Automated Reasoning

Artificial Intelligence

Jacopo Parretti

I Semester 2025-2026

Indice

Parte I

1 Introduction

1.1 Why Study Automated Reasoning?

Automated Reasoning is a fundamental field of artificial intelligence concerned with the design and implementation of computational procedures that enable machines to solve problems formulated in logical languages. These procedures involve **logical inference and search**, allowing systems to reason in a deductive manner through mechanical processes.

Since the inception of artificial intelligence at the Dartmouth Conference in 1956, one of the core challenges has been to demonstrate machine intelligence through the automated proving of mathematical and logical theorems. This capability represents a cornerstone of symbolic AI, where knowledge is represented symbolically and stored in computer memory, and systematic procedures are applied to solve complex problems.

Theorem proving traditionally requires human-level mathematical reasoning and insight. The goal of automated reasoning is to mechanize this process, creating algorithms and systems capable of performing logical deduction without human intervention.

A logical language consists of formal symbols and syntactic rules. Within the symbolic approach to artificial intelligence, knowledge is encoded using these symbols, which are stored in computer memory. The system then applies a sequence of mechanical steps to manipulate these symbols and arrive at solutions to the given problems.

2 Historical Foundations and Theoretical Limits

2.1 The Hilbert-Turing Perspective

The foundations of automated reasoning can be traced to fundamental questions in mathematical logic and computability. Since Alan Turing's seminal work on *Turing machines* in 1936, researchers have pursued David Hilbert's 1900 challenge: is it possible to develop a mechanical procedure—executed deterministically by a human computor—capable of solving any mathematical problem?

The Entscheidungsproblem: A central question that emerged from this pursuit asks whether it is possible to define a mechanical method to determine the validity of formulas in first-order logic.

Turing demonstrated that the halting problem is undecidable using Turing machines. This fundamental result can be reduced to the validity problem in first-order logic, providing a negative answer to Hilbert's challenge. This undecidability result forms one of the core theoretical foundations of automated reasoning, establishing fundamental limits on what can be mechanically computed.

2.2 Practical Applications of Automated Reasoning

Automated reasoning systems find application across multiple domains requiring formal verification and systematic problem-solving:

- **Planning**: Given an initial state, goal state, and formal model of agent actions, determine a sequence of moves that transforms the initial state into the goal state (utilizing SAT solvers)
- **Scheduling**: Generate temporal arrangements of tasks that satisfy complex temporal and resource constraints
- **Software Analysis and Verification**: Formal methods for ensuring program correctness, including:
  - Provable privacy guarantees in security protocols
  - Verification of distributed algorithms
  - Analysis of randomized algorithms and their probabilistic properties
- **Hybrid Approaches**: Integration of automated reasoning with machine learning techniques for enhanced problem-solving capabilities

These applications demonstrate the practical utility of automated reasoning in solving complex real-world problems that require rigorous logical analysis and systematic exploration of solution spaces.

3 Core Problems in Automated Reasoning

Automated reasoning fundamentally addresses three interconnected classes of decision problems in mathematical logic, each representing a distinct challenge in determining the truth properties of logical formulas.

3.1 Satisfiability Problems

A satisfiability problem (SAT) concerns determining whether a given logical formula has a truth assignment that makes it true. Formally, for a formula ϕ with free variables, we seek to determine

if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \phi$. This represents the existence problem—does there exist a model making the formula true?

3.2 Validity Problems

In contrast to satisfiability, validity problems ask whether a formula holds true under all possible interpretations. A formula ϕ is valid (denoted $\models \phi$) if every possible truth assignment to its variables results in ϕ evaluating to true. This corresponds to the universal quantification over all possible models—the formula must hold in every conceivable scenario.

3.3 Logical Consequence Problems

The logical consequence problem represents a generalization of validity, asking whether a conclusion necessarily follows from given premises. Given a set of assumptions $\Gamma = \{\phi_1, \phi_2, \ldots, \phi_n\}$ and a conjecture ψ, we determine if $\Gamma \models \psi$ (i.e., if ψ is true in every model where all formulas in Γ are true). This captures the essence of logical deduction—what conclusions are forced by the given information.

These three problems form the foundation of automated reasoning systems, with satisfiability serving as the most tractable and finding widespread application in practical problem-solving contexts.

4 Formal Language and Semantics

4.1 Logical Language: First-Order Logic

The formal language employed in automated reasoning is first-order logic (FOL), a substantial extension of propositional logic that incorporates quantification over individual elements. First-order logic achieves greater expressive power by introducing:

- **Individual variables** ranging over domain elements
- **Function symbols** for constructing complex terms
- **Predicate symbols** for expressing relations between elements
- **Quantifiers** ($\forall$, $\exists$) enabling statements about all or some elements

This enhanced expressiveness allows first-order logic to formalize mathematical structures, relationships, and properties in a manner impossible in propositional logic, which can only express truth-functional combinations of atomic propositions.

