The Algebra of Intelligence

Sijie Wang

February 1, 2026

Preface

What is intelligence? This book approaches the question mathematically.

We take a specific stance: **intelligence is the ability to learn and then act**. This suggests a natural decomposition:

1. **Expression**: What can be described? What structures exist?
2. **Learning**: How do we find the right structure from data?
3. **Learning well**: Why are some structures easier to learn than others?
4. **Limits**: How far can intelligence go?

The mathematics we develop unifies modal logic, coalgebra, semirings, and optimization theory.

Key Insight

The central question of this book:

> *Given what we want to express (semantics), what is the best way to parameterize it (syntax) for learning?*

Contents

Part I

Expression

Chapter 1

Logic: A Historical Overview

Goals of This Chapter

- Understand why logic emerged in the first place
- See how logic evolved from "valid inference" to something much broader
- Trace the journey from Aristotle to the four pillars of modern logic

1.1 The Ancient World

1.1.1 What Logic Is (and Isn't)

In everyday speech, "logic" means something like "rational thinking." We say "that's not logical" when someone's argument doesn't make sense.

Formal logic is not this.

Formal logic is not *only* about being smart or avoiding fallacies—those may be corollaries, but they are not the core. The core is something much more specific:

> Logic is the study of **implication**—the "if... then..." structure.
>
> When can we say that one thing *necessarily follows* from another?

This is where logic *started*. But as we will see, the subject has traveled far from this origin—not abandoning it, but generalizing it beyond recognition.

1.1.2 Athens: Where It Began

Why would anyone study such an abstract thing?

The answer lies in ancient Athens, where democracy meant public debate. To win an argument, you needed to show that your conclusion **necessarily follows** from premises your opponent already accepts.

Historical Note

The word "logic" comes from the Greek *logos*, which means both "word" and "reason." For the Greeks, language and thought were intimately connected.

Definition 1.1 (Validity, informally)**.** An argument is **valid** if the conclusion necessarily follows from the premises. That is, *if* the premises are true, the conclusion *must* be true.

Validity is purely about the **structure** of the argument, not its content.

1.1.3 Plato's World of Forms

Before Aristotle systematized logic, his teacher Plato asked: **what is truth?**

Plato observed that the physical world is messy and changeable. Yet we have concepts of *perfect* circles, *ideal* chairs, *just* societies.

Historical Note

Plato's answer: there is a realm of **Forms**—eternal, unchanging, perfect archetypes. The physical world is just a shadow of this realm.

Intuition

Logical truths are not about this table or that chair. They are about the *form* of arguments themselves. When we prove "if all A are B, and all B are C, then all A are C," we are talking about **structure**, not any particular A, B, or C.
In a sense, logic lives in Plato's realm of Forms.

1.1.4 Aristotle's Syllogism

Plato's student Aristotle wanted to **systematize** valid reasoning—to give rules that anyone could follow to check if an argument is valid.

The result was the **syllogism**, the first formal system in history.

Example 1.2 (A syllogism)**.**

$$\frac{\begin{array}{ll}\text{All men are mortal.} & \text{(major premise)}\\ \text{Socrates is a man.} & \text{(minor premise)}\end{array}}{\text{Therefore, Socrates is mortal.} \quad \text{(conclusion)}}$$

The key insight: validity depends only on **form**, not content. We can replace the terms:

All A are B.
S is an A.

Therefore, S is B.

Aristotle catalogued all valid forms and showed how to reduce complex arguments to these basic patterns.

The Medieval Mnemonics

Medieval scholars labeled the four proposition types with vowels (A, E, I, O from *affirmo* and *nego*), then named each valid syllogism so its vowels encode the structure:

- **Barbara** (AAA): All M are P. All S are M. $\therefore$ All S are P.
- **Celarent** (EAE): No M are P. All S are M. $\therefore$ No S are P.

This is perhaps the first example of **encoding logic in notation**.

Key Insight

Aristotle's revolution: **form can be studied independently of content**.
This is the founding insight of all formal logic.

1.2 The Medieval Interlude

In 1781, Kant declared that logic "has not been able to advance a single step" since Aristotle.[1]

This was largely false.

1.2.1 The Medieval Golden Age

The historian J.M. Bocheński identified **three** golden periods of logic: ancient Greece, the medieval scholastic period, and the mathematical period of the 19th–20th centuries.[2]

Medieval logicians—William of Ockham, Jean Buridan, Walter Burley—developed sophisticated systems beyond Aristotle:

- **Supposition theory**: A proto-semantics distinguishing how terms refer in different contexts—remarkably close to modern use/mention distinctions.

[1] Kant, *Critique of Pure Reason*, Bviii.
[2] See Catarina Dutilh Novaes, "The Rise and Fall and Rise of Logic," *Aeon* (2017).

- **Consequentiae**: A theory of logical consequence beyond the syllogism.
- **Insolubilia**: The study of semantic paradoxes like the Liar, centuries before Russell and Tarski.
- **Obligationes**: Formalized disputation games—an early dialogue logic.

1.2.2 The Humanist Catastrophe

Around 1530, Renaissance humanists swept away the scholastic tradition. They found the "barbarous language and twisted Latin of the scholastics" distasteful, preferring Cicero's elegance to Ockham's precision.

> "After about 1530 not only did new writing on the specifically medieval contributions to logic cease, but the publication of medieval logicians virtually ceased."

The medieval advances were not refuted—they were simply *forgotten.* It was only in the late 20th century that historians rediscovered what had been lost.

1.2.3 What Actually Changed

So why didn't medieval logic break through to the modern form?

Medieval logic served philosophy and theology. Its paradigm cases were "All men are mortal" and "God is good"—these fit the subject-predicate structure of syllogisms.

But in the 17th century, a new customer appeared: **mathematics**. And mathematics needed to say things syllogisms could not express:

- "For every $\epsilon > 0$, there exists a $\delta > 0$ such that..." (limits)
- "There exists a unique x such that $f(x) = 0$" (existence and uniqueness)
- "For all n, if $P(n)$ then $P(n + 1)$" (induction)

These involve **nested quantifiers**—alternations of "for all" and "there exists." Aristotle's "All A are B" cannot express "for every ϵ there exists a δ."

The real revolution came when mathematicians needed to formalize their own reasoning.

1.3 The Birth of Modern Logic

1.3.1 Leibniz's Dream

In the 17th century, Leibniz had a vision:

Historical Note

Leibniz dreamed of a *characteristica universalis*: a universal symbolic language for all human thought. And a *calculus ratiocinator*: a mechanical method to compute truth.
"When there are disputes among persons, we can simply say: let us calculate, and see who is right."

Leibniz never achieved it. But his dream planted a seed: **reasoning as calculation**.

1.3.2 Boole's Algebra

Two centuries later, George Boole took the first real step.

The Laws of Thought, 1854

Boole realized that AND, OR, NOT could be treated as algebraic operations. Propositions became variables, connectives became operations, and logical reasoning became solving equations.

Example 1.3 (Boolean algebra)**.** Let p = "it is raining" and q = "the ground is wet."

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	T
F	F	T	F	F	T

Boole showed these satisfy algebraic laws: $p \wedge q = q \wedge p$, $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$, etc.

But Boole's algebra could only express **propositional** logic. "All men are mortal" is just a single symbol p—the internal structure is invisible.

1.3.3 Frege's Revolution

The real transformation came in 1879, when Gottlob Frege published *Begriffsschrift* (Concept-Script).

Frege wanted to derive mathematics from pure logic. To do this, he introduced:

Key Insight

1. **Quantifiers**: $\forall x$ ("for all x") and $\exists x$ ("there exists x")
2. **Predicates and relations**: $P(x)$, $R(x, y)$ instead of "S is P"
3. **Nested structure**: $\forall x \exists y\, R(x, y)$ differs from $\exists y \forall x\, R(x, y)$

Intuition

$\forall$ and $\exists$ are the **minimal pair**, corresponding to Aristotle's "All" and "Some." They are duals:

$$\forall x\, \varphi \;\equiv\; \neg \exists x\, \neg \varphi$$

Later, modal logic will use the same pattern: $\Box$ = "in all worlds," $\Diamond$ = "in some world." This is not a coincidence.

This is **first-order logic**. Frege's original system was actually higher-order (quantifying over predicates), which led to Russell's paradox.

The Semantics of First-Order Logic

Propositional logic has truth tables. What is the analogous "truth table" for first-order logic?

The answer is a **structure** (or **model**):

Definition 1.4 (Structure)**.** A structure $\mathcal{M}$ consists of:

1. A non-empty set D, the **domain**
2. For each constant c, an element $c^{\mathcal{M}} \in D$
3. For each predicate P, a subset $P^{\mathcal{M}} \subseteq D^n$
4. For each function f, a function $f^{\mathcal{M}} : D^n \to D$

Example 1.5 (Arithmetic)**.** Structure $\mathcal{N}$: domain $D = \{0, 1, 2, \ldots\}$, $0^{\mathcal{N}} = 0$, $s^{\mathcal{N}}(n) = n + 1$, $<^{\mathcal{N}} = \{(m, n) \mid m < n\}$.

In $\mathcal{N}$, the formula $\forall x\, (x < s(x))$ is **true**.

But in a different structure with $D = \{a, b\}$ and $s(b) = a$, the same formula is **false**.

The key insight: **the same formula can be true in one structure and false in another**. Truth is always relative to a structure.

The formal satisfaction relation $\mathcal{M} \models \varphi$ is defined recursively:

$$
\begin{array}{ll}
\mathcal{M} \models P(t_1, \ldots, t_n) & \text{iff } (t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}) \in P^{\mathcal{M}} \\
\mathcal{M} \models \neg\varphi & \text{iff } \mathcal{M} \not\models \varphi \\
\mathcal{M} \models \varphi \wedge \psi & \text{iff } \mathcal{M} \models \varphi \text{ and } \mathcal{M} \models \psi \\
\mathcal{M} \models \forall x\, \varphi & \text{iff for } \textbf{every } d \in D\text{: } \mathcal{M} \models \varphi[x \mapsto d]
\end{array}
$$

1.4 Syntax and Semantics

1.4.1 The Two Turnstiles: $\vdash$ and $\models$

Two symbols are ubiquitous in logic—and often confused. Their history reveals a deep tension at the heart of the subject.

Frege's Turnstile: $\vdash$

Jena, 1879

Gottlob Frege was a mathematics professor with an ambitious dream: to prove that arithmetic is pure logic. No appeals to intuition, no hand-waving—just rigorous derivation from logical axioms.
But how do you write such a proof? Natural language is too vague. Existing mathematical notation is too informal. Frege needed a **perfect notation**—one where every step is explicit and mechanical.
The result was his *Begriffsschrift* ("concept-script"), a formal language for pure thought.

Frege introduced a special symbol to mark **assertion**—the claim that something has been established:

judgment stroke (vertical) $\searrow$

$\vdash \longleftarrow$ content stroke (horizontal)

The **content stroke** "—" indicates that what follows is a proposition (something that can be true or false). The **judgment stroke** "|" asserts that this proposition *has been judged true*—that we have established it.

So $\vdash \varphi$ meant: "φ is hereby asserted." Not "φ is true" (a metaphysical claim), but "φ has been derived" (an epistemological claim).

Intuition

Think of the judgment stroke as a **stamp of approval**. When Frege writes $\vdash \varphi$, he is saying: "I have checked this. The derivation is complete. You may rely on φ."

This is fundamentally a **syntactic** notion—it's about what can be written down following the rules, not about what is "really true" in some Platonic sense.

Tarski's Turnstile: $\models$

Warsaw, 1933

Half a century later, Alfred Tarski faced a different problem: the **concept of truth itself**.

The ancient Liar Paradox ("This sentence is false") had shown that naive talk about truth leads to contradiction. Tarski wanted a *mathematically precise* definition of what it means for a sentence to be true.

His insight: truth is always **relative to an interpretation**. The sentence "Snow is white" is true not absolutely, but *in a particular interpretation* where "snow" refers to snow and "white" means white.

Tarski's semantic definition of truth:

> "Snow is white" is true *if and only if* snow is white.

This looks circular, but it's not. The quoted sentence is a **syntactic object** (a string of symbols). The unquoted part describes the **world**. The definition connects syntax to semantics.

For formal languages, Tarski defined a **satisfaction relation**. But first: what is a **structure**?

Definition 1.6 (Structure)**.** When a mathematician says "a structure," they mean a **domain** together with **interpretations** for all the symbols in the language.

A structure $\mathcal{M}$ for a first-order language consists of:

- A non-empty set D called the **domain** (the "universe" of objects we're talking about)
- For each constant symbol c: an element $c^{\mathcal{M}} \in D$
- For each n-ary function symbol f: a function $f^{\mathcal{M}} : D^n \to D$
- For each n-ary relation symbol R: a subset $R^{\mathcal{M}} \subseteq D^n$

Example 1.7 (A concrete structure)**.** Consider the language with: constant 0, unary function s ("successor"), binary relation $<$.

One structure $\mathcal{N}$ (the natural numbers):

- Domain: $D = \{0, 1, 2, 3, \ldots\}$
- $0^{\mathcal{N}} = 0$ (the number zero)
- $s^{\mathcal{N}}(n) = n + 1$ (successor adds one)
- $<^{\mathcal{N}} = \{(0, 1), (0, 2), (1, 2), \ldots\}$ (the usual ordering)

A different structure $\mathcal{Z}$ using the same language:

- Domain: $D = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (all integers)
- $0^{\mathcal{Z}} = 0$
- $s^{\mathcal{Z}}(n) = n + 1$
- $<^{\mathcal{Z}}$ = the usual ordering on integers

Same symbols, different meanings. The sentence $\forall x\,(0 < s(x))$ is true in $\mathcal{N}$ but false in $\mathcal{Z}$ (consider $x = -1$).

Intuition

A structure is like a **dictionary** that tells you what each symbol *refers to*. The syntax gives you sentences; the structure gives them meaning.
Different structures can make the same sentence true or false—just as "The president is tall" has different truth values depending on which country and time you're talking about.

Given a structure $\mathcal{M}$, we can rigorously define when $\mathcal{M}$ *satisfies* a formula φ—written $\mathcal{M} \models \varphi$.

The symbol $\models$ captures this:

- $\mathcal{M} \models \varphi$ means "structure $\mathcal{M}$ satisfies φ" (the formula is true in this interpretation)
- $\models \varphi$ means "every structure satisfies φ" (the formula is a logical truth)
- $\Gamma \models \varphi$ means "every structure that satisfies all of Γ also satisfies φ"

Intuition

Where $\vdash$ asks "can we derive this by symbol manipulation?", the symbol $\models$ asks "is this true in all possible worlds?"
$\vdash$ is about **proof**. $\models$ is about **truth**.

The Modern Reading

Today, we use both symbols with refined meanings:

Definition 1.8 (The syntactic turnstile $\vdash$)**.** $\Gamma \vdash \varphi$ means "φ is **derivable** from Γ using the proof rules."

Read as: "Γ **proves** φ."

Definition 1.9 (The semantic turnstile $\models$)**.** $\Gamma \models \varphi$ means "in every model where all of Γ is true, φ is also true."

Read as: "Γ **entails** φ."

Key Insight

Syntax ($\vdash$)	**Semantics** ($\models$)
Proof, derivation	Truth, models
Symbol manipulation	Meaning, interpretation
Finite (a proof is finite)	Potentially infinite (all models)

The great theorems connect these worlds:

- **Soundness**: $\vdash \varphi \Rightarrow \models \varphi$ (proofs don't lie)
- **Completeness**: $\models \varphi \Rightarrow \vdash \varphi$ (all truths are provable)

1.4.2 Why Two Notions?

Why do we need *two* ways of saying "φ follows"?

The distinction emerged from the great foundational debates of the early 20th century:

The Three Schools of Mathematical Philosophy

Logicism (Frege, Russell): Mathematics is reducible to logic. Mathematical truths are logical truths. This view privileges *semantics*: $\models$ captures objective truth, and $\vdash$ is our attempt to systematize it.
Formalism (Hilbert): Mathematics is a formal game with symbols. Axioms are strings; proofs are sequences following syntactic rules. Meaning is irrelevant—what matters is consistency. This view privileges *syntax*: $\vdash$ is primary, and $\models$ is a useful heuristic.
Intuitionism (Brouwer): Mathematics is mental construction. A statement is true only if we can *construct* a proof. The law of excluded middle is rejected. Neither $\vdash$ nor $\models$ is primary—only *construction* matters.

Gödel's incompleteness theorems (1931) devastated the formalist program and complicated the logicist program. But the syntax-semantics distinction survived and became foundational.

What Happened to the Three Schools?

Logicism evolved into modern mathematical logic and set theory. While the original dream of reducing all mathematics to logic failed (Gödel), the logicist spirit lives on in foundations of mathematics and type theory.

Formalism transformed into proof theory and theoretical computer science. Hilbert's program failed in its original form, but the study of formal systems flourished. Today, automated theorem provers and proof assistants (Coq, Lean) are direct descendants.

Intuitionism gave birth to constructive mathematics and deeply influenced computer science. The Curry-Howard correspondence—"proofs are programs"—is an intuitionist insight. Martin-Löf type theory and dependently typed programming languages carry forward Brouwer's vision.

Intuition

The three schools didn't "win" or "lose." They revealed that *different questions require different foundations*:

- Want to know what's true? $\models$ (semantic/logicist)
- Want to verify a proof mechanically? $\vdash$ (syntactic/formalist)
- Want to compute a witness? Construction (intuitionist)

Modern logic uses all three perspectives.

1.4.3 Syntax Meets Semantics: Examples

Example 1.10 (A propositional proof)**.** English: "If it rains, the ground gets wet. If the ground gets wet, the game is cancelled. It's raining. Therefore, the game is cancelled."

Translation: Let r = raining, w = wet, c = cancelled.

$$\begin{aligned} \text{Premises:} &\quad r \to w, \quad w \to c, \quad r \\ \text{Conclusion:} &\quad c \end{aligned}$$

Syntactic proof:

1. $r \to w$ (premise)
2. $w \to c$ (premise)
3. r (premise)
4. w (modus ponens: 1, 3)
5. c (modus ponens: 2, 4)

Semantic verification: We check whether the formula $(r \to w) \wedge (w \to c) \wedge r \to c$ is a tautology:

r	w	c	$r \to w$	$w \to c$	Premises	c	Result
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	F	T	T
T	F	F	F	T	F	F	T
F	T	T	T	T	F	T	T
F	T	F	T	F	F	F	T
F	F	T	T	T	F	T	T
F	F	F	T	T	F	F	T

where "Premises" $= (r \to w) \wedge (w \to c) \wedge r$. The only row where all premises are true (row 1) has $c = \text{T}$. So Premises $\to c$ is T in all 8 rows—it's a **tautology**.

Both routes confirm validity.

Example 1.11 (A first-order proof)**.** English: "Every student respects every professor. Alice is a student. Bob is a professor. Therefore, Alice respects Bob."

Translation:

$$\begin{aligned} \text{Premise 1:} \quad & \forall x \forall y \, (S(x) \wedge P(y) \to R(x,y)) \\ \text{Premises 2, 3:} \quad & S(a), \quad P(b) \\ \text{Conclusion:} \quad & R(a,b) \end{aligned}$$

Syntactic proof:

1. $\forall x \forall y \, (S(x) \wedge P(y) \to R(x,y))$ (premise)
2. $S(a)$ (premise)
3. $P(b)$ (premise)
4. $\forall y \, (S(a) \wedge P(y) \to R(a,y))$ ($\forall$-elimination on 1, $x \mapsto a$)
5. $S(a) \wedge P(b) \to R(a,b)$ ($\forall$-elimination on 4, $y \mapsto b$)

6. $S(a) \wedge P(b)$ ($\wedge$-introduction on 2, 3)
7. $R(a, b)$ (modus ponens on 5, 6)

Semantic verification: We construct a model and check that the premises entail the conclusion.

Define structure $\mathcal{M}$:

- Domain: $D = \{\text{Alice}, \text{Bob}, \text{Carol}, \text{Dan}\}$
- Interpretation of constants: $a^{\mathcal{M}} = \text{Alice}$, $b^{\mathcal{M}} = \text{Bob}$
- $S^{\mathcal{M}} = \{\text{Alice}, \text{Carol}\}$ (students)
- $P^{\mathcal{M}} = \{\text{Bob}, \text{Dan}\}$ (professors)
- $R^{\mathcal{M}} = \{(\text{Alice}, \text{Bob}), (\text{Alice}, \text{Dan}), (\text{Carol}, \text{Bob}), (\text{Carol}, \text{Dan})\}$

Now verify each premise:

- $\mathcal{M} \models \forall x \forall y\, (S(x) \wedge P(y) \to R(x, y))$?
  For every $d_1, d_2 \in D$: if $d_1 \in S^{\mathcal{M}}$ and $d_2 \in P^{\mathcal{M}}$, then $(d_1, d_2) \in R^{\mathcal{M}}$.
  Check: Alice-Bob ✓, Alice-Dan ✓, Carol-Bob ✓, Carol-Dan ✓. **True.**
- $\mathcal{M} \models S(a)$? Alice $\in$ {Alice, Carol}. **True.**
- $\mathcal{M} \models P(b)$? Bob $\in$ {Bob, Dan}. **True.**

Conclusion: $\mathcal{M} \models R(a, b)$? Is $(\text{Alice}, \text{Bob}) \in R^{\mathcal{M}}$? **Yes.**

The semantic route confirms: whenever all premises are true, so is the conclusion.

1.5 The Foundational Crisis

Frege's dream was to derive mathematics from logic. In 1903, just as his *Grundgesetze* was going to press, Russell found a contradiction.

Russell's Paradox

Consider the set $R = \{x \mid x \notin x\}$—all sets that don't contain themselves.

Does R contain itself?

- If $R \in R$, then by definition $R \notin R$. Contradiction.
- If $R \notin R$, then by definition $R \in R$. Contradiction.

Frege's system was inconsistent. Russell and Whitehead spent a decade fixing it in *Principia Mathematica* (1910–1913), introducing type theory.

Intuition

The crisis revealed: logic is powerful but **dangerous**. The same expressive power that lets you talk about "all sets" also lets you construct paradoxes.

1.6 Properties of Logical Systems

The foundational crisis forced a question: **what makes a logical system trustworthy?**

Before Russell's paradox, mathematicians assumed their reasoning was sound. Frege spent decades building his logical foundation for arithmetic, confident it was rock-solid. Then one letter from Russell demolished everything.

After the crisis, nothing could be taken for granted. Logicians needed **explicit criteria**—properties that a logical system must have (or should have) to be reliable. These properties became the vocabulary for evaluating any formal system.

1.6.1 Non-Negotiable Properties

Soundness: Proofs Don't Lie

Frege's Life Work, Destroyed

Gottlob Frege began his project in 1879 with the *Begriffsschrift*. His goal: prove that arithmetic is reducible to pure logic. No intuition, no hand-waving—just rigorous derivation from logical axioms.

For over twenty years, Frege labored in obscurity. Most mathematicians ignored his strange notation. But Frege persisted, building an elaborate formal system. In 1893, he published the first volume of *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic). The second volume was at the printer in 1902.

Then came the letter.

Bertrand Russell, a young philosopher who was one of the few people actually reading Frege, had discovered something terrible. He wrote to Frege on June 16, 1902:

> "I find myself in complete agreement with you in all essentials... There is just one point where I have encountered a difficulty."

The "difficulty" was this: Frege's Axiom V allowed forming the set of all sets that satisfy any property. Russell asked: what about the set $R = \{x \mid x \notin x\}$—the set of all sets that don't contain themselves?

- If $R \in R$, then by definition of R, we have $R \notin R$. Contradiction.
- If $R \notin R$, then by definition of R, we have $R \in R$. Contradiction.

Frege's system could *construct* this set R. Therefore his system could prove both $R \in R$ and $R \notin R$. A contradiction.

Frege's reply, written just days later, is one of the most poignant in the history of mathematics:

> "Your discovery of the contradiction caused me the greatest surprise and, I would almost say, consternation, since it has shaken the basis on which I intended to build arithmetic... It is all the more serious since... not only the foundations of my arithmetic, but also the sole possible foundations of arithmetic, seem to vanish."

The second volume of *Grundgesetze* was already being printed. Frege hastily added an appendix admitting the flaw. He never recovered intellectually; he published little afterward and died in obscurity.

But why does one contradiction destroy *everything*? Because of *ex falso quodlibet*: from a contradiction, you can derive any statement. Once Frege's system proved $R \in R$ and $R \notin R$, it could also prove:

- $0 = 1$
- $2 + 2 = 5$
- "The moon is made of green cheese"

Every "theorem" in the system was worthless—not because they were all false, but because the system could no longer distinguish true from false.

The problem is now clear: **how do we know our proof rules are trustworthy?** Just because we can derive φ doesn't mean φ is actually true. Frege *thought* his Axiom V was obviously correct. It wasn't.

Intuition

Think of a proof system as a **machine** that stamps "APPROVED" on formulas. Soundness checks whether this machine ever approves a false statement—by asking: does $\vdash \varphi$ always imply $\models \varphi$?
If not, the machine is broken. You can never trust its output.

Definition 1.12 (Soundness)**.** A proof system is **sound** if everything provable is true:

$$\vdash \varphi \Rightarrow \models \varphi$$

"If we can derive it, it's actually valid."

Soundness is proved by **induction on proof length**: show that axioms are valid, and that each inference rule preserves validity. If both hold, every derivable formula is valid.

Now we can return to Frege's disaster. What went wrong? His Axiom V seemed obviously true, but it wasn't—it allowed constructing Russell's paradoxical set. In modern terms: **Frege's system was unsound**. It could derive $R \in R$, but $R \in R$ is not true (in fact, it's neither true nor false—it's meaningless, because R doesn't exist as a well-defined set).

With the concept of soundness in hand, we know exactly what to check: before trusting any proof system, *prove that its axioms are valid and its rules preserve validity.* If we can establish soundness, then the nightmare scenario cannot happen—the system will never approve "$0 = 1$" or "the moon is made of green cheese," because those statements are false, and a sound system only approves truths.

Common Pitfall

Soundness is the **absolute minimum**. An unsound system is not merely incomplete or inconvenient—it is **broken**. Frege's system was unsound, which is why it could "prove" anything at all.

Consistency: No Contradictions

Ex Falso Quodlibet

Medieval logicians discovered a terrifying principle: from a contradiction, *anything* follows.
Suppose you can prove both φ and $\neg\varphi$. Then:

1. φ (assumption)
2. $\varphi \lor \psi$ (from 1, by $\lor$-introduction)
3. $\neg\varphi$ (assumption)
4. ψ (from 2 and 3, by disjunctive syllogism)

The formula ψ was arbitrary. So from a contradiction, you can derive *any* statement whatsoever.

This is why consistency matters: an inconsistent system proves everything, which means it proves nothing meaningful.

Definition 1.13 (Consistency)**.** A system is **consistent** if there is no formula φ such that both $\vdash \varphi$ and $\vdash \neg\varphi$.

Equivalently: a system is consistent if it does *not* prove $\bot$ (falsehood).

Intuition

Consistency is the "sanity check." A system that contradicts itself has lost all connection to truth—it's just symbol-shuffling that proves anything you want.

Russell's paradox showed that Frege's system was inconsistent. From the contradiction, Frege could derive $0 = 1$, $2 + 2 = 5$, and "the moon is made of green cheese." The system was worthless.

1.6.2 Desirable Properties

Completeness: No Truths Left Behind

Soundness says proofs don't lie. But there's a converse question: **are all truths provable?**

Hilbert's Program

David Hilbert, the most influential mathematician of his era, wanted to secure mathematics forever. His program (1920s): formalize all of mathematics, prove it consistent, and show every statement is decidable. In 1928 he declared: "We must know. We will know."

Hilbert's vision emerged from the chaos after Russell's paradox. If naive reasoning could hide contradictions, perhaps *all* of mathematics was suspect. Hilbert's solution: formalize everything, then prove the formalization is safe.

