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 $X = \{x, y, z, \ldots\}$ 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 $X = \{x, y, z, \ldots\}$ 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:

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

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.

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.

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.

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.

Parte II

Semantic Foundations

6 Interpretations and Models

6.1 The Concept of Interpretation

To assign meaning to syntactic formulas in first-order logic, we require a formal notion of **interpretation**. An interpretation $\mathcal{I}$ provides a semantic mapping from the syntactic elements of a signature to concrete mathematical objects.

In the unsorted case, an interpretation consists of:

- A non-empty set D called the **domain** (or universe of discourse)
- An interpretation function Φ that maps:
  - Each predicate symbol $P \in \mathcal{P}$ of arity n to a subset $\Phi(P) \subseteq D^n$ (representing an n-ary relation over the domain)
  - Each function symbol $f \in \mathcal{F}$ of arity n to a function $\Phi(f) : D^n \to D$
  - Each constant symbol $c \in C$ to a specific element $\Phi(c) \in D$

The interpretation function Φ bridges the gap between syntax and semantics, transforming abstract symbols into concrete mathematical entities.

6.2 Example: Interpreting a Formula over Natural Numbers

Consider the first-order formula:

$$\forall x \, (\neg P(x) \to P(f(x)))$$

This formula contains:

- A unary predicate symbol P
- A unary function symbol f
- A universally quantified variable x

Since the formula is closed (contains no free variables), its truth value depends entirely on the chosen interpretation.

Constructing an interpretation: Let us define an interpretation $\mathcal{I} = \langle D, \Phi \rangle$ where:

- **Domain**: $D = \mathbb{N}$ (the set of natural numbers)
- **Function interpretation**: $\Phi(f) : \mathbb{N} \to \mathbb{N}$ is the successor function, defined by $\Phi(f)(n) = n + 1$ for all $n \in \mathbb{N}$
- **Predicate interpretation**: $\Phi(P) = \{0, 2, 4, 6, \dots\} = \{2k \mid k \in \mathbb{N}\}$ (the set of even natural numbers)

6.3 Semantic Evaluation

Under this interpretation $\mathcal{I}$, the formula $\forall x\,(\neg P(x) \to P(f(x)))$ receives the following semantic reading:

For all natural numbers x, if x is not even, then the successor of x is even.

To determine whether this formula is true in interpretation $\mathcal{I}$, we must verify whether the implication holds for every element of the domain:

- If x is odd (i.e., $x \notin \Phi(P)$), then $x+1$ must be even (i.e., $x+1 \in \Phi(P)$)

This statement is indeed true in the natural numbers: the successor of any odd number is even. Therefore, the formula is **satisfied** by interpretation $\mathcal{I}$, and we write $\mathcal{I} \models \forall x\,(\neg P(x) \to P(f(x)))$.

6.4 Formal Verification of Satisfaction

To formally verify that $\mathcal{I} \models \forall x\,(\neg P(x) \to P(f(x)))$, we must check that for all $n \in \mathbb{N}$, the interpretation satisfies the implication with the substitution $\{x \leftarrow n\}$.

By the semantics of implication, $\mathcal{I}$ satisfies $(\neg P(x) \to P(f(x)))$ with assignment $\{x \leftarrow n\}$ if and only if:

- Either $\mathcal{I}$ does not satisfy $\neg P(x)$ with $\{x \leftarrow n\}$ (i.e., $\mathcal{I}$ satisfies $P(x)$ with $\{x \leftarrow n\}$), or
- $\mathcal{I}$ satisfies $P(f(x))$ with $\{x \leftarrow n\}$

Equivalently, for all $n \in \mathbb{N}$, either $n \in \Phi(P)$ or $\Phi(f)(n) \in \Phi(P)$.

Truth condition for atomic formulas: An atomic formula $P(t)$ is true under interpretation $\mathcal{I}$ if and only if the interpretation of the term t belongs to the interpretation of the predicate $\Phi(P)$.

In our example, for all $n \in \mathbb{N}$:

- Either $n \in \Phi(P)$ (meaning n is even), or
- $\Phi(f)(n) = n+1 \in \Phi(P)$ (meaning the successor of n is even)

This condition holds for all natural numbers, confirming that the formula is satisfied by the interpretation.

6.5 Formal Definition of Interpretation

Having illustrated the concept through examples, we now provide the complete formal definition of an interpretation in first-order logic.