4.2 Signature and Universe of Discourse

A signature Σ provides the vocabulary for interpreting formulas within a specific domain. Formally, a signature is a triple $\Sigma = \langle S, \mathcal{F}, \mathcal{P} \rangle$ where:

- S is a set of sorts (disjoint domains of interpretation)
- $\mathcal{F}$ is a set of function symbols, each with an arity specifying the number of arguments
- $\mathcal{P}$ is a set of predicate symbols, which we use to write formulas, each with an associated arity

The signature defines the syntactic elements available for constructing formulas, while the universe of discourse (or domain) provides the semantic interpretation. Sorts correspond to types in programming languages, enabling the formal specification of heterogeneous domains with different categories of objects.

The signature $\Sigma = \langle S, \mathcal{F}, \mathcal{P} \rangle$ can be further decomposed to provide a complete syntactic framework. The set of function symbols $\mathcal{F}$ is typically partitioned into two distinct subsets:

- **Constant symbols** $C \subseteq \mathcal{F}$: These are function symbols of arity 0, representing named individuals in the domain. Constants have fixed interpretations and serve as proper names for specific domain elements.
- **Proper function symbols** $\mathcal{F} \setminus C$: These represent actual functions mapping tuples of domain elements to domain elements, each with arity $n \geq 1$.

Formally, a function symbol $f \in \mathcal{F}$ has type $s_1 \times s_2 \times \cdots \times s_n \to s$, where $s_1, \ldots, s_n, s \in S$ are sorts indicating the domain and codomain of the function. For example:

- $c :\to \texttt{int}$ (a constant symbol denoting a specific integer)
- $p :\to \texttt{color}$ (a constant symbol denoting a specific color)
- $f : \texttt{int} \times \texttt{int} \to \texttt{int}$ (a binary function like addition)

This decomposition enables precise specification of both individual elements (via constants) and functional relationships (via proper function symbols) within the formal language.

The complete signature structure $\langle S, C \cup (\mathcal{F} \setminus C) \cup \mathcal{P} \rangle$ provides the full syntactic foundation for expressing complex logical statements about structured domains.

Constants can be viewed as function symbols of arity 0—functions that require no arguments and directly denote specific domain elements. Additionally, to construct quantified formulas and express generality, we require a countably infinite set of variables $X = \{x, y, z, \dots\}$ disjoint from the signature symbols.

Thus, the function symbols may be comprehensively categorized as $\mathcal{F} = C \cup (\mathcal{F} \setminus C) \cup \mathcal{P}$, where $\mathcal{P}$ represents the set of predicate symbols. Each predicate symbol $p \in \mathcal{P}$ has arity $n \geq 0$ and type $s_1 \times s_2 \times \cdots \times s_n$, where the final sort represents the truth value (typically a boolean sort).

Predicate symbols differ fundamentally from function symbols in that they represent relations rather than functions—they evaluate to truth values rather than domain elements. For example:

- $\mathit{even} : \texttt{int} \to \texttt{bool}$ (unary predicate testing parity)
- $\mathit{less} : \texttt{int} \times \texttt{int} \to \texttt{bool}$ (binary relation for ordering)

The complete signature structure $\langle S, C \cup (\mathcal{F} \setminus C) \cup \mathcal{P} \rangle$ provides the full syntactic foundation for expressing complex logical statements about structured domains.

Constants can be viewed as function symbols of arity 0—functions that require no arguments and directly denote specific domain elements. Additionally, to construct quantified formulas and express generality, we require a countably infinite set of variables $X = \{x, y, z, \dots\}$ disjoint from the signature symbols.

5 Syntactic Foundations: Terms and Formulas

5.1 Terms: The Building Blocks

The most fundamental syntactic objects in first-order logic are **terms**, which represent individuals in the domain of discourse. The set of terms $\mathcal{T}_\Sigma(X)$ over signature Σ and variables X is inductively defined as the smallest set satisfying:

- **Constants**: If $c \in C$ is a constant symbol, then $c \in \mathcal{T}_\Sigma(X)$
- **Variables**: If $x \in X$ is a variable, then $x \in \mathcal{T}_\Sigma(X)$

- **Function application**: If $f \in \mathcal{F}$ is a function symbol of arity $n \geq 1$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(X)$ are terms, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(X)$

Terms can be conceptualized as finite trees where:

- Leaf nodes are constants or variables
- Internal nodes are function symbols applied to subterms

For example, the term $f(x, g(a), z)$ has the tree representation:

$$\begin{array}{ccc} & f & \\ & | & \\ x & g & z \\ & | & \\ & a & \end{array}$$

Terms may contain subterms, which correspond to subtrees in this representation. Complex terms are constructed recursively from simpler subterms, reflecting the hierarchical nature of the syntactic structure.

Terms may contain subterms, which correspond to subtrees in this representation. Complex terms are constructed recursively from simpler subterms, reflecting the hierarchical nature of the syntactic structure.

5.2 From Terms to Atomic Formulas

Having established the notion of terms as representations of individuals, we now introduce atomic formulas to express relationships between these individuals. An **atomic formula** (or **atom**) is constructed by applying a predicate symbol to an appropriate number of terms.