The program had three goals:

1. Formalize all of mathematics in a precise symbolic system
2. Prove, using only "finitary" methods, that this system is consistent
3. Show that every mathematical statement can be either proved or disproved

If successful, mathematics would be **complete** and **provably consistent**. Human intuition could be replaced by mechanical symbol manipulation.

Enter Kurt Gödel, a 23-year-old Austrian logician.

In 1929, Gödel proved the **completeness theorem**: first-order logic is complete. Every valid formula has a proof. Progress!

But in 1931, Gödel published his **incompleteness theorems**, which demolished the program:

- **First Incompleteness Theorem**: Any consistent formal system capable of expressing basic arithmetic contains statements that are true but unprovable within the system.
- **Second Incompleteness Theorem**: Such a system cannot prove its own consistency (unless it is actually inconsistent).

Gödel's proof was devastatingly clever. He constructed a sentence G that essentially says: "This sentence is not provable."

- If G is provable, then what it says is false, so the system proves a falsehood—the system is unsound.
- If G is not provable, then what it says is true—a true but unprovable statement exists.

Either way, the system is incomplete or inconsistent. Hilbert's dream was dead.

Definition 1.14 (Completeness)**.** A proof system is **complete** if every valid formula is provable:

$$\models \varphi \Rightarrow \vdash \varphi$$

"If it's true in all models, we can derive it."

Together with soundness, completeness gives the ideal correspondence:

$$\vdash \varphi \Leftrightarrow \models \varphi$$

Syntax and semantics coincide perfectly.

Intuition

Soundness: "the proof system doesn't make mistakes."
Completeness: "the proof system doesn't miss anything."
First-order logic has both. Arithmetic (and anything stronger) has soundness but not completeness—there are truths it cannot prove.

Now return to Hilbert's question: can every mathematical statement be proved or disproved?

For pure first-order logic, the answer is **yes**—that's Gödel's completeness theorem. If φ is a logical truth (true in all structures), there's a proof of φ. If φ is not a logical truth, there's a counterexample.

But Hilbert wanted more: completeness for *arithmetic*, not just logic. Here Gödel's answer is **no**. The sentence G ("I am not provable") is true but unprovable. Hilbert's dream of deciding every arithmetic statement was impossible—not because we haven't been clever enough, but because *no consistent formal system can do it.*

What does this mean in practice? It means mathematics is inexhaustible. No matter how many axioms you add, there will always be truths beyond your reach. Far from being a defeat, this is liberating: mathematics can never be reduced to a finished, mechanical procedure. There will always be new theorems to discover, new methods to invent.

Decidability: Can Machines Check?

The Entscheidungsproblem

Hilbert and Ackermann (1928) posed the "decision problem": is there an algorithm that takes any first-order formula and correctly determines whether it's valid? This was the last hope for Hilbert's program—even if not all truths are provable, perhaps validity is still mechanically checkable.

For propositional logic, the answer is yes: truth tables. You can mechanically check all 2^n rows.

But first-order logic is different. In 1936, two mathematicians independently proved that no such algorithm can exist.

Alonzo Church at Princeton published first (April 1936). He used his λ-calculus to define "computable function," then showed that validity in first-order logic is not computable.

Alan Turing, a 23-year-old at Cambridge, had been working on the same problem without knowing of Church's work. He invented the Turing machine—an abstract model of mechanical computation—and proved the same result. His paper introduced the famous "halting problem": no algorithm can determine whether an arbitrary program will halt or run forever.

When Turing learned of Church's result (after completing his own paper), he went to Princeton to study under Church, who became his doctoral advisor. They proved that λ-calculus and Turing machines are equivalent in power. The **Church-Turing thesis** emerged: any "reasonable" notion of computation is equivalent to Turing machines (or λ-calculus, or any of several other equivalent formalisms).

The Entscheidungsproblem is unsolvable. There is no algorithm that can always tell you whether a first-order formula is valid. You can search for a proof (and you'll find one if it exists, by completeness), but if the formula is invalid, you might search forever.

Definition 1.15 (Decidability)**.** A logic is **decidable** if there exists an algorithm that:

- Takes any formula φ as input
- Always terminates (doesn't run forever)
- Correctly outputs "valid" or "not valid"

Return to Hilbert's question: can a machine check validity?

The answer splits in two. For **propositional logic**: yes. Truth tables give a mechanical procedure—check all 2^n rows, and you'll know whether φ is valid. Slow, but guaranteed to terminate with the correct answer.

For **first-order logic**: no. Church and Turing showed that no algorithm can do this. You can search for a proof, and if φ is valid, you'll eventually find one (by completeness). But if φ is *not* valid, you might search forever without knowing whether to keep looking or give up.

This is Hilbert's Entscheidungsproblem, answered in the negative. But the negative answer required defining precisely what an "algorithm" is—and that definition (Turing machines, λ-calculus) became the foundation of computer science.

Completeness vs. Decidability: A Crucial Distinction

These two properties are easy to confuse. Let's be precise:

Completeness	If φ is valid, then φ has a proof. (Every truth is provable.)
Decidability	There exists an algorithm that, given any φ, correctly determines whether φ is valid. (A machine can always tell you yes or no.)

Two natural questions arise:

Q1: Completeness $\Rightarrow$ Decidability?

"If a proof exists (completeness), why can't I just search and find it? Doesn't that give me an algorithm (decidability)?"

Q2: Decidability $\Rightarrow$ Completeness?

"If I have an algorithm that decides everything (decidability), doesn't that cover all truths (completeness)?"

Both questions have subtle answers.

A1: Completeness gives Semi-Decidability, not Decidability

Yes, you can search for proofs. Enumerate all possible proofs: proof 1, proof 2, proof 3, ... If φ is valid, you'll eventually find a proof. But here's the catch: *what if φ is invalid?* Then no proof exists. You'll search forever—proof 1 (no), proof 2 (no), proof 3 (no)—never knowing whether to keep looking or give up.

The search procedure only works in one direction: it can *confirm* validity (by finding a proof) but cannot *refute* it (you can't search through all possible counterexamples). This is called **semi-decidability**: valid formulas can be recognized, but invalid ones cannot.

A2: Decidability is about the Logic, Completeness is about a Proof System

You're right that if an algorithm says "valid," you *know* it's valid. But completeness asks a different question: can a *specific proof system* derive all valid formulas?
Here's the distinction:

- **Decidability** is a property of the *logic itself*. It asks: can we determine validity?
- **Completeness** is a property of a *particular proof system*. It asks: can *this* proof system derive all valid formulas?

A logic can be decidable while a particular proof system for it is incomplete.

Let's see a concrete example.
The logic: Propositional logic.
The decision algorithm: Truth tables. Given any formula φ, list all possible truth assignments to its variables. If φ is true under every assignment, output "valid." Otherwise, output "not valid."
For example, to check $p \to p$:

p	$p \to p$
T	T
F	T

All rows give T, so the algorithm outputs "valid." This works for any formula—propositional logic is decidable.
The (weak) proof system: Only one rule—modus ponens. No axioms.

Modus ponens: From A and $A \to B$, derive B.

Can this system prove $p \to p$?
No! Modus ponens lets you derive new formulas from old ones. But with no axioms, you have no starting point. You can't derive anything at all, let alone $p \to p$.
So here's the situation:

- The algorithm says: "$p \to p$ is valid" (by truth table).
- The proof system says: "I can't derive $p \to p$" (no axioms to start from).

The logic is decidable. This particular proof system is incomplete.
Why not use the algorithm as the proof system? You absolutely can! Define "proof of φ" to mean "the truth table has all T's." Then every valid formula has a proof, and your system is complete.

This is a key insight: **if a logic is decidable, you can always construct a complete proof system**—just define "proof" as "algorithm says valid." So:

$$\text{Decidable} \implies \text{Can have a complete proof system}$$

But notice: the moment you switch from "modus ponens only" to "truth tables," you've changed proof systems. You're no longer discussing the original system—you've replaced it with a different one.

Intuition

This reveals something important: **choosing a proof system is choosing an algorithm**. Modus ponens is not just a "rule"—it's an **operation**. Given inputs A and $A \to B$, it produces output B. A proof system is defined by which operations you allow.

When we ask "why not use truth tables?"—we're really asking: which operations should count as valid proof steps? The answer determines what "proof" means.

This connection—**proofs as computations, rules as operations**—runs deep. We'll return to it in proof theory, where the Curry-Howard correspondence reveals that proofs literally *are* programs.

Does completeness imply decidability?

If every valid formula has a proof (completeness), can't I just search through all proofs until I find one? Doesn't that decide validity?

No: Completeness gives Semi-Decidability, not Decidability

First-order logic is the classic counterexample:

- **Complete**: Yes (Gödel 1929). Every valid formula has a proof.
- **Decidable**: No (Church-Turing 1936). No algorithm can always determine validity.

Here's what goes wrong. You want to know if φ is valid:

1. You enumerate all possible proofs: proof 1, proof 2, proof 3, ...
2. If φ is valid, by completeness, a proof exists. Eventually you find it. **Success!**
3. But if φ is *invalid*, no proof exists. You search forever: proof 1 (no), proof 2 (no), ... **You never know when to stop.**

Completeness guarantees the proof is out there *if φ is valid*. But it doesn't help when φ is invalid—you can't distinguish "haven't found it yet" from "doesn't exist."
This is **semi-decidability**: you can confirm validity (by finding a proof) but cannot refute it (you'd search forever).

Does decidability imply completeness?

If I have an algorithm that decides validity, can't I just use "algorithm says valid" as the definition of "has a proof"? Then every valid formula has a proof!

Yes, but Completeness is about a Specific Proof System

You're right! If the logic is decidable, you *can* construct a complete proof system this way.
But completeness is a property of a *particular* proof system, not the logic itself. The same decidable logic can have:

- **System A** (algorithm-as-proof): Complete
- **System B** (modus ponens only): Incomplete

When we ask "is this proof system complete?" we're asking about *that specific system*, not whether *some* complete system exists.
The real lesson: decidable logic $\Rightarrow$ complete proof system *exists*. But a given proof system might still be incomplete.

The precise relationships:

$$\begin{aligned} \text{Decidable} &\Longrightarrow \text{Can construct a complete proof system} \\ \text{Complete proof system} &\Longrightarrow \text{Semi-decidable} \\ \text{Complete proof system} &\not\Longrightarrow \text{Decidable} \end{aligned}$$

The four combinations:

	Complete	Incomplete
Decidable	Propositional logic	Weak subsystems
Undecidable	First-order logic	Peano arithmetic

- **Propositional logic**: Complete (truth tables prove everything valid) and decidable (truth tables are an algorithm).
- **First-order logic**: Complete (Gödel) but undecidable (Church-Turing). You can find proofs of valid formulas, but can't always detect invalidity.
- **Peano arithmetic**: Incomplete (Gödel's incompleteness) and undecidable. Some true statements have no proof, and no algorithm can determine truth.

Key Insight

Property	Meaning	Without it...
Soundness	Proofs $\Rightarrow$ truth	Proofs are meaningless
Consistency	No contradictions	Everything is provable
Completeness	Truth $\Rightarrow$ proofs	Some truths have no proofs
Decidability	Algorithm exists	No machine can always check

Soundness and consistency are **non-negotiable**—a system without them is useless. Completeness and decidability are **desirable but sometimes impossible**.

1.6.3 Structural Properties

Beyond the basic four, several deeper properties reveal the structure of logical systems.

Compactness: Infinity Has No Surprises

Here is a puzzle. Suppose you have infinitely many sentences: $\varphi_1, \varphi_2, \varphi_3, \ldots$ You check: $\{\varphi_1\}$ has a model. $\{\varphi_1, \varphi_2\}$ has a model. $\{\varphi_1, \varphi_2, \varphi_3\}$ has a model. Every finite subset has a model.

Question: Does the infinite set $\{\varphi_1, \varphi_2, \varphi_3, \ldots\}$ have a model?

You might think: "Not necessarily! Maybe the sentences gradually 'squeeze' tighter and tighter, and in the limit there's no room left. Like the intersection $[0,1] \cap [0,1/2] \cap [0,1/3] \cap \cdots = \{0\}$—each finite intersection is a whole interval, but the infinite intersection is just a point."

This intuition is reasonable. But for first-order logic, it's *wrong*.

Gödel's Compactness Theorem, 1930

While proving his completeness theorem, Gödel discovered a surprising byproduct: in first-order logic, infinity has no surprises. If every finite piece is satisfiable, the whole thing is satisfiable. There's no way to "sneak up" on a contradiction through infinity.

Why is this true? The key insight is that **proofs are finite**.

Suppose the infinite set $\Gamma = \{\varphi_1, \varphi_2, \ldots\}$ has no model. Then Γ is inconsistent—we can derive a contradiction from it. But a derivation is a finite sequence of steps, using only finitely many premises. So some *finite* subset of Γ already derives a contradiction. That finite subset has no model.

Contrapositive: if every finite subset has a model, then the infinite set has a model.

Definition 1.16 (Compactness)**.** A logic is **compact** if: whenever $\Gamma \models \varphi$, there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \varphi$.

Equivalently: if every finite subset of Γ has a model, then Γ itself has a model.

Intuition

You can't "sneak up" on a contradiction through infinity. If an infinite set of sentences has no model, some finite subset already has no model.

Compactness has surprising consequences:

Example 1.17 (First-order logic can't express finiteness)**.** Consider $\Gamma = \{\exists x_1, \exists x_1 \exists x_2 (x_1 \neq x_2), \exists x_1 \exists x_2 \exists x_3 (\text{all different}), \ldots\}$

Every finite subset has a finite model. By compactness, Γ has a model—but that model must be infinite!

This means: there is no first-order sentence φ such that φ is true exactly in finite models. First-order logic **cannot express finiteness**.

Finite Model Property: Small Witnesses Suffice

Definition 1.18 (Finite Model Property)**.** A logic has the **finite model property (FMP)** if: whenever φ is satisfiable, it is satisfiable in some *finite* model.

Intuition

If you want to show φ is satisfiable, you don't need to search through infinite models—a finite counterexample always exists (if any exists).

Why care? Because FMP + effective proof search = decidability. If you can enumerate all finite models and all proofs, you'll eventually find either a proof of φ or a finite model of $\neg\varphi$.

Basic modal logic **K** has the FMP, proved via **filtration**—a technique that collapses infinite models to finite ones while preserving truth of formulas up to a certain complexity.

Craig Interpolation: Proofs Stay On Topic

Craig's Interpolation Theorem, 1957

William Craig, working at Berkeley, asked a simple question: if φ implies ψ, what is the "common ground" that connects them? He proved that there's always an intermediate statement using only their shared vocabulary.

Why does this matter? Consider two scientific theories that share some concepts. If theory A implies theory B, Craig's theorem guarantees there's a "bridge" statement expressible in their common language. The implication doesn't require smuggling in external concepts.

This has practical applications in computer science: if module A's specification implies module B's behavior, there's always an interface description using only their shared vocabulary. Proofs are **modular**.

Definition 1.19 (Craig Interpolation)**.** A logic has **Craig interpolation** if: whenever $\varphi \to \psi$ is valid, there exists a formula θ (the **interpolant**) such that:

- $\varphi \to \theta$ is valid
- $\theta \to \psi$ is valid
- Every non-logical symbol in θ appears in both φ and ψ

Intuition

If φ implies ψ, the proof doesn't "wander into irrelevant territory." There's always an intermediate statement θ using only the shared vocabulary.

This means proofs are **modular**—the connection between premise and conclusion can always be expressed using only their common concepts.

Löwenheim-Skolem: The Strangeness of First-Order Logic

Löwenheim-Skolem and the Paradox, 1915–1922

Leopold Löwenheim (1915) proved that any satisfiable first-order sentence has a countable model. Thoralf Skolem (1920, 1922) strengthened and generalized this, then noticed a disturbing consequence that now bears his name.

Here is Skolem's paradox:

Set theory (ZFC) proves that uncountable sets exist—Cantor's theorem shows $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$. The real numbers are uncountable.

But by Löwenheim-Skolem, ZFC has a *countable* model! Call it $\mathcal{M}$. Inside $\mathcal{M}$, there's a set r that $\mathcal{M}$ "thinks" is the real numbers—and $\mathcal{M}$ can prove r is uncountable.

Yet from outside, we can see that $\mathcal{M}$ itself is countable, so $r \subseteq \mathcal{M}$ is countable too.

How can r be "uncountable inside" but countable outside?

The resolution: "uncountable" means "no bijection to $\mathbb{N}$ exists." Inside $\mathcal{M}$, no bijection $r \to \mathbb{N}$ exists *that is an element of* $\mathcal{M}$. But there are bijections outside $\mathcal{M}$ that $\mathcal{M}$ can't see.

The paradox reveals: first-order logic cannot pin down "absolute" cardinality. It can only talk about bijections that exist within the model.

Definition 1.20 (Löwenheim-Skolem)**.** If a first-order theory has an infinite model, it has models of every infinite cardinality (both smaller and larger).

Intuition

First-order logic cannot "pin down" the size of infinite structures. The real numbers, from inside first-order logic, look the same as a countable set.

This is not a bug but a feature: it means first-order theories are "robust" across different sizes of infinity. But it also means first-order logic has limited expressive power.

Cut Elimination: Proofs Without Detours

Gentzen's Hauptsatz, 1935

Gerhard Gentzen, a young German logician, invented the sequent calculus and proved his *Hauptsatz* (main theorem): every proof can be transformed into "cut-free" form. This seemingly technical result had profound consequences for proof theory and would later influence computer science.

What is the "cut" rule? It's the formal version of using lemmas:

1. Prove φ (a lemma)
2. Prove $\varphi \to \psi$ (the lemma implies what you want)
3. Conclude ψ (by modus ponens)

The cut rule is powerful—it lets you introduce "helper" concepts not in your final conclusion. But Gentzen showed: you never *need* it. Any proof using cuts can be transformed into one without.

Why does this matter?

Cut-free proofs have the **subformula property**: every formula in the proof is a subformula of the conclusion or premises. No "outside" concepts sneak in.

This bounds proof search: to prove φ, you only need to consider subformulas of φ. And it yields consistency: to derive $\perp$ (falsehood), you'd need subformulas of $\perp$—but $\perp$ has no subformulas!

Gentzen used this technique to prove the consistency of Peano arithmetic (relative to a stronger principle, transfinite induction up to ε_0). It was the first significant consistency proof after Gödel.

Gentzen's Fate

Gentzen's life ended tragically. A member of the Nazi party (though apparently not ideologically committed), he was captured by Soviet forces in Prague in 1945 and died of starvation in a prison camp at age 29. His work on proof theory became foundational for computer science decades later—the Curry-Howard correspondence connects his sequent calculus to typed programming languages.

Definition 1.21 (Cut Elimination)**.** A proof system has **cut elimination** if every proof can be transformed into a **cut-free** proof—one that doesn't use the cut rule (or any rule introducing "lemmas").

Intuition

Cut-free proofs have the **subformula property**: every formula appearing in the proof is a subformula of the premises or conclusion. No "outside" formulas are needed.

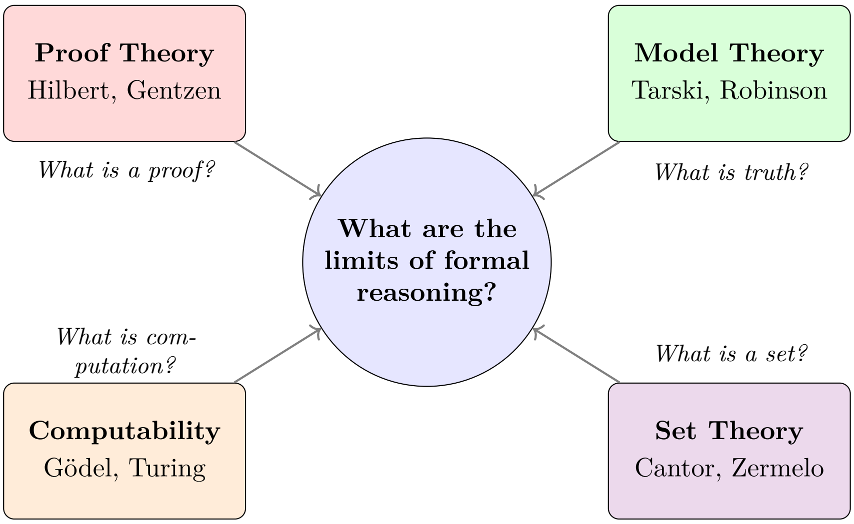
This is enormously useful:

- **Proof search becomes bounded**: you only need to consider subformulas
- **Consistency follows**: if you can't derive $\perp$ using only subformulas of $\perp$ (there are none!), the system is consistent

Gentzen used cut elimination to prove the consistency of arithmetic (relative to a stronger principle called transfinite induction).

1.7 The Four Pillars

The crisis sparked a golden age. Four branches of mathematical logic crystallized:



Proof Theory: Hilbert's program to formalize mathematics; Gentzen's sequent calculus and cut elimination.

Model Theory: Tarski's semantic definition of truth; studying the relationship between syntax and structures.

Computability: Gödel's incompleteness; Church-Turing's definition of computability; the halting problem.

Set Theory: Cantor's infinities; ZFC axiomatics; Cohen's independence results.

1.8 Looking Ahead

Key Insight

Logic studies **the invariance of forms**.
What properties are preserved under what transformations?

- Valid inference preserves truth
- Bisimulation preserves modal properties
- Computable functions preserve finiteness of description
- Homomorphisms preserve algebraic structure

This is why logic serves so many fields: philosophers analyzing concepts, mathematicians studying foundations, linguists uncovering semantic structure, computer scientists verifying programs.

A question lingers: can we study "invariance" itself as a mathematical object? The answer is **category theory**—the subject of the next chapter.

Summary

- Logic began as the study of valid inference (Aristotle's syllogism)
- Medieval logic was sophisticated but forgotten after the Humanist turn
- Modern logic arose from mathematics' need for quantifiers (Frege)
- The syntax/semantics distinction ($\vdash$ vs $\models$) emerged from foundational debates
- Three schools—logicism, formalism, intuitionism—each gave different answers
- Key properties: soundness and consistency (non-negotiable); completeness and decidability (desirable)
- The foundational crisis led to four pillars: proof theory, model theory, computability, set theory
- Core insight: logic studies the **invariance of forms**

Chapter 2

Logic as Language

◎ Goals of This Chapter

- See the evolution of expressive power: from syllogisms to modal logic
- Understand why studying invariance leads naturally to category theory
- Meet algebra and coalgebra as the two faces of formal structure

In the previous chapter, we saw that logic studies **form** and **invariance**—what properties are preserved under what transformations. Now we trace how this insight developed, and where it leads.

2.1 The Ladder of Expressiveness

Logic began with a simple question: which arguments are valid? But the answer required building formal **languages**—and over time, these languages became more and more expressive.

Each step up the ladder was driven by the same complaint: **I can't say what I want to say.**

This desire for greater expressive power is the thread that runs through the entire history of logic—from Aristotle to category theory. Keep it in mind as we climb.

2.1.1 Syllogisms: The Starting Point

Aristotle's syllogisms could express statements of the form:

- All A are B
- Some A are B

- No A are B
- Some A are not B

This is enough for many philosophical arguments. But try to express:

"Every person loves someone."

Is it "All persons are lovers"? That loses the structure. The syllogism sees only **two terms** in a fixed relationship. The internal complexity—that loving involves a *relation* between two things—is invisible.

2.1.2 Propositional Logic: From the Stoics to Boole

Historical Note

Propositional logic actually predates Boole by two millennia. The **Stoic** philosophers (3rd century BC) studied arguments based on connectives like "and," "or," and "if-then"—independently of Aristotle's term logic.

They identified basic argument forms, including what we now call *modus ponens* and *modus tollens*. But their work was largely forgotten, rediscovered only in the 20th century.

George Boole (1854) reinvented propositional logic algebraically, treating propositions as variables and connectives as operations. This algebraic approach made logic part of mathematics.

Boole's algebra could express combinations of propositions:

$$(p \wedge q) \to r$$

The basic **inference rules** of propositional logic are:

Modus Ponens From P and $P \to Q$, infer Q.
"It rains. If it rains, the ground is wet. So the ground is wet."

Modus Tollens From $\neg Q$ and $P \to Q$, infer $\neg P$.
"The ground is dry. If it rains, the ground is wet. So it's not raining."

Hypothetical Syllogism From $P \to Q$ and $Q \to R$, infer $P \to R$.
"If A then B. If B then C. So if A then C." (Chaining conditionals)

Disjunctive Syllogism From $P \vee Q$ and $\neg P$, infer Q.
"A or B. Not A. So B."

These rules are **truth-preserving**: if the premises are true, the conclusion must be true. This can be verified by **truth tables**. Each connective is defined by how the truth value of the whole depends on the truth values of the parts:

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

Wait—why is $p \to q$ true when p is false?

Intuition

This is **material implication**, and it confuses everyone at first.
Think of implication as a **rule**: "When the light is red, all cars must stop."
Now suppose the traffic light is broken today—it's green all day. No car stops. Has anyone violated the rule?
No. The rule only says what must happen *when the light is red*. If the light is never red, the rule is never triggered. It's not violated—it simply doesn't apply.
That's material implication: $p \to q$ is false *only* when p is true and q is false. Otherwise, no violation.

This leads to strange truths. Let p = "the moon is made of cheese." Then:

$$p \to q$$

is **true** for *any* q—"if the moon is made of cheese, then I am the Pope," "if the moon is made of cheese, then $2 + 2 = 5$." All true, because p is false.

In fact, from a false premise, you can "derive" anything. This is called ex falso quodlibet (from falsehood, anything follows).

Even worse, you can construct seemingly valid "proofs" of absurd conclusions—like proving that God exists from the mere fact that you don't pray. We'll analyze this famous example in detail after introducing modal logic, where we'll see how a more expressive language dissolves the paradox.

But for now, note the more basic limitation: each proposition is atomic. "All men are mortal" is just a letter p. You cannot look inside to see the quantifier "all" or the predicate "mortal."

2.1.3 First-Order Logic: Frege's Revolution

Frege's first-order logic finally cracked open propositions:

$$\forall x \, (\mathrm{Man}(x) \to \mathrm{Mortal}(x))$$

Now we have:

- **Variables**: x, y, z ranging over individuals

- **Predicates**: $\text{Man}(x)$, $\text{Mortal}(x)$
- **Relations**: $\text{Loves}(x, y)$
- **Quantifiers**: $\forall$ (for all), $\exists$ (exists)

"Every person loves someone" becomes:

$$\forall x\,(\text{Person}(x) \to \exists y\,\text{Loves}(x, y))$$

First-order logic is **remarkably expressive**. You can formalize most of mathematics in it. For decades, it was *the* logic.

But it inherits the same quirk from propositional logic: vacuous truth. Consider:

"All even primes other than 2 are divisible by 5."

Formally: $\forall x\,((\text{EvenPrime}(x) \wedge x \neq 2) \to \text{DivisibleBy5}(x))$.

This is **true**. Why? Because there are no even primes other than 2. The antecedent is never satisfied, so the implication is vacuously true for all x.

Or consider: "I am taller than every blue giraffe in this room." Formally:

$$\forall x\,((\text{BlueGiraffe}(x) \wedge \text{InThisRoom}(x)) \to \text{TallerThan}(me, x))$$

Also true—there are no blue giraffes here. You can truthfully claim to be taller than all of them.

Why does this work? Because of the **algebraic structure** of quantifiers. The negation of our claim would be:

$$\exists x\,((\text{BlueGiraffe}(x) \wedge \text{InThisRoom}(x)) \wedge \neg\text{TallerThan}(me, x))$$

In plain English: "There exists a blue giraffe in this room that is at least as tall as me."

This is clearly **false**—there are no blue giraffes at all. And since the negation is false, the original statement must be true.

Intuition

This feels strange because our intuition says: "How can you be taller than something that doesn't exist?" But the logic doesn't say you *are* taller than anything—it says there's *no counterexample*. And indeed there isn't: no blue giraffe stands in this room being taller than you. The algebraic duality $\neg\forall x\,\varphi \equiv \exists x\,\neg\varphi$ forces this. To deny "all X have property P," you must produce an X that lacks P. If no X exists, you can't produce one, so the universal claim stands.

