CS-602 - Design of Problem Solvers

Chapter 1 - Introduction to Logic Problem Solvers

Dr. Mahdi Khemakhem

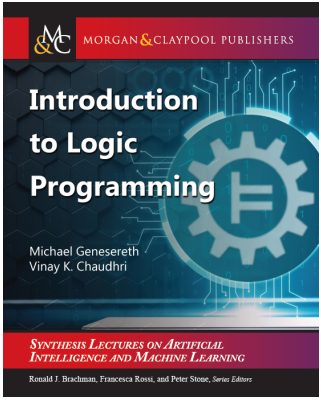
Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY - 2025/2026

Outline

Materials

- **Textbook:** "Introduction to logic programming", 2022, by *Genesereth, Michael, and Vinay K. Chaudhri*
- $\Rightarrow$ **Read Chapter 1 for this chapter's material**



Outline

What is Mathematical Logic in AI?

Core Concept

Mathematical Logic provides the *foundational framework* for enabling AI systems to perform *rigorous reasoning* and make *intelligent decisions* based on *formal representations* of knowledge.

Why is Mathematical Logic Essential for AI Problem Solvers?

- Enables **precise representation** of knowledge and complex relationships
- Supports **systematic inference** from known facts to derive new conclusions
- Provides **formal mechanisms** for automated reasoning and decision-making
- Facilitates **verification and validation** of AI system behavior

Key Applications of Mathematical Logic in AI

Mathematical Logic Powers the Following AI Capabilities:

1. **Knowledge Representation:**
   - Modeling entities, properties, and their relationships
   - Creating structured, machine-readable knowledge bases
2. **Deductive Reasoning & Inference:**
   - Drawing logical conclusions from available data
   - Making predictions based on established rules
3. **Constraint Satisfaction:**
   - Solving scheduling, planning, and resource allocation problems
   - Finding solutions that satisfy multiple constraints simultaneously
4. **Automated Theorem Proving:**
   - Automating complex mathematical and logical proofs
   - Verifying software correctness and system properties

Outline

What is Logical Reasoning?

Definition

Logical Reasoning is a *systematic process* of deriving valid conclusions from given premises using *formal rules of inference*.

Core Components of Logical Reasoning:

- **Premises:** Initial facts or statements assumed to be true
- **Inference Rules:** Logical operations that transform premises into conclusions
- **Conclusion:** New knowledge derived through rigorous application of inference rules

Key Property

If the premises are true and the reasoning is valid, the conclusion must be true.

Logical Reasoning Constructs

Problem Solvers Use Three Main Logical Constructs:

1. **Conditional Reasoning (If/Then):**
   - Format: *"If P, then Q"*
   - Example: *"If the temperature exceeds 30°C, then activate cooling system"*
2. **Logical Connectives (And/Or):**
   - *AND:* Both conditions must be true
   - *OR:* At least one condition must be true
   - Example: *"Grant access if user has valid password AND security token"*
3. **Transitive Reasoning:**
   - If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
   - Example: *"If Riyadh is in Saudi Arabia, and Saudi Arabia is in Asia, then Riyadh is in Asia"*

Outline

Definition and Rules

Boolean Algebra

Boolean Algebra is used to analyze and simplify the digital (logic) circuits. **It uses only the binary numbers i.e. 0 and 1**. It is also called as Binary Algebra or Logical Algebra. Boolean Algebra was invented by George Boole in 1854.

Following are the important rules used in Boolean algebra:

- *Variable* used can have only two values. **Binary 1 for HIGH and Binary 0 for LOW**.
- *Complement of a variable* is represented by an overbar(-). Thus, complement of variable B is represented as $\overline{B}$. Thus if $B = 0$ then $\overline{B} = 1$ and $B = 1$ then $\overline{B} = 0$.
- *ORing of the variables* is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as $A + B + C$.
- *ANDing of the variables* is represented by writing a dot (.) between them such as $A.B.C$. Sometime the dot may be omitted like ABC.

Laws

There are six types of Boolean Laws:

- **Commutative laws:** $A.B = B.A$, $A + B = B + A$.
- **Associative laws:** $(A.B).C = A.(B.C)$, $(A + B) + C = A + (B + C)$.
- **Distributive law:** $A.(B + C) = A.B + A.C$.
- **AND laws:** $A.0 = 0$, $A.1 = A$, $A.A = A$, $A.\overline{A} = 0$.
- **OR laws:** $A + 0 = A$, $A + 1 = 1$, $A + A = A$, $A + \overline{A} = 1$.
- **Inversion law:** $\overline{\overline{A}} = A$.

Expression/Function

Boolean algebra deals with binary variables and logic operation.

Boolean Function and Expression

A **Boolean Function** is described by an algebraic expression called **Boolean Expression** which consists of binary variables, the constants 0 and 1, and the logic operation symbols.