An **interpretation** $\mathcal{I}$ consists of a triple $\mathcal{I} = \langle D, \Phi, \beta \rangle$ where:

- **Domain**: D is a non-empty set representing the universe of discourse
- **Interpretation function** Φ maps signature symbols to semantic objects:
  - For each constant symbol $c \in C$: $\Phi(c) = d \in D$
  - For each function symbol $f \in \mathcal{F}$ of arity $n \geq 1$: $\Phi(f) : D^n \to D$, where D^n denotes the n-fold Cartesian product $D \times D \times \cdots \times D$
  - For each predicate symbol $P \in \mathcal{P}$ of arity $n \geq 0$: $\Phi(P) \subseteq D^n$ (an n-ary relation over D)
- **Variable assignment** β maps each variable to a domain element:

– $\beta : X \to D$ such that $\beta(x) = d \in D$ for all variables $x \in X$

Assignment update notation: Given an assignment β and a substitution $x \leftarrow d$, we define the updated assignment $\beta[x \leftarrow d]$ as:

$$\beta[x \leftarrow d](y) = \begin{cases} d & \text{if } y = x \\ \beta(y) & \text{otherwise} \end{cases}$$

This notation allows us to formally express the evaluation of formulas with different variable bindings, which is essential for defining the semantics of quantified formulas.

7 Satisfaction Relation

7.1 The Satisfaction Relation

Having defined interpretations and variable assignments, we now formalize the central semantic concept: when does an interpretation **satisfy** a formula? The satisfaction relation, denoted $\mathcal{I} \models_\beta \phi$, specifies the conditions under which a formula ϕ is true in interpretation $\mathcal{I}$ with variable assignment β.

Let F and G denote arbitrary formulas. The satisfaction relation is defined inductively on the structure of formulas.

7.2 Interpretation of Terms

Before defining satisfaction for formulas, we must specify how terms are interpreted. Given an interpretation $\mathcal{I} = \langle D, \Phi, \beta \rangle$, the interpretation of a term t, denoted $\Phi_\beta(t)$, is defined recursively:

- **Constants**: $\Phi_\beta(c) = \Phi(c) \in D$
- **Variables**: $\Phi_\beta(x) = \beta(x) \in D$
- **Function application**: $\Phi_\beta(f(t_1, \ldots, t_n)) = \Phi(f)(\Phi_\beta(t_1), \ldots, \Phi_\beta(t_n)) \in D$

Each term evaluates to a specific element of the domain under a given interpretation.

7.3 Satisfaction for Atomic Formulas

Base case (atomic formulas): An interpretation $\mathcal{I}$ satisfies an atomic formula $P(t_1, \ldots, t_n)$ under assignment β if and only if the tuple of interpreted terms belongs to the interpretation of the predicate:

$$\mathcal{I} \models_\beta P(t_1, \ldots, t_n) \quad \text{iff} \quad (\Phi_\beta(t_1), \ldots, \Phi_\beta(t_n)) \in \Phi(P)$$

7.4 Satisfaction for Logical Connectives

The satisfaction relation extends to compound formulas through the following inductive clauses. Note that all satisfaction relations are parameterized by the variable assignment β, written as $\models_\beta$.

- **Negation**:
$$\mathcal{I} \models_\beta \neg F \quad \text{iff} \quad \mathcal{I} \not\models_\beta F$$
- **Conjunction**:
$$\mathcal{I} \models_\beta (F \wedge G) \quad \text{iff} \quad \mathcal{I} \models_\beta F \text{ and } \mathcal{I} \models_\beta G$$

- **Disjunction**:
$$\mathcal{I} \models_\beta (F \vee G) \quad \text{iff} \quad \mathcal{I} \models_\beta F \text{ or } \mathcal{I} \models_\beta G$$
- **Implication**:
$$\mathcal{I} \models_\beta (F \to G) \quad \text{iff} \quad \mathcal{I} \not\models_\beta F \text{ or } \mathcal{I} \models_\beta G$$
Equivalently: if $\mathcal{I} \models_\beta F$ then $\mathcal{I} \models_\beta G$
- **Biconditional**:
$$\mathcal{I} \models_\beta (F \leftrightarrow G) \quad \text{iff} \quad \mathcal{I} \models_\beta F \text{ if and only if } \mathcal{I} \models_\beta G$$

7.5 Satisfaction for Quantified Formulas

Quantifiers introduce variable bindings that range over the entire domain:

- **Universal quantification**:
$$\mathcal{I} \models_\beta \forall x\, G \quad \text{iff} \quad \text{for all } d \in D,\ \mathcal{I} \models_{\beta[x \leftarrow d]} G$$
The formula $\forall x\, G$ is satisfied if G is satisfied under every possible assignment to x.
- **Existential quantification**:
$$\mathcal{I} \models_\beta \exists x\, G \quad \text{iff} \quad \text{there exists } d \in D \text{ such that } \mathcal{I} \models_{\beta[x \leftarrow d]} G$$
The formula $\exists x\, G$ is satisfied if G is satisfied under at least one assignment to x.

This completes the inductive definition of the satisfaction relation, providing a complete semantics for first-order logic.

7.6 Example: Quantifier Evaluation

Consider the formula $\forall x\, \exists y\, R(x, y)$, which can be read as "for all x, there exists y such that $x < y$".

Let $\mathcal{I} = \langle \mathbb{N}, \Phi \rangle$ where $\Phi(R) = \{(n, m) \in \mathbb{N} \times \mathbb{N} \mid n < m\}$ (the standard ordering on natural numbers).

