

Proof Theory

Archived study guides

These archived materials are no longer updated. This archive was exported from a saved copy of these guides.

Export date: 2026-10-05T19:26:30.244Z

Contents

- Unit 1 – Intro to Proof Theory & Math Logic - 4 guides
- Unit 2 – Propositional Logic: Syntax and Semantics - 4 guides
- Unit 3 – First-Order Logic: Syntax and Semantics - 5 guides
- Unit 4 – Natural Deduction and Sequent Calculus - 4 guides
- Unit 5 – Cut Elimination and Normalization - 4 guides
- Unit 6 – Completeness and Compactness Theorems - 4 guides
- Unit 7 – Gödel's Incompleteness Theorems - 4 guides
- Unit 8 – Second-Order Logic and Higher-Order Logics - 4 guides
- Unit 9 – Intuitionistic Logic and Constructive Proofs - 4 guides
- Unit 10 – Modal Logic and Kripke Semantics - 4 guides
- Unit 11 – Proof Theory and Foundations of Mathematics - 4 guides
- Unit 12 – Proof Theory and Computability Theory - 4 guides
- Unit 13 – Proof Theory Applications in CS - 4 guides
- Unit 14 – Recent Advances in Proof Theory - 4 guides

Unit 1 – Intro to Proof Theory & Math Logic

1.1 Historical development of proof theory

Proof theory emerged from a crisis in math's foundations. Paradoxes like Russell's shook confidence, sparking new approaches. Formalism, intuitionism, and logicism tried to rebuild math on solid ground, each with its own take on what math really is.

Hilbert's ambitious program aimed to prove math's consistency using finite methods. But Gödel's incompleteness theorems showed this was impossible, revealing inherent limits in formal systems. Gentzen's work then opened new paths, using infinitary methods to tackle consistency.

Early Foundations of Mathematics

Philosophical Approaches to Mathematics

- Foundational crisis in mathematics arose in the early 20th century due to the discovery of paradoxes and inconsistencies in the foundations of mathematics (Russell's paradox)
- Formalism, developed by David Hilbert, holds that mathematics is a formal system of symbols and rules, independent of any interpretation or meaning
- Mathematical statements are meaningless formulas manipulated according to fixed rules
- Goal is to prove the consistency of mathematical systems using finite methods
- Intuitionism, founded by L.E.J. Brouwer, maintains that mathematics is a mental construction and that mathematical objects exist only as mental constructs
- Rejects the law of the excluded middle and the existence of actual infinity
- Emphasizes the constructive nature of mathematics and the role of intuition
- Logicism, advocated by Gottlob Frege and Bertrand Russell, seeks to reduce mathematics to logic
- Aims to derive all mathematical truths from purely logical axioms and inference rules
- Russell and Whitehead's Principia Mathematica attempted to provide a logical foundation for mathematics

Hilbert's Program and Gödel's Incompleteness

Hilbert's Program

- Hilbert's program aimed to establish the consistency and completeness of mathematics using finite, formal methods
- Sought to formalize all of mathematics and prove its consistency within the system itself
- Proposed the use of metamathematics, a formal system to study the properties of mathematical systems

- Hilbert believed that every mathematical problem could be solved by finding a proof or demonstrating that no proof exists
- Emphasized the importance of the axiomatic method and the role of formal systems in mathematics
- Aimed to resolve the foundational crisis and provide a secure foundation for mathematics

Gödel's Incompleteness Theorems

- Gödel's first incompleteness theorem states that any consistent formal system containing arithmetic is incomplete
- There exist true statements within the system that cannot be proved using the system's axioms and rules of inference
- Demonstrates the limitations of formal systems and the impossibility of proving all mathematical truths within a single system
- Gödel's second incompleteness theorem shows that a consistent formal system containing arithmetic cannot prove its own consistency
- Any attempt to prove the consistency of a system within itself will lead to contradictions
- Highlights the inherent limitations of formal systems and the challenges in establishing the foundations of mathematics
- Gödel's incompleteness theorems had a profound impact on Hilbert's program and the foundations of mathematics
- Demonstrated the impossibility of achieving Hilbert's goal of proving the consistency and completeness of mathematics using finite methods
- Revealed the inherent limitations of formal systems and the existence of undecidable propositions

Gentzen's Contributions

Gentzen's Consistency Proof and Sequent Calculus

- Gerhard Gentzen developed the sequent calculus, a formal proof system that extends the natural deduction system
- Sequent calculus allows for the manipulation of sequences of formulas (sequents) using inference rules
- Provides a more symmetric and elegant presentation of logical reasoning compared to earlier proof systems
- Gentzen proved the consistency of arithmetic using transfinite induction up to the ordinal ϵ_0
- Demonstrated that the consistency of arithmetic can be established using methods that go beyond finitary reasoning
- Introduced the concept of ordinal numbers and their role in proof theory
- Gentzen's consistency proof and sequent calculus had significant implications for proof theory and the foundations of mathematics
- Opened up new avenues for investigating the consistency and properties of mathematical systems

- Highlighted the importance of infinitary methods and ordinal analysis in proof theory
- Gentzen's work laid the foundation for the development of ordinal analysis and the study of strong proof-theoretic reductions
- Ordinal analysis assigns ordinals to theories to measure their proof-theoretic strength
- Strong proof-theoretic reductions establish the relative consistency and strength of mathematical theories

1.2 Fundamental concepts and goals of proof theory

Proof theory explores the structure and properties of mathematical proofs. It aims to understand how logical systems work, focusing on consistency, completeness, and decidability. These concepts help ensure the reliability and effectiveness of logical reasoning.

Key proof transformations include normalization and reduction, which simplify proofs and compare logical systems. Metamathematical analysis, like ordinal analysis, measures the strength of theories. These tools help us grasp the relationships between different mathematical frameworks.

Logical Properties

Consistency and Completeness

- Consistency refers to the property of a logical system where no contradiction can be derived from the axioms
- In a consistent system, it is impossible to prove both a statement and its negation
- Ensures the reliability and trustworthiness of the logical system
- Completeness is the property of a logical system where every logically valid formula is provable within the system
- If a formula is true in every model of the system, then it must be provable using the axioms and inference rules
- Guarantees that the system captures all the intended logical consequences

Decidability and Cut Elimination

- Decidability is the property of a logical system where there exists an effective procedure to determine whether a given formula is provable in the system
- A decidable system has an algorithm that can always determine the provability of any formula in a finite number of steps
- Ensures that proofs can be mechanically checked and verified
- Cut elimination is a fundamental property of sequent calculi, where any proof can be transformed into a cut-free proof
- The cut rule allows the use of lemmas in proofs, but cut elimination shows that lemmas are not necessary for proving theorems
- Cut-free proofs are more direct and provide a clearer insight into the structure of the proved formula
- Cut elimination is closely related to the consistency of the logical system (*cut*-elimination theorem)

Proof Transformations

Normalization and Reduction

- Normalization is the process of transforming a proof into a normal form, which is a simplified and standardized representation of the proof
- Normal forms eliminate redundancies and unnecessary detours in proofs
- Normalization can be applied to various proof systems, such as natural deduction and lambda calculus
- Key examples of normalization include β -reduction in lambda calculus and the elimination of detours in natural deduction proofs
- Proof-theoretic reduction is a technique for comparing the strength of different logical systems
- It involves translating proofs from one system to another while preserving essential properties
- Reductions can be used to establish relative consistency results and to study the relationships between different theories
- Examples of proof-theoretic reductions include the interpretation of arithmetic in set theory and the embedding of intuitionistic logic in classical logic

Metamathematical Analysis

Ordinal Analysis and Proof-Theoretic Strength

- Ordinal analysis is a method for measuring the strength of mathematical theories by assigning ordinal numbers to the theories
- Ordinals represent the level of transfinite induction required to prove the consistency of the theory
- Theories with higher proof-theoretic ordinals are considered stronger than those with lower ordinals
- Examples of ordinal analysis include Gentzen's consistency proof of arithmetic using transfinite induction up to ϵ_0 and the analysis of subsystems of second-order arithmetic
- Proof-theoretic strength refers to the relative power and expressiveness of different logical systems and theories
- It is determined by the principles and axioms that can be proven within the system
- Stronger theories can prove the consistency of weaker theories, but not vice versa
- Comparing proof-theoretic strengths helps in understanding the relationships and hierarchies between different branches of mathematics
- Examples of theories with increasing proof-theoretic strength: Peano arithmetic, subsystems of second-order arithmetic (RCA_0 , WKL_0 , ACA_0), and Zermelo-Fraenkel set theory (ZF)

1.3 Relationship between proof theory and other branches of logic

Proof theory is deeply connected to other branches of logic, like model theory and set theory. These fields work together to provide a solid foundation for mathematical reasoning, exploring how we construct and interpret formal systems.

Proof theory also has strong ties to computability and recursion theory. These connections highlight the relationship between proofs, algorithms, and the limits of computation, influencing areas like formal verification and programming language design.

Relationship to Model Theory and Set Theory

Foundations of Mathematical Logic

- Model theory studies mathematical structures and their properties using formal languages
- Focuses on the relationship between a formal language and its interpretations (models)
- Examines concepts such as satisfiability, validity, and completeness of logical systems
- Set theory serves as a foundation for modern mathematics by providing a framework for defining and manipulating mathematical objects
- Axiomatic set theory (Zermelo-Fraenkel set theory) is commonly used as a basis for mathematics
- Proof theory can be used to study the consistency and independence of axioms in set theory

Proof-Theoretic Semantics

- Proof-theoretic semantics is an approach to the meaning of logical connectives and quantifiers based on the role they play in proofs
- Differs from the traditional truth-conditional semantics used in model theory
- Focuses on the inferential behavior of logical constants rather than their truth conditions
- Proof-theoretic semantics aims to provide a more constructive and computational account of meaning
- Emphasizes the importance of proofs and derivations in determining the meaning of logical statements
- Has connections to intuitionistic and constructive logic, where truth is closely tied to provability

Relationship to Computability and Recursion Theory

Foundations of Computation

- Recursion theory, also known as computability theory, studies the concept of computable functions and the limits of computation
- Investigates which problems can be solved using algorithms and which cannot (decidability)
- Develops a hierarchy of unsolvable problems based on their complexity (arithmetic hierarchy)
- Computability theory has close ties to proof theory, as both fields deal with formal systems and their properties
- Proof theory can be used to study the strength and limitations of formal systems used in computability theory
- Results from computability theory, such as the undecidability of the halting problem, have implications for proof theory

Connections between Proof Theory and Computation

- Proof theory has applications in computer science, particularly in the design and analysis of type systems and formal verification
- Type systems use formal rules to ensure the correctness of programs and prevent certain types of errors
- Formal verification techniques, such as Hoare logic and separation logic, use proof-theoretic methods to reason about program correctness
- The Curry-Howard correspondence establishes a deep connection between proofs and programs
- Proofs in certain logical systems can be seen as programs in a corresponding programming language
- This correspondence allows for the transfer of ideas and techniques between proof theory and computer science (functional programming)

1.4 Basic notions of formal systems and proofs

Formal systems are the backbone of mathematical logic, providing a structured framework for reasoning. They consist of a language, axioms, and inference rules that allow for precise and unambiguous derivation of theorems from initial assumptions.

Proofs and derivations are the heart of formal systems, demonstrating the logical validity of statements. Through step-by-step reasoning, they show how conclusions follow from premises, forming the basis for establishing mathematical truths within the system.

Formal Systems

Components of Formal Systems

- Formal language consists of an alphabet of symbols and a set of formation rules that define well-formed formulas (wffs)
- Axioms are the starting assumptions or premises of the formal system
- Serve as the foundation for deriving other statements
- Cannot be proven within the system itself
- Inference rules specify how new well-formed formulas can be derived from existing ones
- Allow for the generation of theorems from the axioms and previously derived statements
- Examples include modus ponens ($P \rightarrow Q, P \vdash Q$) and hypothetical syllogism ($P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R$)

Properties of Formal Systems

- Formal systems aim to be precise, unambiguous, and mechanically checkable
- Removes potential for misinterpretation or ambiguity in reasoning
- The choice of axioms and inference rules determines the specific formal system
- Different systems can be constructed for various areas of mathematics and logic (propositional logic, first-order logic, Peano arithmetic)

- Formal systems provide a rigorous framework for mathematical and logical reasoning
- Enable the systematic derivation of theorems from a set of initial assumptions

Proofs and Derivations

Formal Proofs

- A formal proof is a sequence of well-formed formulas, each of which is either an axiom or derived from previous formulas using inference rules
- Demonstrates the logical validity of a statement within the formal system
- The final formula in the proof sequence is the theorem being proven
- Represents a logically necessary consequence of the axioms and inference rules
- Formal proofs are constructed using a step-by-step process, justifying each step with the appropriate axiom or inference rule

Derivations

- A derivation is a sequence of well-formed formulas that leads from the premises to the conclusion
- Shows how the conclusion can be obtained from the premises using the inference rules of the formal system
- Derivations are often represented using a proof tree or a sequence of lines, with each line justified by the premises or previous lines
- Helps to visualize the logical structure of the argument
- The existence of a derivation for a statement demonstrates its provability within the formal system

Theorems

- A theorem is a well-formed formula that can be derived from the axioms using the inference rules of the formal system
- Represents a logically necessary truth within the system
- Theorems are proven through formal proofs or derivations
- The proof provides a convincing argument for the truth of the theorem based on the axioms and rules of the system
- Examples of theorems include the law of excluded middle ($P \vee \neg P$) in propositional logic and the fundamental theorem of arithmetic in number theory

Metatheory

Metatheorems

- Metatheorems are statements about the properties of a formal system itself
- Proven using meta-reasoning, which is reasoning about the formal system from an external perspective

- Metatheorems make assertions about the consistency, completeness, or other characteristics of the formal system
- Help to establish the reliability and limitations of the system
- Examples of metatheorems include Gödel's completeness theorem for first-order logic and the undecidability of the halting problem in computability theory

Soundness and Completeness

- Soundness is a metatheoretical property that ensures every theorem provable in the formal system is actually true
- If a statement can be derived using the inference rules, it is guaranteed to be a valid consequence of the axioms
- Soundness prevents the derivation of false statements within the system
- Completeness is a metatheoretical property that ensures every true statement expressible in the formal language is provable using the axioms and inference rules
- If a statement is a logical consequence of the axioms, there exists a formal proof for it within the system
- Completeness guarantees that all valid statements can be derived

Significance of Metatheory

- Metatheory allows for the study of the properties and limitations of formal systems
- Helps to establish the reliability, consistency, and expressive power of the system
- Metatheoretical results provide insights into the foundations of mathematics and logic
- Shed light on the nature of mathematical truth and the capabilities of formal reasoning
- Metatheory plays a crucial role in areas such as proof theory, model theory, and computability theory
- Enables the analysis of the strengths and weaknesses of different formal systems and their applications

Unit 2 – Propositional Logic: Syntax and Semantics

2.1 Syntax and formation rules of propositional logic

Propositional logic forms the foundation of formal reasoning. It uses symbols and connectives to represent and manipulate statements that can be true or false. Understanding its syntax is crucial for building more complex logical structures.

The syntax of propositional logic defines how formulas are constructed. It introduces propositional variables, logical connectives, and rules for combining them into well-formed formulas. This framework allows us to express and analyze complex logical relationships.

Propositional Formulas

Propositional Variables and Atomic Formulas

- Propositional variables are symbols that represent propositions or statements that can be either true or false (p, q, r)
- Atomic formulas consist of a single propositional variable and cannot be broken down into simpler formulas
- Serve as the basic building blocks for constructing more complex formulas in propositional logic
- Each propositional variable is considered an atomic formula (p, q, r are atomic formulas)

Compound Formulas and Well-Formed Formulas

- Compound formulas are formed by combining one or more atomic formulas using logical connectives
- Can be broken down into simpler formulas, unlike atomic formulas
- Well-formed formulas (WFFs) are formulas that are constructed according to the syntax rules of propositional logic
- WFFs are defined recursively:
- Every atomic formula is a WFF
- If ϕ and ψ are WFFs, then $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$ are also WFFs
- Examples of well-formed formulas: $(p \wedge q)$, $(\neg p \vee (q \rightarrow r))$

Logical Connectives and Notation

Logical Connectives

- Logical connectives are symbols used to combine propositional formulas to create compound formulas
- The main logical connectives in propositional logic are:
- Negation ($\neg$): represents the negation or opposite of a proposition

- Conjunction ($\wedge$): represents the logical AND, connecting two propositions that must both be true for the compound formula to be true
- Disjunction ($\vee$): represents the logical OR, connecting two propositions where at least one must be true for the compound formula to be true
- Implication ($\rightarrow$): represents a conditional statement, where the truth of the second proposition depends on the truth of the first proposition
- Biconditional ($\leftrightarrow$): represents a bi-directional implication, where both propositions must have the same truth value for the compound formula to be true
- Examples of compound formulas using logical connectives: $(\neg p)$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$, $(p \leftrightarrow q)$

Parentheses, Infix Notation, and Precedence Rules

- Parentheses are used to group propositional formulas and specify the order of operations
- Infix notation places the logical connectives between the propositional variables or formulas they connect
- Precedence rules determine the order in which logical connectives are evaluated when parentheses are not used:
- Negation ($\neg$) has the highest precedence
- Conjunction ($\wedge$) and disjunction ($\vee$) have the next highest precedence and are evaluated from left to right
- Implication ($\rightarrow$) and biconditional ($\leftrightarrow$) have the lowest precedence and are evaluated from right to left
- Examples demonstrating precedence rules: $p \wedge q \vee r$ is equivalent to $(p \wedge q) \vee r$, while $p \rightarrow q \vee r$ is equivalent to $p \rightarrow (q \vee r)$

2.2 Semantics and truth tables

Propositional logic uses truth values to represent the truth or falsity of statements. Truth tables are powerful tools that display all possible combinations of truth values for a set of propositions, helping us evaluate complex formulas and determine logical properties.

Logical properties like tautologies, contradictions, and satisfiability can be identified using truth tables. These tables also reveal logical relationships between formulas, such as logical equivalence and logical consequence, which are fundamental to understanding propositional logic.

Truth Values and Tables

Assigning Truth Values

- Truth values represent the truth or falsity of a proposition
- In classical logic, there are two possible truth values: true (T) and false (F)
- A valuation assigns a truth value to each propositional variable in a formula
- Valuations determine the overall truth value of a complex formula based on the truth values of its constituent propositions

Constructing Truth Tables

- Truth tables display all possible combinations of truth values for a given set of propositions
- Each row in a truth table represents a unique valuation of the propositional variables
- The number of rows in a truth table is determined by 2^n , where n is the number of distinct propositional variables
- Truth tables help determine the logical properties and relationships between propositions (tautology, contradiction, logical equivalence)

Evaluating Complex Formulas

- Complex formulas are built using logical connectives (conjunction $\wedge$, disjunction $\vee$, negation $\neg$, implication $\rightarrow$, biconditional $\leftrightarrow$)
- The truth value of a complex formula is determined by the truth values of its constituent propositions and the rules of the logical connectives
- Truth tables can be used to evaluate the truth value of a complex formula for each possible valuation of its propositional variables
- Example: The truth table for the formula $p \wedge q$ shows that it is true only when both p and q are true, and false in all other cases

Logical Properties

Tautologies and Contradictions

- A tautology is a formula that is always true, regardless of the truth values assigned to its propositional variables
- Example: The formula $p \vee \neg p$ is a tautology, as it is true for all possible valuations of p
- A contradiction is a formula that is always false, regardless of the truth values assigned to its propositional variables
- Example: The formula $p \wedge \neg p$ is a contradiction, as it is false for all possible valuations of p

Satisfiability

- A formula is satisfiable if there exists at least one valuation that makes the formula true
- A formula is unsatisfiable if it is false for all possible valuations (i.e., a contradiction)
- Satisfiability can be determined by examining the truth table of a formula
- If there is at least one row in the truth table where the formula evaluates to true, then the formula is satisfiable
- If all rows in the truth table evaluate to false, then the formula is unsatisfiable

Logical Relationships

Logical Equivalence

- Two formulas are logically equivalent if they have the same truth value for all possible valuations of their propositional variables
- Logical equivalence is denoted by the symbol $\equiv$
- Example: The formulas $p \rightarrow q$ and $\neg p \vee q$ are logically equivalent, as they have the same truth table
- Logical equivalence is reflexive, symmetric, and transitive, making it an equivalence relation
- Logically equivalent formulas can be substituted for each other in any context without changing the truth value of the overall formula

Logical Consequence

- A formula ψ is a logical consequence of a set of formulas Γ if, whenever all the formulas in Γ are true, ψ is also true
- Logical consequence is denoted by the symbol $\models$, as in $\Gamma \models \psi$
- Example: The formula q is a logical consequence of the set $\{p, p \rightarrow q\}$, as whenever p and $p \rightarrow q$ are both true, q must also be true
- Logical consequence is closely related to the concept of validity in arguments, where the premises logically entail the conclusion
- Truth tables can be used to determine logical consequence by checking if, in all rows where the formulas in Γ are true, ψ is also true

2.3 Proof systems: axiomatic, natural deduction, and tableau methods

Proof systems are the backbone of formal logic, providing methods to verify the validity of arguments. Axiomatic, natural deduction, and tableau methods each offer unique approaches to constructing proofs, with their own strengths and weaknesses.

These systems build on the syntax and semantics of propositional logic, allowing us to derive logical conclusions. By understanding these proof methods, we gain powerful tools for reasoning and analyzing complex logical statements.

Axiomatic Systems and Inference Rules

Fundamental Components of Axiomatic Systems

- Axiomatic systems provide a formal framework for deriving logical conclusions from a set of axioms and inference rules
- Axioms serve as the foundational statements or premises that are assumed to be true within the system
- Inference rules define the valid ways in which new statements can be derived from existing axioms or previously derived statements

- Logical symbols and connectives ($\wedge$, $\vee$, $\rightarrow$, $\neg$) are used to express and manipulate logical statements within the axiomatic system

Derivations and Proofs

- Derivation is the process of deriving new statements or conclusions from the axioms and inference rules of an axiomatic system
- Each step in a derivation must be justified by applying an inference rule to the previous statements or axioms
- The sequence of steps in a derivation forms a proof, demonstrating the logical validity of the derived conclusion
- Proofs in axiomatic systems are typically presented in a linear format, with each step numbered and justified by the applied inference rule (modus ponens, modus tollens)

Advantages and Limitations

- Axiomatic systems provide a rigorous and formal approach to logical reasoning, ensuring the validity of derived conclusions
- The use of axioms and inference rules allows for the systematic construction of proofs and the verification of logical arguments
- However, axiomatic systems can be complex and may require a significant amount of effort to construct and understand
- The choice of axioms and inference rules can impact the expressive power and limitations of the axiomatic system (propositional logic, first-order logic)

Natural Deduction

Introduction and Elimination Rules

- Natural deduction is a proof system that closely resembles the way humans reason and construct arguments
- Introduction rules specify how to introduce logical connectives and construct new statements (conjunction introduction, disjunction introduction, implication introduction)
- Elimination rules define how to eliminate or simplify logical connectives and derive new statements from existing ones (conjunction elimination, disjunction elimination, implication elimination)
- The introduction and elimination rules for each logical connective form the basis of natural deduction proofs

Proof Construction and Proof Trees

- In natural deduction, proofs are constructed by applying introduction and elimination rules to derive the desired conclusion from the given premises
- Proofs can be represented using a proof tree, where each node represents a statement, and the edges represent the application of inference rules
- The root of the proof tree is the conclusion, and the leaves are the premises or assumptions used in the proof

- Proof trees provide a visual representation of the logical structure and dependencies within a natural deduction proof (subproofs, assumptions, discharging assumptions)

Advantages and Applications

- Natural deduction proofs closely resemble human reasoning patterns, making them more intuitive and easier to understand compared to other proof systems
- The introduction and elimination rules provide a systematic way to construct proofs and analyze logical arguments
- Natural deduction is widely used in various areas of logic, mathematics, and computer science (program verification, type theory, theorem proving)
- The proof tree representation allows for the visualization and analysis of the logical structure and dependencies within a proof

Tableau Method

Tableau Construction and Closure

- The tableau method is a proof procedure that determines the satisfiability or validity of a logical formula
- To construct a tableau, the negation of the formula to be proved is assumed, and the formula is broken down into its constituent parts using tableau expansion rules
- The expansion rules are applied to each branch of the tableau until a contradiction is reached (closed branch) or no further expansions are possible (open branch)
- A closed tableau, where all branches are closed, indicates that the original formula is valid or unsatisfiable (proof by contradiction)

Proof Procedure and Strategies

- The tableau method involves systematically applying the expansion rules to the formulas on each branch of the tableau
- Common expansion rules include conjunction expansion (α -rule), disjunction expansion (β -rule), and negation expansion (γ -rule)
- Strategies for efficient tableau construction include selecting the most promising formulas for expansion and applying the expansion rules in a specific order (depth-first, breadth-first)
- The proof procedure continues until all branches are closed (indicating validity) or an open branch is found (indicating satisfiability)

Advantages and Limitations

- The tableau method provides a systematic and mechanical procedure for determining the satisfiability or validity of logical formulas
- Tableaux can be used to prove the consistency or inconsistency of a set of formulas, making them useful for logical analysis and reasoning
- The proof tree structure of tableaux allows for the visualization and understanding of the logical dependencies and case distinctions within a proof

- However, the tableau method can be computationally expensive for large and complex formulas, as the number of branches can grow exponentially (proof complexity)

2.4 Soundness and completeness of propositional logic

Soundness and completeness are crucial properties of propositional logic. They ensure that our logical system is reliable and powerful, guaranteeing that provable statements are valid and all valid statements can be proven.

