure *i* is said to be sound: if whenever *KB* $\vdash_i$ α, it is also true that *KB* $\models$ α
- A proof system is sound if everything that is provable is in fact true (if *KB* $\vdash_i$ α then *KB* $\models$ α)
- **Completeness**: inference procedure *i* is said to be complete if whenever *KB* $\models$ α, it is also true that *KB* $\vdash_i$ α
- A proof system is complete if everything that is true has a proof (if *KB* $\models$ α then *KB* $\vdash_i$ α)

Inference Procedure property

- The record of operation of a sound inference procedure is called a **proof**
- Soundness of an inference can be established through truth table.
- For example the inference procedure that entails P from a KB

P	**Q**	**¬P**	**P^Q**	**P ∨ Q**	*P→Q*	**P⇔Q**
False	False	True	False	False	True	True
False	True	True	False	True	False	False
True	False	False	False	True	True	False
true	true	false	true	true	true	true

Complexity of Propositional inference

- The worst case is exponential (2^n).
- In this case we can prove using truth table
- Using inference rule we may minimize the time complexity.
- The use of inference rules depends on general principles of logic called mono-tonicity
- Monotonicity says that the set of entailed sentences can only increase as information is added to the knowledgebase
- Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base-the conclusion of the rule must follow ***regardless*** of ***what else is in the knowledge base***
- For any sentences α and β , if $KB_1 \models \alpha$ then $KB \wedge \beta \models \alpha$

Normal Forms of Logical expression

- There are different standard forms of expressing PL statement.
- Some inference procedure require their knowledge base to be in one of the normal forms to infer from it.
- Hence, we need to convert the logical expression which are the source of knowledge into the appropriate forms
- Some of the normal forms are stated bellow

Normal Forms of Logical expression

- Clausal normal form (disjunction normal form):
  - it is a set of one or more literals connected with the disjunction operator (disjunction of literals).
  - It doesn’t allow negation marker except as a prefix to a symbol
    Example $\sim P \vee Q \vee \sim R$ is a clausal form
  - According to the definition a single symbol or negation of a symbol is in clausal form

- Disjunctive normal form (DNF):
  - disjunction of conjunction of literals.
    Example $(A \wedge B) \vee (C \wedge D)$

Normal Forms of Logical expression

- Conjunctive normal forms (CNF):
  - Simply defined as a sequences of one or more clauses connected by the conjunction operator
  - Or it is conjunction of clauses
  - Or conjunction of disjunction of literals
    Example $(A \vee B) \wedge (C \vee D)$
  - Note: connector is needed if we have two or more elementary unit to connect to each other to form complex expression
  - This is the basis for the generalized resolution inference procedure

- Horn form: conjunction of literals implies a literal.
  Example $(A \wedge B \wedge C \wedge D) => E$

Inference procedure and normal forms

- The inference procedure that we have seen before are all sound
- If KB is represented in CNF, the generalized resolution inference procedure is complete
- If KB is represented in Horn form, the generalized modes ponens algorithm is complete
- It can be proved that every sentence of human language can be represented using logic as CNF. However, it is not possible in Horn form.
- Therefore, CNF is a more powerful representation technique for knowledge
- But, Horn form representation of knowledge is easily understandable and convenient. It also require polynomial time inference procedure

Logical equivalence (1)

- Two sentences are logically equivalent iff they have the same truth value in all possible world
- Equivalently $\alpha \equiv$ ß iff $\alpha \models \beta$ and $\beta \models \alpha$