To verify $\mathcal{I} \models \forall x\, \exists y\, R(x, y)$, we apply the semantic definitions:

$$\begin{aligned}
\mathcal{I} \models \forall x\, \exists y\, R(x, y) \quad &\text{iff} \quad \text{for all } n \in \mathbb{N},\ \mathcal{I} \models_{\beta[x \leftarrow n]} \exists y\, R(x, y) \\
&\text{iff} \quad \text{for all } n \in \mathbb{N},\ \text{there exists } m \in \mathbb{N} \text{ such that } \mathcal{I} \models_{\beta[x \leftarrow n, y \leftarrow m]} R(x, y) \\
&\text{iff} \quad \text{for all } n \in \mathbb{N},\ \text{there exists } m \in \mathbb{N} \text{ such that } (n, m) \in \Phi(R) \\
&\text{iff} \quad \text{for all } n \in \mathbb{N},\ \text{there exists } m \in \mathbb{N} \text{ such that } n < m
\end{aligned}$$

This statement is **false** in $\mathbb{N}$ because there is no natural number greater than all natural numbers. However, it would be true in $\mathbb{Z}$ or $\mathbb{Q}$, demonstrating how truth depends on the chosen interpretation.

8 Semantic Properties of Formulas

Having established the satisfaction relation, we now define fundamental semantic properties that classify formulas according to their truth behavior across different interpretations.

8.1 Satisfiability

A formula F is **satisfiable** if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models F$. In this case, we say that $\mathcal{I}$ is a **model** of F.

$$F \text{ is satisfiable} \quad \Leftrightarrow \quad \exists \mathcal{I} : \mathcal{I} \models F$$

8.2 Validity

A formula F is **valid** (or a **tautology**) if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models F$. Valid formulas are true in every possible interpretation.

$$F \text{ is valid} \quad \Leftrightarrow \quad \forall \mathcal{I} : \mathcal{I} \models F$$

We denote validity by $\models F$.

8.3 Unsatisfiability

A formula F is **unsatisfiable** (or **contradictory**) if there exists no interpretation $\mathcal{I}$ such that $\mathcal{I} \models F$. Unsatisfiable formulas are false in every interpretation.

$$F \text{ is unsatisfiable} \quad \Leftrightarrow \quad \nexists \mathcal{I} : \mathcal{I} \models F \quad \Leftrightarrow \quad \forall \mathcal{I} : \mathcal{I} \not\models F$$

8.4 Invalidity

A formula F is **invalid** (or **falsifiable**) if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \not\models F$. In this case, we say that $\mathcal{I}$ is a **countermodel** (or **counterexample**) for F.

$$F \text{ is invalid} \quad \Leftrightarrow \quad \exists \mathcal{I} : \mathcal{I} \not\models F$$

8.5 Relationships Between Semantic Properties

These semantic properties are intimately related through negation:

Teorema 8.1 (Validity and Unsatisfiability)**.** *A formula F is valid if and only if $\neg F$ is unsatisfiable.*

Dimostrazione. Suppose F is valid. Then for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models F$, which implies $\mathcal{I} \not\models \neg F$. Therefore, $\neg F$ is unsatisfiable.

Conversely, if $\neg F$ is unsatisfiable, then for all interpretations $\mathcal{I}$, we have $\mathcal{I} \not\models \neg F$, which implies $\mathcal{I} \models F$. Therefore, F is valid. □

Teorema 8.2 (Satisfiability and Invalidity)**.** *A formula F is satisfiable if and only if $\neg F$ is invalid.*

Dimostrazione. F is satisfiable if and only if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models F$, which is equivalent to $\mathcal{I} \not\models \neg F$. This is precisely the condition for $\neg F$ to be invalid. □

These relationships demonstrate the duality between validity and unsatisfiability, and between satisfiability and invalidity, providing a complete characterization of the semantic landscape of first-order logic.

9 Logical Consequence

9.1 The Three Fundamental Query Types

Automated reasoning systems must be capable of resolving three fundamental types of semantic queries:

1. **Satisfiability queries**: Is formula F satisfiable?
2. **Validity queries**: Is formula F valid?
3. **Logical consequence queries**: Does F follow logically from a set of hypotheses H?

9.2 Definition of Logical Consequence

Let $H = \{\phi_1, \phi_2, \ldots, \phi_n\}$ be a set of formulas (the **hypotheses** or **premises**) and let F be a formula (the **conclusion**). We say that F is a **logical consequence** of H, written $H \models F$, and read as "H entails F" or "F follows logically from H", if:

$$H \models F \quad \Leftrightarrow \quad \text{for all interpretations } \mathcal{I}, \text{ if } \mathcal{I} \models H \text{ then } \mathcal{I} \models F$$

In other words, F is a logical consequence of H if every model of H is also a model of F. Here, $\mathcal{I} \models H$ means that $\mathcal{I}$ satisfies all formulas in H.