These properties form the foundation for trust in logical reasoning. By understanding soundness and completeness, we can confidently use propositional logic to analyze arguments and draw conclusions in various fields of study.

Logical Properties

Fundamental Characteristics of Logical Systems

- Soundness ensures that all provable statements within a logical system are valid
- Guarantees the reliability and trustworthiness of the system's inferences
- If a statement can be derived using the system's rules, it must be true in all models (interpretations) of the system
- Completeness ensures that all valid statements within a logical system are provable
- Guarantees that the system is capable of deriving all true statements
- If a statement is true in all models (interpretations) of the system, it must be possible to prove it using the system's rules
- Validity refers to the property of a statement being true in all models (interpretations) of a logical system
- A valid statement is a tautology, meaning it is always true regardless of the truth values assigned to its components
- Validity is a semantic concept, concerned with the meaning and interpretation of statements

Consistency and Its Importance

- Consistency is the property of a logical system being free from contradictions
- In a consistent system, it is impossible to derive both a statement and its negation
- Consistency is crucial for maintaining the integrity and reliability of a logical system
- If a system is inconsistent, it becomes trivial, as any statement can be proved, rendering the system useless
- Maintaining consistency is a fundamental goal in the design and analysis of logical systems
- Inconsistencies can arise due to flaws in the system's axioms or inference rules
- Techniques such as proof by contradiction are used to demonstrate the consistency of a system

Formal Systems

Components and Structure

- A formal system consists of a well-defined language, axioms, and inference rules
- The language specifies the symbols, formulas, and well-formed expressions used in the system
- Axioms are the fundamental assumptions or starting points of the system, accepted as true without proof
- Inference rules define the valid ways to derive new statements from existing ones
- The components of a formal system work together to enable the derivation of theorems and the study of logical properties
- Theorems are statements that can be derived from the axioms using the inference rules
- The structure of a formal system provides a rigorous framework for logical reasoning and analysis

Models and Metatheorems

- A model is an interpretation of a formal system that assigns meaning to its symbols and statements
- Models provide a way to evaluate the truth or falsity of statements within the system
- Different models can yield different truth values for the same statement
- Consistency of a formal system requires the existence of at least one model in which all axioms are true
- Metatheorems are statements about the properties and characteristics of a formal system itself
- Metatheorems are proved using methods outside the formal system, often in a higher-level language or framework
- Examples of metatheorems include the soundness and completeness of a logical system
- Metatheorems provide insights into the capabilities, limitations, and relationships between different formal systems

Unit 3 – First-Order Logic: Syntax and Semantics

3.1 Syntax and formation rules of first-order logic

First-order logic provides a powerful framework for expressing complex statements about objects and their relationships. It builds on propositional logic by introducing quantifiers and predicates, allowing for more nuanced and expressive logical formulations.

The syntax of first-order logic consists of symbols, terms, and formulas. These elements combine to create well-formed formulas that can represent intricate logical statements, laying the groundwork for formal reasoning and proof construction in mathematical and philosophical contexts.

Vocabulary

Symbols and Constants

- Predicate symbols represent relations between objects (Loves, IsGreaterThan)
- Function symbols denote functions that map objects to other objects (MotherOf, Successor)
- Constants refer to specific objects in the domain of discourse (Alice, Bob, 0, 1)
- Variables are used to represent arbitrary objects (x , y , z)

Terms and Atomic Formulas

Building Blocks of First-Order Logic

- Terms are expressions that refer to objects
- Can be constants (a , b , c), variables (x , y , z), or function applications ($f(x)$, $g(a, y)$)
- Function applications consist of a function symbol followed by a tuple of terms (MotherOf(Alice), Successor(0))
- Atomic formulas are the simplest type of well-formed formulas in first-order logic
- Formed by applying a predicate symbol to a tuple of terms (Loves(Alice, Bob), IsGreaterThan(x , y))
- Represent basic assertions about objects and their relations

Logical Connectives and Quantifiers

Constructing Complex Formulas

- Logical connectives are used to combine simpler formulas into more complex ones
- Conjunction ($\wedge$): "and" connects two formulas ($\phi \wedge \psi$)
- Disjunction ($\vee$): "or" connects two formulas ($\phi \vee \psi$)
- Implication ($\rightarrow$): "if...then" connects two formulas ($\phi \rightarrow \psi$)

- Negation ($\neg$): "not" negates a formula ($\neg\phi$)
- Quantifiers express properties about collections of objects
- Universal quantifier ($\forall$): "for all" specifies that a formula holds for all objects ($\forall x \phi(x)$)
- Existential quantifier ($\exists$): "there exists" specifies that a formula holds for at least one object ($\exists x \phi(x)$)

Well-formed Formulas

Syntactically Correct Formulas

- Well-formed formulas (wffs) are syntactically correct expressions in first-order logic
- Atomic formulas are wffs
- If ϕ and ψ are wffs, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\neg\phi)$
- If ϕ is a wff and x is a variable, then $\forall x \phi$ and $\exists x \phi$ are also wffs
- Free and bound variables
- A variable x is bound in a formula if it falls within the scope of a quantifier ($\forall x$ or $\exists x$)
- A variable is free if it is not bound by any quantifier
- In $\forall x \exists y (P(x) \wedge Q(x, y, z))$, x and y are bound, while z is free
- Scope of quantifiers
- The scope of a quantifier is the portion of the formula it applies to
- In $\forall x (P(x) \wedge \exists y Q(x, y))$, the scope of $\forall x$ is the entire formula, while the scope of $\exists y$ is $Q(x, y)$

3.2 Semantics, models, and interpretations

First-order logic helps us reason about objects and their relationships. In this section, we'll explore how models and interpretations give meaning to logical statements. We'll see how domains define what we're talking about and how interpretations assign specific meanings.

We'll also dive into semantics and satisfaction, which tell us when statements are true. We'll learn about validity and logical consequence, which help us determine when arguments are sound. These concepts are crucial for understanding how first-order logic works in practice.

Models and Interpretations

Defining the Domain and Interpretation

- Domain of discourse specifies the set of objects that are being reasoned about in a particular context
- Can be any non-empty set (integers, real numbers, people, countries)
- Provides a universe for the quantifiers $\forall$ (for all) and $\exists$ (there exists) to range over
- Interpretation function assigns meaning to the non-logical symbols in a first-order language
- Maps constant symbols to elements of the domain
- Maps function symbols to actual functions on the domain

- Maps predicate symbols to relations on the domain

Models as Instantiations of a Theory

- Model is a structure that provides a concrete interpretation of a first-order theory
- Consists of a domain of discourse together with an interpretation function
- Essentially an instantiation or "possible world" in which the theory could be realized
- Truth valuation determined by a model assigns truth values (true or false) to each sentence in the language
- Determined recursively based on the interpretation of the non-logical symbols and the meanings of the logical connectives and quantifiers
- A sentence is true in a model if it holds under the interpretation provided by that model

Semantics and Satisfaction

Evaluating Truth in First-Order Logic

- Satisfaction is the key notion for defining truth in first-order logic
- A formula $\phi(x_1, \dots, x_n)$ is satisfied by a sequence of elements $a_1, \dots, a_n$ from the domain if ϕ becomes true when each x_i is interpreted as a_i
- A sentence (formula with no free variables) is true in a model if it is satisfied by every sequence of elements
- "Tarski's truth definition" recursively defines truth based on satisfaction
- Atomic formulas are true if the corresponding relation holds of the elements assigned to the terms
- Compound formulas built using $\wedge$, $\vee$, $\rightarrow$, $\neg$ are evaluated based on truth tables
- $\forall x \phi(x)$ is true if $\phi(a)$ is satisfied for every element a in the domain
- $\exists x \phi(x)$ is true if $\phi(a)$ is satisfied for at least one element a in the domain

Validity and Logical Consequence

- Validity is a stronger notion than truth in a particular model
- A sentence is valid if it is true in every model
- Corresponds to the idea of a tautology or logically necessary truth
- Valid sentences are true based solely on the meanings of the logical symbols, regardless of the interpretation of the non-logical symbols
- Logical consequence captures the idea of a sentence following from a set of premises
- A sentence ψ is a logical consequence of a set of sentences Φ if ψ is true in every model in which all the sentences in Φ are true
- Allows for sound reasoning and inference based on the meanings of the logical symbols and the structure of the argument

3.3 Quantifiers and their properties

Quantifiers are powerful tools in first-order logic, allowing us to express statements about all or some elements in a domain. They come in two flavors: universal ($\forall$) for "all" and existential ($\exists$) for "some."

Negating quantifiers flips their meaning, turning "all" into "some not" and "some" into "none." This property is crucial for understanding logical relationships and constructing valid arguments in first-order logic.

Quantifiers

Universal and Existential Quantifiers

- Universal quantifier $\forall$ expresses that a formula holds for all values of a variable
- Formally, $\forall x P(x)$ means that $P(x)$ is true for every possible value of x in the domain
- Can be read as "for all", "for every", or "for each" ($\forall x \in \mathbb{N}, x \geq 0$ reads "for all natural numbers x , x is greater than or equal to 0")
- Existential quantifier $\exists$ expresses that a formula holds for at least one value of a variable
- Formally, $\exists x P(x)$ means that there exists at least one value of x in the domain for which $P(x)$ is true
- Can be read as "there exists", "for some", or "there is at least one" ($\exists x \in \mathbb{R}, x^2 = 4$ reads "there exists a real number x such that x squared equals 4")

Quantifier Negation

- Negation of a universally quantified formula $\forall x P(x)$ is equivalent to an existentially quantified negation of the formula $\exists x \neg P(x)$
- $\neg(\forall x P(x)) \equiv \exists x \neg P(x)$ (reads "not for all x , $P(x)$ " is equivalent to "there exists an x such that not $P(x)$ ")
- Example: $\neg(\forall x \in \mathbb{N}, x > 0) \equiv \exists x \in \mathbb{N}, x \leq 0$ (reads "not all natural numbers are greater than 0" is equivalent to "there exists a natural number less than or equal to 0")
- Negation of an existentially quantified formula $\exists x P(x)$ is equivalent to a universally quantified negation of the formula $\forall x \neg P(x)$
- $\neg(\exists x P(x)) \equiv \forall x \neg P(x)$ (reads "there does not exist an x such that $P(x)$ " is equivalent to "for all x , not $P(x)$ ")
- Example: $\neg(\exists x \in \mathbb{Z}, x^2 = 2) \equiv \forall x \in \mathbb{Z}, x^2 \neq 2$ (reads "there does not exist an integer x such that x squared equals 2" is equivalent to "for all integers x , x squared does not equal 2")

Quantifier Transformations

Quantifier Distribution and Elimination

- Quantifier distribution refers to the rules for distributing quantifiers over logical connectives
- $\forall x (P(x) \wedge Q(x)) \equiv (\forall x P(x)) \wedge (\forall x Q(x))$ (universal quantifier distributes over conjunction)
- $\exists x (P(x) \vee Q(x)) \equiv (\exists x P(x)) \vee (\exists x Q(x))$ (existential quantifier distributes over disjunction)
- However, $\forall x (P(x) \vee Q(x)) \not\equiv (\forall x P(x)) \vee (\forall x Q(x))$ and $\exists x (P(x) \wedge Q(x)) \not\equiv (\exists x P(x)) \wedge (\exists x Q(x))$

- Quantifier elimination is the process of removing quantifiers from a formula while preserving its satisfiability
- One approach is to replace the quantified variable with a constant that satisfies the formula, if such a constant exists
- Example: $\exists x(x > 0 \wedge \forall y(y > x \rightarrow y > 1))$ can be simplified to $\exists x(x > 0 \wedge x < 1)$ by eliminating the universal quantifier

Prenex Normal Form and Skolemization

- Prenex normal form is a formula where all quantifiers appear at the beginning, followed by a quantifier-free matrix
- Any formula can be converted to prenex normal form by applying quantifier distribution and renaming variables to avoid name clashes
- Example: $\forall x \exists y (P(x) \vee Q(y))$ is in prenex normal form, while $(\forall x P(x)) \vee (\exists y Q(y))$ is not
- Skolemization is a process of eliminating existential quantifiers from a formula in prenex normal form
- Each existentially quantified variable is replaced by a Skolem function that depends on the universally quantified variables preceding it
- The resulting formula is satisfiable if and only if the original formula is satisfiable
- Example: $\forall x \exists y P(x, y)$ can be Skolemized to $\forall x P(x, f(x))$, where f is a Skolem function

3.4 Proof systems for first-order logic

First-order logic proof systems build on propositional logic, adding quantifiers and predicates. Natural deduction and sequent calculus are two key approaches, using inference rules to construct proofs. These systems handle quantifiers through special introduction and elimination rules.

Axiom systems offer another proof method, using schemas as templates for generating axioms. Herbrand's theorem bridges first-order and propositional logic, while automated theorem proving techniques like resolution enable computer-assisted proof discovery in first-order logic.

Natural Deduction and Sequent Calculus

Inference Rules and Proof Systems

- Natural deduction is a proof system that uses a set of inference rules to derive conclusions from premises
- Consists of introduction and elimination rules for each logical connective
- Allows for the construction of proofs in a way that closely resembles human reasoning (modus ponens, modus tollens)
- Sequent calculus is another proof system that operates on sequents, which are expressions of the form $\Gamma \vdash \Delta$
- Γ represents a set of formulas called the antecedent, and Δ represents a set of formulas called the succedent
- Sequent calculus rules manipulate these sequents to prove the validity of logical arguments

- Inference rules in both systems specify how to derive new formulas from existing ones
- Examples include $\wedge$ -introduction, $\vee$ -elimination, $\rightarrow$ -introduction, and $\neg$ -introduction

Quantifier Rules in Natural Deduction and Sequent Calculus

- Quantifier rules in natural deduction and sequent calculus handle the introduction and elimination of universal ($\forall$) and existential ($\exists$) quantifiers
- Universal quantifier introduction ($\forall$ -introduction) allows for the generalization of a formula by introducing a universal quantifier
- If a formula ϕ has been derived without any assumptions about the variable x , then $\forall x\phi$ can be derived
- Existential quantifier elimination ($\exists$ -elimination) allows for the instantiation of an existentially quantified formula with a specific term
- If $\exists x\phi$ has been derived, and a formula ψ can be derived from $\phi[t/x]$ for some term t , then ψ can be derived
- Similar quantifier rules exist in sequent calculus, adapted to work with sequents

Axiom Systems

Axiom Schemas and Substitution

- Axiom systems are another approach to proof theory, where a set of axioms is used as the starting point for deriving theorems
- Axiom schemas are templates for axioms, containing metavariables that can be instantiated with specific formulas
- Example: the axiom schema $\forall x(\phi \rightarrow \psi) \rightarrow (\forall x\phi \rightarrow \forall x\psi)$ can be instantiated with specific formulas for ϕ and ψ
- Substitution is the process of replacing metavariables in an axiom schema with specific formulas to obtain concrete axioms
- Allows for the generation of an infinite number of axioms from a finite set of axiom schemas

Herbrand's Theorem

- Herbrand's theorem is a fundamental result in proof theory that relates the validity of a first-order formula to the satisfiability of a potentially infinite set of propositional formulas
- States that a first-order formula is valid if and only if there exists a finite set of instances of the formula, obtained by substituting terms for variables, that is propositionally unsatisfiable
- Provides a bridge between first-order logic and propositional logic
- Allows for the reduction of first-order validity to propositional satisfiability
- Has important implications for automated theorem proving, as it suggests a way to prove first-order formulas by considering their propositional instances

Automated Theorem Proving

Resolution and Proof Search

- Automated theorem proving refers to the use of computer programs to automatically find proofs of mathematical theorems
- Resolution is a powerful inference rule used in automated theorem proving for first-order logic
- Operates on clauses, which are disjunctions of literals (atomic formulas or their negations)
- Allows for the derivation of new clauses from existing ones by resolving complementary literals
- Example: from the clauses $\neg P(x) \vee Q(x)$ and $P(a)$, resolution derives the clause $Q(a)$
- Resolution can be used as a refutation procedure, where the negation of the desired theorem is added to the set of clauses, and resolution is applied until a contradiction (empty clause) is derived
- Efficient proof search strategies, such as unit resolution and ordered resolution, are employed to guide the application of resolution inferences
- These strategies help reduce the search space and improve the performance of automated theorem provers

3.5 Soundness and completeness of first-order logic

First-order logic's soundness and completeness theorems are crucial pillars. Soundness ensures that provable formulas are valid in all models, while completeness guarantees that valid formulas are provable. These theorems bridge the gap between syntax and semantics.

The model existence theorem shows that consistent formula sets have models. This links consistency and satisfiability, key concepts in model theory. These ideas help us understand how formal systems relate to their interpretations in first-order logic.

Soundness and Completeness Theorems

Validity and Provability

- Soundness theorem proves that if a formula is provable in a formal system, then it is valid in all models of the system
- Ensures that the formal system does not allow proving false statements
- Provides confidence in the correctness of the formal system
- Completeness theorem demonstrates that if a formula is valid in all models of a formal system, then it is provable within that system
- Guarantees that all valid formulas can be derived using the formal system's inference rules
- Establishes that the formal system is powerful enough to prove all valid statements
- Gödel's completeness theorem, a specific instance of the completeness theorem, states that a formula is provable in first-order logic if and only if it is valid in all models
- Proves the equivalence between semantic validity and syntactic provability in first-order logic
- Fundamental result in mathematical logic, bridging the gap between syntax and semantics

Model Existence and Satisfiability

- Model existence theorem, a consequence of the completeness theorem, states that if a set of formulas is consistent, then there exists a model in which all the formulas are true
- Demonstrates the relationship between consistency and satisfiability
- Ensures that consistent sets of formulas have a model that satisfies them
- Satisfiability, a key concept in model theory, refers to the existence of a model that makes a formula or set of formulas true
- Closely related to the concept of consistency, as consistent sets of formulas are satisfiable
- Plays a crucial role in understanding the relationship between syntax and semantics in first-order logic

Constructing Models and Compactness

Henkin Construction and Consistency

- Henkin construction, a method for building models of consistent sets of formulas, involves extending the set with witness constants and ensuring its consistency
- Starts with a consistent set of formulas and expands it by adding new constants and formulas
- Ensures that the extended set remains consistent while providing witnesses for existential statements
- Allows for the construction of a model that satisfies the original set of formulas
- Consistency, a property of sets of formulas, means that no contradiction can be derived from the set using the inference rules of the formal system
- Essential for the existence of models, as inconsistent sets of formulas cannot have a model
- Maintained throughout the Henkin construction process to guarantee the existence of a model

Compactness and Löwenheim-Skolem Theorems

- Compactness theorem states that if every finite subset of a set of formulas has a model, then the entire set has a model
- Allows for reasoning about infinite sets of formulas by considering their finite subsets
- Has important applications in model theory and proof theory, such as proving the existence of non-standard models of arithmetic
- Löwenheim-Skolem theorem, a consequence of the compactness theorem, states that if a countable set of formulas has an infinite model, then it has models of all infinite cardinalities
- Demonstrates the existence of models of different sizes for a given set of formulas
- Highlights the limitations of first-order logic in characterizing the cardinality of models
- Leads to the concept of non-standard models, which satisfy the same formulas but have different properties than the intended model

Unit 4 – Natural Deduction and Sequent Calculus

4.1 Natural deduction: rules and proof construction

Natural deduction is a way of reasoning in logic using rules for introducing and eliminating logical connectives. It's like building with Lego blocks, where each block is a logical statement and the rules tell you how to connect them.

In this section, we'll learn about the specific rules for different logical operators like "and," "or," and "if-then." We'll also see how to use these rules to construct proofs, which are step-by-step arguments that show a conclusion follows from given premises.

Introduction and Elimination Rules

Conjunction Rules

- Conjunction introduction rule ($\wedge I$) allows inferring a conjunction ($A \wedge B$) if both conjuncts (A and B) have been derived separately
- Conjunction elimination rules ($\wedge E$) allow inferring either conjunct (A or B) from a conjunction ($A \wedge B$)
- Left conjunction elimination ($\wedge E1$) infers the left conjunct (A) from a conjunction ($A \wedge B$)
- Right conjunction elimination ($\wedge E2$) infers the right conjunct (B) from a conjunction ($A \wedge B$)

Disjunction Rules

- Disjunction introduction rules ($\vee I$) allow inferring a disjunction ($A \vee B$) if either disjunct (A or B) has been derived
- Left disjunction introduction ($\vee I1$) infers a disjunction ($A \vee B$) from the left disjunct (A)
- Right disjunction introduction ($\vee I2$) infers a disjunction ($A \vee B$) from the right disjunct (B)
- Disjunction elimination rule ($\vee E$) allows inferring a formula (C) from a disjunction ($A \vee B$) if C can be derived from each disjunct (A and B) separately

Implication and Negation Rules

- Implication introduction rule ($\rightarrow I$) allows inferring an implication ($A \rightarrow B$) if the consequent (B) can be derived assuming the antecedent (A)
- Implication elimination rule ($\rightarrow E$), also known as modus ponens, allows inferring the consequent (B) from an implication ($A \rightarrow B$) and its antecedent (A)
- Negation introduction rule ($\neg I$), also known as reductio ad absurdum, allows inferring the negation of an assumption ($\neg A$) if a contradiction can be derived from that assumption
- Negation elimination rule ($\neg E$), also known as ex falso quodlibet, allows inferring any formula (B) from a contradiction ($A \wedge \neg A$)

Quantifiers and Hypothetical Reasoning

Quantifier Rules

- Universal quantifier introduction rule ($\forall I$) allows inferring a universally quantified formula ($\forall x A(x)$) if $A(c)$ has been derived for an arbitrary constant c
- Universal quantifier elimination rule ($\forall E$) allows inferring an instance ($A(t)$) of a universally quantified formula ($\forall x A(x)$) for any term t
- Existential quantifier introduction rule ($\exists I$) allows inferring an existentially quantified formula ($\exists x A(x)$) from an instance ($A(t)$) for some term t
- Existential quantifier elimination rule ($\exists E$) allows inferring a formula (C) from an existentially quantified formula ($\exists x A(x)$) if C can be derived assuming $A(c)$ for a new constant c

Hypothetical Reasoning and Assumptions

- Hypothetical reasoning involves making assumptions and deriving conclusions from those assumptions
- Assumptions are temporary premises used in the course of a proof
- Discharge of assumptions occurs when the assumption is no longer needed and can be removed from the proof
- Discharging an assumption often corresponds to the application of an introduction rule ($\rightarrow I$, $\neg I$, $\forall I$, $\exists E$)

Proof Representation

Proof Trees

- Proof trees are graphical representations of proofs in natural deduction
- Nodes in a proof tree represent formulas, and edges represent the application of inference rules
- The root of the tree is the conclusion of the proof, and the leaves are the assumptions or axioms
- Proof trees provide a clear and intuitive way to visualize the structure of a proof

Fitch-style Notation

- Fitch-style notation is a linear representation of proofs in natural deduction
- Proofs are written as a sequence of lines, with each line containing a formula and the justification for that formula
- Assumptions are indicated by vertical bars ($|$) that scope over the lines that depend on the assumption
- Fitch-style notation is more compact than proof trees and is often used in practice

4.2 Sequent calculus: rules and proof construction

Sequent calculus is a powerful tool for constructing formal proofs in logic. It uses sequents, which are statements with antecedents and succedents, to represent logical relationships. This system provides a structured way to build proofs from the bottom up.