$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

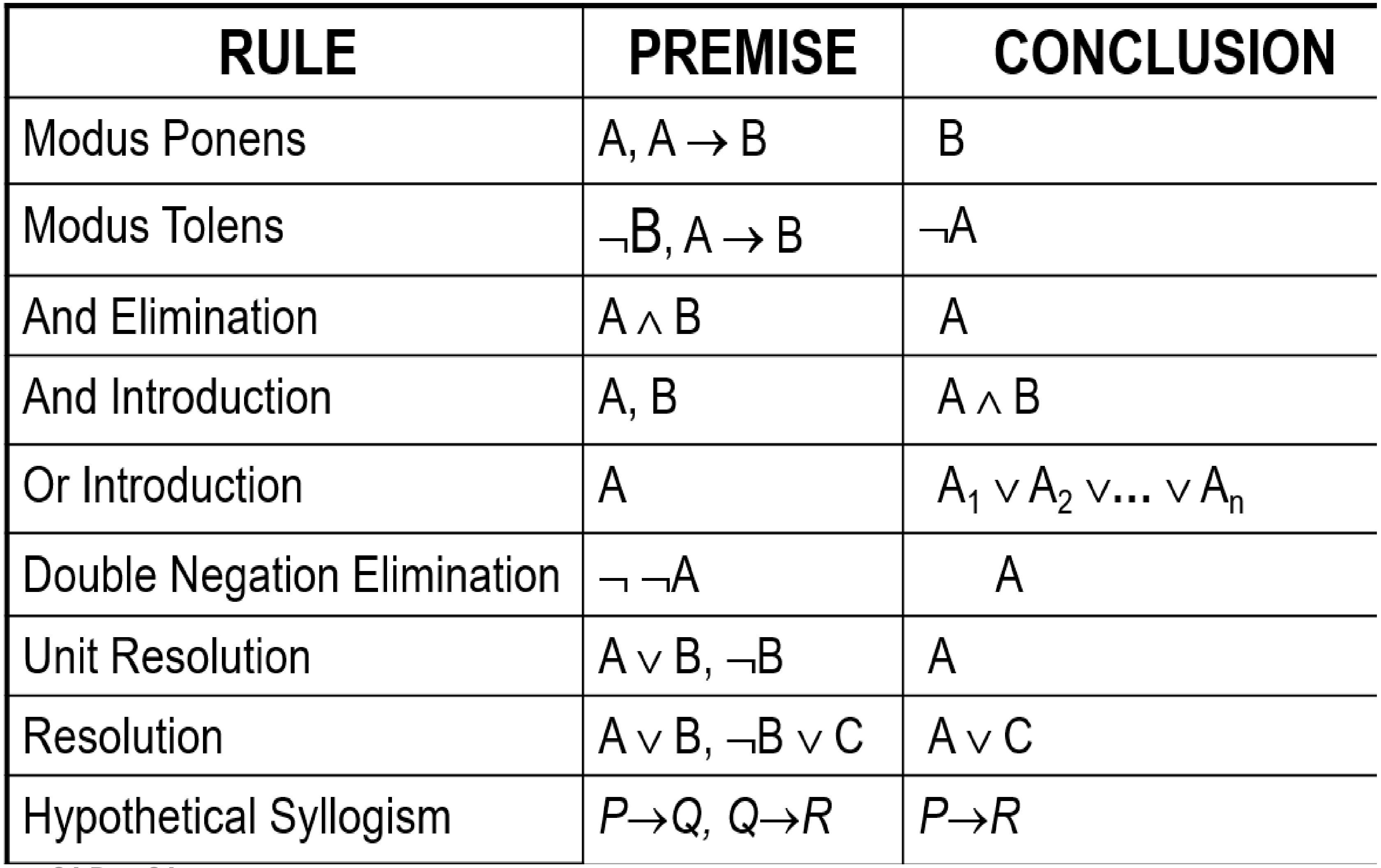
Logical equivalence (2)

Domination laws:	$p \vee \mathbf{T} \equiv \mathbf{T}$, $p \wedge \mathbf{F} \equiv \mathbf{F}$
Identity laws:	$p \wedge \mathbf{T} \equiv p$, $p \vee \mathbf{F} \equiv p$
Idempotent laws:	$p \wedge p \equiv p$, $p \vee p \equiv p$
Double negation law:	$\neg(\neg p) \equiv p$
Negation laws:	$p \vee \neg p \equiv \mathbf{T}$, $p \wedge \neg p \equiv \mathbf{F}$
Absorption laws:	$p \vee (p \wedge q) \equiv p$, $p \wedge (p \vee q) \equiv p$

- Prove the following using logical equivalence
  1. Prove that $(P \vee H) \wedge \neg H) \Rightarrow P$ is valid
  2. Prove that $\neg p \leftrightarrow q \Leftrightarrow p \leftrightarrow \neg q$ is valid
  3. Prove that $(p \wedge q) \rightarrow p$ is a tautology
  4. Prove that $(p \wedge \neg q) \vee q \Leftrightarrow p \vee q$ is valid

