Knowledge Representation and Learning

Gabriel Rovesti

May 3, 2025

Contents

Chapter 1

Introduction

1.1 Knowledge Representation in Artificial Intelligence

The study of Knowledge Representation and Learning is central to artificial intelligence research. As defined by Russell and Norvig, an agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [**?**]. The ability of an agent to make informed decisions depends on how it represents and reasons with knowledge.

1.1.1 Types of Knowledge

An intelligent agent needs various types of knowledge:

- **Propositional constraints**: Limitations on what can be true or false (e.g., if block1 is on block2, then block2 cannot be on block1)
- **Probabilistic dependencies**: Rules that govern events (e.g., the probability of a fair coin toss resulting in tails is 1/2)
- **Action preconditions and effects**: Requirements for actions and their outcomes (e.g., to load a web page, an agent must be online)
- **Perception interpretation**: How to process sensor data (e.g., recognizing objects in images)
- **Causal implications**: Cause-effect relationships (e.g., blocks fall due to gravity)

1.1.2 Models for Knowledge Representation

Several types of mathematical models are used to represent knowledge:

- **Logical models**: Using formal logic to express facts and relationships
- **Probabilistic models**: Capturing uncertainty using probability theory
- **Neural network models**: Learning representations from data
- **Hybrid models**: Combining multiple approaches

In this course, we focus on:

- Logical models (propositional and first-order logic)
- Probabilistic models
- Statistical relational learning (combining logic and probability)
- Neuro-symbolic approaches (combining logic and neural networks)

1.2 Learning and Reasoning

Knowledge representation is intrinsically linked to both learning and reasoning:

- **Learning** involves acquiring or developing models that capture knowledge, through:
  - Manual specification
  - Inductive logic programming
  - Statistical machine learning
  - Neural network training
- **Reasoning** involves using knowledge to make inferences, through:
  - Logical deduction
  - Probabilistic inference
  - Forward/backward chaining
  - Constraint satisfaction

The Challenge of Integration

A key insight in modern AI is that logic and probability theory have addressed complementary aspects of knowledge representation:

- **Logic**: Ability to describe complex domains concisely in terms of objects and relations
- **Probability**: Ability to handle uncertain information

Their unification holds enormous promise for AI [**?**].

Chapter 2

Propositional Logic

2.1 Introduction to Propositional Logic

Propositional logic, also known as propositional calculus or sentential logic, is the simplest form of logic. It studies how propositions can be combined and reasoned about using logical connectives.

2.1.1 What is a Proposition?

A proposition is a declarative statement that can be either true or false, but not both. Examples of propositions include:

- "Today is Monday."
- "The derivative of $\sin x$ is $\cos x$."
- "Every even number has at least two factors."

Non-examples (expressions that are not propositions):

- "The dog of my girlfriend" (not a statement)
- "When is the pretest?" (a question)
- "Do your homework!" (a command)

2.2 Syntax of Propositional Logic

2.2.1 Alphabet

The alphabet of propositional logic consists of:

- **Logical symbols**: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), and $\equiv$ (equivalence)

- **Non-logical symbols**: A set P of propositional variables (typically denoted by lowercase letters $p, q, r, \ldots$)
- **Separator symbols**: Parentheses "(" and ")"

2.2.2 Well-Formed Formulas

The set of well-formed formulas (wffs) over a set P of propositional variables is inductively defined as:

1. Every $p \in P$ is a well-formed formula
2. If A is a well-formed formula, then $\neg A$ is a well-formed formula
3. If A and B are well-formed formulas, then $A \circ B$ is a well-formed formula, where $\circ \in \{\wedge, \vee, \rightarrow, \equiv\}$
4. If A is a well-formed formula, then (A) is a well-formed formula
5. Nothing else is a well-formed formula

Example 2.2.1. *Examples of well-formed formulas:*

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

Examples of expressions that are not well-formed formulas:

- PQ *(missing logical connective)*
- $(P \rightarrow \wedge((Q \rightarrow R)$ *(incorrect syntax)*
- $P \wedge Q \rightarrow \neg R\neg$ *(incorrect placement of negation)*

2.2.3 Reading Formulas

To properly interpret formulas, we need to understand operator precedence and associativity:

Additionally, binary connectives are right-associative, meaning:

- $a \rightarrow b \rightarrow c$ reads as $a \rightarrow (b \rightarrow c)$
- $a \wedge b \wedge c$ reads as $a \wedge (b \wedge c)$

Parentheses can be used to override default precedence and associativity.

Symbol	Priority
$\neg$	1 (highest)
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\equiv$	5 (lowest)

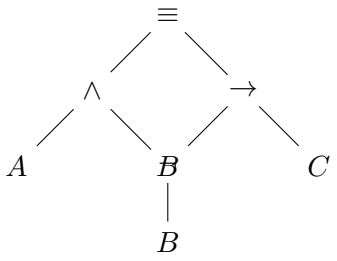
Table 2.1: Operator precedence in propositional logic

2.2.4 Formulas as Trees

A formula can be represented as a tree, where:

- Leaves are propositional variables
- Internal nodes are logical connectives

Example 2.2.2. *Consider the formula* $(A \wedge \neg B) \equiv (B \rightarrow C)$*. Its tree representation is:*



2.2.5 Subformulas

Definition 2.2.3. *A subformula of a formula* ϕ *is defined as:*

- ϕ *is a subformula of itself*
- *If* $\phi = \neg\psi$*, then* ψ *is a subformula of* ϕ
- *If* $\phi = \psi_1 \circ \psi_2$ *where* $\circ \in \{\wedge, \vee, \rightarrow, \equiv\}$*, then* ψ_1 *and* ψ_2 *are subformulas of* ϕ
- *If* ψ *is a subformula of* χ *and* χ *is a subformula of* ϕ*, then* ψ *is a subformula of* ϕ

A proper subformula of ϕ *is any subformula that is not* ϕ *itself.*

In the tree representation, subformulas correspond to subtrees.

2.3 Semantics of Propositional Logic

2.3.1 Interpretations

Definition 2.3.1. *An interpretation (or truth assignment) is a function* $I : P \to \{True, False\}$ *that assigns a truth value to each propositional variable.*

For a set P with n propositional variables, there are 2^n possible interpretations.

Example 2.3.2. *For* $P = \{p, q, r\}$*, there are* $2^3 = 8$ *possible interpretations:*

1. $I_1(p) = True, I_1(q) = True, I_1(r) = True$
2. $I_2(p) = True, I_2(q) = True, I_2(r) = False$
3. $I_3(p) = True, I_3(q) = False, I_3(r) = True$
4. $I_4(p) = True, I_4(q) = False, I_4(r) = False$
5. $I_5(p) = False, I_5(q) = True, I_5(r) = True$
6. $I_6(p) = False, I_6(q) = True, I_6(r) = False$
7. $I_7(p) = False, I_7(q) = False, I_7(r) = True$
8. $I_8(p) = False, I_8(q) = False, I_8(r) = False$

An alternative set-theoretic view: an interpretation I can be seen as the subset of propositional variables that are assigned True. For example, $I = \{p, r\}$ means $I(p) = \text{True}, I(q) = \text{False}, I(r) = \text{True}$.