The rules of sequent calculus, including introduction and structural rules, guide the construction of proof trees. By applying these rules strategically, we can break down complex formulas and work towards simpler axioms, ultimately proving our desired conclusion.

Sequents and Their Components

Sequent Structure

- Sequent consists of two parts separated by a turnstile symbol ($\vdash$)
- Left side of the turnstile is called the antecedent contains a list of formulas separated by commas
- Right side of the turnstile is called the succedent also contains a list of formulas separated by commas
- Sequent represents a logical statement where the conjunction of the formulas in the antecedent implies the disjunction of the formulas in the succedent

Interpreting Sequents

- Empty antecedent is interpreted as true
- Empty succedent is interpreted as false
- Sequent with an empty antecedent and a non-empty succedent ($\vdash A$) means the formula A is provable
- Sequent with a non-empty antecedent and an empty succedent ($A \vdash$) means the formula A is contradictory

Sequent Calculus Rules

Introduction Rules

- Left rules introduce logical connectives on the left side of the turnstile (antecedent)
- Examples: $\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} (\wedge L)$, $\frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B} (\wedge R)$
- Right rules introduce logical connectives on the right side of the turnstile (succedent)
- Examples: $\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, A \vee B} (\vee R1)$, $\frac{\Gamma, B \vdash \Delta}{\Gamma \vdash \Delta, A \vee B} (\vee R2)$

Cut Rule

- Cut rule allows the use of an intermediate formula in the proof
- Example: $\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda} (\text{Cut})$
- Eliminates the need for assumptions by proving the intermediate formula and then using it in the proof

Structural Rules

Weakening Rules

- Weakening rules allow the addition of formulas to the antecedent or succedent
- Example: $\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} (\text{Weakening Left})$, $\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} (\text{Weakening Right})$
- Weakening rules are used to add unused assumptions or conclusions to the sequent

Contraction Rules

- Contraction rules allow the removal of duplicate formulas from the antecedent or succedent
- Example: $\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}$ (Contraction Left), $\frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}$ (Contraction Right)
- Contraction rules are used to simplify the sequent by removing redundant formulas

Exchange Rules

- Exchange rules allow the reordering of formulas in the antecedent or succedent
- Example: $\frac{\Gamma, B, A, \Pi \vdash \Delta}{\Gamma, A, B, \Pi \vdash \Delta}$ (Exchange Left), $\frac{\Gamma \vdash \Delta, B, A, \Lambda}{\Gamma \vdash \Delta, A, B, \Lambda}$ (Exchange Right)
- Exchange rules are used to rearrange the order of formulas in the sequent without affecting the validity of the proof

Proof Construction

Building Proof Trees

- Proof tree in sequent calculus starts with the sequent to be proved at the bottom (root)
- Apply sequent calculus rules to the sequent, working upwards to create new nodes
- Each node in the proof tree is a sequent derived from the sequent below it by applying a rule
- Branches in the proof tree represent case distinctions or the use of multiple rules

Proof Strategies

- Work backwards from the desired conclusion (bottom-up approach)
- Break down complex formulas using introduction rules
- Use structural rules (weakening, contraction, exchange) to manipulate the sequent as needed
- Apply the cut rule to introduce intermediate formulas that can help in proving the sequent
- Aim to reach axioms (sequents with the same formula on both sides) at the top of the proof tree

4.3 Comparison of natural deduction and sequent calculus

Natural deduction and sequent calculus are two key proof systems in logic. While both can prove the same theorems, they differ in structure and properties. Natural deduction mimics human reasoning, while sequent calculus offers a more symmetric approach.

Sequent calculus shines in proof search and automated reasoning due to its subformula property and cut elimination. Natural deduction, however, often feels more intuitive to humans. Both systems have found applications in programming languages and type theory.

Structural Differences and Properties

Proof Structure and Symmetry

- Natural deduction proofs have a tree-like structure with assumptions at the leaves and the conclusion at the root

- Sequent calculus proofs have a more symmetric structure with left and right rules for each connective
- Left rules decompose formulas on the left side of the turnstile ($\vdash$)
- Right rules decompose formulas on the right side of the turnstile
- The symmetric nature of sequent calculus rules allows for a more uniform proof search compared to natural deduction
- Sequent calculus rules maintain the balance between the left and right sides of the sequent throughout the proof

Subformula Property and Cut Elimination

- The subformula property states that every formula appearing in a sequent calculus proof is a subformula of the formulas in the conclusion sequent
- This property holds for cut-free proofs in sequent calculus
- Natural deduction proofs do not always satisfy the subformula property due to the introduction and elimination rules
- Cut elimination is the process of transforming a sequent calculus proof with cuts into a cut-free proof
- The cut rule allows the use of lemmas in sequent calculus proofs
- Gentzen's cut elimination theorem proves that any sequent calculus proof can be transformed into a cut-free proof
- Cut-free proofs in sequent calculus satisfy the subformula property, leading to a more streamlined proof structure

Expressiveness and Applications

Expressiveness and Proof Search

- Both natural deduction and sequent calculus are expressive enough to capture classical and intuitionistic logic
- Sequent calculus is more suitable for proof search algorithms due to its symmetric rule structure
- Proof search in sequent calculus can be performed by applying rules bottom-up, reducing the goal sequent to simpler subgoals
- The subformula property in cut-free sequent calculus proofs limits the search space for proof search algorithms
- Natural deduction proofs can be more intuitive for humans to construct and understand, as they resemble the reasoning process more closely

Applications in Programming Languages

- The Curry-Howard correspondence establishes a connection between proofs and programs
- Propositions in logic correspond to types in programming languages
- Proofs of propositions correspond to programs of the corresponding type
- Sequent calculus has been used as a foundation for type systems in programming languages

- The symmetry and subformula property of sequent calculus make it suitable for type checking and type inference algorithms
- Natural deduction has been used as a basis for the design of functional programming languages (Haskell, ML)
- Introduction and elimination rules in natural deduction correspond to constructors and pattern matching in functional programming

4.4 Proof normalization in natural deduction

Proof normalization in natural deduction is all about simplifying proofs to their most basic form. It's like decluttering your logical arguments, getting rid of unnecessary steps and redundancies to make them cleaner and more efficient.

This process ties into the broader theme of natural deduction by showing how we can transform and optimize proofs. It's a key tool for understanding the structure of logical arguments and making them more elegant and easier to work with.

Normalization

Normal Form and Reduction Steps

- Normal form represents the simplest or most reduced form of a proof
- Cannot be further simplified or reduced using defined reduction rules
- Reduction steps are the individual transformations applied to a proof to reach its normal form
- Each step simplifies the proof while preserving its logical validity
- Examples of reduction steps include β -reduction (function application) and η -reduction (function abstraction)

Strong and Weak Normalization

- Strong normalization guarantees that every sequence of reductions on a proof will eventually terminate
- Proofs in a strongly normalizing system always reach a normal form, regardless of the order of reductions
- Ensures the consistency and decidability of the proof system (e.g., simply typed lambda calculus)
- Weak normalization ensures that there exists at least one sequence of reductions that leads to a normal form
- Some reduction paths may not terminate, but there is always at least one path that does
- More relaxed property compared to strong normalization (e.g., untyped lambda calculus is weakly normalizing)

Proof Structures

Detours in Proofs

- Detours are redundant or unnecessary steps in a proof that can be eliminated without affecting its validity
- Occur when an introduction rule is immediately followed by an elimination rule for the same connective
- Example: Introducing a conjunction ($A \wedge B$) and then immediately eliminating it to obtain one of its components (A or B)
- Eliminating detours simplifies proofs and brings them closer to their normal form
- Reduces the size and complexity of proofs
- Improves the efficiency of proof search and automated reasoning

Maximum Formulas

- Maximum formulas are formulas that are both the conclusion of an introduction rule and the premise of an elimination rule
- Represent the peak of a detour in a proof
- Example: In the detour $A \wedge B$ mentioned earlier, the formula $A \wedge B$ is a maximum formula
- Identifying and eliminating maximum formulas is a key step in proof normalization
- Corresponds to the reduction of detours
- Helps in obtaining a more compact and direct proof structure

Proof Interpretations

Curry-Howard Isomorphism

- The Curry-Howard isomorphism establishes a correspondence between proofs and programs
- Propositions in logic correspond to types in programming languages
- Proofs of propositions correspond to programs of the corresponding types
- This isomorphism allows for the exchange of ideas between logic and computation
- Proofs can be seen as programs, and programs can be seen as proofs
- Enables the application of logical techniques to programming and vice versa (e.g., using type systems to ensure program correctness)

Computational Interpretation of Proofs

- The computational interpretation of proofs assigns a computational meaning to logical proofs
- Proofs are seen as specifications for computation
- The process of proof normalization corresponds to the execution of the associated program

- This interpretation provides insights into the constructive nature of proofs
- Constructive proofs can be seen as algorithms that compute the objects they prove the existence of
- Non-constructive proofs, on the other hand, may not have a direct computational content
- The computational interpretation has practical applications in areas such as program extraction from proofs and the design of dependently typed programming languages (Coq, Agda)

Unit 5 – Cut Elimination and Normalization

5.1 Cut elimination theorem and its significance

The Cut Elimination Theorem is a game-changer in proof theory. It says we can transform any proof into one without the cut rule, simplifying proofs and making them easier to analyze.

This theorem has big implications. It helps prove consistency, decidability, and other important results in logic. It's a powerful tool that opens up new ways to understand and work with proofs.

Cut Elimination and Its Properties

Gentzen's Cut Elimination Theorem

- States that every proof in a sequent calculus can be transformed into a proof that does not use the cut rule
- Fundamental result in proof theory discovered by Gerhard Gentzen in 1934
- Applies to various logical systems including classical and intuitionistic logic
- Elimination process involves complex proof transformations and induction on the structure of the proof

Properties of Cut-Free Proofs

- Subformula property: every formula in a cut-free proof is a subformula of the end-sequent
- Constrains the search space for proofs and enables proof search algorithms
- Allows for a local proof-checking procedure that only needs to consider subformulas
- Analytical proofs: cut-free proofs are analytical, meaning they only use logical rules that decompose the formulas in the end-sequent
- Enables proof normalization and simplification techniques
- Provides a way to study the structure and complexity of proofs

Consistency and Cut Elimination

- Cut elimination can be used to prove the consistency of a logical system
- Consistency means there is no proof of a contradiction (e.g., proving both A and $\neg A$)
- If a system has a cut elimination theorem, then a proof of a contradiction cannot exist in the cut-free fragment
- Reason: in a cut-free proof, the end-sequent only contains subformulas of the original formulas, so a contradiction cannot be introduced

Consequences and Applications

Decidability Results

- Cut elimination is a key tool for proving decidability of logical systems

- Decidability means there is an algorithm to determine if a given formula is provable or not
- Procedure: search for a cut-free proof, which is guaranteed to exist by cut elimination
- The subformula property bounds the search space, making it finite
- Examples: decidability of propositional logic, first-order logic with a relational signature

Craig's Interpolation Theorem

- States that if $A \vdash B$, then there exists a formula C (the interpolant) such that: 1. $A \vdash C$ 2. $C \vdash B$ 3. C only contains symbols common to both A and B
- Cut elimination is used in the proof of interpolation theorems
- The interpolant C is extracted from a cut-free proof of $A \vdash B$
- Interpolation has applications in modular verification and reasoning about components

Herbrand's Theorem and Skolemization

- Herbrand's theorem relates the provability of a first-order formula to the provability of a potentially infinite set of propositional instances
- Instances are obtained by replacing existential variables with terms from the Herbrand universe
- Skolemization is a technique for eliminating existential quantifiers by introducing Skolem functions
- Produces an equisatisfiable formula that is easier to reason about
- Cut elimination is used to prove the correctness of Skolemization and Herbrand's theorem
- Shows that Skolemization preserves provability and that Herbrand instances can be used to construct a proof

Proof-Theoretic Strength and Ordinal Analysis

- Cut elimination is central to the study of proof-theoretic strength of logical systems
- Strength is measured by the ordinal recursive functions that can be proved total in the system
- Ordinal analysis assigns ordinals to cut-free proofs, characterizing their complexity
- Larger ordinals correspond to stronger systems
- Examples: Peano arithmetic has strength ϵ_0 , second-order arithmetic has strength Γ_0
- Cut elimination enables the comparison and classification of theories based on their proof-theoretic strength

5.2 Proof of cut elimination for propositional logic

Cut elimination is a crucial concept in proof theory, showing that any proof using the cut rule can be transformed into a cut-free proof. This process simplifies proofs, making them more direct and easier to analyze.

The proof of cut elimination for propositional logic involves intricate induction principles and transformation steps. By gradually reducing the complexity of cuts, we can ultimately eliminate all cuts, resulting in a more streamlined and constructive proof structure.

Sequent Calculus and Cut Rule

Overview of Sequent Calculus

- Sequent calculus is a formal system for logical proofs
- Consists of rules for introducing and eliminating logical connectives
- Sequents are expressions of the form $\Gamma \vdash \Delta$, where Γ and Δ are sets of formulas
- Intuitive meaning: if all formulas in Γ are true, then at least one formula in Δ is true

The Cut Rule and Its Significance

- Cut rule allows the use of lemmas in proofs
- Stated as: if $\Gamma \vdash \Delta, A$ and $\Gamma, A \vdash \Delta$, then $\Gamma \vdash \Delta$
- Principal formula is the formula A that is "cut out" in the application of the cut rule
- Cut elimination theorem states that any proof with cuts can be transformed into a cut-free proof
- Cut-free proofs are more direct and avoid detours through lemmas

Properties of Cut-Free Proofs

- Cut-free proofs have the subformula property: every formula in the proof is a subformula of the conclusion
- Cut-free proofs are more constructive and computationally meaningful
- Cut elimination is a key tool for proving consistency and normalization results

Induction Principles

Induction on Cut-Rank

- Cut-rank is the complexity of the principal formula in a cut rule application
- Induction on cut-rank proceeds by showing that cuts on complex formulas can be reduced to cuts on simpler formulas
- Base case: cuts on atomic formulas can be eliminated directly
- Inductive step: show how to reduce cuts on $A \wedge B$, $A \vee B$, $A \rightarrow B$, etc. to cuts on A and B

Induction on Cut-Height

- Cut-height is the maximum number of cut rule applications along any branch in a proof tree
- Induction on cut-height proceeds by showing that proofs with many cuts can be transformed into proofs with fewer cuts
- Base case: proofs with cut-height 0 are already cut-free
- Inductive step: show how to reduce the cut-height by eliminating topmost cuts or pushing cuts upwards in the proof tree

Combining Induction Principles

- Cut elimination proofs often use a double induction on cut-rank and cut-height
- Outer induction on cut-rank, with inner induction on cut-height for each rank
- This allows for a gradual simplification of the cuts in a proof until all cuts are eliminated

Transformation Steps

Weakening and Contraction

- Weakening rules allow formulas to be added to the left or right side of a sequent
- Contraction rules allow duplicate formulas to be removed from the left or right side of a sequent
- These rules are often used in cut elimination to adjust the contexts of sequents as needed

Reduction and Permutation Steps

- Reduction steps are used to reduce the cut-rank by eliminating cuts on complex formulas
- Example: a cut on $A \wedge B$ can be reduced to two cuts on A and B
- Permutation steps are used to push cuts upwards in the proof tree, reducing the cut-height
- Example: a cut can be permuted above a rule that does not affect the principal formula of the cut

Elimination of Atomic Cuts

- Atomic cuts are cuts on formulas that cannot be further decomposed (propositional variables)
- These cuts can be eliminated directly by a simple transformation
- After all non-atomic cuts are eliminated, only atomic cuts remain
- The final step of cut elimination removes all atomic cuts, resulting in a cut-free proof

5.3 Cut elimination for first-order logic

Cut elimination for first-order logic extends the technique to more complex logical systems. It shows that even with quantifiers, we can transform proofs to avoid using the cut rule, simplifying our reasoning process.

This topic builds on earlier cut elimination results, tackling the challenges posed by quantifiers. It's crucial for understanding the power and limitations of formal proof systems in first-order logic.

Quantifier Rules and Eigenvariable Condition

Quantifier Rules in First-Order Logic

- Introduce rules for handling quantifiers in first-order logic proofs
- $\forall$ -introduction rule allows inferring $\forall x A(x)$ from $A(y)$ where y is an arbitrary variable not occurring free in any assumption
- $\forall$ -elimination rule allows inferring $A(t)$ from $\forall x A(x)$ where t is any term

- $\exists$ -introduction rule allows inferring $\exists xA(x)$ from $A(t)$ where t is any term
- $\exists$ -elimination rule allows inferring C from $\exists xA(x)$ and $A(y) \vdash C$ where y is a variable not occurring free in C or any assumption except $A(y)$

Eigenvariable Condition and Substitution

- Eigenvariable condition ensures soundness of quantifier rules by restricting variable occurrences
- Prevents inferring invalid formulas like $\forall x\exists y(x = y)$ from $\exists y(y = y)$
- Substitution is the process of replacing variables with terms in a formula
- Substitution must respect bound variable scoping to avoid variable capture and maintain validity
- Bound variable renaming may be necessary to avoid clashes during substitution

Skolemization in First-Order Logic

- Skolemization is a technique for eliminating existential quantifiers from formulas
- Introduces Skolem functions that take the universally quantified variables in scope as arguments
- Replaces $\exists yA(x, y)$ with $A(x, f(x))$ where f is a fresh function symbol
- Produces an equisatisfiable formula without existential quantifiers
- Enables proof search and automated reasoning techniques in first-order logic

Cut Elimination Theorems

Gentzen's Midsequent Theorem

- Gentzen proved the midsequent theorem for first-order logic in the sequent calculus LK
- States that every valid sequent has a proof with a single application of the cut rule
- The cut formula appears exactly once in the antecedent and succedent of the midsequent
- Provides a normal form for proofs in the sequent calculus
- Demonstrates the redundancy of the cut rule for proving valid sequents

Partial Cut Elimination Results

- Cut elimination theorems show that the cut rule is admissible in various proof systems
- Gentzen proved cut elimination for the intuitionistic sequent calculus LJ and the classical sequent calculus LK without quantifier rules
- Tait and others extended cut elimination to first-order logic with quantifiers
- Partial cut elimination results allow cuts on formulas of bounded quantifier complexity
- Eliminating cuts may significantly increase the size and complexity of proofs

Advanced Topics

Infinitary Proof Theory and Cut Elimination

- Infinitary logic allows proofs with infinite branches and infinitely long formulas
- Extends first-order logic to infinitary languages with infinitely long conjunctions and disjunctions
- Infinitary proof systems have been studied for model theory and descriptive set theory
- Cut elimination theorems have been proven for various infinitary logics
- Infinitary cut elimination often requires transfinite induction and ordinal analysis
- Provides insight into the strength and complexity of infinitary logical systems

5.4 Consequences and applications of cut elimination

Cut elimination is a powerful technique in proof theory with far-reaching consequences. It helps prove consistency of logical systems and shows when extensions are conservative. This impacts how we understand and build formal systems.

Cut elimination also plays a key role in important theorems like interpolation and Herbrand's theorem. These results bridge different areas of logic and have practical applications in automated reasoning and theorem proving.

Logical Consequences

Consistency and Conservativity

- Consistency proofs establish that a logical system is free from contradictions
- Ensures the system does not allow deriving both a formula and its negation
- Cut elimination plays a crucial role in proving consistency of various logics (first-order logic, higher-order logic)
- Conservative extension refers to extending a logical system with new axioms or rules without introducing new theorems in the original language
- Guarantees that the extended system does not prove anything new about the original concepts
- Cut elimination helps show that an extension is conservative by proving that any proof in the extended system can be transformed into a proof in the original system

Interpolation and Definability Theorems

- Craig's interpolation theorem states that if $A \rightarrow B$ is provable in first-order logic, then there exists a formula C (the interpolant) such that: 1. $A \rightarrow C$ and $C \rightarrow B$ are provable 2. C contains only symbols common to both A and B
- Interpolation has applications in modular verification and reasoning about large systems
- Beth's definability theorem relates implicit and explicit definability in first-order logic
- If a predicate is implicitly definable, then it is also explicitly definable
- Has connections to interpolation and is used in model theory and algebraic logic

- Cut elimination is often employed in proofs of interpolation and definability theorems

Foundational Results

Herbrand's Theorem

- Herbrand's theorem is a fundamental result in proof theory and automated theorem proving
- States that if a formula in prenex normal form is provable in first-order logic, then a finite disjunction of instances of the matrix is propositionally valid
- Provides a bridge between first-order logic and propositional logic
- Allows reducing first-order validity to propositional satisfiability (SAT)
- Herbrand's theorem has important consequences for automated reasoning
- Forms the basis for the Herbrand universe and Herbrand base in automated theorem proving
- Enables the development of resolution and unification-based proof methods

Löwenheim-Skolem Theorem

- The Löwenheim-Skolem theorem is a significant result in model theory with implications for proof theory
 - States that if a countable first-order theory has an infinite model, then it has models of every infinite cardinality
 - Comes in two versions: the downward (every infinite model has a countable elementary submodel) and upward (every model can be extended to a model of arbitrary larger cardinality)
- Löwenheim-Skolem theorems
- The Löwenheim-Skolem theorem highlights the limitations of first-order logic in characterizing infinite structures
 - Implies that first-order logic cannot capture concepts like finiteness or uncountability
 - Has connections to compactness and the existence of non-standard models of arithmetic

Applications

Automated Theorem Proving

- Automated theorem proving (ATP) aims to develop computer programs that can prove mathematical theorems automatically
- Utilizes various proof search strategies and heuristics to find proofs in formal logical systems
- Examples of ATP systems include resolution-based provers (Vampire, E), tableau-based provers (leanCoP), and SMT solvers (Z3, CVC4)
- Cut elimination has significant implications for the design and implementation of ATP systems
- Allows focusing on cut-free proofs, which generally have a simpler structure and are more amenable to automation

- Enables the development of efficient proof search procedures that avoid redundant or unnecessary inferences

Proof Complexity

- Proof complexity studies the size and efficiency of proofs in various logical systems
- Investigates the length of proofs required to establish tautologies or contradictions
- Considers different proof systems (Frege systems, resolution, sequent calculi) and their relative strengths
- Has connections to computational complexity theory and the P vs. NP problem
- Cut elimination has a significant impact on proof complexity
- Cut-free proofs may be exponentially larger than proofs with cuts for some tautologies
- The efficiency of cut elimination procedures affects the complexity of proof search and verification
- Studying the complexity of cut elimination itself is an important area of research in proof complexity

Unit 6 – Completeness and Compactness Theorems

6.1 Gödel's completeness theorem for first-order logic

Gödel's completeness theorem is a cornerstone of first-order logic. It shows that if a sentence is logically valid, it can be proved using first-order logic rules. This powerful result connects the semantic notion of truth with the syntactic concept of provability.

The theorem has far-reaching implications for mathematical logic and foundations. It establishes that first-order logic is complete, meaning all valid statements can be derived within the system. This provides a solid basis for formal reasoning in mathematics and computer science.