Rules of inference for propositional logic

- **Rules of Inferences**



RULE	PREMISE	CONCLUSION
Modus Ponens	$A, A \rightarrow B$	B
Modus Tolens	$\neg B, A \rightarrow B$	$\neg A$
And Elimination	$A \wedge B$	A
And Introduction	A, B	$A \wedge B$
Or Introduction	A	$A_1 \vee A_2 \vee \ldots \vee A_n$
Double Negation Elimination	$\neg \neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$A \vee B, \neg B \vee C$	$A \vee C$
Hypothetical Syllogism	$P \rightarrow Q,\ Q \rightarrow R$	$P \rightarrow R$

Generalized Resolution for PL

- Given any two clauses A and B, if there are any literal P_1 in A which has a complementary literal P_2 in B, delete P_1 and P_2 from A and B and construct a disjunction of the remaining clauses.
- The clause constructed is called the resolvent of A and B.
  - For example, consider the following clauses

A:	$P \vee Q \vee R$
B:	$\sim P \vee Q \vee M$
C:	$\sim Q \vee S$

- From clause A and B, if we remove P and ~P, it resolves into clause like

  D : $Q \vee R \vee Q \vee M \equiv$ **$Q \vee R \vee M$** .
- If Q of clause D and ~Q of clause C resolved, we get clause

  E: **$R \vee M \vee S$**

Generalized Resolution for PL

- Resolution can also be used to prove a sentence whether it is valid or not.
- That can be done using direct method or indirect method (refutation)
- For example Given a KB consists of the sentences

  A: $P \vee Q \vee R$

  B: $\sim P \vee R$

  C: $\sim Q$
- We need to prove R
- **Method One (Direct)**
- **Entail D:** $Q \vee R$ from A and B
- **Entail E:** R from D and C

Note: in order to apply resolution for proving a theory, make sure first all the knowledge is in its clausal form

Predicate (First-Order) Logic

- We used propositional logic as our representation language because it is one of the simplest languages that demonstrates all the important points
- Unfortunately propositional logic has a very limited ontology, making only the commitment that the world consists of facts
- **First-order logic**: makes a stronger set of ontological commitments

Cont…

- First-order logic **(FOL)** models the world in terms of:
  - **Objects,** which are things with individual identities
  - **Properties** of objects that distinguish them from other objects
  - **Relations** that hold among sets of objects
  - **Functions,** which are a subset of relations where there is only one "value" for any given "input"

Cont…

- Examples:
  - Objects: Students, lectures, companies, cars ...
  - Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: father-of, best-friend, second-half, one-more-than ...

Cont…

- Although **FOL** commits to the existence of objects and relations,
  - It does not make an ontological commitment to such things as:
    - Categories,
    - Time,
    - Events, which also seem to show up in most facts about the world

SYNTAX AND SEMANTICS of FOL

- In propositional logic every expression is a sentence, which represents a fact
- **FOL** has sentences, but it also has terms, which represent objects
- Constant symbols, variables, and function symbols are used to build terms, and quantifiers and predicate symbols are used to build sentences
- **Constant symbols**, which represent individuals in the world

  e.g. Mary, 3, Green, A
- **Predicate symbols:** which map individuals to truth values

  e.g. Round, Brother, greater, green

Cont…

- **Function symbols:** which map individuals to individuals
  - ➢ any given object is related to exactly one other object by the relation

  e.g. Cosine, Tangent, father-of ….
- FOL Provides:
  - ➢ **Variable symbols**

    e.g. x, y
  - ➢ **Connectives**
    - ➢Same as in PL: not (¬), and (^), or (V), implies (→), if and only if (bi-conditional ↔)
  - ➢ **Quantifiers**
    - ❖Universal ∀**x** or **(Ax)**
    - ❖Existential ∃**x** or **(Ex)**

Terms, Atomic, and Complex Sentences

- A **term** is a logical expression that refers to an object
- Terms denote a real-world individual is a constant symbol, a variable symbol, or an n-place function of n terms
  - ➢ x and f(x_1, ..., x_n) are terms, where each x_i is a term
  - ➢ A term with no variables is a **ground term,** or constants
- For example,
- **John** (a constant referring to a specific person)
- **5** (a constant referring to the number 5)
- **f(John)** (if f is a function, and John is a constant, then f(John) is a ground term because it applies the function to a constant)

Cont…

- **Atomic sentences**
- **Atomic sentences** are the most basic sentences of **first-order logic.**
- These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.