9.3 The Deduction Theorem

The deduction theorem establishes a fundamental connection between logical consequence and validity, reducing the problem of checking entailment to the problem of checking validity.

Teorema 9.1 (Deduction Theorem)**.** *Let H be a set of formulas and F be a formula. Then:*

$$H \models F \quad \textit{iff} \quad \models (H \to F)$$

where $H \to F$ denotes the implication from the conjunction of all formulas in H to F.

Dimostrazione. We prove both directions:

($\Rightarrow$) Assume $H \models F$. We must show that $H \to F$ is valid, i.e., for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models H \to F$.

Let $\mathcal{I}$ be an arbitrary interpretation. By the semantics of implication, $\mathcal{I} \models H \to F$ holds if either:

- $\mathcal{I} \not\models H$ (the antecedent is false), or
- $\mathcal{I} \models F$ (the consequent is true)

If $\mathcal{I} \not\models H$, then $\mathcal{I} \models H \to F$ trivially. If $\mathcal{I} \models H$, then by the hypothesis $H \models F$, we have $\mathcal{I} \models F$, so $\mathcal{I} \models H \to F$. In both cases, $\mathcal{I} \models H \to F$.

($\Leftarrow$) Conversely, suppose $H \to F$ is valid. Then for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models H \to F$. By the semantics of implication, for all $\mathcal{I}$, either $\mathcal{I} \not\models H$ or $\mathcal{I} \models F$. Therefore, for all $\mathcal{I}$ such that $\mathcal{I} \models H$, we must have $\mathcal{I} \models F$, which is precisely the definition of $H \models F$. □

9.4 Notational Ambiguity of $\models$

The symbol $\models$ is used in three distinct but related contexts:

- **Satisfaction**: $\mathcal{I} \models F$ means interpretation $\mathcal{I}$ satisfies formula F
- **Validity**: $\models F$ means formula F is valid (satisfied by all interpretations)
- **Entailment**: $H \models F$ means formula F is a logical consequence of the set of formulas H

While this overloading may initially seem confusing, these three uses are semantically coherent and represent different aspects of the same fundamental semantic relation.

9.5 Reduction to Unsatisfiability

A fundamental result connects logical consequence to unsatisfiability, providing another characterization of entailment.

Teorema 9.2 (Logical Consequence via Unsatisfiability)**.** *Let H be a set of formulas and F be a formula. Then:*

$$H \models F \quad \textit{iff} \quad H \cup \{\neg F\} \textit{ is unsatisfiable}$$

Dimostrazione. We prove both directions:

($\Rightarrow$) Assume $H \models F$. We must show that $H \cup \{\neg F\}$ is unsatisfiable.

Suppose for contradiction that there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models H \cup \{\neg F\}$. Then $\mathcal{I} \models H$ and $\mathcal{I} \models \neg F$, which implies $\mathcal{I} \not\models F$. However, since $H \models F$ and $\mathcal{I} \models H$, we must have $\mathcal{I} \models F$, which is a contradiction. Therefore, $H \cup \{\neg F\}$ is unsatisfiable.

($\Leftarrow$) Assume $H \cup \{\neg F\}$ is unsatisfiable. We must show that $H \models F$.

Let $\mathcal{I}$ be any interpretation such that $\mathcal{I} \models H$. Since $H \cup \{\neg F\}$ is unsatisfiable, there is no interpretation that satisfies both H and $\neg F$. Therefore, $\mathcal{I} \not\models \neg F$, which implies $\mathcal{I} \models F$. Since this holds for all interpretations satisfying H, we conclude that $H \models F$. □

10 Refutational Reasoning

10.1 Proof by Contradiction

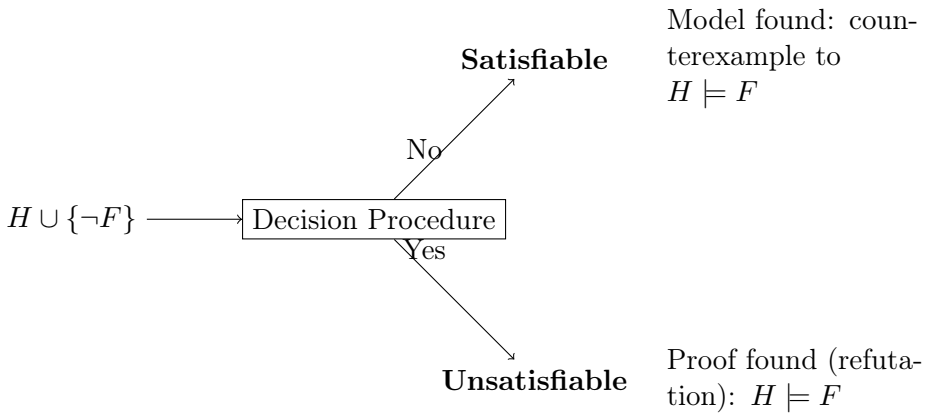
Refutational reasoning (or **proof by refutation**) is a fundamental technique in automated reasoning based on the principle of *reductio ad absurdum*. To prove that a conclusion F follows from hypotheses H, we assume the negation of the conclusion and derive a contradiction.