2.3.2 Satisfaction of Formulas

Definition 2.3.3. *An interpretation* I *satisfies (or makes true) a formula* ϕ*, denoted* $I \models \phi$*, according to the following inductive definition:*

- *For* $p \in P$*,* $I \models p$ *if and only if* $I(p) = True$
- $I \models \neg\phi$ *if and only if* $I \not\models \phi$
- $I \models \phi \wedge \psi$ *if and only if* $I \models \phi$ *and* $I \models \psi$
- $I \models \phi \vee \psi$ *if and only if* $I \models \phi$ *or* $I \models \psi$ *(or both)*
- $I \models \phi \to \psi$ *if and only if* $I \not\models \phi$ *or* $I \models \psi$ *(or both)*
- $I \models \phi \equiv \psi$ *if and only if* $I \models \phi$ *if and only if* $I \models \psi$

Remark 2.3.4. *The semantics of* $\to$ *(material implication) does not assert causality. It only specifies that* $\phi \to \psi$ *is false when* ϕ *is true and* ψ *is false, and true otherwise.*

Example 2.3.5. *Let $I = \{p\}$ (meaning $I(p) = True, I(q) = False$).*

- $I \models p$ *(since p is assigned True)*
- $I \not\models q$ *(since q is assigned False)*
- $I \models \neg q$ *(since $I \not\models q$)*
- $I \not\models p \wedge q$ *(since $I \not\models q$)*
- $I \models p \vee q$ *(since $I \models p$)*
- $I \not\models p \rightarrow q$ *(since $I \models p$ but $I \not\models q$)*
- $I \models q \rightarrow p$ *(since $I \not\models q$)*

2.3.3 Validity, Satisfiability, and Unsatisfiability

Definition 2.3.6. *A formula ϕ is:*

- ***Valid*** *(or a tautology) if every interpretation satisfies it: $\forall I : I \models \phi$. Notation: $\models \phi$*
- ***Satisfiable*** *if some interpretation satisfies it: $\exists I : I \models \phi$*
- ***Unsatisfiable*** *(or a contradiction) if no interpretation satisfies it: $\nexists I : I \models \phi$*

Example 2.3.7.

- $p \vee \neg p$ *is valid (true under any interpretation)*
- $p \wedge q$ *is satisfiable (true when both p and q are true)*
- $p \wedge \neg p$ *is unsatisfiable (never true under any interpretation)*

2.3.4 Models of a Formula

Definition 2.3.8. *For a formula ϕ, the set Models$(\phi) = \{I \mid I \models \phi\}$ is the set of all interpretations that satisfy ϕ.*

Properties:

- If ϕ is satisfiable, then Models$(\phi) \neq \emptyset$
- If ϕ is valid, then Models$(\phi) = 2^{\text{prop}(\phi)}$ (all possible interpretations)
- If ϕ is unsatisfiable, then Models$(\phi) = \emptyset$
- Models$(\neg\phi) = 2^{\text{prop}(\phi)} \setminus$ Models(ϕ)
- Models$(\phi \wedge \psi) =$ Models$(\phi) \cap$ Models(ψ)
- Models$(\phi \vee \psi) =$ Models$(\phi) \cup$ Models(ψ)

2.3.5 Logical Consequence

Definition 2.3.9. *A formula ϕ is a logical consequence of a set of formulas Γ, denoted $\Gamma \models \phi$, if every interpretation that satisfies all formulas in Γ also satisfies ϕ.*

Example 2.3.10.
- $p \models p \vee q$ *(if p is true, then $p \vee q$ is true)*
- $p \vee q, p \to r, q \to r \models r$ *(if at least one of p or q is true, and both imply r, then r must be true)*
- $p \to q, p \models q$ *(modus ponens)*

Proposition 2.3.11. *The following properties hold for logical consequence:*
- *Reflexivity:* $\{A\} \models A$
- *Monotonicity: If* $\Gamma \models A$, *then* $\Gamma \cup \Sigma \models A$
- *Cut: If* $\Gamma \models A$ *and* $\Sigma \cup \{A\} \models B$, *then* $\Gamma \cup \Sigma \models B$
- *Deduction theorem: If* $\Gamma, A \models B$, *then* $\Gamma \models A \to B$
- *Refutation principle:* $\Gamma \models A$ *if and only if* $\Gamma \cup \{\neg A\}$ *is unsatisfiable*

2.4 Modeling in Propositional Logic

2.4.1 Representing Natural Language Statements

Translating natural language into propositional logic requires identifying atomic propositions and connecting them with logical operators.

Natural Language	**Propositional Logic**
"and", "but", "however", "moreover"	$\wedge$ (conjunction)
"or" (inclusive)	$\vee$ (disjunction)
"either...or...but not both"	$(A \vee B) \wedge \neg(A \wedge B)$
"if...then...", "implies", "when"	$\to$ (implication)
"only if"	reverse implication (A only if B means $A \to B$)
"if and only if", "iff", "exactly when"	$\equiv$ (equivalence)
"not", "it is false that"	$\neg$ (negation)
"unless"	similar to "if not" (A unless B means $\neg B \to A$)

Table 2.2: Translation from natural language to propositional logic

Example 2.4.1.
- *"If Sonia is happy and paints a picture, then Renzo isn't happy."* $\Rightarrow (s \wedge p) \to \neg r$
- *"If Sonia is happy, then she paints a picture."* $\Rightarrow s \to p$
- *"When Sonia paints a picture, she is happy."* $\Rightarrow p \to s$

2.4.2 Cardinality Constraints

Special formulas can express constraints on the number of true propositions:

- **At least k**: Given a set of boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, the constraint "at least k propositional variables in X are true" is formalized by:
$$\bigvee_{I \subseteq [n], |I|=k} \bigwedge_{i \in I} x_i$$
- **At most k**: Formalized as:
$$\bigwedge_{I \subseteq [n], |I|=k+1} \bigvee_{i \in I} \neg x_i$$
- **Exactly k**: Conjunction of "at least k" and "at most k"

Example 2.4.2. *For $X = \{a, b, c, d\}$:*

- *"At least 2 variables are true"* $\Rightarrow (a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
- *"At most 2 variables are true"* $\Rightarrow (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee \neg b \vee \neg d) \wedge (\neg a \vee \neg c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg d)$

2.5 Decision Procedures

Decision procedures are algorithms that determine properties of formulas, such as:

- Model checking: Given I and ϕ, determine whether $I \models \phi$
- Satisfiability: Given ϕ, determine whether ϕ is satisfiable
- Validity: Given ϕ, determine whether ϕ is valid
- Logical consequence: Given Γ and ϕ, determine whether $\Gamma \models \phi$

2.5.1 Truth Tables

Truth tables enumerate all possible interpretations and evaluate a formula under each.

Example 2.5.1. *Truth table for $p \rightarrow (q \vee \neg r)$:*

p	q	r	$\neg r$	$q \vee \neg r$	$p \to (q \vee \neg r)$
T	T	T	F	T	T
T	T	F	T	T	T
T	F	T	F	F	F
T	F	F	T	T	T
F	T	T	F	T	T
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	T	T

The formula is satisfiable but not valid (it's false in row 3).