Completeness and Soundness

Gödel's Completeness Theorem and First-Order Logic

- Gödel's completeness theorem states that if a sentence is logically valid in first-order logic, then it can be proved using the rules of first-order logic
- This means that the proof system of first-order logic is complete
- For example, if a sentence like $\forall x(P(x) \rightarrow Q(x))$ is true in every model, then it can be proved using the rules of first-order logic
- First-order logic is a formal system that includes predicates, variables, quantifiers, and logical connectives
- It allows for expressing complex mathematical statements and reasoning about them
- First-order logic is more expressive than propositional logic, which only deals with simple propositions and connectives

Soundness and Consistency

- Soundness is a property of a logical system where every provable sentence is logically valid
- In other words, if a sentence can be proved using the rules of the system, then it must be true in every model
- Soundness ensures that the proof system does not allow for proving false statements
- Consistency is a property of a logical system where no contradiction can be derived
- A contradiction is a sentence of the form $A \wedge \neg A$, where both a sentence and its negation are provable
- If a system is consistent, it means that it is impossible to prove both a sentence and its negation using the rules of the system

Models and Satisfiability

Models and Satisfiability

- A model is an interpretation of a first-order language that assigns meanings to the symbols in the language
- A model consists of a non-empty domain (or universe) and an interpretation function that maps the symbols to elements, subsets, or relations on the domain
- For example, a model for the language of arithmetic might have the natural numbers as its domain and interpret the symbol $+$ as the addition function
- A sentence is satisfiable if there exists a model in which the sentence is true
- For instance, the sentence $\exists x(P(x) \wedge Q(x))$ is satisfiable if there is a model with an element in the domain that satisfies both predicates P and Q
- A sentence is valid if it is true in every model
- For example, the sentence $\forall x(P(x) \rightarrow P(x))$ is valid because it is true in every model, regardless of the interpretation of the predicate P

Validity and Logical Consequence

- Validity is a stronger notion than satisfiability, as a valid sentence is true in all models, while a satisfiable sentence is true in at least one model
- Valid sentences are also called tautologies
- If a sentence is valid, then its negation is unsatisfiable
- A sentence ϕ is a logical consequence of a set of sentences Γ if every model that makes all the sentences in Γ true also makes ϕ true
- We write this as $\Gamma \models \phi$
- Logical consequence is a semantic notion, based on the truth values of sentences in models

Proof Techniques

Henkin Construction

- The Henkin construction is a method for building a model of a consistent set of sentences in first-order logic
- It is used in the proof of Gödel's completeness theorem to show that every consistent set of sentences has a model
- The construction proceeds by extending the original language with new constant symbols and adding sentences that specify the properties of these new constants
- The new constants are used to ensure that every existential sentence in the set has a witness in the model
- The Henkin construction builds a model step by step, ensuring that the resulting model satisfies all the sentences in the original set

- This is done by creating a complete consistent set of sentences that includes the original set and the new sentences added during the construction
- The Henkin construction demonstrates the close connection between consistency and satisfiability in first-order logic
- If a set of sentences is consistent, then the Henkin construction can be used to build a model that satisfies the set
- Conversely, if a set of sentences has a model, then it must be consistent, as an inconsistent set cannot have a model

6.2 Proof of the completeness theorem

Henkin's proof is a powerful method for showing completeness in formal systems. It builds a maximal consistent set of sentences and uses it to construct a model that satisfies all sentences in the set.

This approach is crucial for understanding completeness theorems. By demonstrating that every consistent set of sentences has a model, Henkin's proof establishes the link between syntax and semantics in logical systems.

Henkin's Proof and Consistency

Proving Completeness Using Henkin's Method

- Henkin's proof provides a method for demonstrating the completeness of a formal system
- Involves constructing a maximal consistent set of sentences from a given consistent set
- Utilizes Lindenbaum's lemma to extend the consistent set to a maximal consistent set
- The maximal consistent set serves as the basis for constructing a model that satisfies all sentences in the set

Properties of Consistent Sets

- A consistent set of sentences is a set that does not contain any contradictions
- No sentence and its negation can be derived simultaneously from a consistent set
- Consistency is a fundamental requirement for constructing a model that satisfies the sentences
- Maximal consistent sets play a crucial role in Henkin's proof by ensuring completeness

Lindenbaum's Lemma and Maximal Consistency

- Lindenbaum's lemma states that any consistent set of sentences can be extended to a maximal consistent set
- A maximal consistent set is a consistent set that cannot be further extended without introducing inconsistency
- The lemma guarantees the existence of a maximal consistent set for any given consistent set
- Maximal consistent sets have the property that for every sentence, either the sentence or its negation is included in the set ($\forall \phi, \phi \in \Gamma \vee \neg \phi \in \Gamma$)

Model Construction

Witnesses and Term Models

- A witness is a constant symbol that is used to represent an element in the domain of a model
- Witnesses are introduced for each existential sentence in the maximal consistent set to ensure its satisfaction
- A term model is constructed using the maximal consistent set and the introduced witnesses
- In a term model, the domain consists of equivalence classes of terms, where terms are equivalent if their equality is provable from the maximal consistent set

Canonical Models and Completeness

- The canonical model is a model constructed from the maximal consistent set and the term model
- The canonical model satisfies all sentences in the maximal consistent set by construction
- The truth value of a sentence in the canonical model is determined by its membership in the maximal consistent set
- The existence of the canonical model for any consistent set of sentences proves the completeness of the formal system
- Completeness means that every valid sentence is provable within the system ($\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi$)

Constructing the Canonical Model

- The domain of the canonical model is the set of equivalence classes of terms in the term model
- Function symbols are interpreted as functions on the equivalence classes, preserving the equality relations
- Predicate symbols are interpreted based on their occurrence in the maximal consistent set
- Constants and variables are assigned to their corresponding equivalence classes in the term model
- The canonical model satisfies all sentences in the maximal consistent set, establishing completeness

6.3 Compactness theorem and its consequences

The compactness theorem is a powerful tool in logic, stating that if every finite subset of sentences is satisfiable, the entire set is too. It's crucial for proving the existence of models with specific properties and understanding the limits of first-order logic.

This theorem connects to broader concepts like the Löwenheim-Skolem theorems and non-standard models. These ideas show how first-order logic can't fully capture infinite structures, leading to unexpected models and revealing the richness of logical systems.

Compactness Theorem and Satisfiability

Compactness Theorem and Its Implications

- States that if every finite subset of a set of sentences is satisfiable, then the entire set is satisfiable

- Implies that if a set of sentences is unsatisfiable, then there must be a finite subset that is also unsatisfiable
- Allows for the construction of models with certain properties by considering sets of sentences that describe those properties
- Provides a powerful tool for proving the existence of models with desired characteristics

Finite Subsets and Their Role

- A finite subset is a subset of a set that contains a finite number of elements
- In the context of the compactness theorem, we consider finite subsets of a set of sentences
- The satisfiability of finite subsets is crucial in determining the satisfiability of the entire set
- If every finite subset of a set of sentences is satisfiable, then the compactness theorem guarantees the satisfiability of the entire set

Satisfiability and Its Significance

- Satisfiability refers to the existence of a model that makes a set of sentences true
- A set of sentences is satisfiable if there exists an interpretation that assigns truth values to the symbols in the sentences such that all the sentences are true
- Satisfiability is a fundamental concept in logic and plays a key role in the compactness theorem
- The compactness theorem relates the satisfiability of finite subsets to the satisfiability of the entire set of sentences

Löwenheim-Skolem Theorem and Its Variants

Löwenheim-Skolem Theorem

- States that if a countable first-order theory has an infinite model, then it has models of every infinite cardinality
- Implies that first-order logic cannot characterize a unique infinite model up to isomorphism
- Highlights the limitations of first-order logic in capturing the properties of infinite structures
- Has important consequences for the study of models and their sizes

Upward and Downward Löwenheim-Skolem Theorems

- The upward Löwenheim-Skolem theorem states that if a countable theory has an infinite model, then it has models of arbitrarily large cardinality
- Allows for the construction of models larger than the original model while preserving the satisfiability of the theory
- The downward Löwenheim-Skolem theorem states that if a theory has an infinite model, then it has a countable model
- Ensures the existence of smaller models that satisfy the same theory as the original model
- Together, these theorems demonstrate the flexibility of first-order logic in terms of the sizes of models

Non-Standard Models and Ultraproducts

Ultraproducts and Their Construction

- An ultraproduct is a construction that combines a collection of structures using an ultrafilter
- Ultrafilters are special types of filters that extend the notion of "almost all" elements satisfying a property
- The ultraproduct construction allows for the creation of new structures that inherit properties from the original structures
- Ultraproducts have applications in model theory and can be used to prove the compactness theorem

Non-Standard Models and Their Properties

- Non-standard models are models that contain elements or structures that are not present in the standard model
- These models arise naturally when considering the ultraproduct construction or the compactness theorem
- Non-standard models can have surprising properties that differ from the standard model (hyperreal numbers in non-standard analysis)
- The existence of non-standard models demonstrates the richness and complexity of first-order logic and its models
- Non-standard models have important applications in various areas of mathematics, such as non-standard analysis and the study of infinite structures (non-standard integers in number theory)

6.4 Applications of completeness and compactness

Completeness and compactness theorems are powerful tools in logic. They allow us to prove the existence of models and extend theories. This section explores their applications in model theory, non-standard analysis, and set theory.

These applications showcase the theorems' versatility. From elementary equivalence to Skolem's paradox, we see how completeness and compactness shape our understanding of mathematical structures and formal systems.

Model Theory and Elementary Equivalence

Fundamentals of Model Theory

- Model theory studies mathematical structures and their relationships using formal languages
- Focuses on the interplay between a formal language and its interpretations (models)
- Analyzes properties of structures that can be expressed in a given language
- Explores the connections between syntactic and semantic notions

Elementary Equivalence and Isomorphism

- Two structures are elementarily equivalent if they satisfy the same first-order sentences

- Elementary equivalence is a weaker notion than isomorphism
- Isomorphic structures are always elementarily equivalent
- Elementarily equivalent structures may not be isomorphic
- Elementarily equivalent structures cannot be distinguished by first-order properties
- Examples of elementarily equivalent but non-isomorphic structures:
- Dense linear orders without endpoints (rational numbers and real numbers)
- Algebraically closed fields of the same characteristic and uncountable cardinality

Categoricity and Complete Theories

- A theory is categorical if all its models of a given cardinality are isomorphic
- Categoricity depends on the cardinality of the models
- A theory can be categorical in one cardinality but not in another
- Examples of categorical theories:
- Theory of algebraically closed fields of a fixed characteristic (categorical in all uncountable cardinalities)
- Theory of dense linear orders without endpoints (categorical in cardinality $\aleph_1$)
- Complete theories have a unique model up to isomorphism in each infinite cardinality
- Completeness implies categoricity, but not vice versa

Non-standard Analysis and Transfer Principle

Introduction to Non-standard Analysis

- Non-standard analysis is an alternative approach to infinitesimals and infinite numbers
- Extends the real number system to include infinitesimal and infinite elements
- Allows for a rigorous treatment of infinitesimals and their properties
- Provides a framework for studying continuous phenomena using discrete methods

The Transfer Principle

- The transfer principle is a fundamental tool in non-standard analysis
- States that any first-order statement true in the standard model is also true in the non-standard model, and vice versa
- Allows for the transfer of properties between standard and non-standard models
- Enables the use of classical reasoning and intuition in non-standard settings
- Examples of transferred properties:
- Arithmetic operations and their properties
- Order relations and their properties

- Limits, continuity, and differentiability of functions

Applications in Infinite Combinatorics

- Non-standard analysis provides powerful tools for infinite combinatorics
- Allows for the construction of non-standard models with rich combinatorial properties
- Enables the study of large structures and their asymptotic behavior
- Examples of applications:
 - Szemerédi's theorem on arithmetic progressions
 - Ramsey theory and the existence of large homogeneous substructures
 - Ergodic theory and the study of measure-preserving dynamical systems

Skolem's Paradox

The Paradox and Its Implications

- Skolem's paradox highlights the discrepancy between the intended and the formal interpretation of set theory
- States that countable models of set theory exist, despite the theory proving the existence of uncountable sets
- Arises from the fact that first-order logic cannot capture the notion of uncountability
- Demonstrates the limitations of first-order logic in expressing certain mathematical concepts

Resolving the Paradox

- Skolem's paradox is not a true contradiction but rather a counterintuitive result
- The existence of countable models does not contradict the existence of uncountable sets in the intended interpretation
- The paradox can be resolved by recognizing the difference between the intended and the formal interpretation of set theory
- Higher-order logics or stronger axioms can be used to exclude countable models and capture the intended meaning of uncountability
- Examples of approaches to resolve the paradox:
 - Second-order logic and the full semantics of second-order set theory
 - Large cardinal axioms and the study of inner models of set theory

Unit 7 – Gödel's Incompleteness Theorems

7.1 Gödel numbering and representability

Gödel numbering assigns unique numbers to symbols and formulas in formal systems, enabling complex mathematical statements to be expressed using basic arithmetic. This clever encoding allows metamathematical concepts to be represented within the system itself, setting the stage for Gödel's groundbreaking work.

Representability and self-reference are crucial concepts in Gödel's Incompleteness Theorems. By constructing formulas that refer to themselves, Gödel demonstrated the existence of undecidable statements in consistent formal systems, revealing fundamental limitations in mathematical reasoning.

Gödel Numbering and Encoding

Mapping Symbols to Numbers

- Gödel numbering assigns a unique natural number to each symbol, variable, and formula in a formal system
- Encoding converts statements and proofs in a formal system into natural numbers using the assigned Gödel numbers
- Allows metamathematical statements about a formal system to be expressed within the system itself
- Arithmetization is the process of representing statements about a formal system as statements about natural numbers
- Enables complex formulas to be represented and manipulated using basic arithmetic operations

Primitive Recursive Functions

- Primitive recursive functions are a class of computable functions that can be obtained from basic functions using composition and primitive recursion
- Basic functions include constant functions, successor function, and projection functions
- Composition combines two or more primitive recursive functions to create a new function
- Primitive recursion defines a function in terms of its value on smaller inputs
- Primitive recursive functions play a crucial role in Gödel numbering and encoding
- Used to define the encoding of formulas and proofs as natural numbers
- Allow for the construction of complex arithmetic statements that represent metamathematical properties

Representability and Self-Reference

Expressing Metamathematical Concepts

- Representability refers to the ability to express metamathematical concepts and statements within a formal system
- A predicate or relation is representable if there exists a formula in the system that is true exactly when the predicate or relation holds
- Self-reference occurs when a statement or formula refers to itself or its own properties
- Achieved through clever construction of formulas using Gödel numbering and representability
- Allows for the creation of statements that assert their own unprovability or other metamathematical properties

Diagonalization and Undecidability

- Diagonalization is a technique used to prove the existence of certain mathematical objects or properties
- Involves constructing a new object or property that differs from a given list of objects in a specific way
- Used by Gödel to construct a self-referential formula that asserts its own unprovability
- Diagonalization arguments often lead to undecidability results
- A statement is undecidable if neither it nor its negation can be proven within a formal system
- Gödel's First Incompleteness Theorem demonstrates the existence of undecidable statements in any consistent, sufficiently powerful formal system
- Self-reference and diagonalization are key ingredients in Gödel's Incompleteness Theorems
- Used to construct statements that cannot be proven or disproven within the system itself
- Highlight the limitations of formal systems and the inherent incompleteness of mathematics

7.2 First incompleteness theorem: statement and proof outline

Gödel's First Incompleteness Theorem shook the foundations of mathematics. It states that any consistent formal system containing arithmetic is incomplete, meaning there are true statements that can't be proven within the system.

This groundbreaking result shows that no single formal system can capture all of mathematics. It relies on constructing a self-referential statement, similar to the liar's paradox, that asserts its own unprovability within the system.

Formal Systems and Properties

Definition and Components of Formal Systems

- Formal system consists of a language, axioms, and inference rules used to derive theorems from the axioms
- Language includes an alphabet of symbols and rules for forming well-formed formulas (wffs)

- Axioms are a set of wffs assumed to be true without proof
- Inference rules specify how new theorems can be derived from axioms and previously proven theorems

Consistency and Completeness

- Consistency means a formal system cannot prove both a statement and its negation
- Inconsistent systems can prove any statement, making them useless
- Completeness means for every wff, either it or its negation can be proven within the system
- In a complete system, every true statement is provable

Peano Arithmetic as a Formal System

- Peano arithmetic (PA) is a formal system for the natural numbers and arithmetic operations
- PA includes axioms for the successor function, addition, multiplication, and induction
- Despite its apparent simplicity, PA is powerful enough to express complex mathematical statements (Goldbach's conjecture, Fermat's Last Theorem)

First Incompleteness Theorem

Statement and Implications of the First Incompleteness Theorem

- Gödel's first incompleteness theorem states that any consistent formal system containing arithmetic is incomplete
- There exist true statements (undecidable propositions) that cannot be proven within the system itself
- This implies that no consistent formal system can capture all of mathematics

Undecidable Propositions and Gödel Sentences

- An undecidable proposition is a true statement that cannot be proven within a given formal system
- Gödel constructed a self-referential statement (Gödel sentence) that asserts its own unprovability
- If the system is consistent, the Gödel sentence must be true but unprovable

Proof Sketch of the First Incompleteness Theorem

- Gödel numbering assigns a unique number to each symbol, formula, and proof in the system
- The properties and relations of formulas and proofs can be expressed arithmetically using Gödel numbers
- Gödel constructed a formula G that represents the statement "The formula with Gödel number X is not provable"
- G is true but unprovable, as proving it would lead to a contradiction (if G is provable, then it is not provable)

Related Concepts

Liar's Paradox and Self-Reference

- The liar's paradox is a self-referential statement that leads to a contradiction ("This sentence is false")
- If the sentence is true, then it is false; if it is false, then it is true
- Gödel's proof relies on constructing a similar self-referential statement within a formal system

Implications for Mathematics and Logic

- The incompleteness theorems show that mathematics cannot be reduced to a single, complete formal system
- There will always be true statements that cannot be proven within any given consistent system
- This has led to the development of alternative approaches to foundations of mathematics (set theory, type theory)

7.3 Second incompleteness theorem and its implications

Gödel's Second Incompleteness Theorem shakes the foundations of mathematics. It reveals that consistent formal systems can't prove their own consistency, challenging the idea of a complete, self-contained mathematical framework.

This theorem exposes a gap between formal provability and truth in math. It shows there are true statements that can't be proven within a system, highlighting the limitations of formal reasoning and sparking philosophical debates about mathematical knowledge.

Limitations of Formal Systems

Unprovability of Consistency

- Second incompleteness theorem states that if a formal system T containing arithmetic is consistent, then the statement " T is consistent" cannot be proved within T itself
- This theorem demonstrates the limitations of formal systems in proving their own consistency
- Consistency statement is a formal statement expressing that a given system is free from contradictions
- In order for a system to prove its own consistency, it would need to be able to prove this statement within itself, which the second incompleteness theorem shows is impossible for sufficiently strong systems

Formal Provability and Truth

- Formal provability refers to the existence of a proof for a statement within a given formal system, following the rules and axioms of that system
- The second incompleteness theorem reveals a gap between formal provability and truth in sufficiently strong systems containing arithmetic
- There can be true statements (such as the consistency statement) that are not formally provable within the system itself

- This limitation suggests that truth in mathematics extends beyond what can be captured by formal proof systems
- Gödel's incompleteness theorems demonstrate that no consistent formal system containing arithmetic can prove all arithmetical truths

Philosophical Implications

Challenge to Hilbert's Program

- Hilbert's program aimed to establish the consistency and completeness of mathematics using finitary methods within formal systems
- Gödel's incompleteness theorems, particularly the second theorem, posed a significant challenge to Hilbert's program
- The theorems showed that a consistent formal system containing arithmetic cannot prove its own consistency, contradicting Hilbert's goal
- This revelation led to a reassessment of the foundations of mathematics and the limitations of formal systems in achieving Hilbert's objectives

Metamathematics and Gödel's Theorems

- Metamathematics is the study of mathematics itself using mathematical methods, examining the properties and limitations of mathematical systems
- Gödel's incompleteness theorems are fundamental results in metamathematics, revealing inherent limitations of formal systems
- The theorems demonstrate the impossibility of a complete and consistent formal system that encompasses all of mathematics
- Metamathematical investigations, inspired by Gödel's work, have deepened our understanding of the nature and boundaries of mathematical knowledge
- The incompleteness theorems highlight the role of metamathematics in exploring the foundations and limitations of mathematical reasoning

Impact on the Foundations of Mathematics

- Gödel's incompleteness theorems contributed to a foundational crisis in mathematics in the early 20th century
- The realization that no consistent formal system could capture all mathematical truths challenged prevailing views on the certainty and completeness of mathematics
- The theorems led to a reevaluation of the role of formal systems and axiomatization in the foundations of mathematics
- Alternative approaches, such as intuitionism and constructivism, gained attention as potential ways to address the limitations exposed by Gödel's work
- The incompleteness theorems have had lasting implications for the philosophy of mathematics, prompting ongoing discussions on the nature of mathematical truth, proof, and the limits of formal reasoning

7.4 Philosophical and mathematical consequences

Gödel's Incompleteness Theorems shook the foundations of mathematics, revealing inherent limitations in formal systems. These groundbreaking results showed that no consistent system containing arithmetic can prove all true statements within itself, challenging notions of absolute mathematical truth.

The theorems have far-reaching consequences beyond math, influencing philosophy, computer science, and artificial intelligence. They raise profound questions about the nature of knowledge, the limits of human reasoning, and the boundaries of what can be formally proven or computed.

Philosophical Perspectives on Mathematics

Platonism and Mathematical Truth

- Platonism holds that mathematical objects exist independently of the human mind in an abstract realm
- Mathematical objects (numbers, sets, functions) are real entities that exist objectively and eternally
- Discovering mathematical truths involves accessing this abstract realm through reason and logic
- Mathematical truth is objective, absolute, and independent of human knowledge or beliefs

Formalism and Intuitionism

- Formalism views mathematics as a formal system of symbols and rules without inherent meaning
- Mathematical statements are meaningless formulas manipulated according to formal rules (axioms, inference rules)
- Intuitionism rejects the law of excluded middle and the existence of actual infinities
- Intuitionists require constructive proofs and view mathematics as a mental construction
- Formalists and intuitionists reject Platonic realism and the objective existence of mathematical objects

Limits and Incompleteness of Formal Systems

Gödel's Incompleteness Theorems

- Gödel's First Incompleteness Theorem states that any consistent formal system containing arithmetic is incomplete
- There will always be true statements within the system that cannot be proven using the system's axioms and rules
- Gödel's Second Incompleteness Theorem shows that a consistent formal system cannot prove its own consistency
- The theorems reveal inherent limitations in the expressive power of formal systems

Halting Problem and Incompleteness in Other Domains

- Turing's halting problem demonstrates the existence of undecidable problems in computability theory
- No algorithm can determine whether an arbitrary program will halt or run forever for all possible inputs

- Incompleteness and undecidability appear in various domains beyond mathematics (logic, computer science)
- The theorems have implications for the limits of knowledge and the boundaries of what can be formally proven

Broader Implications

Philosophical and Scientific Consequences

- Gödel's theorems challenge the foundations of mathematics and the notion of absolute mathematical truth
- They raise questions about the nature of mathematical knowledge and the limits of human reasoning
- The incompleteness results have implications for the philosophy of mind and artificial intelligence
- They suggest inherent limitations in the capabilities of formal systems, including the human mind
- The theorems have influenced debates in epistemology, metaphysics, and the philosophy of science

Impact on Mathematics and Logic

- Gödel's work transformed the field of mathematical logic and set theory
- The theorems led to the development of new areas of research (proof theory, model theory)
- They prompted a reevaluation of the role of axioms and the nature of mathematical proof
- The results highlighted the importance of metamathematics and the study of formal systems themselves
- Gödel's theorems continue to inspire research on the foundations and limitations of mathematics and logic

Unit 8 – Second–Order Logic and Higher–Order Logics

8.1 Syntax and semantics of second-order logic

Second-order logic takes first-order logic up a notch. It adds new variables and quantifiers that let us talk about predicates and functions, not just objects. This gives us more power to express complex ideas.

The syntax of second-order logic introduces predicate and function variables. We can quantify over these, making statements about all properties or relations. This expands what we can say and reason about in formal logic.

Syntax of Second-Order Logic

Extending First-Order Logic

- Second-order logic extends the syntax of first-order logic
- Introduces new types of variables and quantifiers
- Allows for quantification over predicates and functions
- Increases the expressive power of the language

Quantifiers and Variables

- Second-order quantifiers $\forall$ and $\exists$ range over predicates and functions
- Predicate variables represent properties or relations (P, Q, R)
- Function variables represent functions (f, g, h)
- Variables can be quantified to make statements about all predicates or functions

Constructing Well-Formed Formulas

- Well-formed formulas (wffs) are constructed using logical connectives and quantifiers
- Predicate variables can be applied to terms to form atomic formulas ($P(x), Q(a, b)$)
- Function variables can be applied to terms to form complex terms ($f(x), g(a, b)$)
- Quantifiers can bind predicate and function variables ($\forall P \exists x P(x), \exists f \forall x f(x) = x$)

Semantics of Second-Order Logic

Interpreting Second-Order Formulas

- Interpretation assigns meaning to the symbols in a second-order formula
- Domain of discourse specifies the objects that variables can refer to
- Predicate variables are assigned subsets of the domain or relations on the domain
- Function variables are assigned functions from the domain to itself

Henkin and Standard Semantics

- Henkin semantics allows the interpretation of predicate and function variables to be restricted
- Not all possible subsets or functions of the domain must be considered
- Standard semantics requires that all possible subsets and functions of the domain are included
- Standard semantics is more expressive but can lead to incompleteness

Satisfaction and Models

- A second-order formula is satisfied by an interpretation if it evaluates to true under that interpretation
- An interpretation that satisfies a formula is called a model of the formula
- A formula is valid if it is satisfied by every interpretation ($\forall P \forall x P(x) \rightarrow P(x)$)
- A formula is satisfiable if there exists an interpretation that satisfies it ($\exists P \forall x P(x)$)

8.2 Expressive power and limitations of second-order logic

Second-order logic packs a punch in the world of mathematical reasoning. It can express complex ideas like the natural numbers and real numbers uniquely, something first-order logic can't do. This added power comes with trade-offs though.