The method proceeds as follows:

1. **Goal**: Prove $H \models F$ (where H represents assumptions and F represents the conjecture)
2. **Method**: Show that $H \cup \{\neg F\}$ is unsatisfiable
3. **Conclusion**: If $H \cup \{\neg F\}$ is unsatisfiable, then $H \models F$ by the previous theorem

10.2 The Refutational Framework

The automated reasoning process can be formalized as follows:



Two possible outcomes:

- **Unsatisfiable**: The procedure derives a contradiction, producing a **refutation** (proof) that $H \models F$
- **Satisfiable**: The procedure finds a model $\mathcal{I}$ of $H \cup \{\neg F\}$, which serves as a **counterexample** showing that $H \not\models F$

This refutational approach forms the foundation of modern automated theorem provers and SAT solvers, reducing the problem of proving logical consequence to the problem of detecting unsatisfiability.

Parte III

Free Variables and Satisfiability

11 Satisfiability of Formulas with Free Variables

11.1 The Problem of Free Variables

In our previous definition of satisfiability, we considered the semantics of logical connectives and quantifiers, but we did not fully address the treatment of **free variables**. Recall that a variable can be either:

- **Bound**: The variable falls within the scope of a quantifier
- **Free**: The variable does not fall within the scope of any quantifier

The presence of free variables requires careful consideration when determining satisfiability, as their interpretation depends on the variable assignment β.

11.2 Motivating Example

Consider the following formula:

$$F : \forall x \, (f(x, x) = x \wedge R(x, y))$$

This formula contains:

- A bound variable x (quantified by $\forall$)
- A free variable y (not quantified)

11.3 Semantic Evaluation with Free Variables

To determine whether an interpretation $\mathcal{I} = \langle D, \Phi \rangle$ with assignment β satisfies F, we write $\mathcal{I} \models_\beta F$ and proceed as follows:

By the semantics of universal quantification, we need to check that for all $d \in D$:

$$\mathcal{I} \models_{\beta[x \leftarrow d]} (f(x, x) = x \wedge R(x, y))$$

This requires verifying both conjuncts under the updated assignment $\beta[x \leftarrow d]$:

First conjunct: $\mathcal{I} \models_{\beta[x \leftarrow d]} f(x, x) = x$

This holds if and only if:

$$(\Phi(f)(d, d), d) \in \Phi(=)$$

where $\Phi(=)$ represents the identity relation. In other words, we require that $\Phi(f)(d, d) = d$ for all $d \in D$.

Second conjunct: $\mathcal{I} \models_{\beta[x \leftarrow d]} R(x, y)$

This holds if and only if:

$$(d, \beta(y)) \in \Phi(R)$$

Let $\beta(y) = d'$ for some $d' \in D$. Then we require $(d, d') \in \Phi(R)$ for all $d \in D$.

11.4 Existential Closure

The key observation is that the satisfiability of a formula with free variables is intimately connected to its **existential closure**.

Definizione 11.1 (Existential Closure)**.** The existential closure of a formula F, denoted $\exists^* F$, is obtained by adding an existential quantifier for every free variable in F.

For example, if F contains free variables $y_1, y_2, \ldots, y_n$, then:

$$\exists^* F = \exists y_1 \, \exists y_2 \, \cdots \, \exists y_n \, F$$

Returning to our example, if we consider the existential closure by quantifying the free variable y:

$$\exists y \, \forall x \, (f(x, x) = x \wedge R(x, y))$$

Under this interpretation, we need to verify that there exists some $d' \in D$ such that for all $d \in D$:

$$\mathcal{I} \models_{\beta[x \leftarrow d, y \leftarrow d']} R(x, y)$$

which means $(d, d') \in \Phi(R)$ for all $d \in D$.

11.5 Satisfiability for Open Formulas

We now provide the complete definition of satisfiability that handles both closed and open formulas.

Definizione 11.2 (Satisfiability - Complete Definition)**.** A formula F is **satisfiable** if:

- **Case 1 (Closed formula)**: If F is a sentence (contains no free variables), then F is satisfiable if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models F$.
- **Case 2 (Open formula)**: If F contains free variables, then F is satisfiable if there exists an interpretation $\mathcal{I}$ and a variable assignment β such that $\mathcal{I} \models_\beta F$.

Teorema 11.1 (Satisfiability and Existential Closure)**.** *A formula F with free variables is satisfiable if and only if its existential closure $\exists^* F$ is satisfiable.*

Dimostrazione. Let F be a formula with free variables $y_1, \ldots, y_n$, and let $\exists^* F = \exists y_1 \, \cdots \, \exists y_n \, F$.