In mathematics, vacuous truth is often convenient (it lets us state theorems without worrying about edge cases). But it shows that first-order logic's $\to$ still doesn't match our intuitive "if-then."

And there's more it cannot say.

2.1.4 What First-Order Logic Cannot Say

Consider these statements:

- "It is *necessary* that $2 + 2 = 4$."
- "Alice *knows* that Bob is lying."
- "The program *will eventually* terminate."
- "It is *obligatory* to keep promises."

These all involve a **mode** of truth. Not just "is φ true?" but "in what *way* is φ true?"

First-order logic has no way to express these modes. It knows only bare, unqualified truth.

2.1.5 Modal Logic: Beyond First-Order

Modal logic adds **operators** that modify propositions:

- $\Box\varphi$: necessarily φ / in all accessible worlds, φ
- $\Diamond\varphi$: possibly φ / in some accessible world, φ

With different interpretations of "accessible," we get different modal logics:

Interpretation	$\Box\varphi$ means	**Field**
Necessity	necessarily φ	Metaphysics
Knowledge	agent knows φ	Epistemology
Time	always in the future, φ	Temporal logic
Obligation	it ought to be that φ	Deontic logic
After action a	after doing a, φ holds	Dynamic logic

Intuition

Modal logic is not one logic but a **family** of logics, each tuned to a different notion of "possibility" and "necessity." The syntax ($\Box$, $\Diamond$) is the same; the semantics varies.

This explosion of modal logics—hundreds of them, with different axioms and different applications—raises a question:

Is there a unified framework for all of them?

But first, let us see how modal logic resolves the paradox we encountered earlier.

2.1.6 Revisiting the Prayer Argument

Recall the "proof" that God exists. Here is the full derivation, with both the formal logic and plain English at each step:

1. $\neg G \to \neg(P \to A)$ (Premise 1)

   "If God doesn't exist, then 'if I pray, I'll be answered' is false."

2. $\neg P$ (Premise 2)

   "I don't pray."

3. $P \to A$ (From 2, by truth table of $\to$)

   "Since I don't pray, 'if I pray, I'll be answered' is vacuously true." (The antecedent is false, so the conditional is true.)

4. $(P \to A) \to G$ (Contrapositive of 1)

   "If 'if I pray, I'll be answered' is true, then God exists." (This is logically equivalent to Premise 1.)

5. G (Modus ponens on 3 and 4)

   "Therefore, God exists."

The logic is valid. The conclusion is absurd. What went wrong?

2.1.7 The Modal Solution: Strict Implication

The problem is that material implication $P \to A$ is too coarse to express the nuance we actually need. It only talks about the *actual* world. In the actual world, I don't pray, so $P \to A$ is trivially true.

But when we say "if I pray, I'll be answered," we mean something stronger: *in any situation where I pray, I would be answered.* This is **strict implication**:

$$\Box(P \to A)$$

Let's be explicit about the notation:

- $\Box$ reads as "necessarily" or "in all possible situations"
- $\Box(P \to A)$ reads as: "Necessarily, if I pray then I'm answered"
- In plain English: "In *every* possible situation where I pray, I would be answered"

This is much stronger than material implication $P \to A$, which only talks about the actual world.

Now let's redo the argument with strict implication, step by step:

Premise 1: "If God doesn't exist, then it's *not* necessarily true that praying leads to being answered."

$$\neg G \to \neg\Box(P \to A)$$

Premise 2: "I don't pray (in the actual world)."

$$\neg P$$

Now can we derive G (God exists)?

Step 1: "Since I don't pray, 'if I pray then I'm answered' is trivially true—here, now."

$$\neg P \vdash P \to A \quad \text{(in the actual world)}$$

Step 2: "But does 'I don't pray here' imply 'in ALL situations, praying leads to answers'?"

$$\neg P \vdash \Box(P \to A) \quad \textbf{???}$$

No! This does not follow. In other possible situations, I might pray.

Step 3: Without $\Box(P \to A)$, we cannot use the contrapositive of Premise 1. The trick that "proved" God exists no longer works.

Why the argument collapses: In other possible situations, I might pray. And in those situations, if God doesn't exist, my prayers wouldn't be answered. So $P \to A$ would be *false* there—even though it's vacuously true here.

Therefore: $\neg P \nvdash \Box(P \to A)$. "I don't pray" does *not* imply "necessarily, praying leads to answers."

Modal logic has the expressive power to distinguish what classical logic conflates.

2.1.8 The Moral: Expressiveness Drives Progress

Historical Note

This is exactly why C.I. Lewis invented modal logic in 1912.[a] He was disturbed by the "paradoxes of material implication" in Russell and Whitehead's *Principia Mathematica*: that a false proposition implies anything, and a true proposition is implied by anything.

Lewis argued that this doesn't match the meaning of "implies" in ordinary reasoning and proof. He developed **strict implication** $A \Rightarrow B =_{\text{def}} \Box(A \to B)$ to capture a stronger notion. Kripke later provided the elegant possible-worlds semantics that made modal logic mathematically tractable.

[a] See Stanford Encyclopedia of Philosophy: Modern Origins of Modal Logic.

Key Insight

Here is the crucial point:
We did *not* say: "The prayer argument is logically valid in propositional logic, so we must accept that God exists."
We said: "The prayer argument reveals that propositional logic's notion of validity doesn't match our intuitions about implication. So we build a better logic."
Logic is a tool, not a master. When the tool doesn't fit the job—when its $\vdash$ doesn't match our intuitive sense of "follows from"—we develop a more expressive language. This is the engine that drives the history of logic.

2.2 The Great Shift: From Consequence to Structure

Before we continue, let us pause to notice something profound.

Look at how the question has changed:

Aristotle's question:	Is this argument valid? (Does the conclusion follow from the premises?)
The modern question:	What kind of structure does this logic describe? (What are the "worlds" and how are they related?)

This is a fundamental shift in what logic *is*.

2.2.1 The Old View: Logic Studies Consequence

In the old view, logic was about **valid inference**:

> Given premises $A_1, \ldots, A_n$, does conclusion B necessarily follow?

The central concept was **consequence**: the relation $A_1, \ldots, A_n \vdash B$. Everything else—syntax, semantics, proof systems—served this goal.

2.2.2 The New View: Logic Describes Systems

In the new view, logic is a **language for describing structured systems**:

> Given a type of system (possible worlds, time, knowledge states, program states...), how do we talk about it precisely?

The central concept is **structure**: what kind of "world" does this logic describe?

Consequence becomes **derived**: once you fix a logic and its class of structures, the consequence relation falls out automatically.

Key Insight

Logic shifted from studying **"what follows from what"** to studying **"how to describe certain kinds of systems."**
Consequence is no longer the definition—it is a *consequence* of the definition.

2.2.3 When Did This Happen?

The shift was gradual, but modal logic made it explicit.

In classical propositional or first-order logic, you can almost ignore the "worlds." There's just one world, and formulas are either true or false in it. The focus stays on consequence.

But modal logic *forces* you to think about structure:

- What are the possible worlds?
- What is the accessibility relation?
- Is it reflexive? Transitive? Symmetric?

The logic doesn't make sense until you specify the **structure**. And different structures give different logics.

Intuition

Think of it this way:
Old: Logic is a machine for checking arguments.
New: Logic is a lens for viewing systems. Different logics are different lenses, suited to different systems.
When you "apply" a logic to a domain, you are *instantiating* the lens—choosing a particular system to view through it. Consequence is what you see through the lens.

2.2.4 Why This Matters

This shift explains why so many different fields need logic.

They don't need logic because they want to "do deduction." They need logic because they have **structured systems** to describe:

- **Linguists**: Natural language is a system. Sentences have structure. "If this word is a verb, what can come next?" is a structural question.

- **Computer scientists**: Programs are systems. States, transitions, specifications—all structural.
- **Philosophers**: Possible worlds, knowledge states, moral situations—structured domains that need precise description.

Logic provides the **general framework** for describing systems with antecedent-consequent structure: "in situation X, Y holds" or "from state X, you can reach state Y."

This is not "reasoning" in the everyday sense. It is **structural description**.

2.3 Enter the Linguists

Meanwhile, in linguistics, a parallel development was happening.

2.3.1 Chomsky: Language Has Hidden Structure

Noam Chomsky revolutionized linguistics in the 1950s with a simple observation: **language is not a list of sentences**. It is a **generative system**—a finite set of rules that produces infinitely many sentences.

Historical Note

Chomsky's *Syntactic Structures* (1957) introduced formal grammars to linguistics. The same mathematical tools used for programming languages could describe natural languages.

But Chomsky focused on **syntax**—the structure of sentences. What about **meaning**?

2.3.2 Montague: Natural Language Has Formal Semantics

Richard Montague made a bold claim: natural language can be given a **precise formal semantics**, just like a programming language.

Historical Note

"I reject the contention that an important theoretical difference exists between formal and natural languages." — Richard Montague, 1970

Montague grammar uses **intensional logic**—a form of modal logic—to analyze meaning. Why modal logic?

Because natural language is full of **modality**:

- "John *might* come" (possibility)

- "Mary *must* have left" (necessity/inference)
- "He *believes* that it's raining" (propositional attitudes)
- "She *will* arrive tomorrow" (future tense)

Key Insight

Natural language doesn't just describe what *is*. It describes what *could be*, what *must be*, what someone *thinks* is, what *will be*.
Modal logic is the natural tool for this, because it has the expressive power to talk about **alternative possibilities**.

2.4 Enter the Computer Scientists

At the same time, computer scientists discovered they needed modal logic too.

2.4.1 Program Specification

How do you specify what a program should do?

Floyd and Hoare developed **Hoare logic**: $\{P\}\,C\,\{Q\}$ means "if precondition P holds before running program C, then postcondition Q holds after."

This is an **implication**—but about *states before and after an action.* It has modal flavor.

2.4.2 Temporal Logic for Verification

Amir Pnueli introduced **temporal logic** for specifying properties of reactive systems:

- $\mathbf{G}\,\varphi$: "globally, φ" (always in the future)
- $\mathbf{F}\,\varphi$: "finally, φ" (eventually)
- $\varphi\,\mathbf{U}\,\psi$: "$\varphi$ until ψ"

Example 2.1. "Every request is eventually answered":

$$\mathbf{G}\,(\mathit{request} \to \mathbf{F}\,\mathit{answer})$$

This is modal logic, with $\Box = \mathbf{G}$ (necessity as "always").

2.4.3 The Curry-Howard Correspondence

Perhaps the deepest connection: **proofs are programs**.

This sounds mystical, but it's a precise mathematical statement. Let's build up to it.

The Core Idea: Think of a proposition A as a *specification*—a claim that something exists or can be done. A proof of A is *evidence* that the claim is true. Now, what is evidence?

For simple propositions, evidence might be a witness: a proof of "there exists an even prime" is the number 2.

For implications $A \to B$, evidence is a *method*: given evidence for A, produce evidence for B. But a method that transforms input to output is exactly what a **function** is!

Curry-Howard Isomorphism

$$\begin{aligned}
\text{Propositions} &\quad\longleftrightarrow\quad \text{Types}\\
\text{Proofs} &\quad\longleftrightarrow\quad \text{Programs}\\
A \to B &\quad\longleftrightarrow\quad \text{Function type } A \to B\\
A \wedge B &\quad\longleftrightarrow\quad \text{Product type } (A, B)\\
A \vee B &\quad\longleftrightarrow\quad \text{Sum type } A + B\\
\text{Proof simplification} &\quad\longleftrightarrow\quad \text{Program execution}
\end{aligned}$$

Let's see concrete examples.

Example 2.2 (Identity: $A \to A$)**. Logical statement:** "If A, then A." (Trivially true.)

Proof: Assume A. Then we have A. Done.

Corresponding program: The identity function.

```
def identity(x: A) -> A:
    return x
```

Why they match: The proof says "given evidence for A, return that same evidence." The program says "given input of type A, return it unchanged." Same thing!

Example 2.3 (Modus Ponens: $(A \to B) \wedge A \to B$)**. Logical statement:** "If we have both 'A implies B' and 'A', then we have B."

Proof:

1. Assume we have $(A \to B) \wedge A$.
2. From this, extract the proof of $A \to B$ (call it f).
3. From this, extract the proof of A (call it a).
4. Apply f to a to get a proof of B.

Corresponding program: Function application.

```
def modus_ponens(pair: (A -> B, A)) -> B:
    f, a = pair       # extract function and argument
    return f(a)       # apply function to argument
```

Why they match: "Applying an implication to its antecedent" in logic is exactly "calling a function with its argument" in programming.

Example 2.4 (Composition: $(A \to B) \to (B \to C) \to (A \to C)$)**. Logical statement:** "If A implies B, and B implies C, then A implies C." (Hypothetical syllogism!)

Proof:

1. Assume $A \to B$ (call this proof f).
2. Assume $B \to C$ (call this proof g).
3. Now we must prove $A \to C$. So assume A (call this proof a).
4. Apply f to a, getting a proof of B.
5. Apply g to that, getting a proof of C.
6. Done: from A we produced C.

Corresponding program: Function composition.

```
def compose(f: A -> B, g: B -> C) -> (A -> C):
    def composed(a: A) -> C:
        return g(f(a))
    return composed
```

Why they match: Chaining implications is function composition. The logical structure and the computational structure are identical.

Intuition

The correspondence goes deeper:

- **Conjunction** $A \wedge B$ corresponds to **pair/product types** (A, B). A proof of $A \wedge B$ is a pair of proofs; a value of type (A, B) is a pair of values.
- **Disjunction** $A \vee B$ corresponds to **sum/union types** $A + B$. A proof of $A \vee B$ is either a proof of A or a proof of B (tagged with which); a value of $A + B$ is either a value of A or a value of B.
- **False** $\perp$ corresponds to the **empty type** (no values). There's no proof of $\perp$, and no program can return a value of the empty type.
- **Proof simplification** (cutting out unnecessary steps) corresponds to **program execution** (reducing expressions to values).

Historical Note

This correspondence was discovered independently by Haskell Curry (1934, for combinatory logic) and William Howard (1969, for natural deduction). It's now foundational to type theory and proof assistants like Coq, Agda, and Lean, where you literally write programs to construct proofs.

The Curry-Howard correspondence reveals that logic and computation are two perspectives on the same underlying structure. This is not a loose analogy—it's a precise isomorphism.

2.5 The Need for Greater Generality

Remember the thread: the desire for greater expressive power.

Let us take stock.

Logic has transformed from the study of valid argument into a family of languages for describing structured systems:

- Modal logic describes systems of possible worlds
- Temporal logic describes systems evolving in time
- Dynamic logic describes systems of program states

- Epistemic logic describes systems of knowledge states

Each logic is tailored to a particular kind of structure.

But this raises a question: **is there a language that can describe structure itself?**

Not this or that particular structure, but the general notion of "what it means to have structure" and "what it means to preserve structure."

> **Intuition**
>
> We have many lenses, each suited to viewing a particular kind of system.
> Can we build a **lens for lenses**—a framework for talking about what all these lenses have in common?

The four pillars of logic (proof theory, model theory, computability, set theory) all use set theory as their metalanguage. But set theory talks about *membership*—what elements belong to what collections. This is not the same as *structure*.

What we need is a language where **structure** and **invariance under transformation** are the primitive concepts.

2.6 Category Theory: The Ultimate Abstraction

The answer came from algebra, not logic—but it turned out to be exactly what logic needed.

In the 1940s, Samuel Eilenberg and Saunders Mac Lane, studying algebraic topology, noticed something curious: many proofs weren't really about specific objects (groups, spaces, modules). They were about **relationships between objects**—homomorphisms, continuous maps, natural transformations.

The objects were interchangeable; the arrows were what mattered.

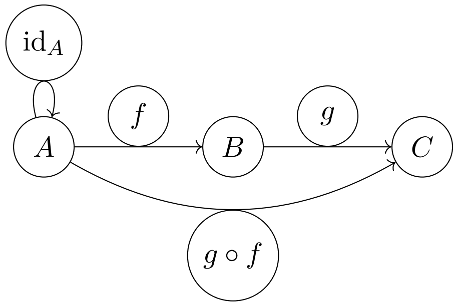
They invented **category theory** to capture this.

2.6.1 Informal Picture

Think of a category as a world of objects connected by arrows:

- **Objects** are the "things"—sets, groups, spaces, whatever. We don't care what they're made of internally.
- **Morphisms** (arrows) are the "relationships" between objects. An arrow $f : A \to B$ goes from A to B.
- **Composition**: If you can go from A to B, and from B to C, you can go from A to C. Arrows chain together.

- **Identity**: Every object has a "do nothing" arrow to itself.



2.6.2 Formal Definition

Definition 2.5 (Category)**.** A **category** $\mathcal{C}$ consists of:

- A collection $\mathrm{Ob}(\mathcal{C})$ of **objects**
- For each pair of objects A, B, a collection $\mathrm{Hom}(A, B)$ of **morphisms** from A to B
- For each object A, an **identity morphism** $\mathrm{id}_A \in \mathrm{Hom}(A, A)$
- For each triple A, B, C, a **composition operation**:

$$\circ : \mathrm{Hom}(B, C) \times \mathrm{Hom}(A, B) \to \mathrm{Hom}(A, C)$$

satisfying:

1. **Associativity:** $(h \circ g) \circ f = h \circ (g \circ f)$

   "It doesn't matter how you group compositions—the result is the same."

2. **Identity laws:** $f \circ \mathrm{id}_A = f$ and $\mathrm{id}_B \circ f = f$ for any $f : A \to B$

   "Composing with identity does nothing."

2.6.3 Why These Axioms?

The axioms capture the minimal requirements for "things and transformations between them":

- **Associativity** says composition is well-behaved. If you transform $A \to B \to C \to D$, you get the same result whether you first compose $A \to B \to C$ and then go to D, or first compose $B \to C \to D$ and apply A to that. No ambiguity.
- **Identity** says every object has a trivial "self-transformation." This is like the number 0 for addition or 1 for multiplication—it's the "do nothing" operation.

Example 2.6 (Some categories)**.**

- **Set**: objects are sets, morphisms are functions, composition is function composition, identity is the identity function $x \mapsto x$.
- **Grp**: objects are groups, morphisms are group homomorphisms (functions that preserve the group operation), composition and identity as in **Set**.
- **Top**: objects are topological spaces, morphisms are continuous maps.
- **Pos**: objects are partially ordered sets, morphisms are monotone functions ($x \leq y \Rightarrow f(x) \leq f(y)$).
- **Vect**$_k$: objects are vector spaces over field k, morphisms are linear maps.

Intuition

Notice the pattern: in each case, morphisms are "structure-respecting maps." Functions respect set structure (elements map to elements). Homomorphisms respect group structure ($f(ab) = f(a)f(b)$). Continuous maps respect topological structure (preimages of open sets are open).
A category isolates this pattern: objects have some structure, morphisms respect it. We can then study properties that depend only on this pattern, not on the specific structures involved.

Key Insight

Category theory is the **mathematics of mathematics**. It studies structure and invariance in full generality.
This is exactly what logic needed: a language where *transformations* and *what they preserve* are first-class concepts.

2.7 Lawvere: Logic Meets Category Theory

In the 1960s, F. William Lawvere showed that logic itself could be expressed categorically.

Historical Note

Lawvere's thesis (1963) and subsequent work reformulated:

- Theories as categories
- Models as functors
- Logical operations as categorical constructions (products, exponentials, etc.)

Let's unpack what this means.

2.7.1 Conjunction is Product

In logic, $A \wedge B$ is true when both A and B are true. To have evidence for $A \wedge B$, you need evidence for A *and* evidence for B.

In category theory, this is exactly a **product** $A \times B$: an object equipped with projections to both A and B, universal among such objects.

$$\begin{tikzcd} & A \times B \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] & \\ A & & B \end{tikzcd}$$

Having a proof of $A \wedge B$ means having a pair: a proof of A and a proof of B.

2.7.2 Disjunction is Coproduct

In logic, $A \vee B$ is true when at least one of A or B is true.

In category theory, this is a **coproduct** (or sum) $A + B$: an object with injections from both A and B.

A proof of $A \vee B$ is either a proof of A or a proof of B (tagged with which one).

2.7.3 Implication is Exponential

This is deeper. In logic, $A \to B$ means "if A then B." A proof of $A \to B$ is a *method* that transforms any proof of A into a proof of B.

In category theory, this is the **exponential object** B^A (also written $[A, B]$ or $A \Rightarrow B$).

Definition 2.7 (Exponential Object)**.** In a category, the **exponential** B^A is an object representing "morphisms from A to B." It is characterized by a natural bijection:

$$\mathrm{Hom}(C \times A, B) \;\cong\; \mathrm{Hom}(C, B^A)$$

What does this mean? Let's translate:

Left side: Maps from $C \times A$ to B. These are functions that take a pair (c, a) and produce a b.

Right side: Maps from C to B^A. These are functions that take a c and produce a "function from A to B."

The bijection says: a function of two arguments is the same as a function that returns a function. This is **currying**!

Example 2.8 (In **Set**)**.** In the category of sets, B^A is exactly the set of all functions from A to B:

$$B^A = \{f : A \to B\}$$

The bijection is currying:

$$f(c, a) = b \quad \longleftrightarrow \quad g(c) = (\lambda a.\, f(c, a))$$

Why the notation B^A? Because $|B^A| = |B|^{|A|}$—if B has m elements and A has n elements, there are m^n functions from A to B.

2.7.4 Quantifiers are Adjoints

Now for the deepest part: quantifiers as adjoints.

First, what is an **adjunction**? Informally, two functors F and G are adjoint if they're "almost inverses" in a specific sense:

Definition 2.9 (Adjunction, informally)**.** $F : \mathcal{C} \to \mathcal{D}$ is **left adjoint** to $G : \mathcal{D} \to \mathcal{C}$ (written $F \dashv G$) if there's a natural bijection:

$$\mathrm{Hom}_{\mathcal{D}}(F(A), B) \;\cong\; \mathrm{Hom}_{\mathcal{C}}(A, G(B))$$

"A map out of $F(A)$ is the same as a map into $G(B)$." F and G are two perspectives on the same relationship.

Now, how do quantifiers fit in?

Consider a projection $\pi : A \times B \to A$ (forgetting the B component). This induces a functor $\pi^* : \mathbf{Set}/A \to \mathbf{Set}/(A \times B)$ that pulls back predicates.

- **Existential quantifier** $\exists$ is the **left adjoint** to pullback.

  "There exists a b such that $P(a, b)$" collapses the B-dimension by asking "is there at least one?"

- **Universal quantifier** $\forall$ is the **right adjoint** to pullback.

  "For all b, $P(a, b)$" collapses the B-dimension by asking "does it hold for every one?"

Intuition

Why adjoints? Because $\exists$ and $\forall$ satisfy these bijections:
For $\exists$: To prove $(\exists b.\, P(a,b)) \to Q(a)$, it suffices to prove $P(a,b) \to Q(a)$ (for arbitrary b). If Q holds whenever P holds for any specific b, then Q holds when P holds for *some* b.
For $\forall$: To prove $Q(a) \to (\forall b.\, P(a,b))$, you must prove $Q(a) \to P(a,b)$ for each b. If from Q you can derive P for any specific b, then from Q you can derive P for *all* b.
These are exactly the adjunction bijections in disguise!

Key Insight

The duality between $\exists$ (left adjoint) and $\forall$ (right adjoint) is not a coincidence—it's the same duality that appears throughout mathematics:

- Free structures (left adjoint) vs. forgetful functors (right adjoint)
- Tensor product (left adjoint) vs. Hom (right adjoint)
- Image (left adjoint) vs. preimage (right adjoint)

Quantifiers are just another instance of this universal pattern.

This is not just a translation. It reveals the **structural essence** of logic, stripped of the accidents of set-theoretic encoding. Implication, conjunction, disjunction, quantification—all emerge from categorical structure.

Intuition

A topos is a "universe of mathematics" with its own internal logic. Different toposes can have different logics: classical, intuitionistic, or even more exotic.
This unified logic and geometry: a topos is both a geometric object (like a space) and a logical universe (where you can do mathematics).

2.8 Topos Theory: Logic and Geometry Unite

Lawvere and Myles Tierney developed **topos theory**: categories that behave like the category of sets, but with an internal logic that can differ from classical logic.

Intuition

A topos is a "universe of mathematics" with its own internal logic. Different toposes can have different logics: classical, intuitionistic, or even more exotic.

This unified logic and geometry: a topos is both a geometric object (like a space) and a logical universe (where you can do mathematics).

2.9 Algebra and Coalgebra: Two Faces of Structure

Within category theory, a fundamental duality emerged: **algebra** and **coalgebra**.

2.9.1 Algebra: Building Up

An **algebra** for a functor F is a structure where you can "fold" or "evaluate" F-structured data.

Example 2.10 (Lists as an algebra)**.** The type of lists of natural numbers satisfies:

$$\mathsf{List}(\mathbb{N}) \cong 1 + \mathbb{N} \times \mathsf{List}(\mathbb{N})$$

A list is either empty (1) or a head ($\mathbb{N}$) followed by a tail (another list).

This is an **algebra** for the functor $F(X) = 1 + \mathbb{N} \times X$.

Functions on lists (like sum) are defined by **folding**: specify what to do with empty and what to do with cons.

Algebras are about **construction** and **induction**: building finite, well-founded structures.

The syntax of a logic—formulas, proofs—forms an algebra: the **initial algebra** of the grammar.

2.9.2 Coalgebra: Observing Behavior

A **coalgebra** for a functor F is a structure where you can "unfold" or "observe" behavior.

Example 2.11 (Streams as a coalgebra)**.** An infinite stream of natural numbers has type:

$$\mathsf{Stream}(\mathbb{N}) \to \mathbb{N} \times \mathsf{Stream}(\mathbb{N})$$

From a stream, you can observe: the head (a number) and the tail (another stream).

This is a **coalgebra** for the functor $F(X) = \mathbb{N} \times X$.

Functions producing streams are defined by **unfolding**: specify the head and how to continue.

Coalgebras are about **behavior** and **coinduction**: potentially infinite, observable processes.

State machines, automata, transition systems—these are coalgebras.

2.9.3 The Duality

	Algebra	**Coalgebra**
Perspective	Construction	Observation
Principle	Induction	Coinduction
Structure	Finite, well-founded	Potentially infinite
Examples	Syntax, data types	Behavior, state machines
Canonical object	Initial algebra	Final coalgebra

Key Insight

Algebra is about **syntax**: how things are built.
Coalgebra is about **semantics**: how things behave.
Logic lives in both: the formulas are algebraic (syntax), but their meaning involves coalgebraic structures (models, possible worlds, state transitions).

2.10 Modal Logic as Coalgebraic Logic

Here is where everything connects.

Remember Kripke semantics for modal logic? A **Kripke frame** is a pair (W, R) where:

- W is a set of "possible worlds"
- $R \subseteq W \times W$ is an "accessibility relation"

But this is exactly a **coalgebra**! A Kripke frame is a coalgebra for the powerset functor:

$$\alpha : W \to \mathcal{P}(W)$$

where $\alpha(w) = \{v \mid wRv\}$—the set of worlds accessible from w.

Key Insight

Kripke frames are coalgebras.
And the key notion of equivalence for coalgebras is **bisimulation**—which is exactly the right notion of equivalence for modal logic!
Van Benthem's theorem: modal logic is the **bisimulation-invariant** fragment of first-order logic.

This is not a coincidence. Modal logic is the **natural language for describing coalgebraic behavior**.

Different modal logics arise from different functors:

- Kripke frames: $F(X) = \mathcal{P}(X)$
- Labeled transition systems: $F(X) = \mathcal{P}(A \times X)$ for action set A
- Probabilistic systems: $F(X) = \mathcal{D}(X)$ (probability distributions)
- Automata: $F(X) = 2 \times X^A$ (accepting/rejecting + transitions)

Coalgebraic modal logic provides a uniform treatment of all these cases.