2.5.2 Conjunctive Normal Form (CNF)

Definition 2.5.2. *A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals. A literal is either a propositional variable or its negation.*

Example 2.5.3. $(p \vee \neg q \vee r) \wedge (q \vee r) \wedge (\neg p \vee \neg q) \wedge r$ *is in CNF.*

Every propositional formula can be converted to an equivalent formula in CNF through the following steps:

1. Replace implications: $A \to B \Rightarrow \neg A \vee B$
2. Replace equivalences: $A \equiv B \Rightarrow (\neg A \vee B) \wedge (\neg B \vee A)$
3. Push negations inward (De Morgan's laws):
   - $\neg(A \vee B) \Rightarrow \neg A \wedge \neg B$
   - $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$
   - $\neg\neg A \Rightarrow A$
4. Distribute disjunction over conjunction:
   - $A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)$
   - $(A \wedge B) \vee C \Rightarrow (A \vee C) \wedge (B \vee C)$

Remark 2.5.4. *Converting a formula to CNF can cause exponential blowup in formula size. Tseitin's transformation provides a more efficient alternative by introducing new variables, producing an equisatisfiable (but not equivalent) formula with only linear size increase.*

2.5.3 The DPLL Algorithm

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm is a complete, backtracking-based search algorithm for deciding satisfiability of CNF formulas.

Here, $\phi|_l$ denotes the formula obtained from ϕ by setting l to true and simplifying.

Algorithm 1 DPLL Algorithm

```
1: function DPLL(φ, I)
2:     I, φ ← UnitPropagation(I, φ)
3:     if {} ∈ φ then                          ▷ Empty clause means unsatisfiable
4:         return Unsatisfiable
5:     end if
6:     if φ = {} then                          ▷ Empty formula means satisfiable
7:         return I
8:     else
9:         select a literal l from some clause C ∈ φ
10:        I' = DPLL(φ|_l, I ∪ {l})            ▷ Try setting l to true
11:        if I' ≠ Unsatisfiable then
12:            return I'
13:        else
14:            I' = DPLL(φ|_¬l, I ∪ {¬l})      ▷ Try setting l to false
15:            return I'
16:        end if
17:    end if
18: end function
```

Algorithm 2 Unit Propagation

```
1: function UnitPropagation(I, φ)
2:     while φ contains a unit clause {l} do
3:         I, φ ← I ∪ {l}, φ|_l                ▷ Set l to true and simplify
4:         if {} ∈ φ then                      ▷ Found empty clause, unsatisfiable
5:             return I, φ
6:         end if
7:     end while
8:     return I, φ
9: end function
```

2.5.4 Resolution

Resolution is a rule of inference that produces a new clause from two clauses containing complementary literals.

Definition 2.5.5 (Resolution Rule)**.**

$$\frac{A \vee C, \neg C \vee B}{A \vee B}$$

The resulting clause $A \vee B$ is called the resolvent.

More generally, using set notation for clauses:

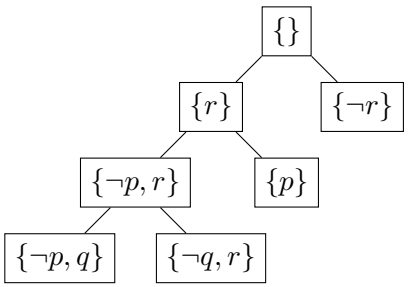
$$\frac{\{A_1, \ldots, C, \ldots, A_m\}, \{B_1, \ldots, \neg C, \ldots, B_n\}}{\{A_1, \ldots, A_m, B_1, \ldots, B_n\}}$$

Resolution provides a complete procedure for determining unsatisfiability: A set of clauses Γ is unsatisfiable if and only if the empty clause $\{\}$ can be derived through a series of resolution steps.

Algorithm 3 Propositional Resolution

```
1: function PropositionalResolution(Γ)
2:     while no new clauses are derivable do
3:         Select clauses C1, C2 ∈ Γ and literal p such that p ∈ C1, ¬p ∈ C2
4:         Γ ← Γ ∪ {(C1 ∪ C2) \ {p, ¬p}}
5:         if {} ∈ Γ then
6:             return Unsatisfiable
7:         end if
8:     end while
9:     return Satisfiable
10: end function
```

Example 2.5.6. *Consider the clause set $\{\{p\}, \{\neg p, q\}, \{\neg q, r\}, \{\neg r\}\}$:*



The derivation of the empty clause proves the set is unsatisfiable.

2.6 Maximum Satisfiability (MaxSAT)

2.6.1 The MaxSAT Problem

In many situations, not all constraints can be satisfied simultaneously, and we need to find an assignment that satisfies as many constraints as possible.

Definition 2.6.1. *Given a set of clauses $\{C_1, \ldots, C_n\}$, the Maximum Satisfiability (MaxSAT) problem asks for an assignment that satisfies the maximum possible number of clauses.*

2.6.2 Weighted MaxSAT

Definition 2.6.2. *In Weighted MaxSAT, each clause C_i is associated with a weight $w_i > 0$. The goal is to find an assignment that maximizes the sum of weights of satisfied clauses, or equivalently, minimizes the sum of weights of unsatisfied clauses.*

Variants of MaxSAT:

- **Basic MaxSAT**: All clauses have weight 1, no hard constraints
- **Partial MaxSAT**: Some clauses are hard (must be satisfied), others have weight 1
- **Weighted MaxSAT**: All clauses have weights, no hard constraints
- **Weighted Partial MaxSAT**: Some clauses are hard, others have weights

Hard constraints can be represented by assigning them infinite weight or, practically, a weight greater than the sum of all soft constraint weights.

2.6.3 Algorithms for MaxSAT

Branch and Bound

Branch and Bound is an extension of DPLL that keeps track of the best solution found so far (upper bound) and estimates a lower bound on the cost of completing the current partial assignment.

Fu and Malik Algorithm

The Fu and Malik algorithm is a core-guided approach for MaxSAT that iteratively identifies unsatisfiable cores (minimal unsatisfiable subsets) and relaxes them.

Algorithm 4 Branch and Bound for MaxSAT

function B&B($\phi, \psi, I, I_{UB}, UB$)
  $I, \phi \leftarrow$ UnitPropagation(I, ϕ)
  $\psi \leftarrow \psi|_I$ ▷ Update weighted clauses
  $LB \leftarrow \sum_{w:\{\}\in\psi} w$ ▷ Lower bound from empty clauses
  if $\{\} \in \phi$ or $UB \leq LB$ **then** ▷ Hard constraint violated or cannot improve
    return I_{UB}, UB
  end if
  if $\phi = \{\}$ and ψ contains only empty weighted clauses **then**
    return $I, \sum_{(w:C)\in\psi} w$ ▷ Complete solution with cost
  else
    select a literal l from some clause
    $I', UB' \leftarrow$ B&B($\phi|_l, \psi|_l, I \cup \{l\}, I_{UB}, UB$)
    $I'', UB'' \leftarrow$ B&B($\phi|_{\neg l}, \psi|_{\neg l}, I \cup \{\neg l\}, I', UB'$)
    return I'', UB''
  end if
end function

Algorithm 5 Fu and Malik Algorithm for MaxSAT

function FuMalikMaxSAT(ϕ)
  if SAT(ϕ) = Unsat **then** ▷ Hard constraints unsatisfiable
    return $(\infty, \emptyset)$
  end if
  cost $\leftarrow 0$
  s $\leftarrow 0$
  while True **do**
    $(st, \phi_c) \leftarrow$ SAT(ϕ) ▷ Try to satisfy all clauses
    if st = Satisfiable **then**
      return (cost, ϕ) ▷ Found optimal solution
    end if
    s $\leftarrow s + 1$ ▷ New set of relaxation variables
    $A_s \leftarrow \emptyset$ ▷ Clauses to relax
    for soft clause $C_i \in \phi_c$ **do**
      $b_i^s \leftarrow$ new propositional variable
      $\phi \leftarrow \phi \setminus \{(C_i, 1)\} \cup \{(C_i \vee b_i^s, 1)\}$ ▷ Relax clause
      $A_s \leftarrow A_s \cup \{i\}$
    end for
    $\phi \leftarrow \phi \cup \{(C, \infty) | C \in \text{CNF}(\sum_{i\in A_s} b_i^s = 1)\}$ ▷ At most one relaxation
    cost $\leftarrow$ cost $+1$
  end while
end function

2.7 Model Counting

2.7.1 The SAT Problem

Definition 2.7.1. *The propositional model counting or SAT problem is: given a propositional formula ϕ, compute the number of distinct truth assignments that satisfy ϕ.*

Formally:

$$\#\text{sat}(\phi, P) = \sum_{I:P\to\{0,1\}} I(\phi)$$

where $I(\phi) = 1$ if $I \models \phi$ and $I(\phi) = 0$ otherwise.