($\Rightarrow$) Suppose F is satisfiable. Then there exists an interpretation $\mathcal{I}$ and assignment β such that $\mathcal{I} \models_\beta F$. Let $d_i = \beta(y_i)$ for $i = 1, \ldots, n$. Then by the semantics of existential quantification applied repeatedly, $\mathcal{I} \models \exists^* F$.

($\Leftarrow$) Suppose $\exists^* F$ is satisfiable. Then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \exists y_1 \, \cdots \, \exists y_n \, F$. By the semantics of existential quantification, there exist $d_1, \ldots, d_n \in D$ such that $\mathcal{I} \models_{\beta[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n]} F$. Therefore, F is satisfiable. □

This result demonstrates that when checking satisfiability of formulas with free variables, we can equivalently check the satisfiability of their existential closures, treating free variables as existentially quantified.

11.6 Universal Closure and Validity

Dually, we can consider the **universal closure** of a formula with free variables.

Teorema 11.2 (Validity and Universal Closure)**.** *A formula F with free variables is valid if and only if its universal closure $\forall^* F$ is valid.*

The intuition is straightforward: if F contains free variables, then F is valid if for all interpretations $\mathcal{I}$ and for all assignments β, we have $\mathcal{I} \models_\beta F$. This is precisely the condition for the universal closure $\forall^* F$ to be valid.

12 Decidability and Semidecidability in First-Order Logic

12.1 Computational Complexity of First-Order Logic

While certain restricted fragments of first-order logic admit decision procedures, the general case presents fundamental computational limitations:

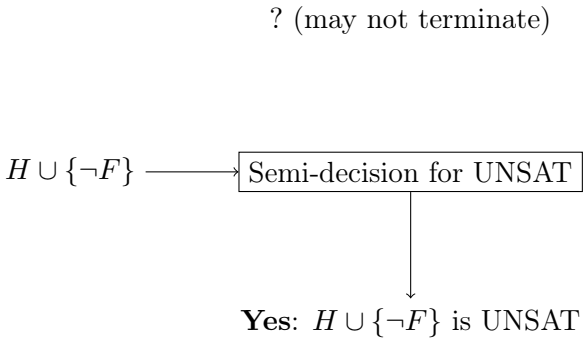
- **Unsatisfiability** in first-order logic is **semidecidable** (recursively enumerable)
- **Satisfiability** in first-order logic is **not even semidecidable**

Nota. Recall that if a decision problem A is semidecidable and its complement $\neg A$ is also semidecidable, then A is decidable. The decision procedure can be obtained by running both semidecision procedures in parallel: if the instance belongs to A, the first procedure will eventually terminate with "yes"; if the instance belongs to $\neg A$, the second procedure will eventually terminate with "yes".

12.2 Review: Semi-Decision Procedures

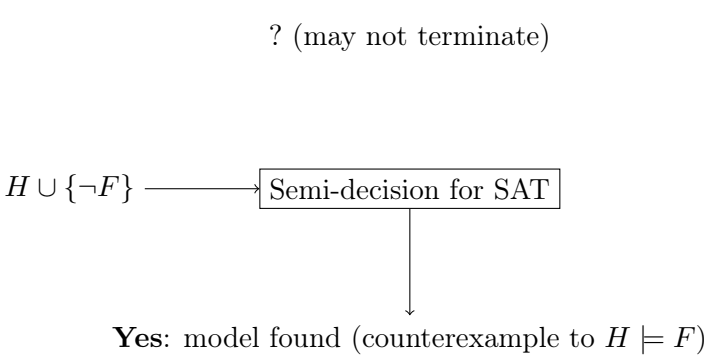
Consider the problem of determining whether $H \models F$ (logical consequence).

Semi-decision procedure for unsatisfiability:



If the procedure terminates with "yes", we have proven $H \models F$ by refutation. However, if $H \cup \{\neg F\}$ is satisfiable, the procedure may run indefinitely.

Semi-decision procedure for satisfiability:



If this procedure terminates with "yes", we have found a countermodel showing $H \not\models F$. However, such a procedure does not exist for general first-order logic.

12.3 The Need for Restricted Theories

To obtain decision procedures, we must restrict our attention to **decidable fragments** of first-order logic. This motivates the study of **first-order theories**—logical systems with additional structure that enable algorithmic reasoning.

13 First-Order Theories

13.1 Definition of a First-Order Theory

Definizione 13.1 (First-Order Theory)**.** A **first-order theory** is a pair $\mathcal{T} = \langle \Sigma, \mathcal{A} \rangle$ where:

- Σ is a **signature** (defining the vocabulary of the theory)
- $\mathcal{A}$ is a set of **axioms**—closed formulas (sentences) that we assume to hold in all models of the theory

The axioms define properties of specific symbols in the signature, giving them fixed interpretations across all models of the theory.