2.11 The Modern Landscape

Let us step back and see where we are.

$$
\begin{tikzcd}
 & \textbf{``Form + Structure-preserving''} \arrow[d, "\text{20th century}"] & \\
 & \text{Four Pillars} \arrow[d, "\text{``what is structure?''}"] & \\
 & \text{Category Theory} \arrow[dl] \arrow[dr] & \\
\text{Algebra (Syntax)} & & \text{Coalgebra (Behavior)} \arrow[d, "\text{natural language for}"] \\
 & & \text{Modal Logic}
\end{tikzcd}
$$

The journey, driven throughout by the desire for greater expressive power:

1. **Syllogisms**: "I want to describe valid argument forms"
2. **First-order logic**: "I want to describe quantified relationships"

3. **Modal logic**: "I want to describe possibility, necessity, time, knowledge..."

4. **The shift**: Logic becomes a language for describing systems, not just a tool for checking arguments

5. **Category theory**: "I want to describe structure itself"

6. **Algebra/Coalgebra**: Syntax vs. behavior—two faces of structure

7. **Coalgebraic modal logic**: Modal logic is the natural language for behavioral systems

Each step: *I can't say what I want to say → build a more expressive language*.

2.12 Why This Matters for Us

This book is about **the algebra of intelligence**. We want to understand:

- How can agents reason about possibility, knowledge, change?
- How can we specify and verify agent behavior?
- How can formal structures *learn*?

Modal logic gives us the **language**.

Coalgebra gives us the **semantics**—a unified framework for all kinds of state-based, behavioral systems.

And the algebraic/coalgebraic duality will be crucial when we ask: how do we make these structures **learnable**?

But first, we need to understand modal logic properly. That begins in the next chapter, with Kripke frames.

Summary

The history of logic is the history of the desire for greater expressive power.

- **The ladder**: syllogisms → propositional → first-order → modal → ...
- **The shift**: from "studying consequence" to "describing structured systems"
- **The convergence**: philosophers, linguists, computer scientists all need to describe systems
- **The ultimate abstraction**: category theory—the language of structure itself
- **The duality**: algebra (syntax, construction) vs. coalgebra (behavior, observation)
- **The connection**: modal logic is the natural language for coalgebraic systems

One thread runs through it all: *I can't say what I want to say—so I build a more expressive language.*

Chapter 3

Kripke Frames

◎ Goals of This Chapter

- Understand Kripke semantics: worlds, accessibility, satisfaction
- Learn what relations are and their key properties (reflexive, transitive, symmetric)
- See how different frame properties give different logics
- Understand bisimulation as the right notion of equivalence for modal logic

3.1 The Problem Kripke Faced

It's 1959. Modal logic has been around for nearly 50 years since C.I. Lewis's first systems, but it's in a strange state.

On the *syntactic* side, things are clear: we have axiom systems like S4 and S5, we can derive theorems, we know which formulas are provable. But on the *semantic* side—what do these formulas *mean*?

The early semantics were awkward. Some used algebraic structures (Boolean algebras with operators). Others tried to interpret $\Box\varphi$ as "φ is provable" or "φ is analytic." None of these felt natural or gave good intuitions.

Historical Note

Saul Kripke was a teenager when he solved this problem. His paper "A Completeness Theorem in Modal Logic" (1959) was written when he was 18, still in high school. He later said the basic idea came to him at age 15 or 16.
The insight was almost embarrassingly simple: take the philosophers seriously when they talk about "possible worlds."

Kripke asked: what if we *literally* model the set of possible worlds, and define $\Box\varphi$ to mean "φ is true in all worlds we can reach from here"?

This turned philosophical intuition into precise mathematics. And it worked beautifully.

3.2 The Idea: Truth is Relative

In classical logic, a proposition is either true or false. Period.

But think about these statements:

- "It might rain tomorrow."
- "I know that Paris is in France."
- "After pressing the button, the light will be on."

These aren't just true or false—they depend on *perspective*. "It might rain" is true if there's *some* possible future where it rains. "I know P" is true if P is true in *all* situations compatible with what I know.

Kripke's insight: model this by having **multiple worlds**, and a relation that says which worlds are "accessible" from which.

3.3 Relations: A Quick Primer

Before we dive in, let's make sure we understand **relations**—they're the backbone of Kripke semantics.

Definition 3.1 (Relation)**.** A *relation* R on a set W is just a set of pairs: $R \subseteq W \times W$.

We write wRv (or $w\ R\ v$, or $(w, v) \in R$) to mean "w is related to v."

Example 3.2 (Relations in everyday life)**.**

- "is a friend of" on the set of people
- "is less than" ($<$) on numbers
- "is reachable from" on cities (via direct flights)

- "is a parent of" on people

Relations can have special properties. Here are the important ones:

3.3.1 Reflexive

Plain English: Everything is related to itself.

Formally: For all w: wRw.

Example 3.3 (Reflexive or not?)**.**

- "$\leq$" (less than or equal): **Yes.** Every number is $\leq$ itself.
- "$<$" (strictly less than): **No.** No number is strictly less than itself.
- "is the same age as": **Yes.** You're the same age as yourself.
- "is a friend of": **Debatable.** Are you your own friend?

3.3.2 Symmetric

Plain English: If w is related to v, then v is related to w. The relation goes both ways.

Formally: For all w, v: $wRv \Rightarrow vRw$.

Example 3.4 (Symmetric or not?)**.**

- "is married to": **Yes.** If Alice is married to Bob, Bob is married to Alice.
- "is a parent of": **No.** If Alice is Bob's parent, Bob is not Alice's parent.
- "is within 10km of": **Yes.** Distance is symmetric.
- "loves": **No.** Love is not always reciprocated.

3.3.3 Transitive

Plain English: If w is related to v, and v is related to u, then w is related to u. The relation "chains."

Formally: For all w, v, u: $(wRv \wedge vRu) \Rightarrow wRu$.

Example 3.5 (Transitive or not?)**.**

- "is an ancestor of": **Yes.** If A is an ancestor of B, and B is an ancestor of C, then A is an ancestor of C.
- "is a parent of": **No.** Your parent's parent is not your parent.
- "$<$" on numbers: **Yes.** If $a < b$ and $b < c$, then $a < c$.
- "is a friend of": **No.** Your friend's friend might be a stranger.

3.3.4 Equivalence Relations

Definition 3.6 (Equivalence Relation)**.** A relation that is **reflexive**, **symmetric**, and **transitive** is called an *equivalence relation*.

Intuition

An equivalence relation partitions the set into groups where everything in the same group is "equivalent." Think of it as a way of saying "these things are the same for our purposes."

Example 3.7 (Equivalence relations)**.**

- "has the same birthday as": reflexive (you share your birthday with yourself), symmetric (if you share with me, I share with you), transitive (if A shares with B and B shares with C, then A shares with C).
- "is congruent to modulo 3": partitions integers into three classes: $\{..., -3, 0, 3, 6, ...\}$, $\{..., -2, 1, 4, 7, ...\}$, $\{..., -1, 2, 5, 8, ...\}$.

Now we're ready for Kripke frames.

3.4 Kripke Frames and Models

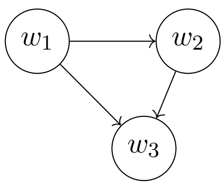
Definition 3.8 (Kripke Frame)**.** A *Kripke frame* is a pair $\mathcal{F} = (W, R)$ where:

- W is a non-empty set of *possible worlds*
- $R \subseteq W \times W$ is the *accessibility relation*

Intuition

Think of it as a directed graph. Worlds are nodes. If wRv, there's an arrow from w to v, meaning "from w's perspective, v is a possibility."

Example 3.9 (A simple frame)**.** Let $W = \{w_1, w_2, w_3\}$ with $R = \{(w_1, w_2), (w_1, w_3), (w_2, w_3)\}$.



From w_1, both w_2 and w_3 are accessible. From w_2, only w_3 is accessible. From w_3, nothing is accessible.

To evaluate formulas, we need to know which propositions are true at which worlds:

Definition 3.10 (Kripke Model)**.** A *Kripke model* is a triple $\mathcal{M} = (W, R, V)$ where:

- (W, R) is a Kripke frame
- $V : \text{Prop} \to \mathcal{P}(W)$ is a *valuation*—for each proposition p, $V(p)$ is the set of worlds where p is true

3.5 Satisfaction: When is a Formula True?

Now the key question: given a model $\mathcal{M}$ and a world w, when is a formula φ true?

We write $\mathcal{M}, w \models \varphi$ to mean "φ is true at world w in model $\mathcal{M}$."

Definition 3.11 (Satisfaction Relation)**.** Let $\mathcal{M} = (W, R, V)$ and $w \in W$. We define $\models$ recursively:

$$
\begin{array}{lll}
\mathcal{M}, w \models p & \text{iff} & w \in V(p) \\
& & \textit{“p is true at w”} \\
\mathcal{M}, w \models \neg\varphi & \text{iff} & \mathcal{M}, w \not\models \varphi \\
& & \textit{“}\varphi\textit{ is not true at w”} \\
\mathcal{M}, w \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\
& & \textit{“both are true at w”} \\
\mathcal{M}, w \models \Box\varphi & \text{iff} & \text{for all } v \text{ with } wRv\text{: } \mathcal{M}, v \models \varphi \\
& & \textit{“}\varphi\textit{ is true at ALL accessible worlds”} \\
\mathcal{M}, w \models \Diamond\varphi & \text{iff} & \text{there exists } v \text{ with } wRv\text{: } \mathcal{M}, v \models \varphi \\
& & \textit{“}\varphi\textit{ is true at SOME accessible world”}
\end{array}
$$

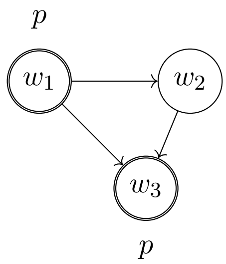
Intuition

$\Box$ means "in all accessible worlds" and $\Diamond$ means "in some accessible world."

If no worlds are accessible from w, then:

- $\Box\varphi$ is **vacuously true** (there's no accessible world where φ could fail)
- $\Diamond\varphi$ is **false** (there's no accessible world where φ could hold)

Example 3.12 (Evaluating formulas)**.** Consider this model with $V(p) = \{w_1, w_3\}$ (meaning p is true at w_1 and w_3):



Let's evaluate some formulas at w_1:

- $\mathcal{M}, w_1 \models p$? **Yes**, because $w_1 \in V(p)$.
- $\mathcal{M}, w_1 \models \Box p$? **No**, because w_2 is accessible from w_1, but p is false at w_2.
- $\mathcal{M}, w_1 \models \Diamond p$? **Yes**, because w_3 is accessible and p is true there.
- $\mathcal{M}, w_2 \models \Box p$? **Yes**, because the only world accessible from w_2 is w_3, and p is true there.

3.6 Frame Properties and Axioms

Here's where it gets interesting: the *properties of the accessibility relation* determine which formulas are valid.

Let's go through them one by one, with plain English first.

3.6.1 Reflexivity and Axiom T

Property: R is reflexive—every world can access itself.

Plain English meaning: "What is necessarily true is actually true." If something holds in ALL accessible worlds, and the current world is accessible from itself, then it holds here too.

Axiom T: $\Box\varphi \to \varphi$

Read as: "If φ is necessarily true, then φ is true."

Intuition

This axiom says you can't escape reality. If something is true in all possibilities, it's true in this one—because this one is a possibility. Without reflexivity, you could have $\Box\varphi$ true at w (all accessible worlds satisfy φ) while φ is false at w itself (because w doesn't access itself).

3.6.2 Transitivity and Axiom 4

Property: R is transitive—if w accesses v, and v accesses u, then w accesses u.

Plain English meaning: "What is necessarily true is necessarily necessarily true." If φ holds in all accessible worlds, then from any of those worlds, φ still holds in all *their* accessible worlds.

Axiom 4: $\Box\varphi \to \Box\Box\varphi$

Read as: "If φ is necessarily true, then it's necessarily necessary."

Intuition

This is about *positive introspection* in epistemic logic: if you know something, you know that you know it.
Without transitivity, you might know φ (it's true in all worlds you consider possible), but from some of those worlds, there are further worlds you can't "see" from here—and φ might fail there.

3.6.3 Symmetry and Axiom B

Property: R is symmetric—if w accesses v, then v accesses w.

Plain English meaning: "What is true is necessarily possible." If φ is true here, then from any accessible world, this world is accessible back, so φ is possible there.

Axiom B: $\varphi \to \Box\Diamond\varphi$

Read as: "If φ is true, then necessarily φ is possible."

Intuition

This says you can always "get back." If from here, world v looks possible, then from v, here looks possible too.

3.6.4 Euclideanness and Axiom 5

Property: R is Euclidean—if w accesses both v and u, then v accesses u.

Plain English meaning: "What is possible is necessarily possible." If φ is possible (true at some accessible world v), then from any accessible world u, you can still reach v.

Axiom 5: $\Diamond\varphi \to \Box\Diamond\varphi$

Read as: "If φ is possible, then it's necessarily possible."

Intuition

This is about *negative introspection*: if you don't know something (i.e., it's possible that not-φ), then you know that you don't know it. With a Euclidean relation, all accessible worlds can see each other.

3.6.5 Summary Table

Property	Axiom	Name	Plain reading
Reflexive	$\Box\varphi \to \varphi$	T	"Necessary implies true"
Transitive	$\Box\varphi \to \Box\Box\varphi$	4	"Necessary implies necessarily necessary"
Symmetric	$\varphi \to \Box\Diamond\varphi$	B	"True implies necessarily possible"
Euclidean	$\Diamond\varphi \to \Box\Diamond\varphi$	5	"Possible implies necessarily possible"

Historical Note

The names T, 4, B, 5 come from C.I. Lewis's systems S1–S5 of modal logic. S5, the strongest, assumes R is an equivalence relation (reflexive + symmetric + transitive, or equivalently reflexive + Euclidean). In S5, the modal operators collapse: $\Box\Box\varphi \equiv \Box\varphi$ and $\Diamond\Box\varphi \equiv \Box\varphi$.

3.7 Bisimulation: The Right Notion of Equivalence

When are two models "the same" from a modal logic perspective?

Not when they're literally identical—that's too strict. We want: **two models are equivalent if no modal formula can tell them apart.**

Definition 3.13 (Bisimulation)**.** Let $\mathcal{M} = (W, R, V)$ and $\mathcal{M}' = (W', R', V')$ be Kripke models. A relation $Z \subseteq W \times W'$ is a *bisimulation* if whenever wZw':

1. **(Atoms)** w and w' agree on all propositions: $w \in V(p) \Leftrightarrow w' \in V'(p)$ for all p
2. **(Zig)** If w can step to v (i.e., wRv), then w' can step to some v' with vZv'
3. **(Zag)** If w' can step to v' (i.e., $w'R'v'$), then w can step to some v with vZv'

We write $w \underline{\leftrightarrow} w'$ if there exists a bisimulation Z with wZw'.

Intuition

Bisimulation says: "whatever move you make, I can match it." It's like a game:

- If you step from w to v, I can step from w' to some v' that's still bisimilar to v.
- And vice versa.

Neither player can "escape" to a state the other can't match.

Theorem 3.14 (Bisimulation Invariance)**.** *If $w \underline{\leftrightarrow} w'$, then for all modal formulas φ:*

$$\mathcal{M}, w \models \varphi \iff \mathcal{M}', w' \models \varphi$$

Proof Sketch

By induction on φ:

- Base case (p): By the Atoms condition, $w \in V(p) \Leftrightarrow w' \in V'(p)$.
- Boolean cases ($\neg$, $\wedge$): By induction hypothesis.
- Modal case ($\Box\varphi$): If $\mathcal{M}, w \models \Box\varphi$, then φ holds at all successors of w. For any successor v' of w', by Zag there's a matching successor v of w with vZv'. By IH, φ holds at v'. So $\mathcal{M}', w' \models \Box\varphi$. Similarly for the other direction using Zig.

Key Insight

Bisimulation is the "right" equivalence for modal logic:

- Two bisimilar states satisfy exactly the same modal formulas.
- Conversely, over image-finite models, states satisfying the same formulas are bisimilar (Hennessy-Milner theorem).

This notion will generalize to **coalgebra**, where it becomes the universal notion of behavioral equivalence.

Summary

- A **relation** on W is a set of pairs $R \subseteq W \times W$
- Relations can be reflexive, symmetric, transitive (and combinations)
- An **equivalence relation** is all three
- A **Kripke frame** is a set of worlds + an accessibility relation
- A **Kripke model** adds a valuation saying which propositions are true where
- $\Box\varphi$ means "true at all accessible worlds"; $\Diamond\varphi$ means "true at some"
- Frame properties (reflexive, transitive, ...) correspond to axioms (T, 4, ...)
- **Bisimulation** is the right notion of equivalence: bisimilar states satisfy the same modal formulas

Chapter 4

Bisimulation

Goals of This Chapter

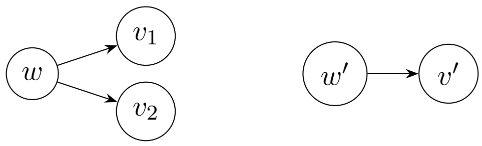
- Understand bisimulation as the fundamental equivalence for modal logic
- Master the zig-zag conditions
- See bisimulation games as an intuitive characterization
- Connect to the later coalgebraic generalization

4.1 The Problem of Equivalence

When are two models "the same" from a modal perspective?

Not isomorphism—too strict. Two very different-looking models might satisfy exactly the same formulas.

Example 4.1. Consider:



If v_1, v_2, v' all satisfy the same atoms, then w and w' satisfy the same modal formulas—even though the models have different structure.

4.2 Definition

Definition 4.2 (Bisimulation)**.** Let $\mathcal{M} = (W, R, V)$ and $\mathcal{M}' = (W', R', V')$. A relation $Z \subseteq W \times W'$ is a *bisimulation* if whenever $w\ Z\ w'$:

(Atoms) For all $p \in \Phi$: $w \in V(p) \Leftrightarrow w' \in V'(p)$

(Zig) If wRv, then there exists $v' \in W'$ such that $w'R'v'$ and $v \; Z \; v'$
(Zag) If $w'R'v'$, then there exists $v \in W$ such that wRv and $v \; Z \; v'$

We write $w \underline{\leftrightarrow} w'$ if there exists a bisimulation relating w and w'.

Intuition

Zig: "I can match your move."
Zag: "You can match my move."
Two states are bisimilar if they can "simulate" each other, step by step.

4.3 Bisimulation Games

Bisimulation has a game-theoretic interpretation.
Players: Spoiler (tries to distinguish) vs Duplicator (tries to match)
Game: Start at (w, w'). Each round:

1. Spoiler picks a side and makes a move (follows an edge)
2. Duplicator must match on the other side

Winning:

- Spoiler wins if atoms differ, or Duplicator can't move
- Duplicator wins if the game continues forever

Theorem 4.3. *$w \underline{\leftrightarrow} w'$ iff Duplicator has a winning strategy.*

4.4 The Invariance Theorem

Theorem 4.4 (Bisimulation Invariance)**.** *If $w \underline{\leftrightarrow} w'$, then for all modal formulas φ:*

$$\mathcal{M}, w \models \varphi \iff \mathcal{M}', w' \models \varphi$$

Proof. Induction on φ.
Base: $\varphi = p$. By (Atoms).
Step $\neg$: Immediate from IH.
Step $\wedge$: Immediate from IH.
Step $\Box$: Suppose $\mathcal{M}, w \models \Box\psi$ and $w \; Z \; w'$.

- Take any v' with $w'R'v'$
- By (Zag), exists v with wRv and $v \; Z \; v'$
- Since $\mathcal{M}, w \models \Box\psi$, we have $\mathcal{M}, v \models \psi$
- By IH, $\mathcal{M}', v' \models \psi$

So $\mathcal{M}', w' \models \Box\psi$. The other direction uses (Zig). $\square$

4.5 Largest Bisimulation

Proposition 4.5. *The union of all bisimulations is itself a bisimulation—the* largest *bisimulation.*

Definition 4.6. $w \sim w'$ (bisimilarity) iff (w, w') is in the largest bisimulation.

4.6 Looking Ahead: Coalgebraic Bisimulation

The zig-zag conditions generalize beyond Kripke frames.

For any coalgebra $(X, \gamma : X \to FX)$, there's a notion of bisimulation. For the powerset functor $F = \mathcal{P}$, it coincides exactly with Kripke bisimulation.

★ Key Insight

Bisimulation is not an accident of Kripke semantics. It's a fundamental concept that exists for *any* coalgebra. This is why it will reappear throughout the book.

Chapter 5

Model Constructions

Goals of This Chapter

- Master bisimulation and its equivalent characterizations
- Understand bounded morphisms (p-morphisms)
- Learn model constructions: generated submodels, disjoint unions, unravelling
- Prove the Hennessy-Milner theorem

5.1 Bisimulation Revisited

Recall: a bisimulation relates states that are "behaviorally equivalent."

Definition 5.1 (Bisimulation)**.** $Z \subseteq W \times W'$ is a bisimulation if wZw' implies:

1. **(Atoms)** Same atomic propositions
2. **(Zig)** Every successor of w is matched by a successor of w'
3. **(Zag)** Every successor of w' is matched by a successor of w

Theorem 5.2 (Bisimulation Invariance)**.** *Bisimilar states satisfy the same modal formulas.*

Proof. Induction on formula structure. □

5.2 Bounded Morphisms

Definition 5.3 (Bounded Morphism / p-morphism)**.** A function $f : W \to W'$ is a *bounded morphism* if:

1. $w \in V(p) \Leftrightarrow f(w) \in V'(p)$
2. $wRv \Rightarrow f(w)R'f(v)$ (homomorphism)
3. $f(w)R'v' \Rightarrow \exists v : wRv \wedge f(v) = v'$ (back condition)

Proposition 5.4. *If f is a bounded morphism, then its graph is a bisimulation.*

5.3 Generated Submodels

Definition 5.5 (Generated Submodel)**.** The submodel *generated by w* is the restriction to all worlds reachable from w.

Theorem 5.6. *A world satisfies the same formulas in the full model and in its generated submodel.*

5.4 Disjoint Union

Definition 5.7. The *disjoint union* $\mathcal{M}_1 \uplus \mathcal{M}_2$ combines two models without connecting them.

Proposition 5.8. *Each component of a disjoint union is bisimilar to itself in the union.*

5.5 Tree Unravelling

Definition 5.9 (Unravelling)**.** The *unravelling* of $\mathcal{M}$ from w is a tree where:

- Nodes are finite paths starting from w
- Edges extend paths by one step
- Valuation inherited from endpoints

Theorem 5.10. *Every pointed model is bisimilar to its unravelling.*

Key Insight

Modal logic cannot distinguish a model from its tree unravelling. This is why modal logic has the *tree model property*.

5.6 Hennessy-Milner Theorem

Theorem 5.11 (Hennessy-Milner)**.** *On* image-finite *models, bisimilarity coincides with modal equivalence.*

Intuition

On finite-branching models, if two states satisfy exactly the same modal formulas, they must be bisimilar. The converse always holds; this theorem says that for nice enough models, the two notions coincide exactly.

Chapter 6

Completeness

Goals of This Chapter

- Understand normal modal logics and their axiomatizations
- Master the canonical model construction
- Prove completeness for K, T, S4, S5
- Understand what canonicity means and why it matters

6.1 Normal Modal Logics

Definition 6.1 (Normal Modal Logic)**.** A *normal modal logic* is a set L of formulas containing:

- All propositional tautologies
- The **K axiom**: $\Box(p \to q) \to (\Box p \to \Box q)$

and closed under:

- Modus ponens: from φ and $\varphi \to \psi$, derive ψ
- Necessitation: from φ, derive $\Box\varphi$

Example 6.2.

- **K** = minimal normal modal logic
- **T** = K + $\Box p \to p$ (reflexivity)
- **S4** = T + $\Box p \to \Box\Box p$ (transitivity)
- **S5** = S4 + $p \to \Box\Diamond p$ (symmetry)

6.2 Maximal Consistent Sets

Definition 6.3 (Maximal Consistent Set)**.** A set Γ of formulas is *maximally L-consistent* if:

- Γ is L-consistent (no derivation of $\bot$)
- Γ is maximal: for all φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$

Lemma 6.4 (Lindenbaum)**.** *Every consistent set can be extended to a maximally consistent set.*

Proof. Enumerate all formulas. Add each one if consistent, otherwise add its negation. □

6.3 The Canonical Model

Definition 6.5 (Canonical Model)**.** The *canonical model* for logic L is $\mathcal{M}^c_L = (W^c, R^c, V^c)$ where:

- W^c = all maximally L-consistent sets
- $\Gamma R^c \Delta$ iff for all φ: $\Box\varphi \in \Gamma \Rightarrow \varphi \in \Delta$
- $V^c(p) = \{\Gamma : p \in \Gamma\}$

Lemma 6.6 (Truth Lemma)**.** $\mathcal{M}^c_L, \Gamma \models \varphi$ *iff* $\varphi \in \Gamma$*.*

Proof. Induction on φ. The key case is $\Box\varphi$:

- If $\Box\varphi \in \Gamma$ and $\Gamma R^c \Delta$, then $\varphi \in \Delta$, so by IH $\Delta \models \varphi$.
- If $\Box\varphi \notin \Gamma$, construct Δ with $\varphi \notin \Delta$ and $\Gamma R^c \Delta$.

□

6.4 Completeness Theorem

Theorem 6.7 (Completeness of K)**.** *If φ is valid in all Kripke frames, then φ is provable in K.*

Proof. Contrapositive. If φ is not provable, then $\{\neg\varphi\}$ is consistent. Extend to MCS Γ. By Truth Lemma, $\mathcal{M}^c_K, \Gamma \models \neg\varphi$, so φ is not valid. □

6.5 Canonicity

Definition 6.8 (Canonical Formula)**.** A formula φ is *canonical* if: whenever $\varphi \in L$, the canonical frame for L validates φ.

Key Insight

Canonicity is the bridge between syntax (axioms) and semantics (frame conditions). If an axiom is canonical, adding it to a logic preserves completeness.

Example 6.9. The T axiom $\Box p \to p$ is canonical: if $\mathrm{T} \in L$, then R^c is reflexive.

Chapter 7

Decidability

Goals of This Chapter

- Understand the finite model property (FMP)
- Master the filtration technique
- Prove decidability of K, T, S4, S5
- Know the complexity classes involved

7.1 The Finite Model Property

Definition 7.1 (Finite Model Property)**.** A logic L has the *finite model property* (FMP) if: every L-consistent formula is satisfiable in a *finite* model.

Theorem 7.2. *If L is finitely axiomatizable and has FMP, then L is decidable.*

Proof. To decide if $\varphi \in L$:

- Search for a proof of φ (complete enumeration)
- Search for a finite countermodel to φ

One must succeed. Both searches are effective. □

7.2 Filtration

The key technique for proving FMP.

Definition 7.3 (Closure)**.** The *closure* $\mathrm{cl}(\varphi)$ is the smallest set containing φ and all its subformulas, closed under single negation.

Note: $|\mathrm{cl}(\varphi)| \leq 2 \cdot |\varphi|$.

Definition 7.4 (Filtration)**.** Let $\mathcal{M} = (W, R, V)$ and $\Sigma = \mathrm{cl}(\varphi)$. Define equivalence:

$$w \equiv_\Sigma v \iff \forall \psi \in \Sigma : (\mathcal{M}, w \models \psi \Leftrightarrow \mathcal{M}, v \models \psi)$$

A *filtration* of $\mathcal{M}$ through Σ is $(W/{\equiv_\Sigma}, R', V')$ where:

- $V'(p) = \{[w] : w \in V(p)\}$ for $p \in \Sigma$
- R' satisfies: $wRv \Rightarrow [w]R'[v]$ (min condition)
- R' satisfies: $[w]R'[v] \wedge \Box\psi \in \Sigma \wedge \mathcal{M}, w \models \Box\psi \Rightarrow \mathcal{M}, v \models \psi$ (max condition)

Theorem 7.5 (Filtration Lemma)**.** *For any filtration $\mathcal{M}'$ of $\mathcal{M}$ through Σ:*

$$\mathcal{M}, w \models \psi \iff \mathcal{M}', [w] \models \psi \quad \textit{for all } \psi \in \Sigma$$

Corollary 7.6. $|W/{\equiv_\Sigma}| \leq 2^{|\Sigma|}$*, so the filtration is finite.*

7.3 FMP for Standard Logics

Theorem 7.7. *K, T, S4, S5 all have the finite model property.*

Proof Sketch

- K: smallest filtration works
- T: take reflexive closure
- S4: take reflexive-transitive closure
- S5: take equivalence closure

7.4 Complexity

Logic	Complexity
K, T, S4	PSPACE-complete
S5	NP-complete
PDL	EXPTIME-complete

Key Insight

Modal logic satisfiability is generally PSPACE-complete. This is "just barely" tractable—hard, but not as bad as undecidable.