Model counting has applications in probabilistic reasoning, statistical relational learning, and other areas.

Example 2.7.2. *For $P = \{p, q\}$:*

- $\#sat(p \wedge q) = 1$ *(only one satisfying assignment)*
- $\#sat(p \vee q) = 3$ *(three satisfying assignments)*
- $\#sat(p \to q) = 3$ *(three satisfying assignments)*
- $\#sat(p \equiv q) = 2$ *(two satisfying assignments)*

2.7.2 Properties of Model Counting

Proposition 2.7.3. *Let ϕ be a propositional formula with variables in P:*

- *If ϕ is valid, $\#sat(\phi) = 2^{|P|}$*
- *If ϕ is unsatisfiable, $\#sat(\phi) = 0$*
- $\#sat(\neg\phi) = 2^{|P|} - \#sat(\phi)$
- *If ϕ and ψ do not share variables, $\#sat(\phi \wedge \psi) = \#sat(\phi) \cdot \#sat(\psi)$*
- $\#sat(\phi) = \#sat(\phi \wedge p) + \#sat(\phi \wedge \neg p)$ *for any variable p*

2.7.3 Algorithms for Model Counting

DPLL-Based Model Counters

Model counting algorithms often extend DPLL by keeping track of the number of solutions rather than just finding one.

Algorithm 6 DPLL-Based Model Counting

```
1: function CountDP(φ, n)
2:     if φ = {} then                          ▷ Empty set of clauses, tautology
3:         return 2^n                          ▷ All possible assignments satisfy
4:     end if
5:     if {} ∈ φ then                          ▷ Contains empty clause, unsatisfiable
6:         return 0
7:     end if
8:     if φ contains unit clause {l} then
9:         return CountDP(φ|_l, n − 1)         ▷ Unit propagation
10:    else
11:        Select variable p from φ
12:        return CountDP(φ|_p, n − 1) + CountDP(φ|_¬p, n − 1)
13:    end if
14: end function
```

Knowledge Compilation for Model Counting

Converting formulas to certain normal forms allows efficient model counting:

Definition 2.7.4 (d-DNNF)**.** *A formula is in deterministic decomposable negation normal form (d-DNNF) if:*

- *Negation appears only in front of atoms (NNF)*
- *For each conjunction $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$, no two conjuncts share variables (decomposable)*
- *For each disjunction $\phi_1 \vee \phi_2 \vee \ldots \vee \phi_n$, each pair of disjuncts is mutually unsatisfiable (deterministic)*

For a d-DNNF formula, model counting can be done in polynomial time by:

- For literals, count = 1
- For $\phi \wedge \psi$, count = count(ϕ) $\times$ count(ψ) (due to decomposability)
- For $\phi \vee \psi$, count = count(ϕ) + count(ψ) (due to determinism)

2.7.4 Weighted Model Counting

Definition 2.7.5. *Given a weight function $w : 2^P \to \mathbb{R}^+$ that assigns a positive real number to each interpretation, the weighted model counting problem is to compute:*

$$wmc(\phi, w) = \sum_{I \models \phi} w(I)$$

Weighted model counting is closely related to probabilistic inference: if weights sum to 1, they can be interpreted as probabilities.

Example 2.7.6. *Let* $w(I) = e^{\sum_{p \in I} v(p) + \sum_{p \notin I} v(\neg p)}$*, where* v *assigns values to literals. Then:*

$$wmc(\phi, w) = \sum_{I \models \phi} e^{\sum_{p \in I} v(p) + \sum_{p \notin I} v(\neg p)}$$

This can represent probabilistic models where $Pr(I) = \frac{w(I)}{Z(w)}$ *and* $Z(w) = \sum_I w(I)$ *is the partition function.*

Chapter 3

First-Order Logic

3.1 Introduction to First-Order Logic

First-order logic (FOL), also known as predicate logic, extends propositional logic by allowing us to express properties of objects and relationships between objects. It introduces:

- Variables that range over objects in a domain
- Quantifiers that express properties about all objects or some objects in the domain
- Functions that map objects to objects
- Predicates that express properties of objects or relationships between objects

While propositional logic can only represent static facts, FOL can express general rules and structural knowledge.

3.1.1 Limitations of Propositional Logic

Propositional logic cannot naturally express:

- Generalized statements about collections of objects ("All persons are mortal")
- Existential statements ("There exists a person who is a spy")
- Relationships involving functions ("The father of Luca is Italian")

Attempting to express these in propositional logic leads to unwieldy or incomplete representations.

3.2 Syntax of First-Order Logic

3.2.1 Alphabet

The alphabet of FOL consists of:

- **Logical symbols**:
  - The logical constant $\bot$ (false)
  - Propositional connectives: $\wedge, \vee, \rightarrow, \neg, \equiv$
  - Quantifiers: $\forall$ (universal), $\exists$ (existential)
  - Variables: $x_1, x_2, \ldots$
  - The equality symbol $=$ (optional)
- **Non-logical symbols** (signature Σ):
  - Constants: $c_1, c_2, \ldots$
  - Function symbols: $f_1, f_2, \ldots$ each with an arity (number of arguments)
  - Predicate symbols: $P_1, P_2, \ldots$ each with an arity

Example 3.2.1. *A signature for arithmetic might include:*

- *Constants:* 0
- *Functions: succ (unary),* $+$ *(binary),* $\times$ *(binary)*
- *Predicates:* $<$ *(binary),* $\leq$ *(binary)*

3.2.2 Terms

Definition 3.2.2. *Terms in FOL are recursively defined as:*

- *Every constant is a term*
- *Every variable is a term*
- *If* $t_1, \ldots, t_n$ *are terms and* f *is a function symbol with arity* n*, then* $f(t_1, \ldots, t_n)$ *is a term*

Example 3.2.3. *Given constants* a, b*, variable* x*, and functions* f *(binary) and* g *(unary):*

- a*,* b*,* x *are terms*
- $f(a, b)$*,* $g(x)$ *are terms*
- $f(g(x), a)$*,* $f(a, f(b, g(x)))$ *are terms*

Definition 3.2.4. *A ground term is a term that does not contain variables.*

3.2.3 Formulas

Definition 3.2.5. *Atomic formulas are:*

- $P(t_1, \ldots, t_n)$ *where* P *is a predicate symbol with arity* n *and* $t_1, \ldots, t_n$ *are terms*
- $t_1 = t_2$ *where* t_1 *and* t_2 *are terms (if equality is in the language)*

Definition 3.2.6. *Formulas are recursively defined as:*

- *Atomic formulas are formulas*
- *If* ϕ *and* ψ *are formulas, then* $\bot$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \to \psi$, $\neg\phi$, *and* $\phi \equiv \psi$ *are formulas*
- *If* ϕ *is a formula and* x *is a variable, then* $\forall x.\phi$ *and* $\exists x.\phi$ *are formulas*

Example 3.2.7.