While second-order logic is more expressive, it loses some nice properties of first-order logic. It's not compact and doesn't have the Löwenheim-Skolem theorems. But it can categorically define important mathematical structures, making it a powerful tool for formalizing mathematics.

Fundamental Theorems

Completeness and Compactness

- Completeness theorem states that every valid formula in second-order logic is provable from the axioms and rules of inference
- Ensures that second-order logic is a sound and complete system
- Guarantees that any logical consequence of a set of axioms can be formally derived within the system
- Compactness theorem does not hold for second-order logic
- In first-order logic, compactness states that if every finite subset of a set of formulas is satisfiable, then the entire set is satisfiable
- Lack of compactness in second-order logic means that there can be infinite sets of formulas where every finite subset is satisfiable, but the entire set is not

Löwenheim-Skolem Theorems and Categoricity

- Löwenheim-Skolem theorems do not hold for second-order logic
- In first-order logic, these theorems state that if a theory has an infinite model, then it has models of every infinite cardinality
- Second-order logic allows for categorical axiomatizations, meaning a theory can have a unique model up to isomorphism

- Categoricity is possible in second-order logic
- A theory is categorical if it has a unique model up to isomorphism
- Second-order logic can express concepts like "the natural numbers" or "the real numbers" categorically
- Categoricity is not possible in first-order logic due to the Löwenheim-Skolem theorems

Arithmetic and Induction

Peano Arithmetic in Second-Order Logic

- Peano arithmetic can be fully axiomatized in second-order logic
- Includes axioms for the successor function, addition, multiplication, and the induction axiom
- Second-order induction axiom captures the full strength of mathematical induction
- Categorically defines the structure of the natural numbers
- Second-order Peano arithmetic is categorical
- Has a unique model up to isomorphism, the standard model of the natural numbers
- Avoids non-standard models that exist in first-order Peano arithmetic (models with "infinite" natural numbers)

Induction Axiom and Its Consequences

- Second-order induction axiom is more powerful than first-order induction
- Allows induction over predicates and sets, not just properties definable by formulas
- Enables proofs of statements that are not provable in first-order Peano arithmetic (Goodstein's theorem)
- Induction axiom implies the well-ordering of the natural numbers
- Every non-empty subset of the natural numbers has a least element
- Crucial for proving many fundamental properties of the natural numbers (every non-zero natural number has a unique predecessor)

Set Theory and Continuum Hypothesis

Expressive Power of Second-Order Logic

- Second-order logic can express many concepts not definable in first-order logic
- Finiteness, countability, uncountability, connectedness of graphs, well-foundedness
- Allows for categorical axiomatizations of structures like the real numbers or the power set of the natural numbers
- Second-order logic is powerful enough to express most of classical mathematics
- Can formalize set theory, analysis, topology, abstract algebra

- Provides a foundation for mathematics alternative to first-order set theory (ZFC)

Continuum Hypothesis and Its Independence

- Continuum hypothesis states that there is no set whose cardinality is strictly between that of the natural numbers and the real numbers
- Can be formulated in second-order logic using the power set operation and the concept of bijection
- Remains independent of the axioms of second-order ZFC (Zermelo-Fraenkel set theory with the axiom of choice)
- Independence of the continuum hypothesis from second-order ZFC
- Proven by Cohen's forcing method, building on Gödel's constructible universe
- Shows that the continuum hypothesis can neither be proved nor disproved from the axioms of second-order ZFC
- Highlights the limitations of second-order logic in resolving certain fundamental questions in set theory

8.3 Introduction to higher-order logics

Higher-order logics extend the expressive power of first-order logic. They allow quantification over predicates and functions, enabling more complex reasoning. This topic introduces key concepts and applications of higher-order logics in formal systems.

Higher-order logics find applications in theorem proving, program verification, and formal semantics. They provide a richer framework for expressing and analyzing complex mathematical and computational concepts, bridging the gap between formal logic and practical reasoning systems.

Type Systems

Foundations of Type Theory

- Type theory provides a formal framework for specifying and reasoning about the types of values and expressions in a programming language or logic system
- Simple type theory, developed by Alonzo Church, forms the basis for many modern type systems
- Introduces basic types (such as integers, booleans) and function types to construct more complex types
- Ensures that well-typed expressions cannot lead to runtime errors due to type mismatches (type safety)
- Type hierarchy organizes types into a hierarchical structure
- Allows subtypes to be used where supertypes are expected (subtyping)
- Enables polymorphism, where functions can operate on values of multiple types

Advanced Type System Features

- Dependent types allow types to depend on values
- Type of a value can be determined by the result of a computation

- Enables expressing precise properties and constraints on values within the type system itself
- Examples: Agda, Coq, and Idris programming languages
- Type inference algorithms automatically deduce the types of expressions without explicit type annotations
- Reduces the burden of manual type annotations while maintaining type safety
- Hindley-Milner type inference used in functional languages like Haskell and ML

Lambda Calculus and Functional Programming

Lambda Calculus as a Foundation

- Lambda calculus, developed by Alonzo Church, is a formal system for expressing computation based on function abstraction and application
- Provides a theoretical foundation for functional programming languages
- Functions are first-class citizens and can be passed as arguments, returned as values, and assigned to variables
- Encourages a declarative programming style focusing on what to compute rather than how to compute it
- Higher-order functions take other functions as arguments or return functions as results
- Enable powerful abstractions and code reuse
- Examples: map, filter, and reduce functions in functional programming languages

Polymorphism and Type Abstraction

- Polymorphism allows functions and data types to operate on values of multiple types
- Parametric polymorphism (generics) enables writing generic code that works uniformly across different types
- Example: a generic length function that works on lists of any type
- Ad-hoc polymorphism (overloading) allows functions to have different implementations for different types
- Example: arithmetic operators (+, -, *) can be overloaded to work on integers, floating-point numbers, and complex numbers
- Type abstraction hides the concrete implementation details of a type and provides a well-defined interface
- Allows modular programming and separates the concerns of implementation and usage
- Examples: abstract data types, modules, and interfaces in programming languages

8.4 Comparison of first-order, second-order, and higher-order logics

First-order, second-order, and higher-order logics form a hierarchy of increasing expressive power. Each level builds on the previous, allowing quantification over more complex entities and capturing more sophisticated mathematical concepts.

However, this increased expressiveness comes at a cost. While first-order logic has nice properties like completeness and compactness, higher-order logics sacrifice these for greater power. This trade-off shapes their use in different areas of mathematics and computer science.

Logical Properties

Expressive Power and Limitations

- First-order logic has limited expressiveness compared to higher-order logics
- Cannot quantify over predicates or functions, only variables
- Struggles to express concepts like infinity, finiteness, or connectedness
- Second-order logic extends first-order logic by allowing quantification over predicates and functions
- Can express properties of relations and functions directly (continuity)
- Captures important mathematical concepts first-order logic cannot (Peano arithmetic, real analysis)
- Higher-order logics further extend quantification to predicates and functions of higher types
- Even more expressive than second-order logic (can define truth predicates)
- Useful for formalizing mathematics and computer science (type theory, category theory)

Decidability and Completeness Properties

- First-order logic is semi-decidable
- Valid formulas are recursively enumerable, but invalid ones may not be
- Has a complete proof system (all valid sentences are provable)
- Second-order logic and higher-order logics are not decidable in general
- Validity is not recursively enumerable due to higher expressiveness
- Do not have complete proof systems (Gödel's incompleteness theorems)
- Restricted fragments of higher-order logics can be decidable (propositional, monadic)

Compactness and Löwenheim-Skolem Properties

- First-order logic has the compactness property
- If every finite subset of a set of sentences has a model, the whole set has a model
- Useful for constructing models (ultraproducts, nonstandard analysis)
- First-order logic satisfies the Löwenheim-Skolem theorems
- If a sentence has an infinite model, it has models of all infinite cardinalities
- Leads to non-categoricity of first-order theories (Peano arithmetic has nonstandard models)
- Second-order logic and higher-order logics do not have compactness or Löwenheim-Skolem properties in general

- Due to the expressive power to characterize infinite structures up to isomorphism (standard model of arithmetic)

Theoretical Foundations

Model-Theoretic Semantics

- First-order logic has a well-developed model theory based on structures and interpretations
- Structures consist of a domain of discourse and interpretations of function and predicate symbols
- Satisfaction relation defined inductively on the structure of formulas
- Second-order logic extends first-order semantics with second-order variables and quantifiers
- Variables range over subsets and functions on the domain
- Full semantics vs. Henkin semantics (nonstandard models)
- Higher-order logics have a more complex model theory due to the hierarchy of types
- Often interpreted in typed structures or extensional models (all functions and predicates included)
- Can also have intensional or nonstandard semantics (general models)

Proof-Theoretic Aspects

- First-order logic has a variety of complete proof systems
- Hilbert-style systems, natural deduction, sequent calculus
- Proofs are finite syntactic objects, can be checked algorithmically
- Second-order logic does not have a complete proof system with standard semantics
- Categorical theories like Peano arithmetic are not recursively axiomatizable
- Henkin semantics allow for a wider class of models and complete proof systems
- Higher-order logics have proof systems based on typed lambda calculi
- Curry-Howard correspondence relates proofs and programs (dependent type theory)
- Proof assistants implement higher-order proof systems (Coq, Isabelle/HOL)

Computational Aspects

Computational Complexity and Decidability

- First-order logic has many decidable fragments used in computer science applications
- Propositional logic is decidable and NP-complete (SAT solvers)
- Monadic first-order logic and two-variable logic are decidable (description logics, modal logics)
- Second-order logic is not decidable, but has decidable fragments
- Monadic second-order logic over finite structures (WS1S) used in verification (Buchi automata)
- Weak second-order logic restricts quantification to finite sets (decidable)

- Higher-order logics are generally not decidable, but have important computable fragments
- Higher-order recursive functions (System T) and polymorphic lambda calculus (System F)
- Constructive type theories used in proof assistants (Calculus of Constructions)

Abstraction Levels and Applications

- First-order logic is used in many areas of computer science for specification and modeling
- Database query languages (SQL), knowledge representation (description logics)
- Program verification (Hoare logic), model checking (temporal logics)
- Second-order logic can express important program properties not captured by first-order logic
- Transitive closure, graph connectivity (monadic second-order logic)
- Program schemes and software synthesis (weak second-order logic)
- Higher-order logics are used in programming language theory and formal verification
- Type systems and semantics of functional programming languages (polymorphic lambda calculus)
- Hardware and software verification using proof assistants (Coq, Isabelle/HOL)
- Automated theorem proving in higher-order logic (TPS, LEO)

Unit 9 – Intuitionistic Logic and Constructive Proofs

9.1 Foundations of intuitionistic logic

Intuitionistic logic challenges classical logic's assumptions, rejecting the law of excluded middle and double negation elimination. It's rooted in constructivism, which demands mathematical objects and proofs be constructible, not just abstract concepts.

The Brouwer-Heyting-Kolmogorov interpretation gives intuitionistic logic meaning through proofs. It defines logical connectives based on how propositions are proven, emphasizing the link between truth and provability in constructive mathematics.

Constructivism and Intuitionistic Logic

Philosophical Foundations of Constructivism

- Constructivism holds that mathematical objects and proofs must be constructible
- Rejects the existence of abstract mathematical objects independent of the mind
- Contrasts with Platonism which asserts the existence of abstract mathematical objects
- Constructive proofs provide explicit procedures for constructing mathematical objects (algorithms, finite processes)

Intuitionistic Logic and Classical Logic

- Intuitionistic logic rejects certain principles of classical logic
- Law of excluded middle ($P \vee \neg P$) not accepted as universally valid
- Asserts that every proposition is either true or false
- Constructivists require proof of P or proof of $\neg P$ to accept the disjunction
- Double negation elimination ($\neg \neg P \rightarrow P$) not accepted as valid
- In classical logic, if a proposition is not false, it is considered true
- Intuitionistic logic distinguishes between "not false" and "provably true"
- Intuitionistic truth requires explicit proof or construction of the object in question
- Truth is equated with provability or constructibility
- Contrasts with classical notion of truth as correspondence to objective reality

Semantics and Interpretation

Brouwer-Heyting-Kolmogorov (BHK) Interpretation

- Provides an informal semantics for intuitionistic logic based on the notion of proof
- Interprets logical connectives in terms of the proofs of the propositions involved

- $P \wedge Q$ is proven if both P and Q are proven
- $P \vee Q$ is proven if either P or Q is proven
- $P \rightarrow Q$ is proven if there is a construction that transforms any proof of P into a proof of Q
- $\neg P$ is proven if there is a construction that transforms any hypothetical proof of P into a contradiction
- Gives intuitionistic logic a constructive meaning in terms of proofs and constructions

Proof-Theoretic Semantics and Knowability

- Proof-theoretic semantics assigns meaning to propositions in terms of the proofs that establish them
- The meaning of a proposition is determined by the conditions under which it can be proven
- Contrasts with model-theoretic semantics which assigns meaning in terms of truth conditions
- Knowability principle states that every true proposition is provable in principle
- $P \rightarrow \Diamond \Box P$ (if P is true, then it is possible to prove P)
- Captures the constructivist idea that truth and provability are closely linked
- Implies that there are no unknowable or unprovable truths in constructive mathematics
- Intuitionistic logic can be seen as formalizing the notion of constructive provability
- Proofs are understood as constructions or algorithms that establish the truth of propositions
- The meaning of logical connectives is given by the BHK interpretation in terms of proofs

9.2 Syntax and semantics of intuitionistic logic

Intuitionistic logic rejects certain classical principles, like the law of excluded middle. It demands constructive proofs, requiring concrete evidence for each logical step. This approach aligns with the chapter's focus on constructive reasoning in mathematics.

The syntax and semantics of intuitionistic logic differ from classical logic. Kripke semantics and Heyting algebras provide frameworks for understanding intuitionistic logic, while properties like disjunction and existence reflect its constructive nature.

Intuitionistic Propositional and Predicate Logic

Intuitionistic Propositional Calculus

- Formal system of logic based on constructive principles
- Differs from classical propositional calculus by rejecting the law of excluded middle ($A \vee \neg A$) and double negation elimination ($\neg \neg A \rightarrow A$)
- Proofs in intuitionistic propositional calculus require constructive evidence for each step
- Connectives in intuitionistic propositional calculus include conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), and negation ($\neg$)

Intuitionistic Predicate Logic

- Extension of intuitionistic propositional calculus to include quantifiers ($\forall$ and $\exists$)

- Allows reasoning about properties of objects and relationships between them
- Intuitionistic predicate logic is more expressive than intuitionistic propositional calculus
- Proofs in intuitionistic predicate logic require constructive evidence for quantified statements (e.g., to prove $\forall x P(x)$, one must provide a method for constructing a proof of $P(x)$ for any given x)

Intuitionistic Implication and Negation

- Intuitionistic implication ($A \rightarrow B$) asserts the existence of a constructive proof that transforms a proof of A into a proof of B
- Intuitionistic negation ($\neg A$) asserts the existence of a constructive proof that derives a contradiction from A
- Intuitionistic implication and negation are weaker than their classical counterparts
- Example: In classical logic, $\neg\neg A \rightarrow A$ is a tautology, but in intuitionistic logic, it is not valid without additional assumptions

Semantics and Algebraic Structures

Kripke Semantics

- Provides a semantics for intuitionistic logic using partially ordered sets (posets) of possible worlds or states of knowledge
- Each world in a Kripke model is associated with a set of propositions considered true at that world
- Accessibility relation between worlds represents the growth of knowledge or the availability of new information
- A proposition is true in a Kripke model if it is true in all accessible worlds from the current world
- Example: In a Kripke model with worlds w_1 and w_2 , where $w_1 \leq w_2$, if a proposition P is true at w_1 , it must also be true at w_2

Heyting Algebra

- Algebraic structure that provides a semantics for intuitionistic logic
- Generalizes the notion of a Boolean algebra by replacing the complement operation with a pseudo-complement (intuitionistic negation)
- Elements of a Heyting algebra represent propositions, and the operations correspond to intuitionistic connectives
- The ordering relation in a Heyting algebra represents implication: $a \leq b$ means $a \rightarrow b$ is true
- Example: The set of open subsets of a topological space forms a Heyting algebra, with intersection as conjunction, union as disjunction, and interior of the complement as pseudo-complement

Logical Properties

Disjunction Property

- In intuitionistic logic, if a disjunction ($A \vee B$) is provable, then either A is provable or B is provable

- Reflects the constructive nature of intuitionistic logic: a proof of a disjunction must provide evidence for at least one of the disjuncts
- Contrasts with classical logic, where a disjunction can be true without either disjunct being true (e.g., $P \vee \neg P$ is a tautology in classical logic)
- Example: If a theorem in intuitionistic logic states $\forall x(P(x) \vee Q(x))$, then for any given x , we can constructively find a proof of either $P(x)$ or $Q(x)$

Existence Property

- In intuitionistic predicate logic, if an existential statement ($\exists x P(x)$) is provable, then there exists a specific term t such that $P(t)$ is provable
- Reflects the constructive nature of intuitionistic logic: a proof of an existential statement must provide a witness for the existential quantifier
- Contrasts with classical logic, where an existential statement can be true without a specific witness being known
- Example: If a theorem in intuitionistic predicate logic states $\exists x \forall y R(x, y)$, then there exists a specific term t such that $\forall y R(t, y)$ is provable

9.3 Proof systems for intuitionistic logic

Intuitionistic logic uses special proof systems to reason without assuming every statement is true or false. These systems, like natural deduction and sequent calculus, have unique rules that reflect the constructive nature of intuitionistic reasoning.

These proof systems have important properties like normalization and cut elimination. They also connect to type theory through the Curry-Howard correspondence, linking logical proofs to computer programs. This connection has big implications for both math and computer science.

Intuitionistic Proof Systems

Natural Deduction and Sequent Calculus

- Natural deduction is a formal proof system that closely resembles the informal proofs used in mathematical reasoning
- Consists of a set of inference rules that allow deriving conclusions from premises
- Proofs are constructed as trees, with the conclusion at the root and the premises at the leaves
- Intuitionistic natural deduction excludes the law of excluded middle and double negation elimination (classical logic rules)
- Sequent calculus is another formal proof system for intuitionistic logic
- Operates on sequents, which are expressions of the form $\Gamma \vdash \Delta$, where Γ and Δ are sequences of formulas
- Γ represents the assumptions (antecedent) and Δ represents the conclusions (succedent)
- Inference rules in sequent calculus manipulate sequents to derive new sequents
- Intuitionistic sequent calculus restricts the succedent to at most one formula

Intuitionistic Tableau Method

- The intuitionistic tableau method is a proof procedure for intuitionistic logic based on the idea of refutation
- To prove a formula φ , the tableau method attempts to construct a countermodel for $\neg\varphi$
- If the construction of the countermodel fails, then φ is intuitionistically valid
- Tableaux are trees labeled with formulas, where each branch represents a possible model
- The tableau rules decompose complex formulas into simpler ones, generating new branches
- Closure rules identify contradictory branches, which are then closed (marked with a cross)
- The intuitionistic tableau method differs from the classical tableau method in the rules for negation and implication
- The intuitionistic rules ensure that the constructed models are intuitionistically valid

Properties of Intuitionistic Proofs

Normalization and Cut Elimination

- Normalization is a process of transforming proofs into a standard form (normal form)
- In natural deduction, a proof is in normal form if it contains no "detours" (redundant steps)
- Normalization eliminates the detours by applying reduction rules to the proof tree
- The normalization theorem states that every proof in intuitionistic natural deduction can be reduced to a normal form
- Cut elimination is a fundamental property of sequent calculus
- The cut rule allows the use of an intermediate lemma in a proof: if $\Gamma \vdash \varphi$ and $\varphi, \Delta \vdash \psi$, then $\Gamma, \Delta \vdash \psi$
- The cut elimination theorem states that any proof in intuitionistic sequent calculus can be transformed into a proof without the cut rule
- Cut-free proofs have a simpler structure and are easier to reason about

Intuitionistic Validity

- Intuitionistic validity is a notion of logical consequence in intuitionistic logic
- A formula φ is intuitionistically valid if it is provable in an intuitionistic proof system (e.g., natural deduction or sequent calculus)
- Intuitionistic validity is weaker than classical validity: every intuitionistically valid formula is classically valid, but not vice versa
- The disjunction and existence properties are characteristic of intuitionistic logic
- The disjunction property states that if $\varphi \vee \psi$ is intuitionistically provable, then either φ or ψ is intuitionistically provable
- The existence property states that if $\exists x\varphi(x)$ is intuitionistically provable, then there is a term t such that $\varphi(t)$ is intuitionistically provable

Connections to Type Theory

Curry-Howard Correspondence

- The Curry-Howard correspondence, also known as the proofs-as-programs interpretation, establishes a deep connection between intuitionistic logic and type theory
- Propositions in intuitionistic logic correspond to types in a typed lambda calculus
- Proofs of propositions correspond to lambda terms (programs) of the corresponding types
- The inference rules of intuitionistic natural deduction correspond to type constructors and term constructors in the lambda calculus
- The correspondence allows the interpretation of logical operations as programming constructs
- Conjunction ($\wedge$) corresponds to the product type ($\times$), where a proof of $\varphi \wedge \psi$ is a pair of proofs of φ and ψ
- Implication ($\rightarrow$) corresponds to the function type ($\rightarrow$), where a proof of $\varphi \rightarrow \psi$ is a function that maps proofs of φ to proofs of ψ
- Disjunction ($\vee$) corresponds to the sum type ($+$), where a proof of $\varphi \vee \psi$ is either a proof of φ or a proof of ψ , tagged with a label indicating the choice
- The Curry-Howard correspondence has far-reaching consequences in both logic and computer science
- It provides a foundation for constructive mathematics and the development of proof assistants (e.g., Coq, Agda)
- It enables the extraction of certified programs from proofs, ensuring their correctness with respect to the specified properties

9.4 Relationships between classical and intuitionistic logic

Intuitionistic logic and classical logic have a complex relationship. Translations like Gödel-Gentzen and double-negation help bridge the gap, allowing classical proofs to be interpreted intuitionistically. This connection reveals the underlying constructive content in classical reasoning.

Glivenko's theorem and conservativity results further illuminate the similarities and differences between these logical systems. Understanding these relationships is crucial for grasping the foundations of constructive mathematics and its applications in computer science and formal verification.