Chapter 8

Correspondence and Sahlqvist's Theorem

Goals of This Chapter

- Understand frame correspondence deeply
- Learn what Sahlqvist formulas are
- Appreciate why Sahlqvist's theorem is powerful
- Know that this has been mechanized

8.1 The Correspondence Problem

Recall: some modal axioms correspond to first-order frame conditions.

$$\begin{array}{ll} \Box p \to p & \forall x. Rxx \\ \Box p \to \Box\Box p & \forall xyz. Rxy \land Ryz \to Rxz \\ \Diamond p \to \Box\Diamond p & \forall xy. Rxy \to \forall z. Rxz \to Ryz \end{array}$$

Question: Which modal formulas correspond to first-order conditions?
Answer: Not all! The Löb axiom $\Box(\Box p \to p) \to \Box p$ corresponds to a *second-order* condition (well-foundedness).

8.2 Sahlqvist Formulas

Definition 8.1 (Sahlqvist Formula (simplified))**.** A *Sahlqvist formula* has the form:

$$\varphi \to \psi$$

where:

- φ is *positive* in $\Box$ and built from boxed atoms ($\Box^n p$), negative formulas, and $\wedge, \vee$
- ψ is *positive* (no negations in front of atoms)

Example 8.2. All the standard axioms are Sahlqvist:

- $\Box p \to p$ (T)
- $\Box p \to \Box\Box p$ (4)
- $p \to \Box\Diamond p$ (B)
- $\Diamond p \to \Box\Diamond p$ (5)

8.3 Sahlqvist's Theorem

Theorem 8.3 (Sahlqvist 1975)**.** *Every Sahlqvist formula:*

1. *Corresponds to a first-order frame condition (computable from the formula)*
2. *Is canonical (hence its logic is complete)*

Key Insight

Sahlqvist's theorem is a "cookbook" for modal logic:

1. Write down your axiom (if Sahlqvist)
2. Automatically get the frame condition
3. Automatically get completeness

No need to construct canonical models by hand!

8.4 The Algorithm (SQEMA)

There's an algorithm called **SQEMA** (Sahlqvist-van Benthem algorithm) that:

- Takes a modal formula as input
- Outputs the corresponding first-order condition (if it exists)

The algorithm works by:

1. Translate to first-order logic with the standard translation
2. Apply Ackermann's lemma to eliminate second-order quantifiers
3. If successful, output the first-order result

8.5 Mechanization

Historical Note

Sahlqvist's theorem has been formalized in proof assistants:

- Isabelle/HOL: formalized by various authors
- Lean: partial formalizations exist

This is one of the benchmark results for modal logic mechanization.

Exercise

1. Verify that the McKinsey axiom $\Box\Diamond p \to \Diamond\Box p$ is *not* Sahlqvist. (Hint: it's not canonical for the expected frame class.)

2. Use the standard translation to find the frame condition for $\Box(p \vee q) \to \Box p \vee \Box q$. Is this formula valid on all frames?

Chapter 9

Graded Modal Logic

Goals of This Chapter

- Go beyond $\forall/\exists$: counting quantifiers
- Understand graded modalities: $\Diamond_{\geq n}$, $\Diamond_{\leq n}$
- See majority and probabilistic quantifiers
- Appreciate the trade-offs (expressiveness vs decidability)

9.1 The Limitation of $\Box/\Diamond$

Standard modal logic has:

- $\Box\varphi$ = "all accessible worlds satisfy φ" ($\forall$)
- $\Diamond\varphi$ = "some accessible world satisfies φ" ($\exists$)

But natural reasoning uses many more quantifiers:

- "*Most* options are safe"
- "*At least 3* escape routes exist"
- "*Exactly one* successor state"
- "With *probability* ≥ 0.9, success"

Common Pitfall

"At least 3 safe options" is very different from "some safe option." Standard modal logic can't distinguish them.

9.2 Graded Modalities

Definition 9.1 (Graded Modal Logic)**.** Extend the language with counting modalities:

$$\begin{array}{ll} \Diamond_{\geq n}\varphi & \text{“at least } n \text{ accessible worlds satisfy } \varphi\text{”} \\ \Diamond_{\leq n}\varphi & \text{“at most } n \text{ accessible worlds satisfy } \varphi\text{”} \\ \Diamond_{=n}\varphi & \text{“exactly } n \text{ accessible worlds satisfy } \varphi\text{”} \end{array}$$

Definition 9.2 (Semantics)**.**

$$\mathcal{M}, w \models \Diamond_{\geq n}\varphi \iff |\{v : wRv \text{ and } \mathcal{M}, v \models \varphi\}| \geq n$$

Note: $\Diamond\varphi \equiv \Diamond_{\geq 1}\varphi$ and $\Box\varphi \equiv \Diamond_{\leq 0}\neg\varphi$.

9.3 Majority Quantifier

Definition 9.3 (Majority Modality)**.**

$$\mathsf{M}\varphi \quad \text{“most accessible worlds satisfy } \varphi\text{”}$$

Semantics:

$$\mathcal{M}, w \models \mathsf{M}\varphi \iff |\{v : wRv \wedge v \models \varphi\}| > |\{v : wRv \wedge v \not\models \varphi\}|$$

Key Insight

The majority quantifier M cannot be defined using $\forall/\exists$ (Lindström). It's genuinely more expressive.

9.4 Probabilistic Modality

Definition 9.4 (Probabilistic Modal Logic)**.** Given a probability distribution μ_w over successors of w:

$$\mathcal{M}, w \models P_{\geq r}\varphi \iff \mu_w(\{v : wRv \wedge v \models \varphi\}) \geq r$$

This requires more structure: not just accessibility R, but a probability measure.

9.5 Meta-Theoretic Trade-offs

Adding quantifiers has costs:

Property	**Basic Modal**	**Graded/Majority**
Compactness	✓	Often fails
Finite model property	✓	Often fails
Decidability	PSPACE	Higher complexity

Theorem 9.5 (Lindström)**.** *First-order logic is the unique logic with compactness and Löwenheim-Skolem. Adding quantifiers breaks at least one.*

9.6 Connection to Description Logic

Description logics (used in OWL, knowledge graphs) have number restrictions:

$$(\geq 3 \ \mathsf{hasChild.Doctor}) \quad \text{"at least 3 children are doctors"}$$
$$(\leq 1 \ \mathsf{hasSpouse}) \quad \text{"at most 1 spouse"}$$

These are graded modalities in disguise!

9.7 Why This Matters

For AI agents:

- "Ensure at least 2 fallback options" (robustness)
- "Most paths lead to success" (probabilistic planning)
- "Exactly one next state" (determinism)

Standard modal logic can't express these. Graded modal logic can.

Chapter 10

Dynamic Logic

Goals of This Chapter

- Understand PDL: modal logic for programs
- See how actions become modalities
- Appreciate the connection to program verification
- Connect to coalgebra (labeled transition systems)

10.1 Programs as Modalities

So far: $\Box\varphi$ means "in all accessible worlds, φ."

New idea: what if accessibility is *labeled by programs*?

Definition 10.1 (PDL Syntax)**.** Programs α and formulas φ are mutually defined:

$$\alpha ::= a \mid \alpha;\beta \mid \alpha \cup \beta \mid \alpha^* \mid \varphi?$$
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid [\alpha]\varphi$$

where a is an atomic action.

Program	Meaning
a	atomic action
$\alpha;\beta$	do α, then β
$\alpha \cup \beta$	choose α or β (nondeterministic)
α^*	repeat α zero or more times
$\varphi?$	test: proceed if φ, else fail

10.2 Semantics

Definition 10.2 (PDL Model)**.** A PDL model is $(W, \{R_a\}_{a \in \mathrm{Act}}, V)$ where each atomic action a has its own accessibility relation R_a.

Extend to compound programs:

$$R_{\alpha;\beta} = R_\alpha \circ R_\beta \quad \text{(composition)}$$
$$R_{\alpha \cup \beta} = R_\alpha \cup R_\beta \quad \text{(union)}$$
$$R_{\alpha^*} = R_\alpha^* \quad \text{(reflexive-transitive closure)}$$
$$R_{\varphi?} = \{(w, w) : w \models \varphi\} \quad \text{(identity on } \varphi\text{-states)}$$

Definition 10.3 (Satisfaction)**.**

$$\mathcal{M}, w \models [\alpha]\varphi \iff \forall v : (w, v) \in R_\alpha \Rightarrow \mathcal{M}, v \models \varphi$$

"After any execution of α, φ holds."

Dually: $\langle\alpha\rangle\varphi \equiv \neg[\alpha]\neg\varphi$ means "α can lead to a state where φ."

10.3 Examples

Example 10.4 (Program Correctness)**.**

$$\mathsf{pre} \to [\mathsf{program}]\mathsf{post}$$

"If precondition holds, then after running the program, postcondition holds."

This is a Hoare triple $\{\mathsf{pre}\}\mathsf{program}\{\mathsf{post}\}$ expressed in modal logic!

Example 10.5 (Loop Invariant)**.**

$$\mathsf{inv} \to [(\mathsf{cond}?; \mathsf{body})^*](\neg\mathsf{cond} \to \mathsf{inv})$$

"If invariant holds, after any number of loop iterations, when loop exits, invariant still holds."

Example 10.6 (Agent Planning)**.**

$$\mathsf{atHome} \to \langle\mathsf{drive}; \mathsf{park}\rangle\mathsf{atWork}$$

"From home, there exists a way (drive then park) to reach work."

10.4 PDL and Automata

Programs in PDL correspond to regular expressions:

- ; = concatenation
- $\cup$ = union
- $*$ = Kleene star

Theorem 10.7. *PDL model checking is in P. PDL satisfiability is EXPTIME-complete.*

10.5 Connection to Coalgebra

A PDL model is a *labeled transition system*:

$$\gamma : W \to \prod_{a \in \mathrm{Act}} \mathcal{P}(W)$$

This is a coalgebra for the functor $F(X) = (\mathcal{P}(X))^{\mathrm{Act}}$.

Key Insight

PDL semantics is coalgebraic. Programs are built from atomic actions using regular operations. This connects modal logic of programs to automata theory.

10.6 Extensions

- **Game Logic**: players alternate, $[\alpha^d]\varphi$ = "player can ensure φ"
- **Concurrent PDL**: parallel composition $\alpha \| \beta$
- **μ-calculus**: add fixed points, subsumes PDL and CTL

Chapter 11

Automata and the Chomsky Hierarchy

Goals of This Chapter

- Understand automata as another way to describe "behavior"
- See the Chomsky hierarchy: what can be recognized by what machine
- Prepare for the unification: Kripke frames and automata are both coalgebras

11.1 Finite Automata

Definition 11.1 (Deterministic Finite Automaton)**.** A *DFA* is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is a finite alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of accepting states

Intuition

A DFA is a machine that reads a string symbol by symbol, transitioning between states. At the end, it accepts or rejects based on whether it's in an accepting state.

Definition 11.2 (Non-deterministic Finite Automaton)**.** An *NFA* allows multiple transitions: $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

Theorem 11.3. *DFA and NFA recognize the same class of languages: the* regular languages.

11.2 The Chomsky Hierarchy

Type	Language Class	Machine	Example
Type 3	Regular	Finite automaton	a^*b^*
Type 2	Context-free	Pushdown automaton	a^nb^n
Type 1	Context-sensitive	Linear bounded	$a^nb^nc^n$
Type 0	Recursively enumerable	Turing machine	Halting problem

Key Insight

Each level adds computational power:

- Finite automaton: finite memory (just the state)
- Pushdown: unbounded stack
- Linear bounded: tape proportional to input
- Turing: unbounded tape

11.3 Automata as Transition Systems

Notice the similarity:

Structure	States	Transitions
Kripke frame	Worlds W	$R \subseteq W \times W$
DFA	States Q	$\delta : Q \times \Sigma \to Q$
NFA	States Q	$\delta : Q \times \Sigma \to \mathcal{P}(Q)$
Labeled TS	States S	$\to \subseteq S \times \Sigma \times S$

They're all "states + transitions." This is not a coincidence.

Intuition

Both Kripke frames and automata describe *systems that can be in states and move between states*. Modal logic asks "what is true here?" Automata theory asks "what strings are accepted?" But the underlying structure is the same.

11.4 Looking Ahead

The next chapter introduces **coalgebra**, which unifies all these structures under one framework:

$$\text{state} \to F(\text{state})$$

Different choices of the functor F give Kripke frames, DFAs, NFAs, Markov chains, streams, and more.

Exercise

1. Draw a DFA that accepts strings with an even number of a's.
2. Can a finite automaton recognize $\{a^n b^n : n \geq 0\}$? Why or why not?
3. What's the relationship between a Kripke frame and an NFA without accepting states?

Chapter 12

Beyond Kripke: Coalgebra

Goals of This Chapter

- See that Kripke frames are just one instance of a general pattern
- Understand coalgebra as the mathematics of state-based systems
- Appreciate the unification: automata, Markov chains, streams

12.1 The Pattern

Look at these structures:

Structure	**Transition**
Kripke frame	$W \to \mathcal{P}(W)$
Deterministic automaton	$Q \to 2 \times Q^A$
Markov chain	$S \to \mathcal{D}(S)$
Stream	$S \to A \times S$

They all have the form: **state** $\to$ **F(state)** for some functor F.

Intuition

A coalgebra answers: "What can I observe from this state, and what states can I reach next?" It's the mathematics of *behavior*.

12.2 Coalgebras

Definition 12.1 (Coalgebra)**.** Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor. An F-*coalgebra* is a pair (X, γ) where:

- X is a set of *states*
- $\gamma : X \to F(X)$ is the *transition structure*

Example 12.2 (Kripke Frame as Coalgebra)**.** A Kripke frame (W, R) is a $\mathcal{P}$-coalgebra: $\gamma(w) = \{v : wRv\}$.

12.3 Final Coalgebra

Definition 12.3 (Final Coalgebra)**.** A *final* F-coalgebra (Ω, ω) is one where for any (X, γ), there exists a unique morphism $X \to \Omega$.

Key Insight

The final coalgebra is the "universe of all possible behaviors." Two states are equivalent iff they map to the same element in Ω.

Chapter 13

How Much Can We Express?

◎ Goals of This Chapter

- Understand expressiveness as a measure of logical power
- Compare modal logic to first-order logic
- Introduce correspondence theory

13.1 The Standard Translation

Definition 13.1 (Standard Translation)**.** Define $\mathrm{ST}_x : \mathcal{L}(\Box) \to \mathcal{L}_{\mathrm{FO}}$:

$$\begin{aligned} \mathrm{ST}_x(p) &= P(x) \\ \mathrm{ST}_x(\neg\varphi) &= \neg \mathrm{ST}_x(\varphi) \\ \mathrm{ST}_x(\Box\varphi) &= \forall y (R(x, y) \to \mathrm{ST}_y(\varphi)) \end{aligned}$$

Modal logic is a *fragment* of first-order logic. But which fragment?

13.2 The van Benthem Characterization

Theorem 13.2 (van Benthem)**.** *A first-order formula $\varphi(x)$ is equivalent to a modal formula iff it is invariant under bisimulation.*

★ Key Insight

Modal logic = bisimulation-invariant fragment of first-order logic.

13.3 Frame Correspondence

Some modal formulas correspond to first-order conditions on frames:

Modal Axiom	**Frame Condition**
$\Box p \to p$	Reflexive
$\Box p \to \Box\Box p$	Transitive
$p \to \Box\Diamond p$	Symmetric

Part II

Learning

Chapter 14

What is Learning?

Goals of This Chapter

- Ask: what does it mean to "learn"?
- Survey mathematical notions of "getting closer"
- Discover that most aren't operational
- Motivate our choice: gradient descent + semiring

14.1 The Intuition

Learning feels like **getting closer** to something.

You start far from the answer. You try something. You see how well it works. You adjust. You try again. Gradually, you approach the target.

Intuition

A child learning to throw a ball:

1. Throw. Miss.
2. Adjust. Throw. Closer.
3. Adjust. Throw. Hit!

Each attempt is "closer" to the goal. Learning is the process of closing the gap.

Can we make this precise? What mathematical structures capture "approaching"?

14.2 Mathematical Notions of Approximation

14.2.1 Topology

Topology defines "closeness" without distance.

- A sequence $x_n \to x$ means: every neighborhood of x eventually contains all x_n
- Continuous functions preserve convergence

Problem: Topology *describes* convergence but doesn't tell you *how* to get there. No algorithm.

14.2.2 Metric Spaces

Add a distance function $d(x, y)$.

- "Closer" means smaller d
- Cauchy sequences, completeness

Problem: Having a metric doesn't give you a method to minimize it.

14.2.3 Order / Domain Theory

An information ordering: $x \sqsubseteq y$ means "y is more defined than x."

- Start with $\perp$ (no information)
- Iteratively refine: $\perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp) \sqsubseteq \cdots$
- Reach a fixed point

Problem: Requires the structure to be a dcpo with finite chains, or continuous functions. Doesn't apply to arbitrary parameter spaces.

14.2.4 Galois Connections

A pair of "best approximations" between two levels of abstraction.

$$f(x) \le y \iff x \le g(y)$$

Used in abstract interpretation, type systems.

Problem: Describes the *relationship* between approximations, not how to compute them.

14.2.5 Game-Theoretic

Minimax, Nash equilibrium.

- Multiple agents adjusting strategies
- Converge to equilibrium (sometimes)

Problem: Convergence is not guaranteed. Computationally hard.

14.2.6 Probabilistic

PAC learning: "Probably Approximately Correct."

- With high probability, get close to the target
- Concentration inequalities, sample complexity

Problem: Gives bounds, not algorithms. Still need a method to actually learn.

14.3 The Operational Question

> **⚠ Common Pitfall**
>
> Most notions of approximation are **descriptive**, not **operational**. They tell you what "getting closer" means, but not how to do it.

What we need:

1. A **space** to search in
2. A **measure** of how close we are (loss)
3. An **algorithm** that reduces the loss

14.4 The Problem of Semantic Convergence

There's a deeper issue with approaches that require "understanding" the structure.

In analysis, when we say a sequence $x_n \to x$, we have *tools to prove* this:

- Epsilon-delta definitions
- Cauchy criterion
- Completeness of $\mathbb{R}$

The target x **exists**, and we can **prove** our approximations converge to it.

Common Pitfall

But what if we're trying to "approximate" a *meaning*?
When we build a model to capture some semantic concept—"justice," "intention," "belief"—what guarantees that our formalization converges to anything real?

This is Wittgenstein's challenge. He famously wrote:

> "Philosophy is a battle against the **bewitchment** of our intelligence by means of language."

The bewitchment: language makes us *feel* like we're referring to something. We say "meaning," "understanding," "truth," and we imagine there must be *things* these words point to—things we can approximate, converge to, capture.

But what if there's nothing there? What if the question "what is the meaning?" is itself a grammatical illusion—a question that *looks* well-formed but has no answer?

Intuition

In analysis: we prove $x_n \to x$ using the structure of $\mathbb{R}$. The limit *exists*.
In semantics: we want "model $\to$ meaning." But maybe there is no "meaning" sitting there, waiting to be approximated. The target itself may be a mirage created by language.

The domain-theoretic approach is vulnerable to this bewitchment. To set up the dcpo, to define the continuous function, you must already *believe* there's a well-defined target. You formalize your intuition about "what learning should converge to." But you can't prove your intuition refers to anything real.

Shannon's Move

Shannon's information theory succeeds by *refusing to engage* with meaning.
He doesn't ask: "What does this message mean?"
He asks: "How many bits to transmit it?"
The semantic question has no tools. The syntactic question does.

Gradient descent makes the same move:

Approach	Question	Tools?
Semantic	"Does my model capture the concept?"	No
Operational	"Does my loss decrease?"	Yes

We don't ask whether our neural network "understands" language. We ask whether its loss on the next-token prediction task goes down. The first question is philosophically intractable. The second is measurable.

This is why we choose gradient descent: not because it's philosophically satisfying, but because it's the only game in town with actual convergence guarantees—at the operational level, not the semantic level.

14.5 What Actually Works

Surprisingly few methods are both *principled* and *operational*:

Method	Requirements	Scalable?
Gradient descent	Differentiable loss	Yes
Fixed point iteration	Monotone + chain condition	Sometimes
Constraint propagation	Discrete + local	Sometimes
LP relaxation	Linear/convex	Yes
MCMC / sampling	Probabilistic	Slow
Evolutionary	Fitness function	Slow

Key Insight

Gradient descent dominates because it is:

- Mathematically principled (follows steepest descent)
- Computationally tractable (autodiff scales)
- General (works for any differentiable function)

14.6 Our Path

For learning *logical structures* (Kripke frames, automata, coalgebras):

1. The structures are **discrete** — can't directly use gradients
2. We need to **embed** them into a continuous space
3. The embedding should preserve logical meaning
4. After learning, we **extract** back to discrete

This is the **semiring relaxation**:

$$\text{Discrete structure} \xrightarrow{\text{embed}} \text{Semiring-valued} \xrightarrow{\text{gradient descent}} \text{Trained} \xrightarrow{\text{threshold}} \text{Discrete}$$

14.7 Why Semiring?

Why not just use $[0, 1]$ (fuzzy) and be done?

Because we want:

- **Generality**: Different tasks need different interpretations
- **Compositionality**: Logical connectives should compose correctly
- **Gradient flow**: Operations should be differentiable

Semirings provide all three. They are the *minimal algebraic structure* that supports continuous relaxation of logic.

Key Insight

We arrive at semirings not by abstract preference, but by elimination:

- Need operational approximation $\to$ gradient descent
- Structures are discrete $\to$ need continuous embedding
- Want compositionality $\to$ need algebraic structure
- Semiring $=$ the natural answer

14.8 What's Next

The rest of Part II develops this idea:

1. Gradient descent: the core algorithm
2. Continuization: embedding discrete into continuous
3. Fuzzy and probabilistic: concrete examples
4. Semiring: the general framework
5. Modal operators: how to handle $\Box/\Diamond$
6. The learning loop: putting it all together

Chapter 15

Gradient Descent

Goals of This Chapter

- Review calculus: derivatives measure sensitivity
- Understand gradient descent as iterative optimization
- See Newton's method and its quadratic convergence
- Appreciate why continuous optimization is powerful

15.1 The Optimization Problem

We have a function $f : \mathbb{R}^n \to \mathbb{R}$ and want to find:

$$\theta^* = \arg\min_\theta f(\theta)$$

If f is differentiable, the gradient $\nabla f(\theta)$ points "uphill."

Intuition

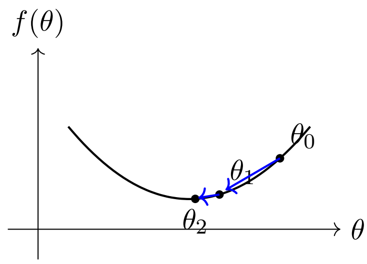
The gradient tells you: "If you want f to increase, go this direction." So to minimize, go the opposite direction.

15.2 Gradient Descent

Definition 15.1 (Gradient Descent)**.** Starting from θ_0, iterate:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

where $\eta > 0$ is the *learning rate*.



15.3 Convergence

Theorem 15.2 (Convergence for Convex Functions)**.** *If f is convex and L-smooth (bounded second derivatives), gradient descent with $\eta = 1/L$ satisfies:*

$$f(\theta_t) - f(\theta^*) \leq \frac{\|\theta_0 - \theta^*\|^2}{2t/L}$$

Rate: $O(1/t)$. Slow, but guaranteed.

15.4 Newton's Method

Use second-order information (the Hessian $H = \nabla^2 f$):

$$\theta_{t+1} = \theta_t - H(\theta_t)^{-1} \nabla f(\theta_t)$$

Theorem 15.3 (Quadratic Convergence)**.** *Near a minimum with positive definite Hessian, Newton's method converges quadratically:*

$$\|\theta_{t+1} - \theta^*\| \leq C\|\theta_t - \theta^*\|^2$$

Key Insight

Newton's method uses curvature information to take better steps. The Hessian "rescales" the gradient to account for how steep different directions are.

15.5 The Problem with Discrete Spaces

All of this requires:

- A continuous space $\mathbb{R}^n$
- A differentiable function f

But the structures we care about (automata, Kripke frames, programs) are *discrete*.

Common Pitfall

You can't take the gradient of "number of states" or "which transitions exist." Discrete spaces have no derivatives.

Solution: Embed discrete structures into continuous spaces. This is the motivation for *semirings*.

Chapter 16

From Discrete to Continuous

Goals of This Chapter

- Understand why we need to "soften" discrete structures
- See concrete examples: soft attention, soft selection
- Introduce the idea of relaxation and rounding

16.1 The Continuization Trick

Discrete optimization is hard. Continuous optimization has gradients.

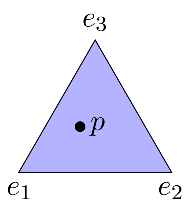
Strategy:

1. Embed discrete objects into a continuous space
2. Optimize in the continuous space
3. Round back to discrete

16.2 Example: Soft Selection

Discrete: Choose one of n items. Represented as one-hot vector $e_i \in \{0,1\}^n$.

Continuous: Use a probability distribution $p \in \Delta^{n-1}$ (the simplex).



The corners are discrete choices. The interior is "soft" mixtures.

16.3 Example: Soft Transitions

Discrete: Transition relation $R \subseteq W \times W$. Either $(w, v) \in R$ or not.
Continuous: Transition weights $R : W \times W \to [0, 1]$.

- $R(w, v) = 1$: definitely connected
- $R(w, v) = 0$: definitely not connected
- $R(w, v) = 0.7$: "70% connected" (soft)

16.4 The Softmax Function

To go from unconstrained $\mathbb{R}^n$ to the simplex:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Output is always a valid distribution
- Differentiable everywhere
- Temperature τ: $\text{softmax}(z/\tau)$ becomes sharper as $\tau \to 0$

16.5 Relaxation and Rounding

Relaxation: Replace discrete constraints with continuous ones.

$$x \in \{0, 1\} \quad \rightsquigarrow \quad x \in [0, 1]$$

Rounding: After optimization, snap back to discrete.

$$x = 0.7 \quad \rightsquigarrow \quad x = 1$$

⚠ Common Pitfall

Rounding can fail! The continuous optimum might round to a bad discrete solution. This is why we need careful design of the relaxation.

16.6 What We Need

To make this systematic, we need:

1. A principled way to "soften" logical operations $(\wedge, \vee, \neg)$
2. Compatibility with gradient computation
3. Convergence guarantees (soft $\to$ crisp)

This is exactly what **semirings** provide.

Chapter 17

Fuzzy Logic

Goals of This Chapter

- See the simplest way to make logic continuous: fuzzy truth values
- Understand fuzzy AND, OR, NOT
- Appreciate what works and what doesn't

17.1 Truth as a Degree

Classical logic: true $= 1$, false $= 0$.

Fuzzy logic: truth values in $[0, 1]$.

Intuition

"The water is hot" — is this true or false? Depends on temperature. At 30°C, maybe 0.3 true. At 80°C, maybe 0.95 true. Fuzzy logic captures this gradation.

17.2 Fuzzy Connectives

How should $\wedge$, $\vee$, $\neg$ work on $[0, 1]$?

Negation:

$$\neg x = 1 - x$$

If x is 0.7 true, then $\neg x$ is 0.3 true.

Conjunction (Gödel):

$$x \wedge y = \min(x, y)$$

"A and B" is as true as the least true conjunct.