- $P(x)$ *is a formula*
- $\forall x.(P(x) \to Q(x))$ *is a formula*
- $\exists x.(P(x) \wedge \forall y.R(x, y))$ *is a formula*

3.2.4 Scope and Free Variables

Definition 3.2.8. *The scope of a quantifier* Q *in* $Qx.\phi$ *is the formula* ϕ.

Definition 3.2.9. *A free occurrence of a variable* x *in a formula is an occurrence not within the scope of a quantifier over* x. *A variable is free in a formula if it has at least one free occurrence.*

Example 3.2.10. *In the formula* $\forall x.P(x) \wedge Q(y)$*:*

- x *is bound (not free)*
- y *is free*

In the formula $\forall x.(P(x, y) \to \exists y.R(x, y))$*:*

- *All occurrences of* x *are bound*
- *The occurrence of* y *in* $P(x, y)$ *is free*
- *The occurrence of* y *in* $R(x, y)$ *is bound*

Definition 3.2.11. *A formula is:*

- *Ground if it contains no variables*
- *Closed (or a sentence) if it contains no free variables*

3.3 Semantics of First-Order Logic

3.3.1 Interpretations

Definition 3.3.1. *An interpretation $\mathcal{I}$ for a signature Σ is a pair $\langle \Delta, I \rangle$ where:*

- *Δ is a non-empty set called the domain or universe*
- *I is an interpretation function that assigns:*
  - *For each constant c, an element $I(c) \in \Delta$*
  - *For each function symbol f of arity n, a function $I(f) : \Delta^n \to \Delta$*
  - *For each predicate symbol P of arity n, a relation $I(P) \subseteq \Delta^n$*

Example 3.3.2. *Consider a signature with constants a, b, function f (unary), and predicate P (binary). An interpretation might be:*

- $\Delta = \{1, 2, 3\}$
- $I(a) = 1$, $I(b) = 2$
- $I(f) = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 1\}$
- $I(P) = \{(1, 2), (2, 3), (3, 1)\}$

3.3.2 Variable Assignments

Definition 3.3.3. *A variable assignment α for an interpretation $\mathcal{I} = \langle \Delta, I \rangle$ is a function that maps variables to elements of Δ.*

We write $\alpha[x \mapsto d]$ for the assignment that is identical to α except that it maps x to d.

3.3.3 Interpretation of Terms

Definition 3.3.4. *Given an interpretation $\mathcal{I} = \langle \Delta, I \rangle$ and a variable assignment α, the interpretation of a term t, denoted $I(t)[\alpha]$, is defined as:*

- *$I(x)[\alpha] = \alpha(x)$ for a variable x*
- *$I(c)[\alpha] = I(c)$ for a constant c*
- *$I(f(t_1, \ldots, t_n))[\alpha] = I(f)(I(t_1)[\alpha], \ldots, I(t_n)[\alpha])$ for a function application*

3.3.4 Satisfaction of Formulas

Definition 3.3.5. *An interpretation $\mathcal{I}$ with variable assignment α satisfies a formula ϕ, denoted $\mathcal{I} \models \phi[\alpha]$, according to the following rules:*

- $\mathcal{I} \models P(t_1, \ldots, t_n)[\alpha]$ *iff* $(I(t_1)[\alpha], \ldots, I(t_n)[\alpha]) \in I(P)$
- $\mathcal{I} \models (t_1 = t_2)[\alpha]$ *iff* $I(t_1)[\alpha] = I(t_2)[\alpha]$
- $\mathcal{I} \not\models \bot[\alpha]$
- $\mathcal{I} \models (\phi \wedge \psi)[\alpha]$ *iff* $\mathcal{I} \models \phi[\alpha]$ *and* $\mathcal{I} \models \psi[\alpha]$
- $\mathcal{I} \models (\phi \vee \psi)[\alpha]$ *iff* $\mathcal{I} \models \phi[\alpha]$ *or* $\mathcal{I} \models \psi[\alpha]$
- $\mathcal{I} \models (\phi \to \psi)[\alpha]$ *iff* $\mathcal{I} \not\models \phi[\alpha]$ *or* $\mathcal{I} \models \psi[\alpha]$
- $\mathcal{I} \models (\neg\phi)[\alpha]$ *iff* $\mathcal{I} \not\models \phi[\alpha]$
- $\mathcal{I} \models (\forall x.\phi)[\alpha]$ *iff for all* $d \in \Delta$, $\mathcal{I} \models \phi[\alpha[x \mapsto d]]$
- $\mathcal{I} \models (\exists x.\phi)[\alpha]$ *iff there exists* $d \in \Delta$ *such that* $\mathcal{I} \models \phi[\alpha[x \mapsto d]]$

For a closed formula, satisfaction is independent of the variable assignment, so we simply write $\mathcal{I} \models \phi$.

3.3.5 Models, Satisfiability, and Validity

Definition 3.3.6. *An interpretation $\mathcal{I}$ is a model of a formula ϕ if $\mathcal{I} \models \phi$. A formula ϕ is:*

- *Satisfiable if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \phi$*
- *Valid (or a tautology) if for every interpretation $\mathcal{I}$, $\mathcal{I} \models \phi$*
- *Unsatisfiable if it has no models*

Definition 3.3.7. *A formula ϕ is a logical consequence of a set of formulas Γ, denoted $\Gamma \models \phi$, if for every interpretation $\mathcal{I}$ and assignment α such that $\mathcal{I} \models \psi[\alpha]$ for all $\psi \in \Gamma$, we have $\mathcal{I} \models \phi[\alpha]$.*

Example 3.3.8.

- $\forall x.P(x) \models P(c)$ *(universal instantiation)*
- $P(c) \models \exists x.P(x)$ *(existential generalization)*
- $\forall x.(P(x) \to Q(x)), \forall x.P(x) \models \forall x.Q(x)$ *(universal modus ponens)*

3.4 Quantifier Properties and Equivalences

3.4.1 Quantifier Transformations

Proposition 3.4.1. *The following equivalences hold:*

- *Quantifiers of the same type commute:*
  - $\forall x.\forall y.\phi(x,y) \equiv \forall y.\forall x.\phi(x,y)$
  - $\exists x.\exists y.\phi(x,y) \equiv \exists y.\exists x.\phi(x,y)$
- *Quantifiers of different types generally do not commute:*
  - $\forall x.\exists y.\phi(x,y) \not\equiv \exists y.\forall x.\phi(x,y)$
- *Quantifier duality (De Morgan laws for quantifiers):*
  - $\neg\forall x.\phi(x) \equiv \exists x.\neg\phi(x)$
  - $\neg\exists x.\phi(x) \equiv \forall x.\neg\phi(x)$
- *Distribution over connectives:*
  - $\forall x.(\phi(x) \wedge \psi(x)) \equiv \forall x.\phi(x) \wedge \forall x.\psi(x)$
  - $\exists x.(\phi(x) \vee \psi(x)) \equiv \exists x.\phi(x) \vee \exists x.\psi(x)$
  - $\forall x.(\phi(x) \vee \psi(x)) \not\equiv \forall x.\phi(x) \vee \forall x.\psi(x)$
  - $\exists x.(\phi(x) \wedge \psi(x)) \not\equiv \exists x.\phi(x) \wedge \exists x.\psi(x)$
- *Quantifier absorption:*
  - $\forall x.\exists x.\phi(x) \equiv \exists x.\phi(x)$
  - $\exists x.\forall x.\phi(x) \equiv \forall x.\phi(x)$