Translations between Classical and Intuitionistic Logic

Negative Translations

- Gödel-Gentzen negative translation embeds classical logic into intuitionistic logic by adding double negations to atomic formulas and implications
- Preserves provability from classical to intuitionistic logic
- Allows classical proofs to be interpreted intuitionistically
- Double-negation translation is a simplified version of the Gödel-Gentzen translation

- Adds double negations only to atomic formulas
- Results in a formula that is classically equivalent to the original

Glivenko's Theorem

- Glivenko's theorem states that if a propositional formula is provable in classical logic, then its double negation is provable in intuitionistic logic
- Establishes a relationship between classical and intuitionistic provability for propositional formulas
- Allows classical propositional proofs to be converted into intuitionistic proofs of the double negation
- The theorem does not extend to first-order logic, as there are classically valid formulas whose double negation is not intuitionistically provable (Peirce's law)

Relationships and Conservativity

Conservativity Results

- Intuitionistic logic is a conservative extension of minimal logic, which lacks the ex falso quodlibet principle ($\perp \rightarrow A$)
- Any formula provable in intuitionistic logic without using ex falso is also provable in minimal logic
- Classical logic is not conservative over intuitionistic logic, as there are formulas provable in classical logic that are not provable intuitionistically (law of excluded middle)
- However, classical logic is conservative over intuitionistic logic for Π_2^0 formulas (universal quantifiers followed by existential quantifiers and a quantifier-free matrix)
- If a Π_2^0 formula is provable in classical logic, it is also provable intuitionistically

Intuitionistic Models of Classical Theories

- Intuitionistic models can be used to study the relationship between classical and intuitionistic theories
- The double-negation translation allows classical theories to be interpreted in intuitionistic models
- If a theory T is consistent in classical logic, then its double-negation translation is consistent in intuitionistic logic
- Kripke models and topological models provide intuitionistic semantics for classical theories
- Can be used to establish independence results and study the intuitionistic content of classical theories

Foundations and Applications

Constructive Mathematics

- Constructive mathematics is a foundation of mathematics that relies on intuitionistic logic and constructive proofs
- Emphasizes the computational content of proofs and the explicit construction of mathematical objects
- Avoids non-constructive principles such as the law of excluded middle and the axiom of choice
- Intuitionistic type theory (Martin-Löf type theory) provides a foundation for constructive mathematics

- Incorporates intuitionistic logic and allows for the development of constructive proofs and programs
- Constructive analysis and constructive algebra are subfields of mathematics developed using intuitionistic logic
- Constructive analysis focuses on computable real numbers and functions
- Constructive algebra studies algebraic structures with computational content
- Constructive proofs have applications in computer science, such as program extraction and verified software development
- Proofs in intuitionistic logic can be translated into functional programs (Curry-Howard correspondence)
- Constructive proofs provide a basis for formal verification and dependently-typed programming

Unit 10 – Modal Logic and Kripke Semantics

10.1 Syntax and semantics of modal logic

Modal logic adds necessity and possibility operators to propositional logic, allowing us to reason about what must be true and what could be true. This powerful extension lets us model and analyze concepts like knowledge, belief, and time in formal systems.

Possible worlds semantics provides the foundation for interpreting modal logic formulas. By considering different possible scenarios and their relationships, we can define precise truth conditions for modal statements and explore their logical properties.

Modal Operators and Formulas

Modal Operators and Propositional Variables

- Modal operators express modality in modal logic
- Modality refers to the way in which a proposition can be true or false
- Two main modal operators necessity ($\Box$) and possibility ($\Diamond$)
- Necessity ($\Box$) asserts that a proposition is true in all accessible possible worlds
- Possibility ($\Diamond$) asserts that a proposition is true in at least one accessible possible world
- Propositional variables (p, q, r) represent atomic propositions in modal logic formulas
- Atomic propositions are the basic building blocks and cannot be further broken down

Well-formed Formulas in Modal Logic

- Well-formed formulas (wffs) are syntactically correct expressions in modal logic
- Follow specific rules for combining propositional variables, logical connectives, and modal operators
- Formation rules for wffs in modal logic: 1. Every propositional variable (p, q, r) is a wff 2. If ϕ is a wff, then $\neg\phi$ is also a wff (negation) 3. If ϕ and ψ are wffs, then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$ are also wffs (conjunction, disjunction, implication, and equivalence) 4. If ϕ is a wff, then $\Box\phi$ and $\Diamond\phi$ are also wffs (modal operators)
- Examples of well-formed formulas:
 - $\Box p$ (it is necessary that p)
 - $\Diamond(p \vee q)$ (it is possible that either p or q)

Possible Worlds Semantics

Possible Worlds and Accessibility Relation

- Possible worlds semantics provides a framework for interpreting modal logic formulas

- A possible world represents a complete and consistent way the world could be
- Different possible worlds can have different truth values for the same proposition
- Accessibility relation (R) is a binary relation between possible worlds
- If world w_1 is accessible from world w_0 ($w_0 R w_1$), then the truth values of propositions in w_1 are relevant to the truth values of modal formulas in w_0
- Accessibility relation determines which worlds are considered when evaluating modal formulas

Truth Conditions for Modal Formulas

- Truth conditions define when a modal formula is true or false in a given world
- For necessity ($\Box\phi$):
 - $\Box\phi$ is true in a world w if and only if ϕ is true in all worlds accessible from w
 - Formally: $w \models \Box\phi$ iff for all w' such that wRw' , $w' \models \phi$
- For possibility ($\Diamond\phi$):
 - $\Diamond\phi$ is true in a world w if and only if ϕ is true in at least one world accessible from w
 - Formally: $w \models \Diamond\phi$ iff there exists a w' such that wRw' and $w' \models \phi$
- Examples:
 - If $\Box p$ is true in world w_0 , then p must be true in all worlds accessible from w_0
 - If $\Diamond q$ is true in world w_0 , then q must be true in at least one world accessible from w_0

Logical Properties

Validity and Satisfiability in Modal Logic

- Validity and satisfiability are important logical properties in modal logic
- A modal formula ϕ is valid if it is true in all possible worlds under any interpretation
- Formally: $\models \phi$ iff for all models M and all worlds w in M , $M, w \models \phi$
- Valid formulas are tautologies in modal logic (e.g., $\Box(p \vee \neg p)$)
- A modal formula ϕ is satisfiable if there exists at least one model and one world where it is true
- Formally: ϕ is satisfiable iff there exists a model M and a world w in M such that $M, w \models \phi$
- Satisfiable formulas are consistent and can be true in some models (e.g., $\Diamond p$)
- Validity and satisfiability are related concepts:
 - A formula is valid if and only if its negation is not satisfiable
 - A formula is satisfiable if and only if its negation is not valid
- Examples:
 - $\Box(p \rightarrow p)$ is a valid formula because it is true in all possible worlds

- $\Diamond(p \wedge \neg p)$ is not satisfiable because it cannot be true in any possible world

10.2 Kripke frames and models

Kripke frames and models are the backbone of modal logic, providing a way to visualize and reason about possibility and necessity. They consist of possible worlds connected by accessibility relations, allowing us to interpret modal statements across different scenarios.

These structures help us understand various modal logics by defining different properties of accessibility relations. Reflexivity, symmetry, transitivity, and other characteristics correspond to specific axioms, shaping how we interpret necessity and possibility in different contexts.

Kripke Frames and Models

Defining Kripke Frames

- Kripke frame F is a pair (W, R) where W is a non-empty set of possible worlds and R is a binary relation on W called the accessibility relation
- W represents the set of all conceivable scenarios or states of affairs in the context of modal logic
- Each element $w \in W$ is a possible world that captures a specific configuration of propositions being true or false
- $R \subseteq W \times W$ determines which worlds are considered accessible from each world in the frame
- If $(w, v) \in R$, then world v is accessible from world w , meaning that if a proposition is necessary at w , it must be true at v

Extending Frames to Models

- Kripke model M is a triple (W, R, V) where (W, R) is a Kripke frame and V is a valuation function
- $V : Prop \rightarrow 2^W$ assigns to each propositional variable $p \in Prop$ the set of worlds where p is true
- For each $p \in Prop$ and $w \in W$, $w \in V(p)$ means that p is true at world w in the model M
- Truth of modal formulas at a world in a Kripke model depends on both the valuation and the accessibility relation
- Example: $M, w \models \Box \phi$ iff for all v such that $(w, v) \in R$, $M, v \models \phi$, meaning ϕ is necessary at w if it is true at all accessible worlds

Interpreting Accessibility

- Accessibility relation R captures the notion of relative possibility between worlds in a Kripke frame or model
- Different interpretations of R correspond to different modal logics and their associated axioms
- In epistemic logic, $(w, v) \in R$ means that world v is considered possible from the perspective of an agent's knowledge at world w
- In deontic logic, $(w, v) \in R$ means that world v is considered ideal or permissible according to the norms or obligations at world w

- In temporal logic, $(w, v) \in R$ means that world v is a future state reachable from world w , capturing the flow of time

Properties of Accessibility Relations

Reflexivity and Symmetry

- Reflexivity: $\forall w \in W : (w, w) \in R$, meaning each world is accessible from itself
- Reflexivity corresponds to the modal axiom $\Box \phi \rightarrow \phi$ (axiom T), capturing the idea that necessity implies truth
- Example: In epistemic logic, reflexivity means that if an agent knows ϕ , then ϕ is actually true
- Symmetry: $\forall w, v \in W : (w, v) \in R \Rightarrow (v, w) \in R$, meaning accessibility is bidirectional
- Symmetry corresponds to the modal axiom $\phi \rightarrow \Box \Diamond \phi$ (axiom B), capturing the idea that truth implies possibility of necessity
- Example: In epistemic logic, symmetry means that if ϕ is considered possible by an agent, then the agent considers it possible to know ϕ

Transitivity and Euclideaness

- Transitivity: $\forall w, v, u \in W : (w, v) \in R \wedge (v, u) \in R \Rightarrow (w, u) \in R$, meaning accessibility is transitive
- Transitivity corresponds to the modal axiom $\Box \phi \rightarrow \Box \Box \phi$ (axiom 4), capturing the idea that necessity is idempotent
- Example: In epistemic logic, transitivity means that if an agent knows that they know ϕ , then they know ϕ
- Euclidean property: $\forall w, v, u \in W : (w, v) \in R \wedge (w, u) \in R \Rightarrow (v, u) \in R$, meaning worlds accessible from the same world are accessible from each other
- Euclideaness corresponds to the modal axiom $\Diamond \phi \rightarrow \Box \Diamond \phi$ (axiom 5), capturing the idea that possibility of necessity implies necessity of possibility
- Example: In epistemic logic, Euclideaness means that if an agent considers ϕ possible and ψ possible, then in situations where ϕ is true, the agent must also consider ψ possible

Seriality and Combinations

- Seriality: $\forall w \in W, \exists v \in W : (w, v) \in R$, meaning each world has at least one accessible world
- Seriality corresponds to the modal axiom $\Box \phi \rightarrow \Diamond \phi$ (axiom D), capturing the idea that necessity implies possibility
- Example: In deontic logic, seriality means that for each situation, there is at least one ideal or permissible situation
- Combining properties of accessibility relations leads to different modal logics with specific axiomatizations
- Example: Modal logic S4 has a reflexive and transitive accessibility relation, characterized by axioms K, T, and 4

- Example: Modal logic S5 has a reflexive, symmetric, and transitive accessibility relation, characterized by axioms K, T, B, 4, and 5

10.3 Proof systems for modal logic

Modal logic proof systems build on propositional logic, adding axioms and rules for necessity and possibility operators. These systems, like K, T, S4, and S5, capture different notions of necessity and possibility in formal reasoning.

Proof systems for modal logic are essential for establishing the validity of modal arguments. They provide a framework for deriving theorems and exploring the relationships between different modal concepts, forming the backbone of modal reasoning.

Axiomatization of Modal Logics

Axiom K and Necessitation Rule

- Axiom K states if a proposition ϕ is true, then it is necessarily possible, denoted as $\Box\Diamond\phi$
- Axiom K is a fundamental axiom in modal logic that relates necessity and possibility
- Necessitation rule infers the necessity of a proposition from its truth, formally if $\vdash \phi$ then $\vdash \Box\phi$
- Necessitation rule allows deriving necessary truths from theorems of propositional logic

Distribution Axiom and Normal Modal Logics

- Distribution axiom, also known as Axiom K, states $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$
- Distribution axiom relates the modal operator $\Box$ with implication $\rightarrow$
- Distribution axiom ensures the modal operator $\Box$ distributes over implication
- Normal modal logics include Axiom K and the Necessitation rule as core axioms
- Normal modal logics form a class of modal logics with desirable properties (soundness, completeness)

Common Modal Systems

System K and System T

- System K is the minimal normal modal logic containing only Axiom K and the Necessitation rule
- System K serves as the foundation for other stronger modal logics
- System T extends System K by adding the axiom $\Box\phi \rightarrow \phi$, known as the reflexivity axiom
- Reflexivity axiom states if a proposition is necessary, then it is true

System S4 and System S5

- System S4 extends System T by adding the axiom $\Box\phi \rightarrow \Box\Box\phi$, known as the transitivity axiom
- Transitivity axiom states if a proposition is necessary, then its necessity is also necessary
- System S5 extends System S4 by adding the axiom $\Diamond\phi \rightarrow \Box\Diamond\phi$, known as the symmetry axiom

- Symmetry axiom states if a proposition is possible, then it is necessarily possible
- System S5 is one of the strongest and most studied modal logics

Metatheoretic Properties

Soundness

- Soundness is a metatheoretic property that ensures the consistency of a proof system
- A modal logic is sound if every provable formula is valid in the corresponding semantics (Kripke frames)
- Soundness guarantees that the axioms and inference rules of a modal logic do not lead to contradictions
- Proving soundness involves showing that each axiom is valid and inference rules preserve validity

Completeness

- Completeness is a metatheoretic property that ensures the adequacy of a proof system
- A modal logic is complete if every valid formula in the corresponding semantics (Kripke frames) is provable
- Completeness guarantees that all valid formulas can be derived using the axioms and inference rules
- Proving completeness often involves constructing a canonical model that satisfies all consistent formulas
- Completeness is a stronger property than soundness and is crucial for automated theorem proving

10.4 Applications of modal logic

Modal logic extends classical logic, allowing us to reason about different modes of truth. It's a powerful tool for analyzing knowledge, belief, time, and necessity in various domains. From epistemic logic to deontic and temporal logic, these systems provide frameworks for complex reasoning.

In computer science, modal logic finds applications in multi-agent systems, artificial intelligence, and formal methods. It's used to model agent knowledge, verify software properties, and ensure system correctness. These applications showcase modal logic's versatility and practical importance.

Types of Modal Logic

Epistemic Logic

- Epistemic logic deals with the formalization of knowledge and belief
- Introduces modal operators such as "K" for knowledge and "B" for belief
- Allows reasoning about what agents know or believe in a given situation
- Can be used to model and analyze knowledge-based systems (expert systems, decision support systems)
- Helps in understanding the distribution of knowledge among multiple agents
- Addresses concepts like common knowledge, distributed knowledge, and reasoning about knowledge

Deontic and Temporal Logic

- Deontic logic is concerned with the formalization of normative concepts such as obligation, permission, and prohibition
- Introduces modal operators like "O" for obligation, "P" for permission, and "F" for prohibition
- Useful for modeling and reasoning about legal systems, contracts, and ethical principles
- Temporal logic deals with the representation and reasoning about time and temporal properties
- Introduces temporal operators such as "G" for always, "F" for eventually, "X" for next, and "U" for until
- Allows expressing and reasoning about properties that change over time (safety properties, liveness properties)
- Combines well with other modalities to express complex temporal relationships and constraints

Alethic Modalities

- Alethic modalities deal with the concepts of necessity, possibility, and impossibility
- Introduces modal operators such as " $\Box$ " (box) for necessity and " $\Diamond$ " (diamond) for possibility
- Expresses the idea that certain propositions are necessarily true or possibly true
- Can be used to formalize and reason about metaphysical concepts and philosophical arguments
- Provides a foundation for other types of modal logics by defining the basic modal operators and their properties
- Allows exploring the relationships between necessary truths, possible truths, and logical consequences

Applications in Computer Science

Multi-Agent Systems and AI

- Modal logic is extensively used in the design and analysis of multi-agent systems
- Allows reasoning about the knowledge, beliefs, and intentions of multiple agents
- Helps in modeling communication and coordination protocols among agents
- Facilitates the specification and verification of properties related to agent interaction and collaboration
- Finds applications in areas such as distributed artificial intelligence, game theory, and social choice theory
- Used in the development of intelligent agents and multi-agent planning algorithms

Program Verification and Formal Methods

- Modal logic is employed in the field of program verification and formal methods
- Helps in specifying and verifying properties of software systems
- Temporal logic, in particular, is used to express and reason about the behavior of concurrent and reactive systems

- Allows formulating properties such as safety (nothing bad happens), liveness (something good eventually happens), and fairness
- Used in model checking techniques to automatically verify whether a system satisfies a given property
- Enables the formal verification of critical systems (air traffic control systems, medical devices) to ensure their correctness and reliability

Unit 11 – Proof Theory and Foundations of Mathematics

11.1 Hilbert's program and its legacy

Hilbert's program aimed to secure math's foundations using finitistic methods and formal systems. It sought to prove consistency, completeness, and decidability for all of mathematics through axiomatization and formalization.

Gödel's incompleteness theorems showed inherent limitations in formal systems, undermining Hilbert's goals. While the program didn't fully succeed, it deeply influenced mathematical logic and sparked the development of proof theory.

Hilbert's Program and Formalism

Hilbert's Vision for Mathematics

- Hilbert's program aimed to provide a secure foundation for all of mathematics using finitistic methods
- Formalism views mathematics as the manipulation of meaningless symbols according to explicit rules
- Finitistic methods rely only on a finite number of objects and processes that can be concretely realized (finite sequences of symbols)
- Metamathematics studies mathematical proofs and theories as mathematical objects themselves

Formalization and Axiomatization

- Hilbert sought to formalize all of mathematics in axiomatic systems
- Formal systems consist of explicit axioms and rules of inference from which theorems can be mechanically derived
- Axiomatization allows mathematical theories to be studied abstractly without reference to any intended interpretation
- Formal proofs are finite sequences of formulas, each either an axiom or derived from previous formulas by a rule of inference

Key Goals of Hilbert's Program

Consistency and Completeness

- Consistency means a formal system cannot prove both a statement and its negation
- Proving consistency of a system would show it is free of contradictions
- Completeness means every true statement expressible in the system can be formally proved within the system
- Hilbert believed consistency and completeness could be established for formalized mathematics using finitistic methods

Decidability and the Entscheidungsproblem

- Decidability asks whether there is an effective procedure to determine if any given mathematical statement is provable
- The Entscheidungsproblem (decision problem) sought an algorithm to decide the truth or falsity of any statement in first-order logic
- Solving the Entscheidungsproblem would mean mathematics could be reduced to mechanical computation
- Establishing decidability was a key aim of Hilbert's program to secure the foundations of mathematics

Challenges to Hilbert's Program

Gödel's Incompleteness Theorems

- Gödel's first incompleteness theorem showed any consistent formal system containing arithmetic is incomplete
- There will always be true statements that cannot be proved within the system itself
- Gödel's second incompleteness theorem showed the consistency of a formal system cannot be proved within the system
- These results undermined Hilbert's goals of completeness and proving consistency using finitistic methods

The Foundational Crisis and Aftermath

- Gödel's theorems triggered a foundational crisis in mathematics in the early 20th century
- The realization that formal systems have inherent limitations cast doubt on the philosophical basis of Hilbert's program
- While Hilbert's specific goals were not achieved, his ideas deeply influenced mathematical logic and metamathematics
- Proof theory emerged from Hilbert's work as the study of formal proofs and continues to be an active area of research today

11.2 Proof-theoretic reductions and ordinal analysis

Proof-theoretic reductions and ordinal analysis are powerful tools for comparing formal systems. They help us understand the relative strength and consistency of different mathematical theories, providing insights into the foundations of math.

These techniques use ordinal numbers and transfinite induction to measure proof complexity. By assigning ordinals to systems, we can create a hierarchy of mathematical theories based on their power and consistency strength.

Proof-Theoretic Reductions and Ordinals

Comparing Formal Systems

- Proof-theoretic reduction compares the strength of formal systems by translating proofs from one system to another
- Involves defining a translation function that maps formulas and proofs between the systems
- If every theorem in system A can be translated into a theorem in system B, then A is considered proof-theoretically reducible to B
- Allows for establishing relative consistency and strength of different formal systems (Peano arithmetic, Zermelo-Fraenkel set theory)

Ordinal Analysis and Proof-Theoretic Strength

- Ordinal analysis assigns ordinal numbers to formal systems to measure their proof-theoretic strength
- Associates a proof-theoretic ordinal to each system, representing the length of the longest possible proof in that system
- Higher proof-theoretic ordinals indicate stronger systems capable of proving more complex statements
- Ordinal analysis provides a fine-grained hierarchy of formal systems based on their consistency strength (first-order arithmetic, second-order arithmetic)

Ordinal Numbers and Transfinite Induction

- Ordinal numbers extend natural numbers into the transfinite, representing well-ordered sets
- Transfinite induction generalizes mathematical induction to prove statements about ordinal numbers
- Allows for proving properties that hold for all ordinals up to a certain point in the ordinal hierarchy
- Ordinal arithmetic operations (addition, multiplication, exponentiation) are used to construct and manipulate ordinal notations

Consistency and Cut Elimination

Consistency Strength and Relative Consistency

- Consistency strength measures the power of a formal system in terms of the consistency of weaker systems
- A system A is considered stronger than B if the consistency of A implies the consistency of B
- Relative consistency proofs establish the consistency of one system relative to another (Euclidean geometry relative to hyperbolic geometry)
- Consistency strength hierarchy provides a partial ordering of formal systems based on their relative consistency

Cut Elimination and Consistency Proofs

- Cut elimination is a key property of sequent calculus proof systems, allowing for the elimination of the cut rule
- Cut-free proofs are more structured and easier to analyze, enabling consistency proofs
- Gentzen's consistency proof for Peano arithmetic uses transfinite induction up to the ordinal ϵ_0
- Cut elimination is used to prove the consistency of various subsystems of arithmetic and set theory (subsystems of second-order arithmetic)

Implications for Foundations of Mathematics

- Consistency proofs provide a solid foundation for mathematical theories by ensuring their non-contradictory nature
- Cut elimination and ordinal analysis techniques have been applied to study the foundations of mathematics
- Results on consistency strength and proof-theoretic reductions shed light on the relationships between different axiomatic systems
- Advances in proof theory contribute to the understanding of the limits and capabilities of mathematical reasoning (incompleteness theorems, independence results)

Ordinal Notations

Representing Ordinals

- Ordinal notation systems provide a way to represent and manipulate ordinal numbers using symbols and rules
- Common ordinal notation systems include Cantor normal form, Veblen functions, and Bachmann-Howard ordinal
- Ordinal notations allow for the comparison and arithmetic of ordinals beyond the finite realm
- Different notation systems have varying expressive power and are suited for different purposes (proof theory, set theory)

Ordinal Collapsing Functions and Hierarchies

- Ordinal collapsing functions map large ordinals to smaller ones while preserving certain properties
- Used to define hierarchies of fast-growing functions and measure the strength of axiom systems
- Examples include the Hardy hierarchy, Veblen hierarchy, and Feferman-Schütte ordinal Γ_0
- Ordinal hierarchies provide a way to classify the growth rates of functions and the proof-theoretic strength of theories (Peano arithmetic, Kripke-Platek set theory)

Ordinal Analysis Techniques

- Ordinal analysis involves assigning ordinals to proof systems and studying their properties

- Techniques such as cut elimination, infinitary proof theory, and ordinal induction are used in ordinal analysis
- Aims to determine the proof-theoretic ordinal of a system, which characterizes its consistency strength
- Ordinal analysis has been applied to various subsystems of arithmetic and set theory (Peano arithmetic, Zermelo-Fraenkel set theory)

11.3 Reverse mathematics and proof-theoretic strength

Reverse mathematics explores the axioms needed to prove theorems in mathematics. It's like figuring out the minimum ingredients for a recipe. This approach helps us understand the foundations of math and how different parts connect.

The big five subsystems of second-order arithmetic are key players in this field. They range from the basic RCA_0 to the powerful $\Pi^1_1 - CA_0$, each with its own strengths and capabilities in proving theorems.