→Consider the following example where the left hand side (LHS) represents a **Boolean Function** and the right hand side (RHS) represents **Boolean expression**:

$$F(A, B, C, D) = A + A.\overline{C} + A.D.C$$

Here the left hand side (LHS) of the equation represents the output Y. So the equation can be written as follows:

$$Y = A + A.\overline{C} + A.D.C$$

Truth Table

Definition

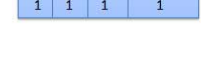
A *Truth Table* represents a table having **all combinations of inputs and their corresponding result**. It is used to analyze the **output of a logic circuit** for all possible combinations of inputs.

For example, consider the following switching equation.

$$F(A, B, C) = A + BC$$

The output will be high (1) if $A = 1$ or $BC = 1$ or both are 1. The truth table for this equation is shown by **Truth Table** at the right.
$\rightarrow$The number of rows in the truth table is 2^n where n is the number of input variables **($n = 3$ for the given equation)**. Hence there are $2^3 = 8$ possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



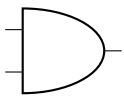
Logic Gates (1/2)

Definition

Logic Gates are the basic building blocks of any digital system. It is an electronic circuit having **one or more than one input** and **only one output**. The relationship between the input and the output is based on a **certain logic**.

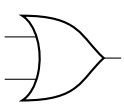
Logic gates are named as **AND gate**, **OR gate**, **NOT gate**, etc.

- **AND Gate:** $Y = A.B$



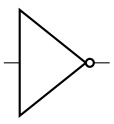
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

- **OR Gate:** $Y = A + B$



Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

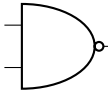
- **NOT Gate:** $Y = \overline{A}$



Inputs	Output
A	B
0	1
1	0

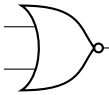
Logic Gates (2/2)

- **NAND Gate:** $Y = \overline{A.B}$



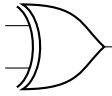
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

- **NOR Gate:** $Y = \overline{A + B}$



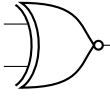
Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

- **XOR Gate:** $Y = A \oplus B = \overline{A}.B + A.\overline{B}$



Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- **XNOR Gate:** $Y = A \ominus B = \overline{A \oplus B} = A.B + \overline{A}.\overline{B}$



Inputs		Output
A	B	$A \ominus B$
0	0	1
0	1	0
1	0	0
1	1	1

De Morgan's Theorems

De Morgan has suggested two theorems which are extremely useful in **Boolean Algebra**. The two theorems are discussed below.

- **Theorem 1:** $\overline{A.B} = \overline{A} + \overline{B}$
  The left hand side (LHS) of this theorem represents a **NAND gate** with inputs A and B, whereas the right hand side (RHS) of the theorem represents an **OR gate** with inverted inputs. This **OR gate** is called as **Bubbled OR**.

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

- **Theorem 2:** $\overline{A + B} = \overline{A}.\overline{B}$
  The left hand side (LHS) of this theorem represents a **NOR gate** with inputs A and B, whereas the right hand side (RHS) of the theorem represents an **AND gate** with inverted inputs. This **AND gate** is called as **Bubbled AND**.

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Outline

Logic Programming

Definition

Logic Programming is a style of programming in which programs take the form of **sets of sentences** in the language of **Symbolic Logic**.

- Programs written in this style are called **Logic Programs**.
- The language in which these programs are written is called **Logic Programming Language**.
- A computer system that manages the creation and execution of Logic Programs is called a **Logic Programming System**.

Logic Programming is often said to be **declarative**[1] or **descriptive** and contrasts with the **imperative**[2] or **prescriptive** approach to programming associated with **traditional programming languages**.

[1] Declarative code focuses on specifying the result of what you want.

[2] Imperative code focuses on writing an explicit sequence of commands to describe how you want the computer to do things

Types of Logic Programming

Logic Programming Landscape:

Main Paradigms:

- *Basic Logic Programming*
- Classic Logic Programming
- Transaction Logic Programming
- Constraint Logic Programming
- Disjunctive Logic Programming
- Answer Set Programming
- Inductive Logic Programming

Popular Languages:

- **Datalog**
- **Prolog**
- *Epilog* (our choice)
- Golog
- Progol
- LPS

Course Focus

We focus on *Basic Logic Programming* (a variant of Transaction Logic Programming) using **Epilog** for all examples and problem solvers.

What is Basic Logic Programming?

Core Concept

Basic Logic Programming models application behavior through *declarative facts* and *logical rules* rather than imperative commands.

Two Key Components:

1. **State Representation:**
   - Application states are modeled as **datasets** (sets of simple facts)
   - *Rules* define abstract *views* of these facts
   - Example: `parent(john, mary)` represents a fact
2. **State Transitions:**
   - Changes are modeled as **primitive updates** (additions/deletions)
   - *Action rules* define *compound actions* from primitive updates
   - Example: Adding or removing facts to modify the dataset

Basic Logic Programming: Key Features

What Makes Basic Logic Programming Special?