3.4.2 Prenex Normal Form

Definition 3.4.2. *A formula is in prenex normal form if it has the form $Q_1x_1.Q_2x_2.\ldots Q_nx_n.\phi$, where each Q_i is either $\forall$ or $\exists$, and ϕ contains no quantifiers.*

Every FOL formula can be converted to an equivalent formula in prenex normal form through the following steps:

1. Rename variables so each quantifier uses a unique variable
2. Eliminate implications and equivalences
3. Push negations inward using De Morgan laws and quantifier duality
4. Move quantifiers outward using the following equivalences (where x is not free in ψ):

- $(\forall x.\phi) \wedge \psi \equiv \forall x.(\phi \wedge \psi)$
- $(\forall x.\phi) \vee \psi \equiv \forall x.(\phi \vee \psi)$
- $(\exists x.\phi) \wedge \psi \equiv \exists x.(\phi \wedge \psi)$
- $(\exists x.\phi) \vee \psi \equiv \exists x.(\phi \vee \psi)$

3.5 First-Order Theories

3.5.1 Definition of Theories

Definition 3.5.1. *A first-order theory T is a set of FOL formulas closed under logical consequence: if $T \models \phi$, then $\phi \in T$.*

Definition 3.5.2. *Given a class of Σ-structures S, the theory of S, denoted $Th(S)$, is the set of all Σ-sentences that are satisfied by all structures in S:*

$$Th(S) = \{\phi \mid \forall \mathcal{I} \in S, \mathcal{I} \models \phi\}$$

Definition 3.5.3. *A set of formulas Ω is a set of axioms for a theory T if for all $\phi \in T$, $\Omega \models \phi$.*

Definition 3.5.4. *A theory is:*

- *Finitely axiomatizable if it has a finite set of axioms*
- *Recursively axiomatizable if it has a recursive set of axioms*

3.5.2 Examples of First-Order Theories

Peano Arithmetic

Peano Arithmetic (PA) is a first-order theory of the natural numbers with signature $\{0, \text{succ}, +, \times\}$ and axioms:

1. $\forall x.0 \neq \text{succ}(x)$
2. $\forall x, y.\text{succ}(x) = \text{succ}(y) \rightarrow x = y$
3. $\forall x.x + 0 = x$
4. $\forall x, y.x + \text{succ}(y) = \text{succ}(x + y)$
5. $\forall x.x \times 0 = 0$
6. $\forall x, y.x \times \text{succ}(y) = (x \times y) + x$
7. The induction axiom schema: $\phi(0) \wedge \forall x.(\phi(x) \rightarrow \phi(\text{succ}(x))) \rightarrow \forall x.\phi(x)$ for every formula ϕ with at least one free variable

Other Examples

- Group theory
- Field theory
- Set theory
- Theories of specific data structures (e.g., lists, trees)

3.5.3 Knowledge Representation Tricks

- **Closed World Assumption**: Specifying that the domain contains only the named objects:

$$\forall x.x = a \vee x = b \vee \ldots \vee x = z$$

- **Unique Name Assumption**: Constants refer to distinct objects:

$$a \neq b \wedge a \neq c \wedge \ldots \wedge y \neq z$$

- **Domain Cardinality**: Specifying the exact number of elements in the domain:

$$\exists x_1, \ldots, x_n. \left(\bigwedge_{i<j} x_i \neq x_j \wedge \forall x. \bigvee_{i=1}^{n} x = x_i \right)$$

3.6 Herbrand's Theorem

3.6.1 Herbrand Universe

Definition 3.6.1. *Given a signature Σ containing at least one constant, the Herbrand universe Δ_H is the set of all ground terms that can be constructed from the constants and function symbols in Σ.*

If Σ does not contain constants, we add a new constant (e.g., a) to form the Herbrand universe.

Example 3.6.2. *For $\Sigma = \{a, b, f(\_), g(\_, \_)\}$:*

$$\Delta_H = \{a, b, f(a), f(b), g(a, a), g(a, b), g(b, a), g(b, b), \\ f(f(a)), f(f(b)), f(g(a, a)), \ldots\}$$

3.6.2 Herbrand Interpretations

Definition 3.6.3. *A Herbrand interpretation for a signature Σ is an interpretation $\mathcal{I} = \langle \Delta_H, H \rangle$ where:*

- Δ_H *is the Herbrand universe*
- $H(c) = c$ *for each constant symbol* c
- $H(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ *for each function symbol* f
- $H(P) \subseteq \Delta_H^n$ *for each predicate symbol* P *of arity* n

Definition 3.6.4. *The Herbrand base HB_Σ is the set of all ground atomic formulas that can be formed using predicate symbols from Σ and terms from Δ_H.*

Theorem 3.6.5 (Herbrand's Theorem)**.** *A universal formula $\forall x_1, \ldots, x_n.\phi(x_1, \ldots, x_n)$, where ϕ is quantifier-free, is satisfiable if and only if it is satisfied by a Herbrand interpretation.*

This theorem allows us to restrict our search for models to Herbrand interpretations when checking satisfiability of universal formulas.

3.6.3 Skolemization

Definition 3.6.6. *Skolemization is a process that eliminates existential quantifiers from a formula by replacing existentially quantified variables with new function symbols.*

The basic idea:

- For $\exists x.\phi(x)$, introduce a new constant c and replace the formula with $\phi(c)$
- For $\forall x_1, \ldots, x_n.\exists y.\phi(x_1, \ldots, x_n, y)$, introduce a new function symbol f and replace the formula with $\forall x_1, \ldots, x_n.\phi(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$

Skolemization preserves satisfiability but not logical equivalence.

Example 3.6.7. *Skolemizing $\forall x.\exists y.P(x, y)$ gives $\forall x.P(x, f(x))$ where f is a new function symbol.*

3.7 Resolution and Unification

3.7.1 Clausal Form

Definition 3.7.1. *A clause is a disjunction of literals. In set notation, it is represented as a set of literals.*

Every FOL formula can be converted to clausal form (a set of clauses) through the following steps:

1. Convert to prenex normal form
2. Skolemize to eliminate existential quantifiers
3. Drop the universal quantifiers (all variables are implicitly universally quantified)
4. Convert the matrix to conjunctive normal form
5. Separate the conjuncts into a set of clauses

3.7.2 Substitutions

Definition 3.7.2. *A substitution σ is a finite set of replacements $\{x_1/t_1, \ldots, x_n/t_n\}$ where each x_i is a variable and each t_i is a term different from x_i.*

Definition 3.7.3. *The application of a substitution σ to a term or formula ϕ, denoted $\phi\sigma$, is the term or formula obtained by simultaneously replacing each occurrence of variable x_i with the term t_i.*

Definition 3.7.4. *The composition of substitutions σ and θ, denoted $\sigma \circ \theta$, is the substitution that satisfies:*

$$t(\sigma \circ \theta) = (t\sigma)\theta$$

for all terms t.

Composition is associative but not commutative.

3.7.3 Unification

Definition 3.7.5. *A substitution σ unifies terms s and t if $s\sigma = t\sigma$.*

Definition 3.7.6. *A substitution σ is more general than a substitution θ if there exists a substitution λ such that $\theta = \sigma \circ \lambda$.*

Definition 3.7.7. *A most general unifier (MGU) of terms s and t is a unifier that is more general than any other unifier of s and t.*

Example 3.7.8.