Reverse Mathematics and Subsystems

Overview of Reverse Mathematics

- Reverse mathematics studies the strength of axioms needed to prove theorems
- Involves proving equivalences between theorems and axioms over a weak base theory
- Base theory is typically a subsystem of second-order arithmetic
- Goal is to determine the minimal axioms required for proving specific theorems

Subsystems of Second-Order Arithmetic

- Second-order arithmetic extends first-order Peano arithmetic with sets and quantification over sets
- Subsystems are formed by restricting the comprehension axiom schema
- Common subsystems include RCA_0 , WKL_0 , ACA_0 , ATR_0 , and $\Pi^1_1 - CA_0$
- Subsystems are linearly ordered by inclusion and consistency strength

The Big Five Subsystems

- The big five are the most commonly studied subsystems in reverse mathematics
- RCA_0 (Recursive Comprehension Axiom): Includes comprehension for computable sets and Σ^0_1 -induction
- WKL_0 (Weak König's Lemma): Extends RCA_0 with a compactness principle for infinite binary trees
- ACA_0 (Arithmetical Comprehension Axiom): Allows comprehension for arithmetically definable sets
- ATR_0 (Arithmetical Transfinite Recursion): Extends ACA_0 with transfinite recursion of arithmetic predicates
- $\Pi^1_1 - CA_0$ (Π^1_1 -Comprehension Axiom): Allows comprehension for Π^1_1 formulas

Proof-Theoretic Strength

- Proof-theoretic strength measures the strength of a theory in terms of the functions it can prove total

- Commonly measured using ordinal notations and fast-growing hierarchies
- The big five have well-understood proof-theoretic strengths:
- RCA_0 and WKL_0 : Prove the totality of primitive recursive functions
- ACA_0 : Proves the totality of ϵ_0 -recursive functions (Ackermann-level)
- ATR_0 : Proves the totality of Γ_0 -recursive functions (non-primitive recursive)
- $\Pi_1^1 - CA_0$: Has the strength of finitary Zermelo-Fraenkel set theory

Comprehension and Induction

Comprehension Axioms

- Comprehension axioms assert the existence of sets defined by formulas
- General form: $\exists X \forall n (n \in X \leftrightarrow \varphi(n))$ for a formula φ
- Unrestricted comprehension leads to inconsistency (Russell's paradox)
- Subsystems of second-order arithmetic restrict comprehension to specific formula classes

Induction Principles

- Induction principles allow proving statements for all natural numbers
- First-order induction: $(\varphi(0) \wedge \forall n (\varphi(n) \rightarrow \varphi(n+1))) \rightarrow \forall n \varphi(n)$
- Second-order arithmetic includes induction for formulas with set quantifiers
- Restricted induction schemes are often used, such as Σ_1^0 -induction in RCA_0

Arithmetical Comprehension

- Arithmetical comprehension allows forming sets defined by arithmetic formulas
- Arithmetic formulas are those with only number quantifiers (no set quantifiers)
- The system ACA_0 includes arithmetical comprehension
- Equivalent to the existence of the Turing jump of any set

Π_1^1 -Comprehension

- Π_1^1 -comprehension allows forming sets defined by Π_1^1 formulas
- Π_1^1 formulas have a universal set quantifier followed by an arithmetic formula
- The system $\Pi_1^1 - CA_0$ includes Π_1^1 -comprehension
- Equivalent to the existence of a Suslin set (a specific Π_1^1 set)

Key Principles

Weak König's Lemma (WKL)

- Weak König's Lemma is a compactness principle for infinite binary trees

- States that every infinite binary tree has an infinite path
- Equivalent to the compactness of propositional logic over RCA_0
- Strictly stronger than RCA_0 but weaker than ACA_0
- Used to prove the completeness theorem and the Heine-Borel covering lemma

Reverse Mathematics of WKL

- Many theorems in analysis and algebra are equivalent to WKL_0 over RCA_0
- Examples include the intermediate value theorem and the existence of algebraic closures
- These equivalences show that WKL is necessary and sufficient for proving these theorems
- Demonstrates the role of compactness in many areas of mathematics

Applications and Consequences

- Reverse mathematics has been applied to many areas, including analysis, algebra, and combinatorics
- Has revealed the role of set existence principles and induction in ordinary mathematics
- Provides a fine-grained analysis of the strength of mathematical theorems
- Connections with computability theory, descriptive set theory, and constructivism

11.4 Constructive and predicative mathematics

Constructive and predicative mathematics challenge traditional foundations, demanding explicit constructions and rejecting certain logical principles. These approaches aim to provide more rigorous and computationally meaningful foundations for mathematics, aligning with intuitionistic philosophy.

Type theory offers an alternative foundation, interpreting propositions as types and proofs as programs. This perspective provides a computational interpretation of mathematics, avoiding paradoxes through a predicative hierarchy of types and enabling powerful proof assistants.

Constructivism and Intuitionism

Constructive Foundations of Mathematics

- Constructivism is a philosophy of mathematics that requires mathematical objects to be constructed rather than merely proven to exist
- Intuitionism, a form of constructivism, was developed by L.E.J. Brouwer in the early 20th century
- Rejects the law of excluded middle and double negation elimination
- Asserts that mathematical objects are mental constructions rather than platonic entities
- Constructive proofs provide explicit algorithms for constructing the mathematical objects they prove the existence of
- Contrast with classical proofs which may prove existence without providing a method of construction (nonconstructive proofs)

Constructive Logic and the BHK Interpretation

- Constructive logic rejects the law of excluded middle ($P \vee \neg P$) as a valid logical principle
- In constructive systems, $\neg\neg P$ does not necessarily imply P (double negation elimination fails)
- Example: "This program is bug-free or not bug-free" is not necessarily true constructively until the program is examined in its entirety
- The Brouwer–Heyting–Kolmogorov (BHK) interpretation provides an intuitionistic semantics for logical connectives
- $P \wedge Q$ is proven if P is proven and Q is proven
- $P \vee Q$ is proven if either P is proven or Q is proven
- $P \rightarrow Q$ is proven if there is a construction transforming any proof of P into a proof of Q
- $\neg P$ is proven if there is a construction transforming any hypothetical proof of P into a proof of a contradiction
- Realizability is a formal semantics for constructive logic based on the BHK interpretation
- A formula is realizable if there exists a program that realizes it according to certain conditions

Predicativism

Predicative Foundations

- Predicativism is a philosophy of mathematics that rejects impredicative definitions
- An impredicative definition defines an object in terms of a collection that contains the object being defined
- Example: Defining the set of all sets that do not contain themselves
- Motivations for predicativism include avoiding paradoxes (like Russell's paradox) and vicious circles in definitions
- Some predicative systems aim to recapture most of classical mathematics while using only predicative constructions

Predicative Theories

- Constructive set theories like Constructive Zermelo-Fraenkel (CZF) are predicatively justified
- Only allow set constructions that do not quantify over the entire universe of sets
- Weaker subsystems of second-order arithmetic like ACA_0 are also predicative
- Limit comprehension axioms to only quantify over numbers rather than sets/properties

Type Theory

Foundations Based on Type Theory

- Type theory is an alternative foundation for mathematics based on a system of types and terms

- Every term has an associated type, with types constructed from atomic types using type constructors
- Provides a computational interpretation of propositions via the Curry-Howard correspondence
- Propositions are interpreted as types, proofs as programs of that type
- Compared to set theory, type theory offers:
 - A more expressive language for formalizing mathematics
 - Intrinsic computational content via the Curry-Howard correspondence
 - Avoidance of paradoxes with a predicative hierarchy of types

Constructive Type Theories

- Martin-Löf type theory is a predicative, constructive type theory
- Developed by Per Martin-Löf in the 1970s
- Provides a foundation for constructive mathematics
- Variants include:
 - Intensional MLTT: Treats propositional equality as judgmental
 - Extensional MLTT: Equates propositional and judgmental equality
- Other constructive type theories build on MLTT with additional features
- Example: The Calculus of Constructions adds dependent types and polymorphism
- Used as the basis for proof assistants like Coq and Lean

Unit 12 – Proof Theory and Computability Theory

12.1 Curry-Howard isomorphism

The Curry-Howard isomorphism links logic and computation, showing how propositions match types and proofs match programs. This connection lets us use programming concepts in logic and vice versa, leading to cool stuff like proof assistants and better program verification.

Natural deduction in logic lines up nicely with the Curry-Howard isomorphism. It shows how logical rules match up with ways we build and use different types in programming. This idea even works for fancier types that depend on values.

Foundations of Curry-Howard Isomorphism

Correspondence between Propositions and Types

- Propositions in logic correspond to types in programming languages
- Proofs of propositions correspond to programs of a given type
- Type theory provides a foundation for this correspondence by treating logical propositions as types and proofs as programs
- Intuitionistic logic, a constructive logic that requires proofs for every true statement, aligns well with the Curry-Howard isomorphism
- Constructive mathematics, which relies on intuitionistic logic, ensures that every proven statement has a corresponding program (witness)

Implications and Applications

- The Curry-Howard isomorphism establishes a deep connection between logic and computation
- Allows for the exchange of ideas and techniques between the two fields
- Enables the use of type systems in programming languages to ensure program correctness
- Provides a basis for developing proof assistants and automated theorem provers
- Offers insights into the nature of computation and its relationship to mathematical reasoning

Logical Systems and Curry-Howard

Natural Deduction and Curry-Howard

- Natural deduction is a logical system that closely mirrors the structure of proofs in intuitionistic logic
- In natural deduction, logical connectives (e.g., implication, conjunction) correspond to type constructors in programming languages
- Introduction and elimination rules for logical connectives in natural deduction correspond to the construction and destruction of values of the corresponding types

- The Curry-Howard isomorphism allows for the interpretation of natural deduction proofs as programs in a typed lambda calculus (simply-typed lambda calculus)

Dependent Types and Curry-Howard

- Dependent types are types that can depend on values, allowing for more expressive type systems
- In a dependently-typed system, types can be parameterized by values, enabling the encoding of complex logical propositions as types
- The Curry-Howard isomorphism extends to dependent types, establishing a correspondence between proofs in higher-order logic and programs in dependently-typed languages (e.g., Coq, Agda)
- Dependent types enable the specification and verification of more complex program properties and mathematical theorems

Category Theory and Curry-Howard

Cartesian Closed Categories and Curry-Howard

- Category theory provides a general framework for studying mathematical structures and their relationships
- Cartesian closed categories (CCCs) are a type of category that models the simply-typed lambda calculus
- In a CCC, objects correspond to types, and morphisms correspond to functions between types
- The Curry-Howard isomorphism can be generalized to the categorical setting, establishing a correspondence between proofs in intuitionistic logic and morphisms in a CCC
- The categorical perspective on Curry-Howard allows for the study of the isomorphism in a more abstract and general setting, enabling its application to various type systems and logics

Functors, Natural Transformations, and Curry-Howard

- Functors are structure-preserving mappings between categories, allowing for the translation of concepts and constructions from one category to another
- In the context of Curry-Howard, functors can be used to map between different type systems or logical frameworks
- Natural transformations are structure-preserving mappings between functors, providing a way to relate and compare different translations between categories
- The interplay between functors, natural transformations, and the Curry-Howard isomorphism allows for the study of relationships between different type systems and logics, enabling the transfer of results and techniques between them

12.2 Lambda calculus and proof normalization

Lambda calculus is a formal system for expressing computation through function abstraction and application. It provides a simple yet powerful foundation for studying computability, programming languages, and proof theory, connecting abstract mathematics with practical computer science.

In this section, we explore lambda calculus basics, its properties, and its relationship to proofs. We'll see how lambda terms encode data and computation, and how concepts like normalization and the

Curry-Howard correspondence link lambda calculus to logic and programming.

Lambda Calculus Basics

Syntax and Evaluation Rules

- Lambda terms consist of variables, abstractions (anonymous functions), and applications
- Variables are single letters used to represent values (x, y, z)
- Abstractions are functions defined using the λ symbol followed by a variable and a body ($\lambda x.M$ where M is another lambda term)
- Applications represent the application of a function to an argument ($M N$ where M and N are lambda terms)
- Beta reduction is the process of evaluating an application by substituting the argument for the bound variable in the function body
- Example: $(\lambda x.x) y$ β -reduces to y by replacing all occurrences of x with y in the function body
- Alpha conversion allows renaming bound variables in a lambda term without changing its meaning
- Example: $\lambda x.x$ and $\lambda y.y$ are equivalent terms related by α -conversion
- Normal form is a lambda term that cannot be further reduced using β -reduction
- Terms in normal form represent fully evaluated expressions

Encoding Data and Computation

- Church encodings allow representing data types like booleans, numbers, and pairs using lambda terms
- Church booleans: $\text{true} = \lambda x.\lambda y.x$, $\text{false} = \lambda x.\lambda y.y$
- Church numerals: $0 = \lambda f.\lambda x.x$, $1 = \lambda f.\lambda x.f x$, $2 = \lambda f.\lambda x.f (f x)$, ...
- Lambda calculus is Turing complete can express any computable function
- Recursive functions can be defined using fixed-point combinators (Y combinator)
- Example: $\text{factorial} = \lambda f.\lambda n.\text{if } (\text{isZero } n) 1 (\text{multiply } n (f (\text{pred } n)))$

Properties of Lambda Calculus

Confluence and Uniqueness of Normal Forms

- Church-Rosser theorem states that if a term can be reduced to two different terms, there exists a common term to which both can be reduced
- Implies confluence: reduction order does not affect final result
- Example: $(\lambda x.x x) (\lambda y.y)$ β -reduces to $(\lambda y.y) (\lambda y.y)$ regardless of evaluation order
- Uniqueness of normal forms is a consequence of the Church-Rosser theorem
- If a term has a normal form, it is unique up to α -conversion

Termination and Decidability

- Strong normalization property holds for some variants of lambda calculus (simply typed lambda calculus)
- Guarantees that every reduction sequence terminates in a normal form
- Implies decidability of normalization and type checking
- Untyped lambda calculus is not strongly normalizing some terms have infinite reduction sequences
- Example: $(\lambda x.x\ x)\ (\lambda x.x\ x)$ β -reduces to itself indefinitely
- Undecidable to determine if an arbitrary term has a normal form

Lambda Calculus and Proofs

Correspondence with Natural Deduction

- Cut elimination in natural deduction corresponds to β -reduction in lambda calculus
- Cut rule allows using a lemma (intermediate proof) in a larger proof
- Elimination of cuts yields a direct proof analogous to a normal form
- Typed lambda calculus assigns types to lambda terms, restricting well-formed expressions
- Simple types: $\tau : : = \alpha \mid \tau \rightarrow \tau$ (base types and function types)
- Well-typed terms cannot "go wrong" during evaluation (type soundness)
- Curry-Howard correspondence relates proofs in intuitionistic logic to typed lambda terms
- Propositions correspond to types, proofs correspond to terms
- Example: $A \rightarrow B$ corresponds to the type of functions from A to B
- Proof normalization (cut elimination) corresponds to β -reduction

Applications in Programming Languages

- Functional programming languages (Haskell, ML) are based on typed lambda calculus
- Higher-order functions, type inference, pattern matching, lazy evaluation
- Dependently typed languages (Coq, Agda) extend the Curry-Howard correspondence
- Types can depend on values, allowing expressing precise specifications
- Proofs of program properties can be expressed and checked within the language
- Proof assistants (Isabelle, Lean) use lambda calculus as a foundation for formal reasoning
- Proofs are represented as lambda terms, can be checked and manipulated programmatically
- Enable formalization of mathematics and verification of software/hardware

12.3 Realizability and extracting computational content from proofs

Realizability connects proofs to programs, showing how logical formulas translate to computational tasks. It's a bridge between abstract math and concrete algorithms, revealing the hidden computational power in logical reasoning.

This idea fits into proof theory by demonstrating how formal proofs aren't just abstract symbols, but contain real-world instructions. It shows that logic and computation are deeply intertwined, not separate realms.

Realizability and Constructive Proofs

Kleene Realizability and Computational Interpretation

- Kleene realizability provides a computational interpretation of intuitionistic arithmetic and logic
- Associates each formula A with a set $\|A\|$ of "realizers" which can be seen as programs that validate the computational content of A
- A realizer $e \in \|A\|$ can be thought of as a program that computes the evidence or proof of the formula A
- Kleene realizability establishes a connection between intuitionistic proofs and computable functions

Constructive Proofs in Intuitionistic Arithmetic

- In intuitionistic arithmetic, proofs are constructive and computationally meaningful
- Constructive proofs provide explicit algorithms or methods for constructing mathematical objects
- Intuitionistic logic rejects the law of excluded middle ($A \vee \neg A$) and double negation elimination ($\neg\neg A \rightarrow A$)
- Constructive proofs avoid non-constructive principles like the axiom of choice and proof by contradiction
- Intuitionistic arithmetic is weaker than classical arithmetic but is well-suited for extracting computational content from proofs

Program and Witness Extraction

Extracting Programs from Proofs

- Program extraction is the process of deriving a program from a constructive proof
- Given a constructive proof of a formula $\exists x. P(x)$, program extraction produces a program that computes a witness a such that $P(a)$ holds
- The extracted program is correct by construction since it is derived directly from the proof
- Program extraction enables the synthesis of certified programs from proofs (λ -terms)

Witness Extraction and Proof-Carrying Code

- Witness extraction is a technique for extracting computational content from classical proofs
- It extends program extraction to handle classical logic and axioms like the law of excluded middle

- Witness extraction produces a program along with a proof of its correctness (proof-carrying code)
- The extracted program computes a witness that satisfies the specification given by the proof
- Proof-carrying code enables the verification of program properties and ensures trust in the extracted code

Variants of Realizability

Modified Realizability and Other Variants

- Modified realizability is a variant of Kleene realizability that allows for the extraction of programs from classical proofs
- It extends realizability to handle the law of excluded middle and other non-constructive principles
- Modified realizability provides a way to extract computational content from proofs in classical logic
- Other variants of realizability include:
 - Dialectica interpretation: Associates formulas with games and strategies
 - Functional interpretations: Maps formulas to higher-order functionals
 - Friedman's translation: Embeds classical arithmetic into intuitionistic arithmetic
- These variants provide different perspectives on the computational content of proofs and enable program extraction in various settings

12.4 Proof complexity and computational complexity

Proof complexity and computational complexity are two intertwined fields in theoretical computer science. They explore the resources needed to prove statements and solve problems, shedding light on the fundamental nature of computation and reasoning.

This section dives into propositional proof systems, focusing on resolution and Frege systems. It then examines proof size, NP-completeness, and the Cook-Reckhow theorem, revealing connections between proof efficiency and computational complexity classes.

Propositional Proof Systems

Resolution and Frege Systems

- Propositional proof systems are formal systems for proving propositional tautologies
- Resolution is a simple and widely used propositional proof system based on the resolution inference rule
 - Operates on clauses (disjunctions of literals)
 - Resolves two clauses containing complementary literals to derive a new clause
 - Refutation proofs derive the empty clause, indicating the unsatisfiability of the original formula
- Frege systems are a class of propositional proof systems based on a fixed set of axioms and inference rules
 - Closely resemble natural deduction systems used in mathematical logic

- Proofs are sequences of formulas, each derived from previous formulas using the inference rules
- Extended Frege systems augment Frege systems with the ability to introduce new variables and definitions
- Allows for more concise proofs by defining intermediate formulas
- Can lead to exponentially shorter proofs compared to standard Frege systems for some formulas

Polynomial Simulation

- Polynomial simulation is a relationship between two propositional proof systems
- A proof system P polynomially simulates another proof system Q if there exists a polynomial-time computable function that translates Q -proofs into P -proofs
- The size of the resulting P -proof is polynomially bounded by the size of the original Q -proof
- Polynomial simulation captures the relative efficiency of proof systems
- If P polynomially simulates Q , then P is at least as efficient as Q in terms of proof size
- Frege systems and extended Frege systems are known to polynomially simulate resolution
- This implies that Frege and extended Frege systems are at least as powerful as resolution in terms of the sizes of proofs they can generate

Proof Complexity

Proof Size and NP-Completeness

- Proof complexity studies the sizes of proofs in various propositional proof systems
- The size of a proof is the number of symbols or lines required to write down the proof
- Measured as a function of the size of the formula being proven
- The goal is to understand the efficiency of proof systems and the inherent difficulty of proving certain propositional formulas
- Many propositional tautologies arising from NP-complete problems have been shown to require super-polynomial proof size in various proof systems
- Examples include tautologies based on the pigeonhole principle and graph coloring problems
- NP-completeness plays a crucial role in proof complexity
- If $NP \neq co-NP$, then there exist propositional tautologies with no polynomial-size proofs in any propositional proof system
- This is a consequence of the Cook-Levin theorem, which establishes the NP-completeness of the propositional satisfiability problem (SAT)

Cook-Reckhow Theorem and Lower Bounds

- The Cook-Reckhow theorem characterizes the efficiency of propositional proof systems in terms of the NP-completeness of the tautology recognition problem

- A propositional proof system is considered efficient if the tautology recognition problem for that system is in NP
- In other words, if there exists a polynomial-time algorithm to verify the correctness of proofs in that system
- The theorem implies that $NP \neq co-NP$ is equivalent to the existence of a propositional proof system in which the tautology recognition problem is not in NP
- Super-polynomial lower bounds on proof size have been established for various propositional proof systems
- Examples include exponential lower bounds for resolution proofs of pigeonhole principle tautologies and Tseitin tautologies on expander graphs
- These lower bounds demonstrate the limitations of specific proof systems and provide insights into the difficulty of proving certain classes of tautologies
- Proving lower bounds on proof size is a central challenge in proof complexity and has connections to fundamental questions in computational complexity theory, such as the P vs. NP problem

Unit 13 – Proof Theory Applications in CS

13.1 Program verification and formal methods

Program verification and formal methods are crucial for ensuring software correctness. These techniques use mathematical proofs to verify that programs meet their specifications, catching errors early in development and improving reliability.

Hoare logic, formal specifications, and model checking are key tools in this field. They allow developers to rigorously define and verify program behavior, enhancing software quality and reducing the risk of critical failures in complex systems.

Hoare Logic and Program Verification

Hoare Triples and Correctness Proofs

- Hoare logic provides a formal system for reasoning about the correctness of programs
- Consists of axioms and inference rules for proving properties of programs
- Hoare triples, denoted as $\{P\} C \{Q\}$, specify the behavior of a program fragment C
- P represents the pre-condition that must hold before executing C
- Q represents the post-condition that will hold after executing C if P was true initially
- Verification conditions are logical formulas generated from Hoare triples
- Must be proved to establish the correctness of the program with respect to its specification
- Correctness proofs involve applying inference rules to derive the desired post-condition from the pre-condition and the program code
- Proofs can be constructed manually or with the assistance of theorem provers (*Coq, Isabelle/HOL*)

Applying Hoare Logic in Practice

- Pre-conditions and post-conditions are written as logical assertions or predicates
- Pre-conditions describe the assumptions about the program's input and state before execution
- Post-conditions describe the expected output and state after execution
- Hoare logic can be used to prove properties such as:
 - Partial correctness: If the program terminates, it satisfies the post-condition
 - Total correctness: The program always terminates and satisfies the post-condition
- Hoare logic has been applied to various programming languages and paradigms
 - Imperative languages (*C, Java*)
 - Functional languages (*Haskell, ML*)
 - Object-oriented languages (*Java, C++*)

Formal Methods

Formal Specification and Model Checking

- Formal specification involves precisely defining the desired behavior and properties of a system
- Specification languages such as *Z*, *VDM*, and *TLA+* are used to write formal specifications
- Model checking is an automated technique for verifying that a system model satisfies a given specification
- Explores all possible states and transitions of the system to check for violations of the specified properties
- Widely used in hardware and software verification (*Intel*, *NASA*)
- Invariants are properties that hold throughout the execution of a system
- Can be used to specify safety properties (e.g., absence of deadlocks) and liveness properties (e.g., eventual completion)
- Model checkers can verify that invariants are preserved by the system model

Benefits and Challenges of Formal Methods

- Formal methods provide a rigorous and systematic approach to system development
- Help detect errors and inconsistencies early in the development process
- Increase confidence in the correctness and reliability of the system
- Challenges in adopting formal methods include:
 - Steep learning curve and expertise required to write formal specifications
 - Scalability issues when dealing with large and complex systems
 - Balancing the cost and effort of formal verification against the benefits gained

Program Analysis Techniques

Static Analysis and Its Applications

- Static analysis examines program code without executing it
- Analyzes the structure, data flow, and control flow of the program
- Can detect potential bugs, security vulnerabilities, and performance issues
- Common static analysis techniques include:
 - Type checking: Verifies that variables and expressions have consistent types
 - Data flow analysis: Tracks the flow of data through the program to identify issues like uninitialized variables or null pointer dereferences
 - Control flow analysis: Analyzes the possible execution paths of the program to detect unreachable code or infinite loops

- Static analysis tools are widely used in industry for code quality assurance (*Coverity*, *SonarQube*)
- Integrated into development workflows to catch issues early and improve code maintainability

Symbolic Execution and Program Verification

- Symbolic execution explores program paths by treating input values as symbolic variables
- Generates path conditions and symbolic expressions for each path
- Can be used to generate test cases that cover different program paths
- Symbolic execution can aid in program verification by:
 - Checking the feasibility of program paths and identifying infeasible paths
 - Generating verification conditions for each path that can be proved using theorem provers
 - Detecting runtime errors such as division by zero or out-of-bounds array accesses
- Symbolic execution tools (*KLEE*, *JavaPathFinder*) have been used to find bugs and generate test cases for real-world software systems
- Effective in uncovering corner cases and exploring complex program behaviors

13.2 Type theory and programming languages

Type theory bridges logic and programming, showing how proofs relate to programs. It's the backbone of modern programming languages, ensuring code correctness and catching errors early. This connection helps create safer, more reliable software.