e.g. Brother(Richard, John)

- has value true or false
- We can represent atomic sentences as:
  - **Predicate (term1, term2, , term n).**
- **Example: Abdi and Ali are brothers: => Brothers(Abdi, Ali)**
- **Tom is a cat: => cat (Tom).**
  - Atomic sentences can have arguments that are complex terms:

e.g. Married(FatherOf (Richard),MotherOf (John))

Cont…

- **Complex sentences**
  - A **complex sentence** is formed from atomic sentences connected by the logical connectives:
    - $\neg P$, $P \vee Q$, $P \wedge Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ where P and Q are sentences
- A **quantified sentence** adds quantifiers $\forall$ and $\exists$

Quantifiers

- **FOL** contains two standard quantifiers, called universal and existential
  - **Universal quantification (∀)**
    - (∀x)P(x) means that P holds for **all** values of x in the domain associated with that variable
    - E.g., (∀x) dolphin(x) → mammal(x)
  - **Existential quantification (∃)**
    - (∃ x)P(x) means that P holds for **some** value of x in the domain associated with that variable
    - E.g., (∃ x) mammal(x) ∧ lays-eggs(x)
    - Permits one to make a statement about some object without naming it

Cont…

- Universal quantifiers are often used with "implies" to form "rules":

  $(\forall x)$ student(x) $\rightarrow$ smart(x) means "All students are smart"

- Universal quantification is *rarely* used to make blanket statements about every individual in the world:

  $(\forall x)$student(x)$\wedge$smart(x) means "Everyone in the world is a student and is smart"

Cont…

- Existential quantifiers are usually used with "and" to specify a list of properties about an individual:

  $(\exists x)$ student(x) $\wedge$ smart(x) means "There is a student who is smart"

- A common mistake is to represent this FOL sentence with English:

  $(\exists x)$ student(x) $\rightarrow$ smart(x)

  - ➢ But what happens when there is a person who is *not* a student?

Cont…

- **Nested quantifiers:**
- We will often want to express more complex sentences using multiple quantifiers
- For example, "For all x and all y, if x is the parent of y then y is the child of x" becomes

  $\forall x,y \quad Parent(x,y) \Rightarrow Child(y,x)$
- $\forall x,y$ is equivalent to $\forall x \, \forall y$.
- Similarly, the fact that a person's brother has that person as a sibling is expressed by:

  $\forall x,y \; Brother(x,y) \Rightarrow Sibling(y,x)$

Cont…

- **Quantifier Scope:**
  - Switching the order of universal quantifiers *does not* change the meaning:
    - $(\forall x)(\forall y)P(x, y) \leftrightarrow (\forall y)(\forall x) P(x, y)$
  - Similarly, you can switch the order of existential quantifiers:
    - $(\exists x)(\exists y)P(x,y) \leftrightarrow (\exists y)(\exists x) P(x,y)$
  - Switching the order of universal and existential *does* change meaning:
    - Everyone likes someone: $(\forall x)(\exists y)$ likes(x,y)
    - Someone liked by everyone: $(\exists y)(\forall x)$ likes(x,y)

Connections between ∀ and ∃

- The two quantifiers are actually intimately connected with each other, through negation.
  - when one says that everyone dislikes parsnips,
  - one is also saying that there does not exist someone who likes them; and vice versa:
  - We can relate sentences involving ∀ and ∃ using **De Morgan's** laws:

$$
\begin{array}{llll}
\forall x\ \neg P & \equiv \neg\exists x\ P & \qquad \neg P \land \neg Q & \equiv \neg(P \lor Q) \\
\neg\forall x\ P & \equiv \exists x\ \neg P & \qquad \neg(P \land Q) & \equiv \neg P \lor \neg Q \\
\forall x\ P & \equiv \neg\exists x\ \neg P & \qquad P \land Q & \equiv \neg(\neg P \lor \neg Q) \\
\exists x\ P & \equiv \neg\forall x\ \neg P & \qquad P \lor Q & \equiv \neg(\neg P \land \neg Q)
\end{array}
$$

Cont…

- **Equality**
  - We can use the **equality symbol** to make statements to the effect that two terms refer to the same object
  - For example:

    Father(John) = Henry

  - says that the object referred by Father(John) and the object referred by Henry are the same.