Disjunction (Gödel):

$$x \vee y = \max(x, y)$$

"A or B" is as true as the most true disjunct.

17.3 Example: Fuzzy Kripke Model

Definition 17.1 (Fuzzy Kripke Model)**.** A *fuzzy Kripke model* is (W, R, V) where:

- $R : W \times W \to [0, 1]$ (fuzzy accessibility)
- $V : \Phi \to (W \to [0, 1])$ (fuzzy valuation)

Satisfaction becomes a degree:

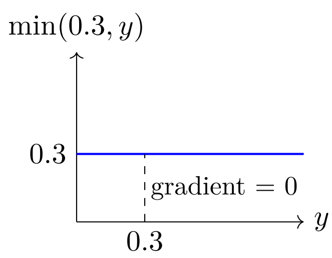
$$\begin{aligned} [\![p]\!]_w &= V(p)(w) \\ [\![\neg\varphi]\!]_w &= 1 - [\![\varphi]\!]_w \\ [\![\varphi \wedge \psi]\!]_w &= \min([\![\varphi]\!]_w, [\![\psi]\!]_w) \\ [\![\Box\varphi]\!]_w &= \min_{v \in W} (R(w, v) \Rightarrow [\![\varphi]\!]_v) \end{aligned}$$

where $a \Rightarrow b = \min(1, 1 - a + b)$ (Łukasiewicz implication) or other choices.

17.4 Problems with Min/Max

Common Pitfall

Min/max have zero gradients almost everywhere!
If $x = 0.3$ and $y = 0.7$, then $\min(x, y) = x$. The gradient w.r.t. y is zero. Optimization gets stuck.



17.5 Alternative: Product Logic

Instead of min/max, use multiplication:

$$x \wedge y = x \cdot y$$
$$x \vee y = x + y - x \cdot y$$

Now gradients are non-zero:

$$\frac{\partial(x \cdot y)}{\partial y} = x \neq 0$$

But this changes the algebraic properties...

17.6 The Pattern

We have two "fuzzy logics":

	Gödel	**Product**
AND	min	$\times$
OR	max	$+$ (clamped)

They're both trying to do the same thing: extend Boolean logic to $[0, 1]$. Is there a general framework that includes both?

Chapter 18

Probabilistic Reasoning

Goals of This Chapter

- See probability as another way to handle uncertainty
- Understand how AND and OR work probabilistically
- Compare with fuzzy logic
- Notice the algebraic pattern

18.1 Probability vs Fuzziness

Fuzzy: "The water is 0.7 hot" (degree of truth)

Probabilistic: "There's 0.7 probability the water is hot" (uncertainty about a crisp fact)

Different interpretations, but similar math.

18.2 Probabilistic Connectives

For independent events A, B with probabilities p, q:

Conjunction:

$$P(A \wedge B) = P(A) \cdot P(B) = p \cdot q$$

Disjunction (inclusive):

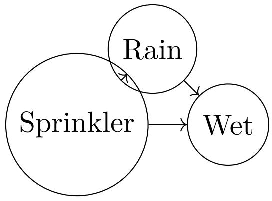
$$P(A \vee B) = P(A) + P(B) - P(A \wedge B) = p + q - pq$$

Negation:

$$P(\neg A) = 1 - P(A) = 1 - p$$

18.3 Example: Probabilistic Inference

Consider a simple Bayesian network:



Computing $P(\text{Wet})$ involves:

- Multiplying probabilities along paths (AND)
- Adding probabilities across paths (OR)

18.4 Semiring Structure Appears

Look at the operations:

	Fuzzy (Gödel)	**Probabilistic**
AND ($\otimes$)	min	$\times$
OR ($\oplus$)	max	$+$
False ($\mathbf{0}$)	0	0
True ($\mathbf{1}$)	1	1

Both satisfy:

- $\oplus$ is associative, commutative, has identity $\mathbf{0}$
- $\otimes$ is associative, commutative, has identity $\mathbf{1}$
- $\otimes$ distributes over $\oplus$
- $\mathbf{0} \otimes x = \mathbf{0}$

Key Insight

This is the structure of a **semiring**! Both fuzzy logic and probabilistic logic are instances of the same algebraic pattern.

18.5 Why This Matters

Once we recognize the pattern:

- We can write algorithms that work for *any* semiring
- Switching from fuzzy to probabilistic = changing the semiring
- New semirings = new kinds of reasoning

18.6 More Examples

Name	$\oplus$	$\otimes$	**Use**
Boolean	$\vee$	$\wedge$	Classical logic
Fuzzy	max	min	Soft constraints
Probabilistic	$+$	$\times$	Uncertainty
Tropical	min	$+$	Shortest paths

The tropical semiring is especially interesting: "OR = take minimum", "AND = add costs". Finding a satisfying assignment becomes finding the shortest path!

Next: the general definition of semiring.

Chapter 19

Semiring: The Abstraction

Goals of This Chapter

- See the common structure behind fuzzy, probabilistic, tropical
- Understand the formal definition of semiring
- Appreciate why this abstraction is useful

19.1 The Pattern We've Seen

Recall:

Name	S	$\oplus$ (OR)	$\otimes$ (AND)	$\mathbf{0}$	$\mathbf{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Fuzzy (Gödel)	$[0,1]$	max	min	0	1
Probabilistic	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1

All satisfy the same algebraic laws. Let's name this structure.

19.2 Definition

Definition 19.1 (Semiring)**.** A *semiring* is a tuple $(S, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ where:

1. $(S, \oplus, \mathbf{0})$ is a commutative monoid:
   - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
   - $a \oplus b = b \oplus a$
   - $a \oplus \mathbf{0} = a$
2. $(S, \otimes, \mathbf{1})$ is a monoid:
   - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$

- $a \otimes \mathbf{1} = \mathbf{1} \otimes a = a$

3. $\otimes$ distributes over $\oplus$:
   - $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
   - $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$

4. $\mathbf{0}$ annihilates:
   - $a \otimes \mathbf{0} = \mathbf{0} \otimes a = \mathbf{0}$

Intuition

$\oplus$ = "or" / "choice" / "combine alternatives"
$\otimes$ = "and" / "sequence" / "combine steps"
$\mathbf{0}$ = "impossible" / "failure"
$\mathbf{1}$ = "trivially true" / "do nothing"

19.3 More Examples

Definition 19.2 (Tropical Semiring)**.** $(\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0)$

- $a \oplus b = \min(a, b)$: choose the better option
- $a \otimes b = a + b$: accumulate costs
- $\mathbf{0} = +\infty$: infinite cost = impossible
- $\mathbf{1} = 0$: zero cost = free

Key Insight

In the tropical semiring, logical inference becomes optimization! "Find a satisfying assignment" becomes "find the minimum-cost path."

Definition 19.3 (Viterbi Semiring)**.** $([0, 1], \max, \times, 0, 1)$
Used in HMMs: find the most probable path.

Definition 19.4 (Log Semiring)**.** $(\mathbb{R} \cup \{-\infty\}, \mathrm{logsumexp}, +, -\infty, 0)$
Numerically stable version of probabilistic semiring.

19.4 Why Semirings?

1. **Genericity**: Write algorithm once, instantiate for different semirings
2. **Correctness**: Algebraic laws guarantee properties
3. **Modularity**: Change interpretation without changing structure

Example 19.5. The same dynamic programming algorithm:

- Boolean semiring $\rightarrow$ reachability (is there a path?)
- Tropical semiring $\rightarrow$ shortest path (what's the minimum cost?)
- Probabilistic semiring $\rightarrow$ most probable path
- Counting semiring $(\mathbb{N}, +, \times, 0, 1) \rightarrow$ count paths

Chapter 20

The Gradient Semiring

Goals of This Chapter

- Understand automatic differentiation algebraically
- See how gradients flow through semiring operations
- Appreciate: inference and learning in one pass

20.1 The Problem

We want to optimize over semiring computations.

Given a semiring expression (e.g., probabilistic inference), we need gradients w.r.t. parameters.

Naïve approach: compute forward, then backpropagate.

Better approach: **embed gradients into the semiring itself**.

20.2 Dual Numbers

Definition 20.1 (Dual Numbers)**.** A *dual number* is $a + b\epsilon$ where $\epsilon^2 = 0$.

Arithmetic:

$$(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$$
$$(a + b\epsilon) \times (c + d\epsilon) = ac + (ad + bc)\epsilon$$

Key Insight

If $f(x) = a + b\epsilon$ when we plug in $x = x_0 + 1 \cdot \epsilon$, then:

- $a = f(x_0)$ (the value)
- $b = f'(x_0)$ (the derivative!)

20.3 The Gradient Semiring

Definition 20.2 (Gradient Semiring)**.** Elements: pairs (v, g) where $v =$ value, $g =$ gradient.

Operations:

$$(v_1, g_1) \oplus (v_2, g_2) = (v_1 + v_2,\ g_1 + g_2)$$
$$(v_1, g_1) \otimes (v_2, g_2) = (v_1 \cdot v_2,\ v_1 \cdot g_2 + v_2 \cdot g_1)$$

Identities: $\mathbf{0} = (0, 0)$, $\mathbf{1} = (1, 0)$.

Proposition 20.3. *The gradient semiring is a semiring.*

Proof. Check the axioms. The key is that $\otimes$ follows the product rule. □

20.4 Forward-Mode Automatic Differentiation

To compute $f(x)$ and $\frac{\partial f}{\partial x}$ simultaneously:

1. Replace x with $(x, 1)$ (value x, derivative 1)
2. Compute using gradient semiring operations
3. Extract: first component = value, second = derivative

Example 20.4. Compute $f(x) = x^2 + x$ at $x = 3$:

$$x = (3, 1)$$
$$x^2 = (3, 1) \otimes (3, 1) = (9, 6)$$
$$x^2 + x = (9, 6) \oplus (3, 1) = (12, 7)$$

So $f(3) = 12$ and $f'(3) = 7$. Check: $f'(x) = 2x + 1$, so $f'(3) = 7$. ✓

20.5 Application: Differentiable Logic

Any semiring computation can be made differentiable:

1. Write your logic in semiring form
2. Instantiate with gradient semiring
3. Get gradients for free

Key Insight

This is how DeepProbLog works: probabilistic logic programming with gradients. Inference and learning happen in a single forward pass.

20.6 Limitations

Forward-mode computes $\frac{\partial f}{\partial x_i}$ for one x_i at a time.

For many parameters, need reverse-mode (backpropagation).

Reverse-mode is also algebraic, but more complex (requires "transposing" the computation).

Chapter 21

Semiring-Valued Coalgebra

Goals of This Chapter

- Combine Part I (coalgebra) with Part II (semiring)
- See how to "soften" any coalgebra
- Understand the learning setup

21.1 Recall: Coalgebra

From Part I: an F-coalgebra is $(X, \gamma : X \to F(X))$.

Examples:

- Kripke frame: $X \to \mathcal{P}(X)$
- DFA: $X \to 2 \times X^A$
- Stream: $X \to A \times X$

These are all *crisp*: transitions either exist or don't.

21.2 Semiring-Valued Transitions

Definition 21.1 (Semiring-Valued Coalgebra)**.** Fix a semiring S. An *S-valued F-coalgebra* replaces sets with S-valued functions.

For $F = \mathcal{P}$ (Kripke-like):

$$\gamma : X \to (X \to S)$$

i.e., $\gamma : X \times X \to S$. Each transition has a weight in S.

Example 21.2 (Fuzzy Kripke Frame)**.** $S = [0, 1]$. Then $\gamma(w, v) = 0.7$ means "70% connected."

Example 21.3 (Probabilistic Automaton)**.** $S = \mathbb{R}_{\geq 0}$. Then $\gamma(q, a, q')$ is the probability of transitioning from q to q' on input a.

21.3 Parameterized Coalgebras

For learning, we parameterize the coalgebra:

Definition 21.4. A *parameterized S-coalgebra* is a family γ_θ indexed by $\theta \in \Theta$.

Typically: θ are real numbers, γ_θ uses softmax or sigmoid to produce S-values.

Example 21.5. For a fuzzy DFA with n states and alphabet A:

$$\theta \in \mathbb{R}^{n \times |A| \times n}$$

$$\gamma_\theta(q, a, q') = \mathrm{softmax}_j(\theta_{q,a,j})_{q'}$$

21.4 Semantics in Semirings

Modal formulas can be evaluated in semiring-valued models:

$$\begin{aligned}
[\![p]\!]_w &\in S \\
[\![\varphi \wedge \psi]\!]_w &= [\![\varphi]\!]_w \otimes [\![\psi]\!]_w \\
[\![\varphi \vee \psi]\!]_w &= [\![\varphi]\!]_w \oplus [\![\psi]\!]_w \\
[\![\Box\varphi]\!]_w &= \bigotimes_v \gamma(w, v) \Rightarrow_S [\![\varphi]\!]_v
\end{aligned}$$

where $\Rightarrow_S$ is an S-valued implication (semiring-dependent).

21.5 The Bridge

$$\begin{tikzcd}
\text{Crisp Coalgebra} \arrow[rr, leftrightarrow, "\text{soften}", "\text{threshold}"'] \arrow[d, dashed, no head] & & S\text{-valued Coalgebra} \arrow[d, dashed, no head] \\
\text{Discrete, not learnable} & & \text{Continuous, differentiable}
\end{tikzcd}$$

Key Insight

Semiring-valued coalgebra is the bridge between the discrete world of Part I and the continuous world of optimization.

Chapter 22

Modal Operators in Semirings

Goals of This Chapter

- Face the core question: how to interpret $\Box/\Diamond$ over semirings
- Survey existing approaches: fuzzy, many-valued, residuated lattices
- Understand what's solved and what's open
- Know the implications for learning

22.1 The Problem

In classical modal logic:

$$\mathcal{M}, w \models \Box\varphi \iff \forall v : wRv \Rightarrow \mathcal{M}, v \models \varphi$$

This involves:

- $\forall$ — universal quantification (AND over all successors)
- $\Rightarrow$ — implication (if accessible, then...)

In semiring-valued setting:

- $R(w, v) \in S$ — soft accessibility
- $[\![\varphi]\!]_v \in S$ — soft truth value

Question: What is $[\![\Box\varphi]\!]_w$?

22.2 Existing Work: Three Traditions

This is *not* a completely open problem. There's substantial literature.

22.2.1 Modal Semirings (Algebraic)

Möller, Desharnais, and others developed *modal semirings* as an algebraic axiomatization.

- Define *domain* operation: $\mathrm{dom}(x) = |x\rangle 1$
- Define box via de Morgan: $|x]p = \neg|x\rangle\neg p$
- Focus on algebraic laws, not many-valued truth

Key reference: "Some Uses of Modal Semirings" (Möller & Desharnais, WADT 2024).

This approach is about *algebraic structure*, not about truth values in $[0, 1]$.

22.2.2 Many-Valued Modal Logic (Lattice-Valued)

Bou, Esteva, Godo, and others study modal logic over *residuated lattices*.

Definition 22.1 (Lattice-Valued Kripke Model)**.** Over a residuated lattice $A = \langle A, 0, 1, \wedge, \vee, \otimes, \to\rangle$:

- $R : W \times W \to A$ (many-valued accessibility)
- $e : \mathrm{Var} \times W \to A$ (many-valued valuation)

The semantics of $\Box$:

$$[\![\Box\varphi]\!]_w = \bigwedge_v (R(w, v) \to [\![\varphi]\!]_v)$$

where $\to$ is the residual of $\otimes$.

Key reference: "On the Minimum Many-Valued Modal Logic over a Finite Residuated Lattice" (Bou et al., 2009).

22.2.3 Fuzzy Modal Logic (T-Norm Based)

Fuzzy modal logic uses t-norms and their residuals.

Definition 22.2 (Fuzzy Modal Semantics)**.** For a t-norm $\otimes$ with residual $\Rightarrow$:

$$[\![\Box\varphi]\!]_w = \inf_v (R(w, v) \Rightarrow [\![\varphi]\!]_v)$$
$$[\![\Diamond\varphi]\!]_w = \sup_v (R(w, v) \otimes [\![\varphi]\!]_v)$$

Common choices:

Name	$a \otimes b$	$a \Rightarrow b$
Gödel	$\min(a, b)$	$\begin{cases} 1 & a \leq b \\ b & a > b \end{cases}$
Łukasiewicz	$\max(0, a + b - 1)$	$\min(1, 1 - a + b)$
Product	$a \cdot b$	$\begin{cases} 1 & a \leq b \\ b/a & a > b \end{cases}$

22.3 What's Known

Theorem 22.3 (Failure of Axiom K)**.** *In many-valued modal logic with non-Boolean accessibility, the axiom*

$$\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$$

generally ***fails****.*

Theorem 22.4 (Non-Interdefinability)**.** *In general, $\Diamond\varphi \not\equiv \neg\Box\neg\varphi$ when R is many-valued.*

Property	**Status**
Semantics for $\Box/\Diamond$	Defined (inf/sup with residual)
Axiom K	Fails in general
$\Box/\Diamond$ duality	Fails in general
Completeness	Case-by-case, often unknown
Decidability	Case-by-case

22.4 What's Still Open

1. **General semirings**: Most work uses residuated lattices (which have a specific implication $\to$). Arbitrary semirings don't have a canonical implication.

2. **Tropical semiring**: $(\mathbb{R} \cup \{\infty\}, \min, +)$ — what's the right modal semantics? Not well-studied.

3. **Gradient semiring**: What happens when we track derivatives through modal operators? Unexplored.

4. **Best semantics for learning**: Which choice of t-norm/residual leads to best optimization landscape? No one has studied this.

5. **Convergence**: When does soft modal semantics converge to crisp? Conditions unclear.

22.5 For Learning: Practical Choices

Given the theory, what should we use for gradient-based learning?

> **⚠ Common Pitfall**
>
> inf and sup have zero gradients almost everywhere. Not suitable for learning.

22.5.1 Option 1: Soft Inf/Sup

Replace inf with softmin:

$$\text{softmin}_\tau(x_1, \ldots, x_n) = -\tau \log \sum_i e^{-x_i/\tau}$$

As $\tau \to 0$, this approaches true min.

22.5.2 Option 2: Product Semantics

Use product t-norm throughout:

$$[\![\Box\varphi]\!]_w = \prod_v (1 - R(w, v) \cdot (1 - [\![\varphi]\!]_v))$$
$$[\![\Diamond\varphi]\!]_w = 1 - \prod_v (1 - R(w, v) \cdot [\![\varphi]\!]_v)$$

Fully differentiable, gradients everywhere.

22.5.3 Option 3: Weighted Average

For $\Box$ as "expected truth over successors":

$$[\![\Box\varphi]\!]_w = \frac{\sum_v R(w, v) \cdot [\![\varphi]\!]_v}{\sum_v R(w, v)}$$

Simple, differentiable, but doesn't match classical semantics.

22.6 Recommendation

Key Insight

For learning:

1. Use **product t-norm** or **softmin/softmax**
2. Accept that axiom K may fail during training
3. Add regularization to push toward crisp values
4. After convergence, verify classical properties

22.7 Summary

Aspect	Status	Reference
Algebraic axioms	Solved	Möller et al.
Residuated lattice semantics	Solved	Bou et al.
Fuzzy/t-norm semantics	Solved	Hájek, Fitting
Arbitrary semirings	Partially open	—
Differentiable semantics	Open	—
Best choice for learning	Open	—

The mathematics of many-valued modal logic is well-developed. The question of which variant works best for gradient-based learning is new territory.

Chapter 23

The Learning Loop

Goals of This Chapter

- Put everything together: the full learning pipeline
- Understand loss, forward pass, backward pass
- See convergence from soft to crisp

23.1 The Setup

We have:

- A functor F (what we're learning: DFA, Kripke, etc.)
- A semiring S (how we soften: fuzzy, probabilistic, etc.)
- Parameters $\theta \in \Theta \subseteq \mathbb{R}^d$
- A parameterized S-coalgebra γ_θ
- Training data $\mathcal{D}$

23.2 Loss Function

The loss measures how well γ_θ explains the data.

Example 23.1 (Automaton Learning)**.** Data: strings labeled accept/reject.

$$L(\theta) = -\sum_{(x,y)\in\mathcal{D}} \log P_\theta(y \mid x)$$

where P_θ is computed by running the soft automaton on x.

Example 23.2 (Modal Satisfaction)**.** Data: (model, world, formula, truth value) tuples.

$$L(\theta) = \sum_{(w,\varphi,v)} \left([\![\varphi]\!]_w^\theta - v \right)^2$$

23.3 Forward Pass

Compute $[\![\cdot]\!]^\theta$ using semiring operations.

If using gradient semiring: get gradients simultaneously.

23.4 Backward Pass

Compute $\nabla_\theta L$ using:

- Gradient semiring (forward-mode), or
- Standard backpropagation (reverse-mode)

23.5 Update

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

Repeat until convergence.

23.6 Convergence to Crisp

As training progresses, soft values should approach $\{0, 1\}$.

Definition 23.3 (Temperature Annealing)**.** Replace softmax with:

$$\text{softmax}_\tau(z)_i = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}}$$

Gradually decrease $\tau \to 0$. At $\tau = 0$, this is hard argmax.

Definition 23.4 (Sparsity Regularization)**.** Add to loss:

$$L_{\text{sparse}}(\theta) = \lambda \sum_{i,j} H(\gamma_\theta(i, j))$$

where $H(p) = -p \log p - (1 - p) \log(1 - p)$ is entropy. Pushes toward 0 or 1.

23.7 Extraction

After training, extract crisp coalgebra:

$$\gamma_{\text{crisp}}(x, y) = \begin{cases} 1 & \text{if } \gamma_\theta(x, y) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Key Insight

The full pipeline:

$$\text{Data} \xrightarrow{\text{loss}} \text{Soft Coalgebra} \xrightarrow{\text{optimize}} \text{Trained Soft} \xrightarrow{\text{threshold}} \text{Crisp Coalgebra}$$

We learn in continuous space, then extract a discrete program.

23.8 Example: Learning a DFA

Task: Learn a DFA from positive/negative examples.

Setup:

- n states, alphabet $\{0, 1\}$
- $\theta \in \mathbb{R}^{n \times 2 \times n + n}$ (transitions + accepts)
- Soft transitions via softmax
- Soft accepts via sigmoid

Training: Standard gradient descent on cross-entropy loss.
Result: After convergence, threshold to get a crisp DFA.
This is what we'll implement in the code for Part II.

Chapter 24

Differentiable Graded Modalities

Goals of This Chapter

- Combine graded quantifiers with differentiable learning
- See how to soften counting modalities
- Understand this as a potential novel contribution

24.1 The Gap

From Part I: graded modalities $\Diamond_{\geq n}$, M, $P_{\geq r}$ are expressive.

From Part II: semiring-valued coalgebra enables learning.

But existing work doesn't combine them:

Work	Learns R?	Graded?	Differentiable?
MLNNs (2024)	✓	× (only $\Box/\Diamond$)	✓
Graded Modal	×	✓	×
This chapter	✓	✓	✓

24.2 Softening Graded Modalities

Recall the crisp semantics:

$$\mathcal{M}, w \models \Diamond_{\geq n}\varphi \iff |\{v : wRv \wedge v \models \varphi\}| \geq n$$

This is a hard threshold. Not differentiable.

Definition 24.1 (Soft Graded Modality)**.** For fuzzy $R : W \times W \to [0,1]$ and $[\![\varphi]\!]_v \in [0,1]$:

$$[\![\Diamond_{\geq n}\varphi]\!]_w = \sigma\left(\sum_v R(w,v) \cdot [\![\varphi]\!]_v - n + \frac{1}{2}\right)$$

where σ is the sigmoid function.

Intuition

The sum $\sum_v R(w,v) \cdot [\![\varphi]\!]_v$ is a soft count of how many successors satisfy φ. The sigmoid turns "soft count $\geq n$" into a value in $[0,1]$.

24.3 Other Soft Quantifiers

Quantifier	Soft Version
$\Diamond_{\geq n}\varphi$	$\sigma(\sum_v R \cdot [\![\varphi]\!]_v - n + 0.5)$
$\Diamond_{\leq n}\varphi$	$\sigma(n + 0.5 - \sum_v R \cdot [\![\varphi]\!]_v)$
$\mathsf{M}\varphi$	$\sigma(\sum_v R \cdot [\![\varphi]\!]_v - \sum_v R \cdot [\![\neg\varphi]\!]_v)$
$P_{\geq r}\varphi$	$\sigma(\frac{\sum_v R \cdot [\![\varphi]\!]_v}{\sum_v R(w,v)} - r)$

24.4 Learning with Graded Constraints

Now we can train:

Example 24.2 (Robust Planning)**.** "Learn a transition system where every state has at least 2 safe successors."

Constraint: $\forall w.\Diamond_{\geq 2}\mathsf{safe}$
Loss: $L = \sum_w (1 - [\![\Diamond_{\geq 2}\mathsf{safe}]\!]_w)^2$
Optimize: gradient descent on R (soft accessibility).

Example 24.3 (Probabilistic Requirement)**.** "With probability ≥ 0.8, action leads to goal."

Constraint: $P_{\geq 0.8}\mathsf{goal}$
Loss derived from soft semantics, optimize R.

24.5 Convergence

Conjecture 24.4. *With appropriate regularization (entropy penalty, temperature annealing), soft graded modalities converge to crisp:*

- $R(w,v) \to \{0,1\}$
- $[\![\Diamond_{\geq n}\varphi]\!]_w \to \{0,1\}$

The learned structure is a classical Kripke model satisfying the graded constraints.

24.6 Open Questions

1. What are the best soft approximations? (Sigmoid? Softplus?)
2. Convergence guarantees?
3. How do meta-theoretic properties (decidability, completeness) transfer?
4. Approximation bounds: how close is soft to crisp?

Key Insight

Differentiable graded modalities let us *learn* structures satisfying counting constraints. This combines the expressiveness of graded modal logic with the learnability of neural networks.

Part III

Learning Well

Chapter 25

Learning as a Formal Object

Goals of This Chapter

- Distinguish *learning task* from *learning*
- Define behavior via final coalgebra
- See how task and learner connect through observation
- Understand performance as a multi-dimensional tuple

25.1 Behavior: What We're Trying to Learn

Before we can talk about learning, we need to say what it means for two systems to "behave the same."

Definition 25.1 (Final Coalgebra and Behavior)**.** Let $H : \mathbf{C} \to \mathbf{C}$ be a functor. A *final H-coalgebra* is a coalgebra $(Z, \omega : Z \to HZ)$ such that for any H-coalgebra $(X, \alpha : X \to HX)$, there exists a unique morphism $\mathsf{beh} : X \to Z$ making the diagram commute:

$$\begin{tikzcd} X \arrow[r, "\alpha"] \arrow[d, "\mathsf{beh}"'] & HX \arrow[d, "H(\mathsf{beh})"] \\ Z \arrow[r, "\omega"'] & HZ \end{tikzcd}$$

The element $\mathsf{beh}(x) \in Z$ is called the *behavior* of state x.

The final coalgebra Z is the "universe of all possible behaviors." The unique map beh extracts the complete observable behavior of any state.

Example 25.2 (Behaviors for Common Functors)**.**

System	Functor H	Final coalgebra Z
DFA	$2 \times (-)^{\Sigma}$	$\mathcal{P}(\Sigma^*)$ (languages)
Mealy	$O \times (-)^I$	$I^* \to O^*$ (causal functions)
Labeled transition	$\mathcal{P}(A \times -)$	infinite trees up to bisim

Two states have the same behavior iff they are bisimilar.

25.2 Learning Task vs Learning

Here is the key distinction:

Definition 25.3 (Learning Task)**.** A *learning task* is a tuple:

$$T = (H, C_{\text{target}})$$

where:

- H is a functor (the type of system)
- C_{target} is an H-coalgebra that **exists but is unknown**

The task is: find a coalgebra that behaves like C_{target}.

But we don't have direct access to C_{target}. We only see partial observations.

Definition 25.4 (Observation)**.** An *observation* D is partial information about the behavior of C_{target}.

Formally, let (Z, ω) be the final H-coalgebra. Then:

$$D \subseteq Z_n$$

where Z_n is the "truncation" of Z to depth n—behaviors observed up to n steps.