Type systems in programming languages enforce data constraints, enable modular coding, and serve as documentation. Advanced features like dependent types and polymorphism allow for more expressive and flexible code, while type inference and checking improve development efficiency and program safety.

Type Systems and Lambda Calculus

Foundations of Type Theory

- Type systems classify program phrases according to the kinds of values they compute
- Help prevent certain kinds of programming errors by defining and enforcing interfaces between different parts of a program
- Lambda calculus is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution
- Serves as a theoretical foundation for functional programming languages (λ -calculus)
- Curry-Howard correspondence establishes a direct relationship between mathematical proofs and computer programs
- Proofs correspond to programs (propositions as types)
- The proof of a proposition corresponds to a program of the corresponding type
- Type safety guarantees absence of certain run-time errors
- Well-typed programs cannot "go wrong" ($\text{if } \vdash t : \tau \text{ then } t \Downarrow v \text{ for some } v : \tau$)

Applications of Type Systems

- Type systems used in programming languages to enforce data type constraints
- Prevent errors such as adding a number to a string ($3 + \text{"hello"}$)
- Type systems enable modular programming by enforcing interfaces between components
- Ensure functions receive arguments of expected types (function expecting int receives int)
- Type systems catch certain errors at compile-time rather than run-time
- Reduces debugging effort and improves software reliability
- Type systems serve as a form of documentation, making code easier to understand and maintain
- Type signatures convey expected inputs and outputs of functions

Advanced Type System Features

Dependent Types

- Dependent types allow types to depend on terms
- Type of function's output can depend on the value of its input ($\text{Vec} : \text{Nat} \rightarrow \text{Type}$)
- Dependent types enable expressing precise properties of programs in their types
- $\text{append} : (n\ m : \text{Nat}) \rightarrow \text{Vec}\ A\ n \rightarrow \text{Vec}\ A\ m \rightarrow \text{Vec}\ A\ (n + m)$
- Dependent types blur the line between types and terms
- Allows proving properties of programs within the type system itself (e.g., $(n : \text{Nat}) \rightarrow \text{Vec}\ A\ n \rightarrow \text{Vec}\ A\ (\text{double}\ n)$)

Polymorphism and Subtyping

- Polymorphism allows code to be more generic by abstracting over types
- Parametric polymorphism: Code is written without mentioning specific types ($\text{id} : \forall A . A \rightarrow A$)
- Ad-hoc polymorphism: Code can have different implementations for different types (operator overloading)
- Subtyping allows a value of one type to be used in a context expecting another type
- Subtype relation $\leq$ is reflexive, transitive, and antisymmetric ($S \leq T$ if every value of type S can safely be used where a value of type T is expected)
- Enables substitutability and code reuse (functions accepting `Animal` can accept `Cat` or `Dog`)

Type Inference and Checking

Type Inference

- Type inference automatically deduces types of expressions based on context
- Relieves burden of manual type annotations ($\text{val } x = 3$ inferred as $\text{val } x : \text{Int} = 3$)

- Hindley-Milner type inference algorithm used in ML-family languages
- Infers most general type of an expression ($\text{id} : \forall \alpha. \alpha \rightarrow \alpha$)
- Bidirectional type inference combines top-down (checking) and bottom-up (synthesis) propagation
- Reduces annotation burden while supporting advanced type system features

Type Checking

- Type checking verifies type-correctness of a program against a type system
- Detects type errors such as applying a function to an argument of the wrong type
- Type checking can be static (at compile-time) or dynamic (at run-time)
- Static type checking catches errors early but requires more annotations
- Dynamic type checking defers checks to run-time but allows more flexibility
- Gradual typing combines static and dynamic typing
- Allows mixing typed and untyped code (any type in TypeScript)
- Performs static checks where possible and defers remaining checks to run-time

13.3 Automated theorem proving and proof assistants

Automated theorem proving uses algorithms to prove mathematical statements. It combines techniques like resolution and unification with SAT and SMT solvers to tackle complex logical problems. These tools automate reasoning and find proofs faster than humans.

Proof assistants like Coq, Isabelle/HOL, and Lean help verify math and software. They offer interactive theorem proving, where users guide proofs with tactics. This blend of human insight and machine power tackles tricky proofs and boosts confidence in results.

Automated Theorem Proving

Resolution and Unification

- Resolution is a rule of inference for propositional logic and first-order logic that produces a new clause from two clauses containing complementary literals
- Involves resolving two clauses by assuming one is true and the other is false, then eliminating complementary literals to produce a new clause
- Unification is a key concept in automated theorem proving that involves finding a substitution that makes two logical expressions identical
- Unification algorithms (syntactic unification, semantic unification) find the most general unifier between two expressions, allowing for variables to be replaced with terms

SAT and SMT Solvers

- SAT solvers are programs that determine the satisfiability of propositional logic formulas, checking if there exists an interpretation that makes the formula true

- Use algorithms like DPLL (Davis-Putnam-Logemann-Loveland) to efficiently search for satisfying assignments
- SMT (Satisfiability Modulo Theories) solvers build upon SAT solvers to decide the satisfiability of formulas in first-order logic with respect to background theories (arithmetic, arrays, uninterpreted functions)
- SMT solvers combine a SAT solver with specialized decision procedures for the background theories, allowing for reasoning about more expressive formulas

Proof Assistants

Popular Proof Assistants

- Coq is a proof assistant based on the Calculus of Inductive Constructions, a dependent type theory
- Supports writing formal specifications and proofs, allowing for the verification of complex mathematical theorems and programs
- Isabelle/HOL is a generic proof assistant that supports various logics, with higher-order logic (HOL) being the most widely used
- Offers a rich set of tools for interactive theorem proving and has been used to formalize substantial mathematical theories
- Lean is a newer proof assistant based on dependent type theory, designed for both mathematical reasoning and software verification
- Provides a powerful framework for writing formal proofs and has a growing library of mathematical results

Features and Applications

- Proof assistants offer expressive logics and type systems for specifying mathematical concepts and software systems
- Support the interactive development of formal proofs, with the ability to define custom tactics and automation
- Used in a wide range of applications, including verifying the correctness of algorithms, formalizing mathematical theories, and certifying the safety of critical systems
- Enable the construction of large, machine-checked proofs that can be independently verified, increasing confidence in the correctness of results

Interactive Theorem Proving

Proof Tactics and Interactivity

- Proof tactics are programs that automate common proof steps or strategies in interactive theorem proving
- Examples of tactics include intros for introducing hypotheses, apply for applying a lemma or hypothesis, and auto for applying a collection of standard tactics
- Interactive theorem proving involves a user guiding the construction of a proof by applying tactics and providing input when needed

- Allows for the development of complex proofs that may be difficult to fully automate, while still benefiting from the automation provided by tactics

Proof Scripts and Reproducibility

- Proof scripts are files that contain a sequence of proof commands and tactics used to construct a formal proof in a proof assistant
- Serve as a record of the proof development process and can be replayed to check the correctness of the proof
- Enable the reproducibility of proofs, as other users can examine the proof script to understand the reasoning behind each step
- Can be shared and built upon by the community, allowing for the collaborative development of large formalized theories
- Examples of proof scripts include .v files in Coq and .thy files in Isabelle/HOL, which contain the definitions, lemmas, and proofs of a formalization

13.4 Logic programming and proof search algorithms

Logic programming languages like Prolog and Answer Set Programming use declarative approaches to solve complex problems. They rely on rules, facts, and inference engines to derive new information and find solutions in areas like AI and optimization.

Inference techniques like SLD resolution and unification are key to logic programming. These methods, along with search strategies and knowledge representation using Horn clauses, enable efficient problem-solving and reasoning in various applications.

Logic Programming Languages

Prolog

- Prolog (Programming in Logic) is a declarative logic programming language based on first-order logic
- Consists of a set of rules and facts used to define relationships between objects
- Programs are written as a set of clauses (facts and rules) that describe the problem domain
- Prolog uses a built-in inference engine to derive new facts from the given clauses through logical inference
- Supports recursive programming, allowing complex problems to be broken down into smaller, more manageable subproblems
- Used in artificial intelligence, natural language processing, and expert systems (medical diagnosis, financial planning)

Answer Set Programming (ASP)

- Answer Set Programming (ASP) is a declarative programming paradigm based on the stable model semantics of logic programming
- Represents knowledge using a set of rules and constraints

- Programs are written as a set of rules that describe the problem domain and the desired properties of the solution
- ASP solvers generate stable models (answer sets) that satisfy the given rules and constraints
- Supports non-monotonic reasoning, allowing the system to handle incomplete or contradictory information
- Used in planning, scheduling, and optimization problems (resource allocation, supply chain management)

Inference Techniques

SLD Resolution

- SLD (Selective Linear Definite clause) resolution is an inference rule used in logic programming for deriving new clauses from existing ones
- Combines a goal clause with a definite clause (fact or rule) to generate a new goal clause
- Applies unification to match the selected literal in the goal clause with the head of the definite clause
- Resolves the selected literal with the body of the definite clause, generating a new goal clause
- Continues the resolution process until an empty clause is derived, indicating a successful proof, or until no more resolvents can be generated

Unification and Backtracking

- Unification is the process of finding a substitution that makes two terms identical
- Used in logic programming to match a goal with the head of a clause during resolution
- Determines if the goal and the head of the clause can be made identical by substituting variables with terms
- If unification succeeds, the substitution is applied to the body of the clause, generating a new goal
- Backtracking is a search technique used in logic programming to explore alternative solutions when a unification or resolution fails
- When a goal fails, the system backtracks to the most recent choice point and tries an alternative clause or substitution
- Allows the system to systematically explore the search space and find all possible solutions

Search Strategies

Depth-First and Breadth-First Search

- Depth-first search (DFS) is a search strategy that explores a branch of the search tree as deeply as possible before backtracking
- Implemented using a stack data structure to keep track of the nodes to be visited
- Can be more memory-efficient than breadth-first search but may not find the shortest solution path

- Breadth-first search (BFS) is a search strategy that explores all the neighboring nodes at the current depth before moving on to the nodes at the next depth level
- Implemented using a queue data structure to keep track of the nodes to be visited
- Guarantees finding the shortest solution path but may require more memory than depth-first search

Tabling

- Tabling is a technique used in logic programming to avoid redundant computations and improve efficiency
- Memoizes the results of previously computed subgoals and reuses them when the same subgoal is encountered again
- Helps avoid infinite loops and redundant computations in recursive programs
- Implemented using a table data structure to store the results of previously computed subgoals
- Improves the performance of programs with redundant computations and enables the handling of left-recursive rules

Knowledge Representation

Horn Clauses

- Horn clauses are a restricted form of first-order logic clauses used in logic programming
- Consist of at most one positive literal (the head) and zero or more negative literals (the body)
- Can represent facts (clauses with no negative literals), rules (clauses with both positive and negative literals), and goals (clauses with only negative literals)
- Provide a simple and efficient way to represent knowledge in logic programming
- Enable efficient inference algorithms, such as SLD resolution, to derive new facts from the given clauses
- Used to represent domain-specific knowledge in various applications (expert systems, natural language processing, planning)

Unit 14 – Recent Advances in Proof Theory

14.1 Proof mining and proof unwinding

Proof mining and unwinding are powerful techniques for extracting useful information from mathematical proofs. By analyzing and transforming proofs, these methods reveal hidden computational content, providing concrete algorithms and bounds for abstract mathematical results.

These approaches bridge the gap between pure math and practical computation. They've found applications in computer science, numerical analysis, and optimization, helping mathematicians and computer scientists derive effective algorithms from theoretical proofs.

Extracting Computational Content

Proof Mining Techniques

- Proof mining extracts computational content from mathematical proofs
- Involves analyzing proofs to identify and isolate the constructive parts
- Focuses on obtaining quantitative information and effective bounds from proofs
- Applies techniques from mathematical logic and proof theory to classical proofs
- Aims to make the computational content of proofs explicit and usable

Constructive Mathematics and Proof Unwinding

- Constructive mathematics avoids non-constructive existence proofs and the law of excluded middle
- In constructive proofs, existential statements come with a method for constructing the object
- Proof unwinding is a systematic process of transforming classical proofs into constructive ones
- Unwinding eliminates non-constructive steps and replaces them with constructive alternatives
- The resulting constructive proof provides an algorithm for computing the desired object (solution, bound, etc.)

Applications and Benefits

- Extracting computational content has applications in computer science, numerical analysis, and optimization
- Provides a way to obtain concrete algorithms and bounds from abstract mathematical proofs
- Helps bridge the gap between pure mathematics and practical computation
- Constructive proofs often yield more efficient algorithms than classical existence proofs
- Quantitative information extracted from proofs can lead to improved numerical methods (error bounds, convergence rates)

Effective Bounds and Metatheorems

Deriving Effective Bounds

- Effective bounds are computable upper bounds on quantities of interest
- In proof mining, the goal is often to extract effective bounds from non-constructive proofs
- Logical metatheorems play a key role in deriving effective bounds
- These metatheorems establish general conditions under which effective bounds can be obtained
- By applying metatheorems, one can systematically extract computable bounds from a wide class of proofs

Functional Interpretation

- Functional interpretation is a powerful tool for extracting computational content
- Introduced by Gödel, it translates formulas in classical logic into formulas in a functional language
- The functional interpretation of a proof yields a program that realizes the computational content
- This program can be used to compute witnesses, bounds, and other constructive information
- Functional interpretation has been extended and refined by various researchers (Shoenfield, Kohlenbach)

Metatheorems and Their Applications

- Logical metatheorems establish general conditions for extracting effective bounds
- Examples include metatheorems for bounded metric spaces, hyperbolic spaces, and normed linear spaces
- These metatheorems cover a wide range of mathematical structures and properties
- By instantiating the metatheorems, one can obtain effective bounds for specific problems
- Metatheorems have been applied to various areas (approximation theory, fixed point theory, ergodic theory)

Historical Perspectives

Kreisel's Unwinding Program

- Georg Kreisel initiated the unwinding program in the 1950s
- Kreisel aimed to extract computational content from classical proofs in a systematic way
- He introduced the concept of unwinding proofs to eliminate non-constructive steps
- Kreisel's work laid the foundation for modern proof mining and constructive analysis
- His ideas influenced the development of functional interpretation and other proof-theoretic techniques

Kohlenbach's Proof Mining

- Ulrich Kohlenbach is a leading figure in modern proof mining
- Kohlenbach extended and systematized Kreisel's unwinding program
- He developed a general framework for extracting computational content using functional interpretation
- Kohlenbach established numerous logical metatheorems for various mathematical domains
- His work has led to applications in approximation theory, nonlinear analysis, and ergodic theory
- Kohlenbach's book "Applied Proof Theory: Proof Interpretations and their Use in Mathematics" is a comprehensive reference on proof mining

14.2 Proof-theoretic semantics

Proof-theoretic semantics assigns meaning to logical connectives based on their roles in formal proofs. It's a fresh take on meaning, focusing on how we use expressions in proofs rather than just their truth conditions.

This approach is built on some big ideas in philosophy. It's all about how we use language, make inferences, and verify statements. These concepts form the backbone of proof-theoretic semantics, tying meaning to how we reason and prove things.

Foundations of Proof-Theoretic Semantics

Meaning derived from proofs

- Proof-theoretic semantics assigns meaning to logical connectives based on the rules that introduce or eliminate them in formal proofs
- Contrasts with truth-conditional semantics which defines meaning in terms of truth conditions
- Focuses on the role expressions play in the practice of proving statements
- Meaning of a logical constant is given by its introduction rules in a proof system (natural deduction or sequent calculus)

Philosophical underpinnings

- Meaning-as-use principle asserts that the meaning of a linguistic expression is determined by how it is used in a language
- Inferentialism holds that the meaning of an expression is determined by the inferences it licenses and the circumstances under which it may be asserted
- Verificationism maintains that a statement is meaningful only if there is a method to verify its truth
- These philosophical views provide the foundation for proof-theoretic semantics by grounding meaning in the inferential rules and proof conditions associated with expressions

Inference Rules and Harmony

Introduction and elimination rules

- Introduction rules specify the conditions under which a logical connective can be introduced into a proof ($\wedge$ introduction, $\rightarrow$ introduction)
- Elimination rules specify what can be inferred from a formula containing a logical connective ($\wedge$ elimination, $\rightarrow$ elimination)
- Introduction and elimination rules for a connective should be in harmony, justifying each other
- Harmony ensures that the rules are not too strong or too weak, and that they coherently specify the meaning of the connective

Proof-theoretic validity

- A formula is considered valid if there exists a proof of it using the given inference rules
- Validity is defined in terms of the existence of a proof, rather than truth conditions
- Proof-theoretic validity provides an alternative to model-theoretic validity
- Allows for the study of logical systems that may not have a clear model-theoretic interpretation (intuitionistic logic)

Key Principles and Contributors

Prawitz's inversion principle

- Proposed by Dag Prawitz, a key figure in the development of proof-theoretic semantics
- States that the elimination rules for a logical connective should be inverses of its introduction rules
- Ensures that the rules are in harmony and that the meaning of the connective is fully determined by its introduction rules
- Provides a criterion for assessing the coherence and justification of inference rules in a proof system

Dummett's justification procedures

- Michael Dummett, another prominent advocate of proof-theoretic semantics, emphasized the role of justification procedures
- A justification procedure is a method for proving or refuting a statement based on the inference rules of the language
- The meaning of a statement is given by the conditions under which it can be proven or refuted using these procedures
- Dummett argued that this approach provides a more constructive and intuitively appealing account of meaning compared to truth-conditional semantics

14.3 Linear logic and substructural logics

Linear logic and substructural logics shake up traditional logic by treating propositions as consumable resources. This fresh approach allows for more precise reasoning about resource usage, making it super useful in computer science and programming.

These logics introduce new connectives and rules that capture resource-sensitive properties. They've found applications in areas like memory management, quantum computing, and program verification, showing how proof theory can tackle real-world problems.

Linear Logic and Substructural Logics

Overview of Linear Logic

- Linear logic is a substructural logic developed by Jean-Yves Girard in 1987
- Differs from classical logic by treating propositions as resources that are consumed when used
- Provides a more fine-grained analysis of proofs and computation
- Allows for reasoning about resource-sensitive properties (memory usage, time complexity)

Substructural Logics and Resource Sensitivity

- Substructural logics are logical systems that lack or restrict some of the structural rules of classical logic (weakening, contraction, exchange)
- Resource sensitivity is a key feature of substructural logics
- Propositions are treated as resources that are consumed when used in a proof
- Allows for more precise reasoning about resource usage and allocation
- Resource interpretation of linear logic
- Each proposition represents a single-use resource
- Proofs correspond to processes that consume and produce resources

Connectives in Linear Logic

Multiplicative Connectives

- Multiplicative conjunction ($\otimes$) and disjunction ($\text{\textbackslash parr}$)
- $A \otimes B$ represents the simultaneous availability of resources A and B
- $A \text{\textbackslash parr} B$ represents a choice between resources A and B
- Multiplicative implication ($\multimap$)
- $A \multimap B$ represents a process that consumes A to produce B
- Multiplicative constants: 1 (unit) and $\perp$ (bottom)

Additive Connectives

- Additive conjunction ($\&$) and disjunction ($\oplus$)

- $A \& B$ represents a choice between resources A and B , but only one is used
- $A \oplus B$ represents the availability of either resource A or B
- Additive constants: $\top$ (top) and 0 (zero)

Exponentials

- Exponential modalities: $!$ (of course) and $?$ (why not)
- $!A$ allows for unlimited duplication and discarding of the resource A
- $?A$ allows for unlimited duplication of the resource A , but not discarding
- Exponentials reintroduce some of the structural rules in a controlled manner
- Allows for embedding of classical logic within linear logic

Proof Systems for Linear Logic

Sequent Calculus for Linear Logic

- Sequent calculus is a proof system that operates on sequents of the form $\Gamma \vdash \Delta$
- Γ and Δ are multisets of formulas
- Intuitive reading: consuming the resources in Γ produces the resources in Δ
- Inference rules for each connective and exponential
- Rules for multiplicative and additive connectives reflect their resource interpretation
- Rules for exponentials allow for controlled use of structural rules

Proof Nets

- Proof nets are a graphical representation of proofs in linear logic
- Provide a more abstract and parallel view of proofs compared to sequent calculus
- Correctness criteria for proof nets (acyclicity and connectedness)
- Ensures that proof nets correspond to valid proofs in linear logic
- Cut elimination in proof nets corresponds to the execution of proofs as processes

Other Substructural Logics

Relevance Logic

- Relevance logic requires that the antecedent and consequent of an implication are relevant to each other
- Rejects the principle of explosion ($A \wedge \neg A \vdash B$) and the principle of monotonicity ($\Gamma \vdash A$ implies $\Gamma, B \vdash A$)
- Applications in computer science (type systems, information retrieval)

Affine Logic

- Affine logic is a variant of linear logic that allows for discarding of resources, but not duplication
- Obtained by adding the weakening rule to linear logic
- Used in the study of quantum computation and quantum information theory

Bunched Implications

- Bunched implications (BI) is a substructural logic that combines intuitionistic logic with a multiplicative conjunction
- Provides a logical foundation for reasoning about resource composition and separation
- Applications in program verification (separation logic) and computer security (access control)

14.4 Proof theory in mathematical practice and foundations

Proof theory plays a crucial role in mathematical practice and foundations. It investigates the structure of mathematical proofs, exploring concepts like reverse mathematics, predicativity, and foundational programs. These approaches help us understand the minimal axioms needed for theorems and the nature of mathematical reasoning.

Consistency proofs, proof-theoretic reductions, and ordinal analysis are key tools in this field. They allow us to compare the strength of formal systems, measure their expressive power, and establish relationships between different mathematical theories. This deeper understanding of proof structures enhances our grasp of mathematical foundations.

Foundational Approaches

Reverse Mathematics and Predicativity

- Reverse mathematics investigates the minimal axioms required to prove theorems in mathematics
- Involves proving equivalences between mathematical statements and subsystems of second-order arithmetic
- Predicative mathematics avoids impredicative definitions, which define an object using a quantifier whose domain includes the object being defined
- Impredicativity, in contrast, allows such definitions (e.g., defining a set in terms of a collection that includes the set itself)
- Predicativism is a foundational stance that rejects impredicative definitions to avoid potential circularity or paradoxes

Foundational Programs and Constructive Reverse Mathematics

- Foundational programs are systematic approaches to providing rigorous foundations for mathematics
- Examples include Hilbert's program, logicism (Russell and Whitehead), and intuitionism (Brouwer)
- These programs aim to secure the foundations of mathematics using different philosophical and logical principles
- Constructive reverse mathematics is a variant of reverse mathematics using intuitionistic logic

- Investigates the constructive content of mathematical theorems and the axioms needed to prove them constructively

Consistency and Reducibility

Consistency Proofs and Relative Consistency

- Consistency proofs demonstrate that a formal system (e.g., a set theory like ZFC) is free from contradictions
- Relative consistency proofs show that if one system (S1) is consistent, then another system (S2) is also consistent
- For example, Gödel's relative consistency proof showed that if ZF set theory is consistent, then ZFC (ZF with the Axiom of Choice) is also consistent
- Relative consistency is a powerful tool for comparing the strength and reliability of different formal systems

Proof-Theoretic Reductions and Strength

- Proof-theoretic reductions are techniques for comparing the strength of formal systems
- System S1 is proof-theoretically reducible to S2 if the consistency of S1 can be proved in S2
- Proof-theoretic strength measures the power and expressiveness of a formal system
- Stronger systems can prove the consistency of weaker ones, but not vice versa (by Gödel's Incompleteness Theorems)
- Examples of proof-theoretic reductions include reducing Peano Arithmetic to primitive recursive arithmetic and reducing second-order arithmetic to first-order arithmetic

Ordinal Analysis

Proof-Theoretic Ordinals and Measuring Strength

- Proof-theoretic ordinals are ordinal numbers used to measure the strength of formal systems
- Each system is assigned an ordinal that represents the transfinite induction principles it can prove
- For example, the proof-theoretic ordinal of Peano Arithmetic is ϵ_0 , the least ordinal α such that $\omega^\alpha = \alpha$
- Ordinal analysis is the process of determining the proof-theoretic ordinal of a formal system
- Allows for fine-grained comparisons of proof-theoretic strength beyond what can be shown by consistency proofs alone
- Gentzen's consistency proof for Peano Arithmetic used transfinite induction up to ϵ_0 , establishing its proof-theoretic ordinal