13.2 The Theory of Equality

A fundamental example is the **theory of equality** $\mathcal{T}_E = \langle \Sigma_E, \mathcal{A}_E \rangle$.

Signature: The signature of the theory of equality is:

$$\Sigma_E = \{=\} \cup C \cup \mathcal{F} \cup \mathcal{P}$$

where:

- $=$ is the equality predicate (interpreted symbol)
- C is a set of constant symbols (uninterpreted)
- $\mathcal{F}$ is a set of function symbols (uninterpreted)
- $\mathcal{P}$ is a set of predicate symbols (uninterpreted)

Example signature: Consider $\Sigma = \{a, b, f, Q, =\}$. The symbols a, b, f, Q are **free** (uninterpreted), whereas $=$ (equality) is **defined** by the axioms of the theory.

13.3 Axioms of Equality

The axiom set $\mathcal{A}_E$ consists of formulas that characterize equality as a congruence relation.

Equivalence relation axioms:

$$\begin{aligned} &\forall x\,(x = x) && \text{(Reflexivity)} \\ &\forall x\,\forall y\,(x = y \to y = x) && \text{(Symmetry)} \\ &\forall x\,\forall y\,\forall z\,((x = y \land y = z) \to x = z) && \text{(Transitivity)} \end{aligned}$$

These three axioms establish that equality is an equivalence relation.

Congruence axioms for function symbols: For each function symbol $f \in \mathcal{F}$ of arity n:

$$\forall \vec{x}\,\forall \vec{y}\left(\bigwedge_{i=1}^{n} x_i = y_i\right) \to f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.

This axiom states that f respects equality: equal inputs produce equal outputs.

Congruence axioms for predicate symbols: For each predicate symbol $R \in \mathcal{P}$ of arity n:

$$\forall \vec{x} \forall \vec{y} \left(\bigwedge_{i=1}^{n} x_i = y_i \right) \to (R(x_1, \ldots, x_n) \leftrightarrow R(y_1, \ldots, y_n))$$

This axiom states that R respects equality: the truth value of R depends only on the equivalence classes of its arguments.

13.4 Models of a Theory

Definizione 13.2 ($\mathcal{T}$-Model)**.** A **$\mathcal{T}$-model** is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{A}$ (i.e., $\mathcal{I}$ satisfies all axioms in $\mathcal{A}$).

13.5 Satisfiability and Validity Relative to a Theory

Definizione 13.3 ($\mathcal{T}$-Satisfiability)**.** A formula G is **$\mathcal{T}$-satisfiable** if there exists a $\mathcal{T}$-model $\mathcal{I}$ such that $\mathcal{I} \models G$.

Definizione 13.4 ($\mathcal{T}$-Validity)**.** A formula G is **$\mathcal{T}$-valid** if for all $\mathcal{T}$-models $\mathcal{I}$, we have $\mathcal{I} \models G$.

Equivalently, G is a logical consequence of the axioms: $\mathcal{A} \models G$.

13.6 Example: Satisfiability vs. $\mathcal{T}_E$-Satisfiability

Consider the set of formulas:

$$S = \{P(a), \neg P(b), a = b\}$$

Claim: S is satisfiable in general first-order logic, but S is **not $\mathcal{T}_E$-satisfiable**.

Dimostrazione. **Satisfiability**: We can construct an interpretation $\mathcal{I}$ with domain $D = \{d_1, d_2\}$ where:

- $\Phi(a) = d_1$, $\Phi(b) = d_2$
- $\Phi(P) = \{d_1\}$
- $\Phi(=) = \{(d_1, d_1), (d_2, d_2), (d_1, d_2), (d_2, d_1)\}$ (universal relation)

Then $\mathcal{I} \models P(a)$, $\mathcal{I} \models \neg P(b)$, and $\mathcal{I} \models a = b$, so $\mathcal{I} \models S$.

$\mathcal{T}_E$-Unsatisfiability: In any $\mathcal{T}_E$-model, equality must satisfy the congruence axioms. If $\mathcal{I} \models a = b$ and $\mathcal{I} \models P(a)$, then by the congruence axiom for P, we must have $\mathcal{I} \models P(b)$, contradicting $\mathcal{I} \models \neg P(b)$. Therefore, no $\mathcal{T}_E$-model satisfies S. □

This example illustrates that satisfiability relative to a theory is more restrictive than general satisfiability, as models must respect the additional constraints imposed by the theory's axioms.

14 Quantifier-Free Fragments and Decidability

14.1 The Quantifier-Free Fragment of $\mathcal{T}_E$

While the full theory of equality $\mathcal{T}_E$ is undecidable, its **quantifier-free fragment** admits a decision procedure.

Definizione 14.1 (Quantifier-Free Fragment)**.** The **quantifier-free fragment** of first-order logic consists of all formulas that do not contain quantifiers ($\forall$ or $\exists$).