Formally, if $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 0$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(X)$ are terms, then $P(t_1, \ldots, t_n)$ is an atomic formula. When $n = 0$, the atomic formula reduces to the predicate symbol itself P, which can be interpreted as a propositional constant.

The truth value of an atomic formula $P(t_1, \ldots, t_n)$ depends on the interpretation of the predicate P and the denotations of the terms $t_1, \ldots, t_n$ in a given model.

5.3 Literals: Positive and Negative Atomic Formulas

A **literal** is either an atomic formula (positive literal) or its negation (negative literal). Formally, if ϕ is an atomic formula, then:

- ϕ is a positive literal
- $\neg\phi$ is a negative literal

Literals represent the basic building blocks for constructing more complex logical formulas and play a crucial role in automated reasoning systems, particularly in resolution-based theorem proving.

5.4 Logical Connectives

Logical connectives provide the means to construct complex formulas from simpler ones. These operators enable the expression of compound logical statements through systematic combination of atomic formulas and literals.

The primary logical connectives include:

- **Negation** ($\neg$): Expresses logical denial or contradiction
- **Conjunction** ($\wedge$): Represents logical conjunction (AND)
- **Disjunction** ($\vee$): Represents logical disjunction (OR)
- **Implication** ($\rightarrow$): Expresses logical implication (IF...THEN)
- **Biconditional** ($\leftrightarrow$): Represents logical equivalence (IF AND ONLY IF)

5.5 Formulas: The Complete Syntax

The set of **formulas** (or **well-formed formulas**) $\mathcal{F}_\Sigma(X)$ over signature Σ and variables X is inductively defined as the smallest set satisfying:

- **Atomic formulas**: If $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 0$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(X)$, then $P(t_1, \ldots, t_n) \in \mathcal{F}_\Sigma(X)$
- **Negation**: If $\phi \in \mathcal{F}_\Sigma(X)$, then $(\neg\phi) \in \mathcal{F}_\Sigma(X)$
- **Binary connectives**: If $\phi, \psi \in \mathcal{F}_\Sigma(X)$, then:
  - $(\phi \wedge \psi) \in \mathcal{F}_\Sigma(X)$ (conjunction)
  - $(\phi \vee \psi) \in \mathcal{F}_\Sigma(X)$ (disjunction)
  - $(\phi \rightarrow \psi) \in \mathcal{F}_\Sigma(X)$ (implication)
  - $(\phi \leftrightarrow \psi) \in \mathcal{F}_\Sigma(X)$ (biconditional)

This inductive definition ensures that all syntactically valid logical expressions can be systematically constructed from the basic building blocks of atomic formulas through the application of logical connectives.

This inductive definition ensures that all syntactically valid logical expressions can be systematically constructed from the basic building blocks of atomic formulas through the application of logical connectives.

5.6 Quantifiers: Universality and Existence

To complete the syntax of first-order logic, we introduce quantifiers that express generality and existence over the domain of discourse:

- **Universal quantifier** ($\forall$): If $\phi \in \mathcal{F}_\Sigma(X)$ is a formula and $x \in X$ is a variable, then $(\forall x\, \phi) \in \mathcal{F}_\Sigma(X)$
- **Existential quantifier** ($\exists$): If $\phi \in \mathcal{F}_\Sigma(X)$ is a formula and $x \in X$ is a variable, then $(\exists x\, \phi) \in \mathcal{F}_\Sigma(X)$

Quantifiers bind variables within their scope, enabling the expression of properties that hold for all elements of the domain (universal quantification) or for at least one element (existential quantification).

5.7 Illustrative Example: Geometric Axioms

Consider the following first-order formula expressing that any two distinct points determine a unique line:

$$\forall x\, \forall y\, ((P(x) \wedge P(y) \wedge x \neq y) \rightarrow \exists z\, (L(z) \wedge Q(z, x, y) \wedge \forall w\, (L(w) \wedge Q(w, x, y) \rightarrow w = z)))$$

Where:

- $P(x)$ denotes that x is a point
- $L(z)$ denotes that z is a line
- $Q(z, x, y)$ denotes that line z passes through points x and y
- $x \neq y$ expresses that points x and y are distinct

This axiom formalizes the geometric principle that given any two distinct points in space, there exists exactly one line passing through both points, capturing a fundamental property of Euclidean geometry within the framework of first-order logic.

This axiom formalizes the geometric principle that given any two distinct points in space, there exists exactly one line passing through both points, capturing a fundamental property of Euclidean geometry within the framework of first-order logic.

5.8 Closed Formulas and Variable Scope

A **closed formula** (or **sentence**) is a formula containing no free variables—all variables are bound by quantifiers. A variable is **free** in a formula if it appears outside the scope of any quantifier that binds it; conversely, a variable is **bound** if it appears within the scope of a quantifier.

Formulas may contain both free and bound occurrences of variables. For proper semantic interpretation, variables must be **standardized apart**, meaning that no variable appears both free and bound in the same formula, and distinct quantifiers use distinct bound variables.