Example 25.5 (Observations in Practice)**.**

- For DFA: D = finite set of (word, accept/reject) pairs
- For Mealy: D = finite set of input-output traces
- For a program: D = input-output examples, or a queryable oracle

Now we can define learning itself:

Definition 25.6 (Learning)**.** A *learning configuration* (or *learner*) is a tuple:

$$L = (H, S, \Theta, \gamma)$$

where:

- H : functor (structural assumption—what kind of system)

- S : semiring (how we measure—fuzzy, probabilistic, tropical...)
- Θ : parameter space, indexing a family of coalgebras $\{C_\theta\}_{\theta\in\Theta}$
- γ : optimizer (how we search Θ)

Key Insight

The learning task T contains the unknown C_{target}.
The learner L contains the machinery to find an approximation.
They are separate objects.

25.3 How Task and Learner Connect

The relationship:

$$\begin{tikzcd} T = (H, C_{\text{target}}) \arrow[d, "\text{observe}"'] & \\ D \arrow[r, "L"] & \theta^* \in \Theta \end{tikzcd}$$

1. The task T contains the unknown target C_{target}
2. We observe D, a partial view of C_{target}'s behavior
3. The learner L takes D and produces θ^*
4. We hope $C_{\theta^*} \approx C_{\text{target}}$

More categorically:

$$\begin{aligned} \text{Task}(H) &= \text{category of learning tasks for functor } H \\ \text{Obs} &= \text{category of observations} \\ \text{Learner}(H, S) &= \text{category of } (H, S)\text{-learners} \end{aligned}$$

We have:

- $\text{observe} : \text{Task}(H) \to \text{Obs} \quad (\text{forgets } C_{\text{target}}, \text{ keeps observable part})$
- $L : \text{Obs} \to \Theta \quad (\text{learner maps observations to parameters})$

25.4 Performance is Not a Number

Given a learner L and a task T, how do we evaluate?

Not with a single number. Performance is multi-dimensional:

Definition 25.7 (Performance Tuple)**.**

$$\mathrm{Perf}(L, T) = (\varepsilon_{\mathrm{opt}}, \varepsilon_{\mathrm{gen}}, \varepsilon_{\mathrm{exp}})$$

where:

- $\varepsilon_{\mathrm{opt}}$ = **optimization**: how well does γ converge on D?
- $\varepsilon_{\mathrm{gen}}$ = **generalization**: how well does C_{θ^*} perform on unseen behavior?
- $\varepsilon_{\mathrm{exp}}$ = **expressiveness**: $\inf_{\theta \in \Theta} d_H^S(C_\theta, C_{\mathrm{target}})$—can we even represent the target?

These dimensions interact:

- More expressive Θ (lower $\varepsilon_{\mathrm{exp}}$) often means harder optimization (higher $\varepsilon_{\mathrm{opt}}$)
- Better optimization on D may hurt generalization (overfitting)

★ Key Insight

"Learning well" is not a scalar. It's a point in a multi-dimensional performance space.
Different applications care about different trade-offs.

25.5 The Geometry Induced by (H, S)

Here's the deep point: the pair (H, S) induces geometric structure on Θ.

Definition 25.8 (S-Bisimilarity)**.** For H-coalgebras over semiring S, the *S-bisimilarity* is:

$$d_H^S : \mathrm{Coalg}(H) \times \mathrm{Coalg}(H) \to S$$

defined via Kantorovich lifting of H.

Now fix a target. The loss function emerges:

$$\ell : \Theta \to S, \quad \theta \mapsto d_H^S(C_\theta, C_{\mathrm{target}})$$

The pair (H, S) determines:

- The "shape" of the loss landscape on Θ

- Which directions in Θ correspond to behavioral changes
- The condition number of optimization

Key Insight

The optimizer γ operates on the geometry that (H, S) induces on Θ. "Learning well" = γ converges efficiently on this geometry.

25.6 The Question of Part III

We now have precise language:

> Given (H, S, Θ, γ), what determines whether γ converges well on the (H, S)-induced geometry of Θ?

Equivalently:

- When does parameter distance match behavioral distance?
- What is the "condition number" of the map $\theta \mapsto C_\theta$?
- How does the choice of S affect optimization?

The rest of Part III explores these questions experimentally and theoretically.

Chapter 26

Experiments: What Works and What Doesn't

Goals of This Chapter

- See concrete examples where architecture matters
- Observe patterns: symmetry seems to help
- Gather empirical evidence before building theory

26.1 Experiment 1: Counting

Task: Given a sequence, count occurrences of symbol a.

Setup A: Fully connected network

- Input: one-hot sequence, flattened
- Output: count (regression)
- Parameters: $O(n^2)$ where n = sequence length

Setup B: Recurrent network (counter)

- State: running count
- Update: if a, increment; else, keep
- Parameters: $O(1)$

Result: Setup B learns instantly. Setup A struggles, needs much more data.

26.2 Experiment 2: Image Classification

Task: Classify images (e.g., MNIST digits).
Setup A: Fully connected

- Flatten image to vector
- Dense layers
- Parameters: $O(n^2)$ for n pixels

Setup B: Convolutional

- Local kernels, shared across positions
- Parameters: $O(k^2)$ for kernel size k

Result: CNN learns much faster, generalizes better, needs less data.

26.3 Experiment 3: Set Functions

Task: Given a set of numbers, compute some function (e.g., sum, max).
Setup A: Feed as ordered sequence to MLP
Setup B: Permutation-invariant architecture (DeepSets)

$$f(\{x_1, \ldots, x_n\}) = \rho\left(\sum_i \phi(x_i)\right)$$

Result: Setup A fails badly (learns spurious order dependence). Setup B works.

26.4 The Pattern

Task	Task Symmetry	Best Architecture
Counting	Time-shift invariant	RNN / Counter
Image	Translation invariant	CNN
Set function	Permutation invariant	DeepSets
Graph function	Node permutation	GNN

Key Insight

When the architecture respects the task's symmetry, learning is easy. When it doesn't, the network wastes capacity learning what it should get for free.

26.5 Questions

1. **Why** does matching symmetry help?
2. Can we **quantify** how much it helps?
3. Given a task, can we **derive** the right architecture?

The next chapters develop the theory to answer these.

26.6 Numerical Verification: Distance Matching

We ran experiments to test the **distance matching hypothesis**: good parameterization means parameter distance $\approx$ function distance.

26.6.1 Methodology

For each architecture:

1. Sample many random parameter pairs (θ_1, θ_2)
2. Compute parameter distance $d_\Theta(\theta_1, \theta_2) = \|\theta_1 - \theta_2\|$
3. Compute function distance $d_\mathcal{F}(f_{\theta_1}, f_{\theta_2}) = \|f_{\theta_1}(x) - f_{\theta_2}(x)\|$
4. Measure correlation and coefficient of variation of the ratio

High correlation + low CV = good distance matching.

26.6.2 Result 1: Redundancy Destroys Matching

Model	**Correlation**	**Rating**
Linear (direct)	0.96	Excellent
Linear $(a - b)$ parameterization	0.48	Poor
Linear $(a - b + c - d)$	0.26	Terrible
Matrix $A \cdot B$ factorization	0.26	Terrible

Key Insight

Redundant parameterization destroys distance matching. The more redundancy, the worse.

26.6.3 Result 2: Depth Destroys Matching (Even Without Nonlinearity!)

Model	Correlation	Rating
1-layer linear	0.97	Excellent
2-layer linear ($W_2 W_1 x$)	0.26	Terrible
3-layer linear	0.31	Terrible
4-layer linear	0.28	Terrible

Key Insight

Deep linear networks have the *same function space* as shallow ones (all linear functions), yet distance matching collapses!
Why? Because $W_2 \cdot W_1 = W_2' \cdot W_1'$ has infinitely many solutions. Depth introduces *implicit redundancy.*

26.6.4 Result 3: Sparsity Helps

Model	Correlation	Rating
Dense linear (20 params)	0.91	Excellent
Sparse linear (5 params, fixed positions)	0.98	Excellent
Soft-sparse (sigmoid mask)	0.48	Poor

Fixed sparsity reduces parameters and improves matching. "Soft" sparsity with learnable masks introduces redundancy.

26.6.5 Result 4: Matrix Factorization Always Hurts

Model	Correlation	Rating
Direct linear	0.96	Excellent
Low-rank $k = 2$	0.31	Terrible
Low-rank $k = 5$	0.29	Terrible
SVD parameterization	0.32	Terrible

Any factorization introduces redundancy.

26.6.6 Result 5: Nonlinearity Is Not The Main Culprit

2-layer MLP with:	Correlation	Rating
Identity (linear)	0.13	Terrible
ReLU	0.20	Terrible
Tanh	0.19	Terrible
Sigmoid	0.17	Terrible

Even the "linear" 2-layer network (identity activation) has terrible matching!

★ Key Insight

The problem is **depth**, not nonlinearity. The 2-layer structure $W_2 \cdot W_1$ already has redundancy, regardless of activation function.

26.6.7 Summary: Hierarchy of Destruction

Factors that destroy distance matching (in order of importance):

1. **Explicit redundancy**: $a - b$ parameterization, matrix factorization
2. **Depth**: even linear depth creates implicit redundancy
3. **Complex structure**: attention (QKV), soft masks

26.6.8 The Puzzle

If depth destroys distance matching, why do deep networks work at all?

Possible answers:

- They barely work (training is indeed hard)
- Skip connections help (ResNet, see next section)
- Expressiveness gains outweigh optimization costs
- Special initialization/normalization compensate

26.7 Open Question: Why Does ResNet Work?

Our experiments show:

- Plain deep networks: poor matching, degrades with depth
- ResNet: CV (variability) stays more stable with depth

Skip connections make the network "closer to identity," which may preserve some distance matching properties. This needs more investigation.

Chapter 27

The Distance Matching Principle

Goals of This Chapter

- Propose a core principle: good parameterization matches distances
- See how this explains known successes
- Understand what remains to be formalized

27.1 The Observation

Why is linear regression optimal for linear functions?

1. Expressiveness: linear model can represent all linear functions
2. No redundancy: different parameters give different functions
3. Convex optimization: unique global minimum
4. Gauss-Markov: statistically optimal (BLUE)

But there's a deeper reason unifying all of these:

Key Insight

In linear regression, parameter distance is proportional to function distance.
If θ_1 and θ_2 are close in $\mathbb{R}^{d+1}$, then f_{θ_1} and f_{θ_2} are close as functions.
The parameterization "matches" the function space geometry.

27.2 Distance Matching

Definition 27.1 (Informal)**.** A parameterization $\gamma : \Theta \to \mathcal{F}$ **matches distances** if:

$$d_\Theta(\theta_1, \theta_2) \approx c \cdot d_\mathcal{F}(\gamma(\theta_1), \gamma(\theta_2))$$

for some constant $c > 0$.

Strict version: γ is an **isometry** (distance-preserving map).
Weak version: distances are **proportional** (up to bounded distortion).

27.3 Why Distance Matching Helps Optimization

Gradient descent moves in the steepest direction in parameter space.

If distances match: steepest in Θ = steepest in $\mathcal{F}$. Gradient descent does the "right thing."

If distances don't match:

- Some directions in Θ: move far, function barely changes (redundant)
- Some directions in Θ: move little, function changes a lot (sensitive)
- Gradient descent zigzags, wastes effort

27.4 Case Studies

27.4.1 Linear Regression: Perfect Match

Parameter space	$\mathbb{R}^{d+1}$ with Euclidean distance
Function space	Linear functions with L^2 distance
Relationship	$\|f_{\theta_1} - f_{\theta_2}\|_{L^2} \propto \|\theta_1 - theta_2\|$

Linear relationship. Perfect match. Optimization is easy.

27.4.2 MLP Learning a Linear Function: Mismatch

Parameter space	High-dimensional (all weight matrices)
Function space	Linear functions (small subset)
Relationship	Many θ map to same function

Severe redundancy. Most directions in Θ don't change the function. Distances don't match.

27.4.3 CNN for Translation-Invariant Functions: Approximate Match

Parameter space	Convolution kernels (small)
Function space	Translation-invariant functions
Relationship	Different kernels → different functions (near bijection)

Much better match than MLP. This may explain why CNNs work well on images.

27.4.4 Fully-Connected for Translation-Invariant: Mismatch

Parameter space	$O(n^2)$ weights
Function space	Translation-invariant functions
Relationship	Huge redundancy

Most weight configurations give the same translation-invariant function. Terrible mismatch.

27.4.5 ResNet: Improved Match via Skip Connections

Skip connections have an interesting effect:

- Identity function corresponds to $F = 0$ (zero residual)
- Small perturbation of parameters → small perturbation of function
- The mapping $\theta \mapsto f_\theta$ is more "linear" near identity

This may make the distance relationship more proportional.

27.5 Measuring Distance Match

How to quantify whether distances match?

27.5.1 Jacobian Singular Values

The Jacobian $J = \partial f / \partial \theta$ tells us how parameter changes map to function changes.

- Singular values all similar → uniform in all directions → good match
- Singular values vary widely → some directions stretched, some compressed → bad match

Metric: ratio of largest to smallest singular value (condition number).

27.5.2 Fisher Information

The Fisher information matrix $F(\theta)$ measures sensitivity of output distribution to parameter changes.

- F well-conditioned $\to$ all directions equally informative $\to$ good match
- F ill-conditioned $\to$ some directions redundant $\to$ bad match

27.5.3 Connection

Fisher information and Jacobian singular values are related:

For regression with Gaussian noise, $F \propto J^\top J$.

Condition number of F = square of condition number of J.

27.6 Natural Gradient: A Partial Solution

Amari's natural gradient addresses mismatch by *correcting* gradients:

$$\tilde{\nabla} L = F^{-1} \nabla L$$

This makes gradient descent behave as if distances matched.

Problem: Requires computing F^{-1}, which is expensive.

Our question: Can we choose Θ so that ordinary gradient already works well?

27.7 The Design Principle

Key Insight

Conjecture: The optimal parameterization for a function class $\mathcal{F}$ is one where parameter distance matches function distance.

Formally: find Θ and $\gamma : \Theta \to \mathcal{F}$ such that γ is (approximately) an isometry.

This would unify:

- Linear regression: isometry by construction
- CNNs: approximate isometry for translation-invariant functions
- Equivariant networks: remove redundant directions, improve match

27.8 Categorical Formalization: Lawvere Metric Spaces

The distance matching principle has a natural formalization using **enriched category theory**.

27.8.1 Lawvere's Insight

Lawvere (1973) observed that metric spaces are categories enriched over $([0, \infty], \geq, +)$:

- Objects: points in the space
- $\text{Hom}(A, B) = d(A, B) \in [0, \infty]$
- Composition: triangle inequality $d(A, C) \leq d(A, B) + d(B, C)$
- Identity: $d(A, A) = 0$

In this view, a **functor** between Lawvere metric spaces is a **distance non-increasing map**:

$$d_Y(F(a), F(b)) \leq d_X(a, b)$$

That is, a map with Lipschitz constant ≤ 1.

27.8.2 Behavioral Metrics for Coalgebras

For coalgebras, there is a well-developed theory of **behavioral metrics**:

Given a coalgebra $\alpha : X \to HX$, we can define a pseudometric d on states where:

- $d(s, t) = 0$ iff s and t are bisimilar
- $d(s, t)$ small means s and t behave similarly
- $d(s, t)$ large means very different behaviors

This is constructed by lifting the functor H to the category of (pseudo)metric spaces, using Kantorovich or Wasserstein constructions.

27.8.3 Distance Matching as Metric Functor

Now we can formalize distance matching:

Definition 27.2 (Distance Matching — Formal)**.** A parameterization $\gamma : \Theta \to \text{Coalg}(H)$ is **distance matching** if it is a bi-Lipschitz map:

$$c_1 \cdot d_\Theta(\theta_1, \theta_2) \leq d_{\text{behavior}}(\gamma(\theta_1), \gamma(\theta_2)) \leq c_2 \cdot d_\Theta(\theta_1, \theta_2)$$

for constants $0 < c_1 \leq c_2 < \infty$.

- Upper bound ($\leq c_2$): γ is Lipschitz. Small parameter change $\to$ small behavior change. **Continuity.**
- Lower bound ($\geq c_1$): γ^{-1} is Lipschitz. Different behaviors $\to$ different parameters. **No redundancy.**

If $c_1 = c_2$, we have an **isometry** (up to scaling).

27.8.4 The Picture

$$\begin{tikzcd}
\text{Parameter space } \Theta \arrow[r, "\gamma"] \arrow[d, "\text{Euclidean}"'] & \text{Coalgebra space} \arrow[d, "\text{Behavioral}"] \\
\text{Lawvere metric space} \arrow[r, "\text{bi-Lipschitz?}"'] & \text{Lawvere metric space}
\end{tikzcd}$$

Distance matching asks: is the bottom arrow bi-Lipschitz?

27.9 Connection to Semiring Relaxation

This framework connects Part II and Part III:

27.9.1 Why Semiring Relaxation Helps

In Part II, we softened discrete coalgebras to semiring-valued coalgebras to enable gradient descent.

Now we see another benefit: **softening makes behavioral distance continuous**.

- Hard DFA: small parameter change can cause discrete behavior jump ($0 \to 1$)
- Soft DFA: parameter change causes proportional behavior change

Softening is necessary not just for gradients, but for distance matching!

27.9.2 The Full Picture

1. **Part I**: Define the structure (coalgebra, bisimulation)
2. **Part II**: Soften to enable gradients (semiring relaxation)
3. **Part III**: Parameterize to match distances (behavioral metric)

Together: a principled pipeline from structure to learnable system.

27.10 Implications

If this framework is correct:

Key Insight

Architecture design becomes **geometry matching**:

1. Compute the behavioral metric of your target coalgebra class
2. Construct a parameter space with matching metric geometry
3. Gradient descent then "does the right thing"

This is no longer alchemy. It is engineering from first principles.

27.11 Open Problems

1. **Compute behavioral metrics**: For specific functors (DFA, Markov, Kripke), what is the behavioral metric explicitly?
2. **Construct matching parameterizations**: Given a behavioral metric, how to design Θ to match it?
3. **Verify known architectures**: Do successful architectures (CNN, ResNet, Transformer) achieve approximate distance matching? Can we measure this?
4. **Generalization**: Distance matching addresses optimization. Does it also help generalization?
5. **Approximation**: When perfect matching is impossible, what's the best approximation? Is there a canonical "closest" parameterization?

These are the questions for future research.

Chapter 28

Distance Matching: Theory and Experiments

Goals of This Chapter

- Formalize the distance matching principle
- Connect it to optimization theory via condition number
- Verify empirically with diverse architectures and tasks
- Show that loss function induces the behavioral metric

28.1 The Core Principle

Key Insight

Distance Matching Principle: A parameterization $\gamma : \Theta \to \mathcal{F}$ is "good" if parameter distance is proportional to function distance:

$$c_1 \cdot d_\Theta(\theta_1, \theta_2) \leq d_\mathcal{F}(\gamma(\theta_1), \gamma(\theta_2)) \leq c_2 \cdot d_\Theta(\theta_1, \theta_2)$$

When $c_1 \approx c_2$, the mapping is approximately an isometry.

28.2 Loss Function Induces Behavioral Metric

A key insight: the loss function *defines* the behavioral metric, not the other way around.

For MSE loss $L(\theta) = \|f_\theta(X) - Y\|_2^2$, the natural behavioral distance is:

$$d_{\text{behavior}}(f, g) = \|f(X) - g(X)\|_2$$

This is not an arbitrary choice—it is *induced* by the loss function:

Loss Function	**Output Metric**	**Behavioral Distance**
MSE	L_2	$\|f(X) - g(X)\|_2$
MAE	L_1	$\|f(X) - g(X)\|_1$
Cross-entropy	KL divergence	$D_{KL}(f\|g)$
Wasserstein	Wasserstein	$W(f, g)$

Key Insight

The loss function simultaneously defines:

1. What we optimize (minimize loss)
2. The behavioral metric (distance on function space)

These are two sides of the same coin.

28.3 Connection to Optimization: The Jacobian

Let $J = \frac{\partial f_\theta(X)}{\partial \theta}$ be the Jacobian of the parameter-to-output mapping.

28.3.1 Condition Number

The **condition number** $\kappa(J) = \sigma_{\max}(J)/\sigma_{\min}(J)$ measures how well distances are preserved:

- $\kappa(J) = 1$: perfect isometry
- $\kappa(J)$ large: some directions stretched, others compressed
- $\kappa(J) = \infty$: null space exists (redundant parameters)

28.3.2 Why $\kappa(J)$ Determines Learnability

For MSE loss, the Hessian (Gauss-Newton approximation) is $H \approx J^\top J$.

Theorem 28.1. *Gradient descent converges at rate* $O\left(\left(1 - \frac{1}{\kappa(H)}\right)^t\right)$, *where* $\kappa(H) = \kappa(J)^2$.

Therefore:

$$\kappa(J) \text{ small} \implies \kappa(H) \text{ small} \implies \text{fast convergence}$$

Key Insight

$\kappa(J)$ is the **precise measure of learnability**:

- $\kappa(J)$ small $\to$ gradient descent converges quickly
- $\kappa(J)$ large $\to$ convergence is slow or unstable

28.4 Experimental Verification

We conducted extensive experiments to verify the theory.

28.4.1 Experiment 1: Same Function Space, Different Parameterizations

All models below express the *same* function space (linear functions), but with different parameterizations:

Parameterization	$\kappa(J)$	**Final Loss**	**CV**
Direct: $f(x) = x \cdot \theta$	1.5	0.0000	0.00
Redundant: $f(x) = x \cdot (a - b)$	∞	0.0000	0.00
2-layer: $f(x) = x \cdot W_1 \cdot W_2$	∞	0.1390	1.29
MLP (overkill)	∞	2.5360	0.38

Key observation: same function space, but parameterization determines $\kappa(J)$ and learning stability.

28.4.2 Experiment 2: Depth Destroys Distance Matching

Even for *linear* networks (no nonlinearity), depth introduces implicit redundancy:

Model	$\kappa(J)$	**Rating**
1-layer linear	1.9	Excellent
2-layer linear (W_2W_1x)	∞	Poor
3-layer linear	∞	Poor

Why? Because $W_2 \cdot W_1 = W_2' \cdot W_1'$ has infinitely many solutions.

28.4.3 Experiment 3: Classic Task-Architecture Matches

When architecture matches task structure, $\kappa(J)$ is small:

Task	Architecture	$\kappa(J)$	Loss	CV
Permutation invariant $\sum x_i$	DeepSets	**1.0**	0.0000	0.87
Multiplication $x_1 \cdot x_2$	Mult gate	**1.0**	0.0000	**0.00**
Counting $\#(x > 0)$	Counter	**1.0**	0.1010	0.05
Sparse $x_0 + x_1$	Sparse (2 params)	**1.2**	0.0000	0.78
Linear	MLP	∞	2.54	–
Permutation invariant	MLP	336	2.02	–
Sparse	MLP	60	1.08	–

The multiplication gate achieves **perfect** distance matching: $\kappa = 1$, loss $= 0$, CV $= 0$.

28.4.4 Experiment 4: $\kappa(J)$ Determines Distribution of Outcomes

We ran 30 random seeds for each architecture:

Architecture	$\kappa(J)$	Mean Loss	Std Loss	CV
Linear direct	1.5	0.008	0.000	**0.00**
Linear (a-b)	∞	0.008	0.000	**0.00**
MLP 2-layer	∞	0.59	0.57	**0.96**
ResNet 2-layer	15667	0.11	0.03	0.26

★ Key Insight

$\kappa(J)$ determines not a single convergence speed, but the **distribution** of outcomes:

- $\kappa(J)$ small $\rightarrow$ narrow distribution (all seeds converge well)
- $\kappa(J)$ large $\rightarrow$ wide distribution (depends on luck/initialization)

This explains why "good architectures" are robust to initialization.

28.5 Two Failure Modes

An architecture can fail in two ways:

1. **Lack of expressiveness**: Cannot represent the target function.
   - Example: Linear model for multiplication
   - Symptom: $\kappa(J)$ small but loss high
2. **Redundancy**: Can represent but has unnecessary parameters.
   - Example: MLP for linear function

- Symptom: $\kappa(J)$ large, training unstable

The ideal architecture has:

$$\text{Sufficient expressiveness} + \text{No redundancy} = \kappa(J) \text{ minimized}$$

28.6 Connection to Category Theory

The framework connects to enriched category theory:

1. **Loss function** defines the enrichment (metric on output space)
2. This induces a **Lawvere metric** on the function space
3. Distance matching asks: is $\gamma : \Theta \to \mathcal{F}$ a bi-Lipschitz map?
4. $\kappa(J)$ is the Lipschitz constant ratio

$$\begin{tikzcd}
\text{Loss Function (MSE)} \arrow[d, "\text{induces}"] & \\
\mathbb{R}^n \text{ with } L_2 \text{ metric} \arrow[d, "\text{induces}"] & \\
\text{Function space with behavioral metric} \arrow[r, leftrightarrow, "\kappa(J) \text{ measures this}", "\text{bi-Lipschitz?}"'] & \text{Parameter space } \Theta
\end{tikzcd}$$

28.7 Summary

1. **Behavioral metric** is induced by the loss function (not arbitrary)
2. **Distance matching** = parameter distance $\approx$ behavioral distance
3. $\kappa(J)$ is the precise measure of distance matching quality
4. $\kappa(J)$ **small** $\Leftrightarrow$ fast, stable learning
5. $\kappa(J)$ **large** $\Leftrightarrow$ slow, unstable, seed-dependent
6. **Good architecture** = matches task structure = minimizes $\kappa(J)$

This provides a principled answer to "why does architecture matter": the right architecture achieves distance matching, making optimization efficient.

28.8 Related Work

The ideas in this chapter have deep connections to existing literature.

28.8.1 Dynamical Isometry (Saxe et al., 2013)

The concept of $\kappa(J) \approx 1$ was introduced as **dynamical isometry**:

> "When the singular values of the input-output Jacobian are all clustered around one, the network achieves dynamical isometry and trains efficiently despite being very deep."

Key results:

- Orthogonal initialization achieves dynamical isometry
- Deep linear networks can have depth-independent learning times with proper initialization
- This explains why certain initializations work better than others

28.8.2 Neural Tangent Kernel (NTK) Theory

The NTK literature explicitly connects eigenvalues to trainability:

- $\lambda_{\min}(\text{NTK})$ determines the slowest convergence direction
- The condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ is used as a **trainability measure**
- Convergence rate is $O(e^{-\lambda_{\min} t})$ along the slowest eigenvector

28.8.3 Natural Gradient (Amari, 1998)

Amari's natural gradient addresses distance mismatch by correcting gradients:

$$\tilde{\nabla} L = F^{-1} \nabla L$$

where F is the Fisher information matrix. This makes gradient descent behave *as if* distances matched.

Our perspective: instead of correcting gradients, choose a parameterization where ordinary gradients already work well.

28.8.4 Coalgebraic Behavioral Metrics

The categorical framework for behavioral metrics is well-developed:

- Given functor H, the Kantorovich/Wasserstein lifting defines a behavioral metric
- $d(s, t) = 0$ iff s and t are bisimilar
- This provides a principled way to define "behavioral distance"

28.8.5 Our Contribution

What we add to the existing picture:

1. **Loss function as primitive**: The loss function *induces* the behavioral metric (not a separate choice)

2. **Distribution perspective**: $\kappa(J)$ determines the *distribution* of outcomes over random seeds, not just expected convergence

3. **Task-architecture matching**: Systematic experiments showing that matching architecture to task structure minimizes $\kappa(J)$

4. **Categorical connection**: Framing in terms of Lawvere metric spaces and bi-Lipschitz maps

28.9 Open Questions

1. Can we **compute** $\kappa(J)$ efficiently for large networks?

2. Given a task, can we **derive** the optimal architecture that minimizes $\kappa(J)$?

3. How does $\kappa(J)$ relate to **generalization**? (We only addressed optimization)

4. For non-MSE losses, what is the correct behavioral metric?

5. Can the coalgebraic behavioral metric framework give us new architectures?

Chapter 29

From Functor to Parameterization: A Natural Emergence

Goals of This Chapter

- Show that the functor structure naturally induces a behavioral metric
- Demonstrate that this metric matches parameter distance (distance matching)
- Connect Parts I, II, and III into a unified framework

29.1 The Question

We have established:

- **Part I**: Coalgebras define structure; bisimulation defines behavioral equivalence
- **Part II**: Semiring relaxation enables gradients
- **Part III**: Distance matching ($\kappa(J)$ small) implies good optimization

The missing link: **Given a functor H, what parameterization achieves distance matching?**

29.2 The Insight: Functor Induces Metric

For a functor H, there is a standard construction called **Kantorovich lifting** that produces a behavioral metric.