- *The terms $f(a, x)$ and $f(a, b)$ are unifiable with MGU $\{x/b\}$*
- *The terms $f(x, g(x))$ and $f(h(y), z)$ are unifiable with MGU $\{x/h(y), z/g(h(y))\}$*
- *The terms $f(x)$ and $g(y)$ are not unifiable (different function symbols)*
- *The terms $f(x)$ and $f(g(x))$ are not unifiable (occurs check fails)*

Algorithm 7 Robinson's Unification Algorithm

```
1: function UNIFY(Δ)
2:     σ ← {}
3:     while |Δσ| > 1 do
4:         Pick a disagreement pair p in Δσ
5:         if no variable in p then
6:             return "not unifiable"
7:         else
8:             Let p = (x, t) with x being a variable
9:             if x occurs in t then
10:                return "not unifiable"                ▷ Occurs check
11:            else
12:                σ ← σ ∘ {x/t}
13:            end if
14:        end if
15:    end while
16:    return σ
17: end function
```

3.7.4 Resolution for First-Order Logic

The resolution rule for FOL extends the propositional version by incorporating unification:

Definition 3.7.9 (First-Order Resolution Rule)**.**

$$\frac{C_1 \vee L, C_2 \vee \neg M}{(C_1 \vee C_2)\sigma}$$

where σ is the MGU of literals L and M.

In set notation:

$$\frac{\{A_1, \ldots, L, \ldots, A_m\}, \{B_1, \ldots, \neg M, \ldots, B_n\}}{\{A_1, \ldots, A_m, B_1, \ldots, B_n\}\sigma}$$

if σ is the MGU of L and M.

Example 3.7.10. *Resolving $\{P(f(x))\}$ and $\{\neg P(f(a)), Q(x)\}$:*

- *The MGU of $P(f(x))$ and $P(f(a))$ is $\{x/a\}$*
- *The resolvent is $\{Q(x)\}\{x/a\} = \{Q(a)\}$*

Factoring Rule

Definition 3.7.11 (Factoring Rule)**.**

$$\frac{C \vee L_1 \vee \ldots \vee L_n}{(C \vee L_1)\sigma}$$

if σ is the MGU such that $L_1\sigma = \ldots = L_n\sigma$.

Resolution as a Refutation Procedure

Resolution for FOL provides a complete refutation procedure:

- To prove that $\Gamma \models \phi$, convert $\Gamma \cup \{\neg\phi\}$ to clausal form
- Apply resolution repeatedly to derive new clauses
- If the empty clause is derived, $\Gamma \models \phi$; otherwise, $\Gamma \not\models \phi$

Example 3.7.12. *To prove* $\forall x.\exists y.\neg(P(y,x) \equiv \neg P(y,y))$*:*
Convert to clausal form:

- *Negation:* $\exists x.\forall y.(P(y,x) \equiv \neg P(y,y))$
- *Skolemization:* $\forall y.(P(y,a) \equiv \neg P(y,y))$ *where* a *is a Skolem constant*
- *Clausal form:* $\{\neg P(y,a), \neg P(y,y)\}, \{P(y,y), P(y,a)\}$

Applying factoring to both clauses with $\{y/a\}$*:*

- $\{\neg P(a,a)\}$
- $\{P(a,a)\}$

Resolving these clauses gives the empty clause $\{\}$*, proving the formula.*

Chapter 4

Statistical Relational Learning

4.1 Introduction to Statistical Relational Learning

Statistical Relational Learning (SRL) combines relational representations (like logic) with probabilistic reasoning to handle both structure and uncertainty. It addresses limitations of traditional approaches:

- Classical logic cannot handle uncertainty
- Probabilistic models struggle with complex relational structures

SRL is motivated by real-world domains where:

- Relations between entities are important
- There is significant uncertainty
- Both statistical patterns and logical rules are needed

4.1.1 Unifying Logic and Probability

As noted by Russell [**?**], logic and probability theory have addressed complementary aspects of knowledge representation:

- Logic: Describing complex domains in terms of objects and relations
- Probability: Handling uncertain information

SRL frameworks aim to bridge this gap through different approaches:

- Adding probability to logic (e.g., Probabilistic Logic Programming)
- Adding logical structure to probabilistic models (e.g., Markov Logic Networks)

- Building new formalisms that integrate both (e.g., Tensor Logic Networks)

4.2 Probabilistic Graphical Models

4.2.1 Introduction to Graphical Models

Probabilistic graphical models represent probability distributions using graphs, where:

- Nodes represent random variables
- Edges represent probabilistic dependencies

The two main types are:

- Directed graphical models (Bayesian networks)
- Undirected graphical models (Markov random fields)

4.2.2 Bayesian Networks

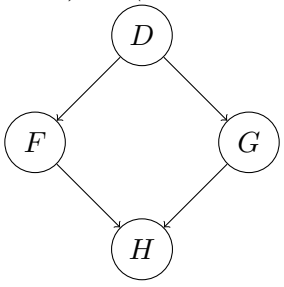
Definition 4.2.1. *A Bayesian network is a directed acyclic graph (DAG) where:*

- *Each node represents a random variable X_i*
- *Each edge represents a direct dependency*
- *Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$*

The joint probability distribution is given by:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i))$$

Example 4.2.2. *Consider a simple Bayesian network with variables D (doing work), F (finished work), G (getting tired), and H (having a rest):*



With conditional probability tables:

- $P(D) = 0.5$
- $P(F|D = \textit{true}) = 0.5$, $P(F|D = \textit{false}) = 0.1$
- $P(G|D = \textit{true}) = 0.7$, $P(G|D = \textit{false}) = 0.2$
- $P(H|F = \textit{true}, G = \textit{true}) = 1.0$, $P(H|F = \textit{true}, G = \textit{false}) = 0.5$, $P(H|F = \textit{false}, G = \textit{true}) = 0.4$, $P(H|F = \textit{false}, G = \textit{false}) = 0.0$

4.2.3 Markov Random Fields

Definition 4.2.3. *A Markov random field (MRF) is an undirected graphical model where:*

- *Nodes represent random variables*
- *Edges represent symmetric dependencies*
- *The joint distribution is defined in terms of potential functions over cliques*

The joint probability distribution is given by:

$$P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(X_c)$$

where:

- $\mathcal{C}$ is the set of cliques (fully connected subgraphs)
- ψ_c is a potential function for clique c
- $Z = \sum_{x_1,\ldots,x_n} \prod_{c \in \mathcal{C}} \psi_c(X_c)$ is the partition function

4.2.4 Inference in Graphical Models

Inference tasks in graphical models include:

- Marginal inference: Computing $P(X_i)$
- Conditional inference: Computing $P(X_i|E = e)$ where E is evidence
- Maximum a posteriori (MAP) inference: Finding $\arg\max_X P(X|E = e)$

Exact inference is generally NP-hard, so approximate methods are often used:

- Sampling methods (e.g., Gibbs sampling, MCMC)
- Variational methods (e.g., mean field, belief propagation)
- Loopy belief propagation

4.2.5 Learning in Graphical Models

Learning in graphical models involves:

- Parameter learning: Estimating the conditional probability tables or potential functions
- Structure learning: Determining the graph structure (which variables are dependent)

Standard approaches include:

- Maximum likelihood estimation
- Bayesian approaches
- Structure search

4.3 Markov Logic Networks

4.3.1 Definition of Markov Logic Networks

Definition 4.3.1. *A Markov Logic Network (MLN) is a set of pairs* (F_i, w_i) *where:*

- F_i *is a formula in first-order logic*
- w_i *is a real-valued weight associated with* F_i

MLNs provide a way to soften first-order logic: formulas don't have to be strictly true or false, but violations are penalized by weights.