Higher-order logic

- **Higher-order logic** allows us to quantify over relations and functions as well as over objects
- For example, in higher-order logic, we can say that two objects are equal if and only if all properties applied to them are equivalent:

$$\forall x, y\ (x = y)\ \&\ (\forall p\ p(x) \Leftrightarrow p(y))$$

- Or we could say that two functions are equal if and only if they have the same value for all arguments:

$$\forall f,g\ (f = g) \Leftrightarrow (\forall\ x\ f(x) = g(x))$$

- Higher-order logics have strictly more expressive power than first-order logic
- But **undecidable**: there isn't an effective algorithm to decide whether all sentences are valid

What is Higher-Order Logic (HOL)

- Higher-Order Logic (HOL) extends **first-order logic (FOL)** by allowing **quantification over predicates** and **functions**, not just individuals or objects.
- **First-Order Logic (FOL):**
  In FOL, the variables represent **individuals** (objects) in the domain.
- For example:
  - **∀x likes(x,y)** means "every individual xxx likes some individual y."
- **Higher-Order Logic (HOL):**
  In HOL, you can quantify not only over **individuals**, but also over **predicates** and **functions**. This adds another layer of abstraction.
  - Example: **∀P P(John)**, where P is a predicate, and we are quantifying over the predicate itself.

Key Differences B/n FOL and HOL:

Feature	**First-Order Logic (FOL)**	**Higher-Order Logic (HOL)**
Quantification	Quantifies over **individuals** (objects).	Quantifies over **predicates**, **functions**, and **individuals**.
Expressiveness	Limited to properties and relations of individuals.	Can express more complex relationships, such as quantifying over functions and sets.
Examples	∀x likes(x,y)	∀f likes(f,x), where f is a function)
Syntax	Simple, no quantification over functions/predicates.	Complex, involving quantification over functions/predicates.
Decidability	First-order logic is **decidable**.	Higher-order logic is **undecidable** in general.

Examples of Higher-Order Logic:

- **Quantifying over predicates:**
  - **∀P P(x)** : This means "for every predicate P, P holds for x."
  - Here, PPP is a **predicate variable**, which can stand for any predicate (such as "is a student", "is a parent", etc.).
- **Quantifying over functions:**
  - **∀f (f(x)=x+1))**: This means "for every function f, f(x) equals x+1."
  - In this case, f is a function variable that represents any function that takes x as an input.

Example with Functions:

- Let’s say we have the function **f** that maps people to their ages.
- **f:Person→N**, where f(x) gives the age of person x.
- A higher-order logic statement might look like this:
  ∀f ∀x (f(x)>0) This means **"for every function f and every person x, the age of x (given by f(x) is greater than zero."**

Use Cases of Higher-Order Logic:

- **Mathematics:** Higher-order logic is commonly used in **set theory**, **theories of functions**, and **mathematical logic**, where we need to quantify over sets, functions, and predicates.
- **Computer Science:** It is used in **functional programming languages** and for formal verification of software, where functions are first-class citizens and can be quantified over.
- **Artificial Intelligence:** Higher-order logic can be used for representing more complex knowledge, especially when dealing with **relations**, **predicates**, and **complex functions** in knowledge representation and reasoning.

Challenges with Higher-Order Logic:

- **Undecidability:** Higher-order logic is **undecidable** in general, meaning that there is no algorithm that can determine whether a given statement in HOL is true or false.
- **Complexity:** The syntax and semantics of HOL are more complex than FOL, making it harder to use and understand.

Expressing uniqueness

- Sometimes we want to say that there is a single, unique object that satisfies a certain condition
- "There exists a unique x such that king(x) is true"
  - $\exists x\ king(x) \land \forall y\ (king(y) \rightarrow x=y)$
  - $\exists x\ king(x) \land \neg\exists y\ (king(y) \land x \neq y)$
  - $\exists!\ x\ king(x)$

Cont…

- "Every country has exactly one ruler"
  - ➢ $\forall c\ country(c) \rightarrow \exists!\ r\ ruler(c,r)$
- Iota operator: "ı x P(x)" means "the unique x such that p(x) is true"
  - ➢ "The unique ruler of Freedonia is dead"
  - ➢ dead(ı x ruler(freedonia,x))

The End of chapter four

Converting to CNF

- Converting the following sentence to CNF:

  $a \land \sim b \leftrightarrow c \equiv (a \land \sim b) \leftrightarrow c$