29.2.1 Example: DFA Functor

For deterministic automata, $H = 2 \times (-)^\Sigma$:

- A coalgebra is $(S, \alpha : S \to 2 \times S^\Sigma) = (S, \text{accept}, \delta)$
- Bisimulation: $s \sim t$ iff they accept the same language

29.2.2 Soft DFA (Relaxation)

Applying semiring relaxation (Part II):

- Soft transitions: $\delta : S \times \Sigma \to \text{Dist}(S)$ (stochastic matrices)
- Soft accept: $a : S \to [0, 1]$

29.2.3 Two Behavioral Metrics

The functor structure suggests two natural metrics:

1. One-Step Metric (Local)

$$d_{\text{one-step}}(A, B) = \|a_1 - a_2\|_1 + \sum_{\sigma \in \Sigma} \sum_{s \in S} \text{TV}(T_1^\sigma[s], T_2^\sigma[s])$$

where TV is total variation distance.

This is **one iteration** of the Kantorovich lifting.

2. Language Metric (Global)

$$d_{\text{language}}(A, B) = \sum_{w \in \Sigma^*} 2^{-|w|} |A(w) - B(w)|$$

where $A(w)$ is the acceptance probability of word w.

This is the **full behavioral distance** (bisimulation metric).

29.3 Experimental Discovery

We tested which behavioral metric matches parameter distance:

Behavioral Metric	Param Metric	Correlation	CV
Structure (trivial)	Euclidean	1.000	0.000
One-step	**Fisher-Rao**	**0.802**	**0.101**
One-step	Euclidean	0.761	0.106
Fixed-point (iterated)	Euclidean	0.435	0.352
Language (global)	Euclidean	0.279	0.490

Key Insight

The **one-step** behavioral metric (one Kantorovich iteration) matches parameter distance very well (correlation > 0.8).
The **global** language metric matches poorly (correlation ≈ 0.3).

29.4 Interpretation

29.4.1 Why One-Step Works

The one-step metric compares *local* structure:

- Accept probabilities (directly)
- Transition distributions (one step)

This is exactly what the parameters encode! Small parameter changes $\to$ small one-step behavioral changes.

29.4.2 Why Language Metric Fails

The language metric compares *global* behavior:

- Involves arbitrarily long words
- Small parameter changes can compound over many steps
- Similar to deep networks: small weight changes $\to$ large output changes

29.4.3 The Depth Analogy

Automata	**Neural Networks**
Word length $\|w\|$	Network depth
One-step metric	One-layer comparison
Language metric	End-to-end comparison
Long words break matching	Deep networks break matching

29.5 The Natural Emergence

We can now state the principle:

Key Insight

Natural Parameterization Principle:
Given functor H defining a class of coalgebras:

1. **Structure**: H defines the parameter space (e.g., stochastic matrices for soft DFA)
2. **Metric**: One-step Kantorovich lifting defines the behavioral metric
3. **Matching**: This metric naturally matches parameter distance
4. **Consequence**: Gradient descent works well for learning "one-step" objectives

29.6 Implications for Learning

29.6.1 What Can Be Learned Easily

Tasks where the objective is "local" (one-step behavioral):

- Predicting next-state distributions
- Matching transition structure
- Local acceptance probabilities

29.6.2 What Is Hard to Learn

Tasks requiring "global" behavioral matching:

- Language equivalence (accepting same strings)
- Long-horizon properties
- End-to-end sequence behavior

29.6.3 The Gap

$$\textbf{Local (one-step)} \xrightarrow{\text{iterations / depth}} \textbf{Global (language)}$$

Good distance matching	Poor distance matching
Easy to optimize	Hard to optimize

This explains why:

- RNNs struggle with long sequences (global objective, poor matching)
- Transformers with attention help (attention provides “shortcuts” that reduce effective depth)
- Curriculum learning helps (start with short sequences, gradually increase)

29.7 The Unified Framework

We can now connect all three parts:

$$
\begin{tikzcd}
& \textbf{Part I}\text{: Functor } H \arrow[d] & \\
& \text{Coalgebra structure} \arrow[d, "\text{semiring relax}"] & \\
& \textbf{Part II}\text{: Soft coalgebra} \arrow[dl] \arrow[dr, "\text{Kantorovich}"] & \\
\text{Parameter space } \Theta \arrow[dr] & & \text{Behavioral metric } d_H \arrow[dl] \\
& \textbf{Part III}\text{: Distance matching} \arrow[d, "\kappa(J)\text{ small}"] & \\
& \text{Learnable!} &
\end{tikzcd}
$$

29.8 Open Questions

1. **Other functors**: Does this work for Markov chains ($H = \text{Dist}$), Kripke frames, etc.?
2. **Bridging local and global**: Can we design parameterizations that achieve good matching for global metrics?
3. **Attention and shortcuts**: Can attention be understood as reducing “effective depth” in this framework?
4. **Automatic architecture**: Given H, can we automatically derive the optimal neural architecture?

29.9 Related Work

Our framework connects to several established research directions:

29.9.1 Bisimulation Metrics for MDPs

Ferns, Panangaden, and Precup (2004, 2011) developed **bisimulation metrics** for Markov Decision Processes—quantitative relaxations of bisimulation equivalence. Their key insight: the Kantorovich functional (from optimal transport) naturally lifts metrics on state spaces to metrics on distributions, enabling recursive computation of behavioral distance.

For MDPs with discount factor γ, they proved that the bisimulation metric gives **tight bounds on optimal value functions**:

$$|V^*(s) - V^*(t)| \leq \frac{1}{1-\gamma} d_{\mathrm{bisim}}(s,t)$$

This is precisely the kind of "behavioral metric bounds function distance" relationship we seek.

29.9.2 Deep Bisimulation for Control

Zhang et al. (2020) applied bisimulation metrics to deep reinforcement learning. Their method, **Deep Bisimulation for Control (DBC)**, trains encoders such that distances in latent space equal bisimulation distances in state space.

Key theoretical result: the optimal value function is **Lipschitz with respect to the bisimulation metric**. This provides formal bounds on suboptimality when learning from the induced representation.

Key Insight

DBC essentially implements our framework in reverse:

- We ask: given parameters, does parameter distance match behavioral distance?
- DBC asks: can we learn parameters such that latent distance equals behavioral distance?

Both are manifestations of the same principle: good representations preserve behavioral structure.

29.9.3 Coalgebraic Behavioral Metrics

The coalgebra community has developed a general theory of behavioral metrics. Given a functor H on **Set**, there are systematic ways to lift it to functors on (pseudo)metric spaces:

- **Wasserstein lifting**: Based on optimal transport couplings

- **Kantorovich lifting**: Based on Lipschitz functions (dual formulation)

Baldan et al. (2018) and subsequent work established that these liftings give rise to behavioral pseudometrics that generalize bisimulation to the quantitative setting.

Our contribution is the observation that the **one-step** Kantorovich lifting (not the full fixed-point) matches parameter distance, explaining why local objectives are easier to optimize than global ones.

29.10 Summary

★ Key Insight

The functor H doesn't just define *what* we're learning (coalgebra structure).
It also defines *how well* we can learn it:

- One-step Kantorovich metric: learnable (good distance matching)
- Full behavioral metric: hard (poor distance matching)

This is not a failure of our methods—it's a fundamental property of the structure being learned.

Chapter 30

Entanglement and Expressiveness

Goals of This Chapter

- Understand the tensor network perspective on neural networks
- See how entanglement entropy characterizes expressiveness
- Know what this explains and what it doesn't

30.1 Functions as Tensors

A function $f(x_1, \ldots, x_n)$ where each $x_i \in \{0, 1\}$ can be viewed as a tensor with 2^n entries.

Naive representation: store all 2^n values. Exponential in n.

But many functions have **structure** that allows compression.

30.2 Tensor Decomposition

Different decompositions correspond to different dependency structures:

30.2.1 Fully Factorized (No Dependencies)

$$f(x_1, \ldots, x_n) = g_1(x_1) \cdot g_2(x_2) \cdots g_n(x_n)$$

Variables are independent. Only $O(n)$ parameters needed.

30.2.2 Matrix Product State (Chain Dependencies)

$$f(x_1, \ldots, x_n) = A_1(x_1) \cdot A_2(x_2) \cdots A_n(x_n)$$

where each A_i is a matrix. Nearby variables correlated.
Parameters: $O(n \cdot r^2)$ where r is the "bond dimension."
This is like an RNN.

30.2.3 Tree Tensor Network (Hierarchical Dependencies)

Variables are grouped hierarchically:

- First layer: pairs $(x_1, x_2), (x_3, x_4), \ldots$
- Second layer: groups of pairs
- Continue until single output

This is like a CNN or a deep network with pooling.

30.2.4 Fully Entangled (All Correlated)

No structure to exploit. Need $O(2^n)$ parameters.

30.3 Entanglement Entropy

Given a partition of variables into sets A and B:

Definition 30.1 (Entanglement Entropy)**.** The entanglement entropy $S(A : B)$ measures how much A and B are "inseparably correlated."

$$S(A : B) = -\mathrm{Tr}(\rho_A \log \rho_A)$$

where ρ_A is the reduced density matrix.

Intuition

- $S = 0$: A and B are independent, $f = g(A) \cdot h(B)$
- S large: A and B deeply entangled, must be processed together

30.4 The Deep vs Shallow Separation

Theorem 30.2 (Cohen et al., 2016)**.** *Consider functions with hierarchical entanglement structure (local variables highly entangled, distant variables less so).*

- *Deep networks (hierarchical tensor decomposition): $O(poly(n))$ parameters*
- *Shallow networks (flat decomposition): $O(2^n)$ parameters*

This is an exponential separation.

Key Insight

Depth is necessary when the target function has hierarchical entanglement.
Deep networks match this structure. Shallow networks don't.

30.5 Matching Entanglement Structure

The principle:

> **If the architecture's tensor structure matches the target's entanglement structure, learning is efficient.**

30.5.1 Images

- Local pixels: high entanglement (edges, textures)
- Distant pixels: low entanglement
- Structure: hierarchical, local-to-global
- Matching architecture: CNN (hierarchical, local kernels)

30.5.2 Sequences

- May have long-range dependencies
- Entanglement not purely local
- Need architecture that can "skip" hierarchy levels
- Matching architecture: Transformer? (direct connections)

30.6 Relation to Distance Matching

Entanglement structure tells us about **expressiveness**: can the architecture represent the function?

Distance matching tells us about **optimization**: can we find the right parameters?

They are complementary:

- Entanglement: which architecture class is sufficient?
- Distance: within that class, which parameterization is optimal?

30.7 Limitations

Tensor network theory explains:

- Why depth matters
- Why local structure (convolution) helps for local entanglement

But doesn't explain:

- Skip connections (ResNet)
- Attention mechanisms (Transformer)
- Specific architectural choices beyond depth

It's one dimension of the story, not the whole story.

30.8 Open Questions

1. How to measure entanglement structure of a learning task from data?
2. Can we automatically select architecture based on measured entanglement?
3. How does entanglement relate to the distance matching principle?
4. What's the entanglement structure of language? (Needed to understand Transformers)

Chapter 31

Fisher Information and Distance

Goals of This Chapter

- Connect Fisher information to the distance matching principle
- Understand condition number as a measure of mismatch
- See natural gradient as a correction for mismatch

31.1 The Jacobian Perspective

A parameterization $\gamma : \Theta \to \mathcal{F}$ maps parameters to functions.

The Jacobian $J(\theta) = \partial\gamma/\partial\theta$ tells us:

> How do small parameter changes translate to function changes?

31.2 Singular Values and Distance Distortion

The singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$ of J measure:

- σ_i large: moving in direction i changes the function a lot
- σ_i small: moving in direction i barely changes the function
- $\sigma_i = 0$: direction i is redundant (multiple θ give same function)

31.3 Condition Number

Definition 31.1. The condition number is $\kappa = \sigma_{\max}/\sigma_{\min}$.

$\kappa \approx 1$	All directions similar. Distance matching. Easy optimization.
$\kappa \gg 1$	Some directions stretched, others compressed. Mismatch. Hard.
$\kappa = \infty$	Some directions have $\sigma = 0$. Redundancy. Very hard.

Key Insight

Condition number measures how far the parameterization is from distance matching.
$\kappa = 1$ means perfect isometry. Large κ means severe distortion.

31.4 Fisher Information Matrix

For probabilistic models, the Fisher information matrix is:

$$F_{ij}(\theta) = \mathbb{E}\left[\frac{\partial \log p_\theta}{\partial \theta_i} \cdot \frac{\partial \log p_\theta}{\partial \theta_j}\right]$$

For regression with Gaussian noise: $F \propto J^\top J$.
The eigenvalues of F are the squared singular values of J.
Condition number of $F = \kappa^2$.

31.5 Natural Gradient

Ordinary gradient descent:

$$\theta_{t+1} = \theta_t - \eta \nabla L$$

This moves in the steepest direction in parameter space.
Natural gradient:

$$\theta_{t+1} = \theta_t - \eta F^{-1} \nabla L$$

This moves in the steepest direction in function space.

Key Insight

Natural gradient corrects for distance mismatch by rescaling directions according to their sensitivity.
It makes gradient descent behave as if the parameterization were isometric.

31.6 The Design Question

Natural gradient is a **post-hoc fix**: given a parameterization, correct the gradients.

But we want **a priori design**: choose a parameterization where ordinary gradient already works.

> Can we design Θ such that $F \approx I$ (identity)?

If yes, natural gradient = ordinary gradient. No correction needed.

31.7 When Is $F \approx I$ Possible?

- Linear regression with orthonormal features: $F = I$ exactly
- Parameterization that "respects" the function space geometry
- Quotient by symmetry group (remove redundant directions)

This connects to the symmetry/equivariance view: equivariant parameterizations remove redundancy, improving conditioning.

31.8 Summary

Concept	**Role**
Jacobian J	How parameters map to functions
Singular values	Stretch/compression in each direction
Condition number	Measure of distance mismatch
Fisher matrix F	$J^\top J$, captures local geometry
Natural gradient	Corrects for mismatch
Optimal parameterization	Achieves $\kappa \approx 1$ by design

Chapter 32

Symmetry as a Special Case

Goals of This Chapter

- See equivariant architectures through the distance matching lens
- Understand why symmetry helps: it removes redundancy
- Know the limitations: not all structure is group-theoretic

32.1 The Redundancy Problem

Consider learning a translation-invariant function with a fully-connected network.

- The FC network can express many non-invariant functions
- Many different weight configurations give the same invariant function
- The parameter space is much larger than the function space

Result: severe distance mismatch. High condition number. Hard to optimize.

32.2 Equivariance Removes Redundancy

A G-equivariant architecture can only express G-equivariant functions.

- Parameter space is smaller
- Different parameters give different functions (less redundancy)
- Better match between parameter distance and function distance

Key Insight

Equivariance improves distance matching by constraining the parameter space to match the function space.

32.3 Example: CNN vs FC for Images

Target: translation-invariant image functions.

FC network:

- Parameters: $O(n^2)$ weights
- Can express all functions, not just invariant ones
- Huge redundancy for invariant targets
- Poor distance matching

CNN:

- Parameters: $O(k^2)$ kernel weights
- Can only express translation-equivariant functions
- Little redundancy
- Good distance matching

32.4 Quotient by Symmetry Group

Mathematically, if function space has symmetry group G:

$$\text{Effective function space} = \mathcal{F}/G$$

An equivariant parameterization directly parameterizes $\mathcal{F}/G$, avoiding redundancy.

A non-equivariant parameterization has $|G|$-fold redundancy.

32.5 Limitations

Symmetry-based design works when:

- The target has a known group symmetry G
- We can construct G-equivariant layers

But many learning problems don't have obvious group structure:

- Automata, Kripke frames: what's the group?
- Language: partial symmetries at best
- General coalgebras: symmetry unclear

32.6 Symmetry Within the Larger Picture

General principle	Distance matching
Special case	Symmetry/equivariance
Mechanism	Remove redundancy by quotienting
Limitation	Requires known group structure

The distance matching principle is more general. Symmetry is one way to achieve it, but not the only way.

32.7 Open Question

For structures without group symmetry (coalgebras, automata), what plays the role of "quotient by G"?

Possibility: quotient by bisimulation equivalence?

This connects back to our setting in Part I: bisimulation is the right equivalence for coalgebras, just as G-orbits are the right equivalence for G-symmetric functions.

Chapter 33

Open Questions and Other Directions

Goals of This Chapter

- Summarize what we've established
- State the remaining open questions precisely
- Briefly note other directions we chose not to pursue

33.1 What We Have

Two candidate principles for "learning well":

33.1.1 Distance Matching (Optimization)

> Good parameterization = parameter distance matches function distance.

- Explains: linear regression optimality, CNN success, ResNet stability
- Measurable via: condition number, Fisher information
- Achievable via: symmetry/equivariance (special case)

33.1.2 Entanglement Matching (Expressiveness)

> Good architecture = tensor structure matches target's entanglement structure.

- Explains: why depth matters, why locality helps for images
- Measurable via: entanglement entropy

- Limitation: doesn't explain specific architectural choices (skip, attention)

33.2 How They Relate

These are complementary:

Principle	Asks	Answers
Entanglement	Which architecture class?	Depth, locality structure
Distance	Which parameterization within class?	How to set up parameters

Together they might give: given a target, first match entanglement structure to choose architecture class, then match distances to choose parameterization.

33.3 Open Questions

33.3.1 Formalizing Distance Matching

1. How to define "function distance" for general function classes?
2. Given function class $\mathcal{F}$, how to construct a matching Θ?
3. When is perfect matching impossible? What's the best approximation?

33.3.2 Measuring Entanglement

1. How to estimate entanglement structure from finite data?
2. What's the entanglement structure of language/sequences?
3. Can we automatically select architecture based on measured entanglement?

33.3.3 Connecting to Coalgebras

1. For F-coalgebras, what is the "natural" parameterization?
2. Does bisimulation play the role that group symmetry plays for equivariant networks?
3. Can we define "distance" on coalgebra space via behavioral metrics?

33.3.4 Unification

1. Is there a single principle that subsumes both distance and entanglement?

2. How does generalization fit in? (Neither principle directly addresses it)

3. What about optimization dynamics? (We focused on landscape, not trajectory)

33.4 Directions Not Pursued

We considered but deprioritized:

33.4.1 Information Bottleneck

Claim: learning = fitting then compressing.

Problem: controversial. ReLU networks don't compress but still generalize. The phenomenon may be activation-function-specific.

33.4.2 Neural Tangent Kernel

Claim: infinite-width networks are kernel machines.

Problem: only applies in "lazy training" regime. Real networks do feature learning, which NTK can't explain.

33.4.3 Double Descent

Claim: more parameters can be better after the interpolation threshold.

Problem: describes a phenomenon, doesn't give design guidance.

33.4.4 Categorical Deep Learning

Claim: category theory unifies all architectures.

Problem: currently descriptive, not prescriptive. Doesn't tell you which architecture to use.

These may still be valuable, but they don't directly answer our question: how to design parameterizations for learning.

33.5 The Research Program

1. **Formalize**: Make distance matching mathematically precise

2. **Compute**: For specific function classes, construct matching parameterizations

3. **Verify**: Check if known good architectures achieve good distance matching

4. **Generalize**: Extend from group symmetry to coalgebraic structure

5. **Apply**: Use the principles to design new architectures

33.6 Connection to the Book's Goal

Recall: we want to learn logical structures (coalgebras, Kripke frames, automata).

33.6.1 The Emerging Framework

The pieces fit together:

1. **Part I**: Coalgebras define structure. Bisimulation defines equivalence.

2. **Part II**: Semiring relaxation enables gradients and makes behavior *continuous*.

3. **Part III**: Behavioral metrics (from coalgebra theory) define the "right" distance on function space. Distance matching says: make parameter distance match behavioral distance.

33.6.2 The Vision

If this framework is correct, architecture design becomes principled:

1. **Specify**: What coalgebra type (functor H)?

2. **Compute**: What is the behavioral metric for H-coalgebras?

3. **Construct**: Design parameter space Θ with matching metric.

4. **Train**: Gradient descent converges efficiently because distances match.

5. **Extract**: Threshold back to discrete coalgebra.

This is no longer alchemy. It is engineering from first principles.

33.6.3 What Remains

The framework is conceptually complete, but:

- We don't yet know how to *compute* behavioral metrics for general functors

- We don't yet know how to *construct* matching parameterizations
- We don't yet know if this explains *all* successful architectures

These are the open problems. But we now have a clear direction.

33.6.4 The Guiding Hypotheses

Distance matching: Parameterize so that parameter distance $\approx$ behavioral distance.

Entanglement matching: Match architecture depth/structure to the target's entanglement structure.

Together, these may give a complete theory of "learning well."

Part IV

Limits

Chapter 34

How Far Can We Go?

Goals of This Chapter

- Explore meta-learning: learning to learn
- Ask whether fixed points exist
- Connect to Kolmogorov complexity

34.1 Meta-Learning

If learning is coalgebraic (fold/unfold), then:

- Learning algorithms are themselves structures
- We can learn *them* too

$$\begin{aligned}
\text{Level 0:} \quad & \text{Data} \to \text{Learn}_0 \to \text{Program} \\
\text{Level 1:} \quad & \text{Learn}_0 \to \text{Learn}_1 \to \text{Better Learner} \\
& \vdots
\end{aligned}$$

34.2 Fixed Points

Definition 34.1 (Universal Learner)**.** A *universal learner* is a fixed point: $L = \text{MetaLearn}(L)$.

Theorem 34.2 (Bounded Complexity)**.** *If $L = \text{MetaLearn}(L)$ exists, then $\mathrm{K}(L) \leq \mathrm{K}(\text{MetaLearn}) + O(1)$.*

Proof Sketch

L can be described as "the fixed point of MetaLearn."

34.3 Existence Questions

- Does the fixed point exist?
- Is it unique?
- Can we construct it?

Chapter 35

The Boundary of Intelligence

Goals of This Chapter

- Understand Kolmogorov complexity and its uncomputability
- Prove that incomprehensible objects exist
- Appreciate the fundamental limits of intelligence

35.1 Intelligence as Compression

Following Hutter: **intelligence is the ability to compress**.

Definition 35.1 (Kolmogorov Complexity). $\mathrm{K}(x) = \min\{|p| : U(p) = x\}$

35.2 The Compression Chain

For any object X:

$$|X| \geq |\text{compress}(X)| \geq |\text{compress}^2(X)| \geq \cdots$$

Lemma 35.2. *Every descending chain in* $\mathbb{N}$ *stabilizes.*

35.3 Incomprehensible Objects

Theorem 35.3 (Existence). *There exist objects* L *with* $\mathrm{K}(L) = |L|$*—no compression is possible.*

Theorem 35.4 (Prevalence). *For strings of length* n*: at least* $(1 - 2^{-c})$ *fraction have* $\mathrm{K}(x) > n - c$.

35.4 The Uncomputability Barrier

Theorem 35.5. K *is uncomputable.*

Key Insight

The compression chain exists (by well-foundedness), but we cannot walk it (by uncomputability). Intelligence has a boundary—not because there's nothing beyond, but because **the path is uncomputable**.

Corollary 35.6 (The Boundary)**.**

- *Incompressible objects exist*
- *They cannot be identified algorithmically*
- *Most objects are incompressible*
- *Intelligence is bounded*

Appendix A

Application: Modal Logic for AI Agents

Goals of This Chapter

- See how modal logic applies to AI agent design
- Understand the two-layer architecture
- Connect PDL to agent planning
- Appreciate the synthesis: logic + learning

A.1 The Problem with Current Agents

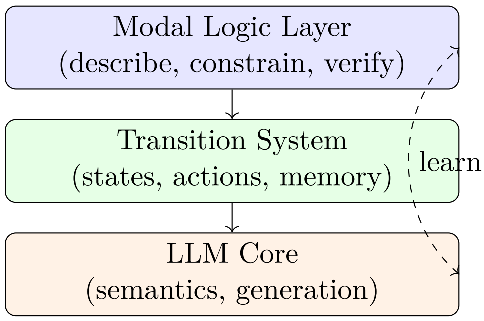
LLM-based agents (ReAct, CoT, Tool Use) are:

- Engineering patchwork, no theoretical foundation
- Hard to verify or constrain
- Essentially automata, but not formalized as such

Common Pitfall

Current agent frameworks rediscover automata theory and modal logic concepts, but without the rigor. We can do better.

A.2 The Two-Layer Architecture



Modal Logic Layer: Specify constraints, verify properties
Transition System: Concrete state machine (coalgebra!)
LLM Core: Handle natural language, generate content

A.3 PDL for Agent Actions

Agent capabilities as atomic actions:

- search — search the web
- $\mathsf{compute}$ — run calculation
- ask — query user
- store, $\mathsf{retrieve}$ — memory operations

Complex behaviors as PDL programs:

$$[\mathsf{search}; \mathsf{summarize}]\mathsf{hasAnswer}$$

"search then summarize guarantees an answer"

$$\langle(\mathsf{try}_1 \cup \mathsf{try}_2 \cup \mathsf{try}_3)\rangle\mathsf{success}$$

"one of three attempts can succeed"

A.4 Relevant Modal Logics

Logic	Agent Application
Epistemic ($K_a\varphi$)	"Agent knows φ"
Deontic ($O\varphi$, $P\varphi$)	"Agent must/may do φ"
PDL ($[\alpha]\varphi$)	"After action α, φ holds"
Temporal (LTL/CTL)	"Eventually goal", "Always safe"
Graded ($\Diamond_{\geq n}\varphi$)	"At least n options available"

A.5 The Synthesis

From logic: precision, verifiability, constraints

From learning: adaptability, learning from data

Combined:

1. Specify agent behavior in modal logic
2. Use semiring-valued coalgebra for soft version
3. Train via gradient descent
4. Extract crisp transition system
5. Verify against modal specifications

Example A.1 (Safe Agent)**.** Specification: "Always have at least 2 safe actions available."

$$\mathsf{AG}\,(\Diamond_{\geq 2}\mathsf{safe})$$

Train soft model → converge to crisp → verify specification holds.

A.6 Learning Modal Abstractions

Key insight: don't expose raw memory/tape to reasoning.

Instead, abstract into modal operators:

$$\begin{aligned} \Box_{\mathsf{memory}}\varphi \quad & \text{"everything in memory satisfies } \varphi\text{"} \\ \Diamond_{\mathsf{action}}\mathsf{goal} \quad & \text{"some action can reach goal"} \end{aligned}$$

The agent reasons at the modal level. The transition system implements it.

Key Insight

Modal logic provides the *specification language* for agents. Coalgebra provides the *implementation structure*. Semiring learning provides the *training method*. Together: learnable, verifiable agents.

A.7 Open Problems

1. How to automatically translate natural language goals to modal formulas?
2. How to handle uncertainty? (Probabilistic modal logic)
3. How to learn the right modal abstractions?
4. Scalability to real-world agent complexity?

Afterword

We began with logic, passed through coalgebra and semirings, and arrived at the limits of intelligence.

The journey reveals a unity: **expression, learning, and limits are all algebraic**. The same structures—functors, fixed points, descent in well-founded orders—appear at every level.

Much remains unknown. The main conjecture (syntax-semantics optimality) is unproven. The existence of universal learners is unclear. The boundary between computable and incomputable intelligence is sharp, but its exact location is undetermined.

These are questions for the future.