4.3.2 Probability Distribution Defined by MLNs

An MLN defines a probability distribution over possible worlds:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{m} w_i n_i(x)\right)$$

where:

- X is the set of all ground atoms
- x is a specific truth assignment (possible world)
- $n_i(x)$ is the number of true groundings of formula F_i in world x
- Z is the partition function

Example 4.3.2. *Consider an MLN with formulas:*

- $w_1 : \forall x, y.Friends(x, y) \rightarrow Friends(y, x)$ *(symmetric friendship)*
- $w_2 : \forall x, y.Friends(x, y) \rightarrow Likes(x, y)$ *(friends like each other)*
- $w_3 : \forall x, y.Likes(x, y) \wedge Likes(y, z) \rightarrow Likes(x, z)$ *(transitivity of "likes")*

The weights determine how strictly these rules are enforced. Higher weights mean stronger constraints.

4.3.3 Inference in MLNs

Inference in MLNs typically involves:

- Grounding the network to create a Markov network
- Performing inference on the ground network

Common inference tasks include:

- Marginal inference: $P(X = x|E = e)$ where E is evidence
- Most probable explanation (MPE): $\arg\max_x P(X = x|E = e)$

Inference algorithms for MLNs include:

- MCMC sampling
- Belief propagation
- MaxWalkSAT (for MPE inference)

4.3.4 Learning in MLNs

Learning in MLNs involves:

- Weight learning: Estimating weights for a fixed structure
- Structure learning: Learning both the formulas and their weights

Weight learning methods include:

- Maximum likelihood learning
- Maximum pseudo-likelihood learning
- Voted perceptron

Structure learning involves searching through the space of possible formulas, guided by scoring functions that balance fit to data and model complexity.

4.4 Probabilistic Logic Programming

4.4.1 Introduction to Probabilistic Logic Programming

Probabilistic Logic Programming (PLP) extends logic programming languages (like Prolog) with probabilistic elements. It allows for:

- Representing uncertainty in logical rules
- Defining probability distributions over logical atoms
- Performing probabilistic inference in structured domains

4.4.2 Distribution Semantics

Most PLP languages follow the distribution semantics, where:

- Some facts are probabilistic (called probabilistic facts)
- Each probabilistic fact is true with a certain probability, independently of other facts
- The truth values of the probabilistic facts induce a probability distribution over possible worlds
- The probability of a query is the sum of probabilities of worlds where the query is true

4.4.3 ProbLog

ProbLog is a probabilistic extension of Prolog that allows probabilistic facts:

```
0.3::edge(a,b).
0.7::edge(b,c).
0.9::edge(a,c).

path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

query(path(a,c)).
```

In this example:

- Edges exist with certain probabilities
- A path exists if there's a direct edge or a path through another node
- The query asks for the probability of a path from a to c

4.4.4 Inference in Probabilistic Logic Programming

Inference methods in PLP include:

- Exact inference: Often based on knowledge compilation (converting to d-DNNF)
- Approximate inference: Monte Carlo sampling, bounded approximations
- Lifted inference: Exploiting symmetries in the model

4.4.5 Learning in Probabilistic Logic Programming

Learning in PLP involves:

- Parameter learning: Estimating probabilities for probabilistic facts
- Structure learning: Learning the logical rules and their structure

Learning methods include:

- Maximum likelihood estimation
- Bayesian learning
- Structure search guided by score functions

4.5 Tensor Logic Networks

4.5.1 Introduction to Tensor Logic Networks

Tensor Logic Networks (TLNs) combine logical reasoning with neural networks through tensor representations. They aim to:

- Represent symbolic knowledge in a differentiable form
- Enable end-to-end learning of logical rules and neural representations
- Bridge symbolic and subsymbolic AI

4.5.2 Key Components of TLNs

TLNs typically consist of:

- **Symbol embeddings**: Vector/tensor representations of entities and predicates
- **Logical operators**: Differentiable implementations of logical connectives

- **Grounding mechanism**: Mapping logical expressions to tensor operations
- **Training procedure**: Learning embeddings and logical weights

4.5.3 Inference and Learning in TLNs

Inference in TLNs involves:

- Computing truth values of formulas through tensor operations
- Propagating gradients through logical operations
- Aggregating results across multiple groundings

Learning includes:

- Learning entity and predicate embeddings
- Learning weights for logical rules
- End-to-end optimization through gradient descent

Chapter 5

Neuro-Symbolic Integration

5.1 Introduction to Neuro-Symbolic AI

Neuro-symbolic AI aims to combine the strengths of neural networks and symbolic reasoning systems:

- Neural networks excel at pattern recognition, learning from data, and handling uncertainty
- Symbolic systems excel at explicit knowledge representation, interpretability, and logical reasoning

The goal is to create systems that can both learn from data and reason with explicit knowledge.

5.1.1 Complementary Strengths

Neural networks and symbolic systems have complementary strengths and weaknesses:

Dimension	**Neural Networks**	**Symbolic Systems**
Learning from data	Strong	Weak
Dealing with high-dimensional data	Strong	Weak
Robustness to exceptions	Strong	Weak
Data efficiency	Weak	Strong
Explainability	Weak	Strong
Common-sense knowledge	Weak	Strong
Complexity of inference	Strong	Weak

Table 5.1: Complementary strengths of neural and symbolic approaches

5.2 Approaches to Neuro-Symbolic Integration

5.2.1 Neural-Symbolic Cycle

The neural-symbolic cycle describes the interaction between neural and symbolic components:

- Symbols to neurons: Encoding symbolic knowledge in neural networks
- Neural learning: Learning from data using neural networks
- Neurons to symbols: Extracting symbolic knowledge from neural networks
- Symbolic reasoning: Performing logical inference with extracted knowledge

5.2.2 Integration Architectures

Various architectures have been proposed for neuro-symbolic integration:

- **Logic as constraints**: Using logical formulas to constrain neural network outputs
- **Differentiable logic**: Implementing logical operations in a differentiable manner
- **Neural-guided search**: Using neural networks to guide symbolic search
- **Hybrid architectures**: Combining neural and symbolic modules in a larger system

5.2.3 Examples of Neuro-Symbolic Systems

- **Neural Theorem Provers**: Neural networks that guide proof search
- **DeepProbLog**: Integrating neural networks with probabilistic logic programming
- **Logical Neural Networks**: Networks where neurons correspond to logical predicates
- **NeuralLP**: Learning logic programs through differentiable operations

5.3 Challenges and Future Directions

5.3.1 Current Challenges

- **Scalability**: Handling large symbolic knowledge bases and neural networks
- **Representation**: Developing shared representations for neural and symbolic components
- **Learning**: Effectively learning from both data and symbolic knowledge
- **Reasoning**: Maintaining sound reasoning while incorporating uncertainty
- **Explainability**: Making neuro-symbolic systems interpretable

5.3.2 Future Directions

- **Large-scale neuro-symbolic systems**: Scaling to real-world problems
- **Multimodal reasoning**: Integrating text, vision, and other modalities
- **Continual learning**: Updating knowledge over time
- **Neuro-symbolic architectures**: Developing new integration patterns
- **Applications**: Applying neuro-symbolic methods to practical problems