1. **Declarative Nature:**
   - Specify *what* should be computed, not *how* to compute it
   - Focus on relationships and constraints rather than algorithms
2. **Datasets as Knowledge Bases:**
   - Facts are stored in **datasets** (structured collections)
   - Rules query and derive new knowledge from existing facts
3. **Update Operations:**
   - **Primitive updates:** Direct addition/deletion of facts
   - **Compound actions:** Complex operations built from primitives
4. **Separation of Concerns:**
   - View definitions: *Query* and *derive* information
   - Action definitions: *Modify* the state of the system

Why Choose Logic Programming?

Core Advantage

Logic Programs are *easier to create*, *easier to modify*, and *more maintainable* than traditional imperative programs.

Key Benefits Compared to Traditional Programming:

1. **Enhanced Composability:**
   - Programs combine seamlessly without conflicts
   - Modular components can be reused across applications
2. **Greater Agility:**
   - Adapts quickly to changing **assumptions** and **goals**
   - Easier to update rules than rewrite algorithms
3. **Superior Versatility:**
   - Same program serves multiple purposes
   - Can query, update, and reason with the same codebase

Where is Logic Programming Used?

Ideal Domains

Logic Programming excels with **complex rules**, **many constraints**, and **changing requirements**.

Best Suited For: Domains with many definitions, multiple constraint sources, or frequently changing policies.

Key Application Areas:

- **Database Systems**
  Query optimization, constraints
- **Data Integration**
  Schema mapping, ETL
- **Enterprise Management**
  Workflow, business rules
- **Computational Law**
  Legal reasoning, compliance
- **General Game Playing**
  Strategy, rule-based AI
- **Logical Spreadsheets**
  Calculations, dependencies

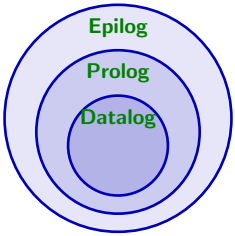
Outline

What is Epilog?

Epilog Language

Epilog[a] is a logic programming language based on **Dynamic Logic Programming (DLP)**, designed for expressing facts, rules, and queries declaratively.

[a] http://logic.stanford.edu/epilog/homepage/index.php



Datalog ⊂ *Prolog* ⊂ *Epilog*

EpilogJS: JavaScript Implementation

What is EpilogJS?

EpilogJS is a **JavaScript library** providing **subroutines for processing Epilog programs** in web environments

Library Components:

- **Predefined Functions:** String manipulation, Mathematical operations, etc.
- **Predefined Relations:** Comparison predicates, Type checking, etc.
- **Operators:** Logical operators, Arithmetic operators, Query operators, etc.
- **Execution Engine:** Rule evaluation, Query processing, Update operations, etc.

Essential Resources

- **Documentation:** Web — PDF
- **Required reference** throughout the course

Sierra: Interactive Development Environment

What is Sierra?

Sierra is a **browser-based IDE** for developing, testing, and debugging Epilog programs with real-time feedback.

Core Capabilities:

1. **Edit & View:** Create and modify **datasets** (facts), Write and edit **rulesets** (logic rules)
2. **Query & Update:** Interactive querying of datasets, Dynamic updates with immediate feedback
3. **Auto-Update:** **Spreadsheet-like** automatic recalculation, Real-time visualization of rule effects
4. **Development Tools:** Dataset/ruleset analysis, Execution tracing and debugging, File management (save/load)

Tutorial: Sierra IDE Guide — **Access:** logic.stanford.edu/epilog/sierra

Outline

Key Takeaways

Key Takeaways

- **Mathematical Logic** provides the foundational framework for AI reasoning and decision-making.
- **Logical Reasoning** enables systematic derivation of conclusions from premises using formal rules.
- **Boolean Algebra** and **Logic Gates** are essential for digital circuit analysis and simplification.
- **Logic Programming** offers a declarative approach to modeling knowledge and state transitions.
- **Epilog** and **Sierra IDE** facilitate the development and execution of Logic Programs.

Outline

Warning

Homework 1.1 and 1.2 should be submitted on Blackboard before the next course session!

Homework 1.1

Use Truth Tables to prove the following propositions.

- $A + (B.C) = (A + B).(A + C)$
- $A.(A + B) = A$
- $\overline{(A + B)} = \overline{A}.\overline{B}$
- $0 + X = X$
- $(A.(\overline{A} + B)) + B = B.$

Homework 1.2

1. Make a tour in Sierra IDE via the following link Sierra IDE.
2. Write the following datasets using Sierra IDE, then save it as file and upload it with your homework submission.

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,coe)
parent(bea,cory)
```

3. Revise the main features of Sierra IDE via the following link: Sierra IDE Guide
4. Revise the predefined concepts and the predefined functions in EpilogJS via one of the following links: WEB – PDF

End of Chapter 1