For a theory $\mathcal{T}$, the quantifier-free fragment allows quantifiers to appear only in the axioms, not in the input formulas to be checked for satisfiability.

Our goal is to develop a decision procedure for the quantifier-free fragment of $\mathcal{T}_E$. We will focus initially on deciding satisfiability for sets (or conjunctions) of literals, and then extend this to arbitrary quantifier-free formulas using normal forms.

14.2 Normal Forms

To handle arbitrary quantifier-free formulas, we introduce two important normal forms that facilitate systematic reasoning.

14.2.1 Negation Normal Form (NNF)

Definizione 14.2 (Negation Normal Form)**.** A formula is in **Negation Normal Form (NNF)** if:

- The only logical connectives are $\wedge$, $\vee$, and $\neg$
- Negation ($\neg$) applies only to atomic formulas (atoms)

Any quantifier-free formula can be converted to NNF by applying the following equivalence-preserving transformations:

$$
\begin{aligned}
H \leftrightarrow G &\equiv (H \rightarrow G) \wedge (G \rightarrow H) && \text{(Eliminate biconditional)} \\
H \rightarrow G &\equiv \neg H \vee G && \text{(Eliminate implication)} \\
\neg(H \wedge G) &\equiv \neg H \vee \neg G && \text{(De Morgan's law)} \\
\neg(H \vee G) &\equiv \neg H \wedge \neg G && \text{(De Morgan's law)} \\
\neg\neg H &\equiv H && \text{(Double negation elimination)}
\end{aligned}
$$

14.2.2 Disjunctive Normal Form (DNF)

Definizione 14.3 (Disjunctive Normal Form)**.** A formula is in **Disjunctive Normal Form (DNF)** if it is a disjunction of conjunctions of literals:

$$\bigvee_{i=1}^{k} \left(\bigwedge_{j=1}^{n_i} L_{i,j} \right)$$

where each $L_{i,j}$ is a literal (an atomic formula or its negation).

Equivalently, a DNF formula has the structure:

$$(L_{1,1} \wedge L_{1,2} \wedge \cdots \wedge L_{1,n_1}) \vee (L_{2,1} \wedge L_{2,2} \wedge \cdots \wedge L_{2,n_2}) \vee \cdots \vee (L_{k,1} \wedge L_{k,2} \wedge \cdots \wedge L_{k,n_k})$$

14.3 Conversion to DNF

Any quantifier-free formula F can be systematically converted to an equivalent DNF formula through a two-stage process:

$$F \longrightarrow \boxed{\text{Convert to NNF}} \longrightarrow F' \longrightarrow \boxed{\text{Convert to DNF}} \longrightarrow F''$$

Stage 1: Convert F to NNF using the transformations above, obtaining F'.

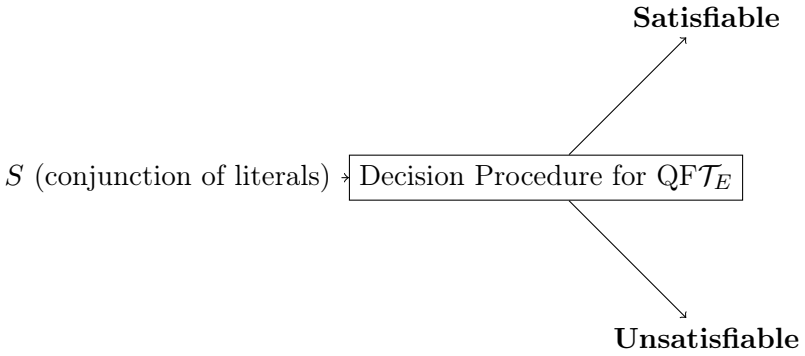
Stage 2: Convert F' to DNF by repeatedly applying the distributivity law:

$$H \wedge (G_1 \vee G_2) \equiv (H \wedge G_1) \vee (H \wedge G_2)$$

This transformation pushes conjunctions inward and disjunctions outward, resulting in a disjunction of conjunctions.

14.4 Decision Procedure for Quantifier-Free $\mathcal{T}_E$

The decision procedure operates on conjunctions of literals. Given a set (or conjunction) S of literals:



Extension to arbitrary quantifier-free formulas: For a general quantifier-free formula F:

1. Convert F to NNF, obtaining F'
2. Convert F' to DNF, obtaining $F'' = \bigvee_{i=1}^{k} C_i$ where each C_i is a conjunction of literals
3. Apply the decision procedure to each conjunction C_i
4. F is satisfiable if and only if at least one C_i is satisfiable

This reduction allows us to decide satisfiability for arbitrary quantifier-free formulas by deciding satisfiability for each disjunct independently.

14.5 Elimination of Non-Equality Predicates

The decision procedure for quantifier-free $\mathcal{T}_E$ assumes that the only predicate symbol in the formula is equality ($=$). However, formulas may contain additional predicate symbols. We now show how to eliminate such predicates through a satisfiability-preserving transformation.

14.5.1 The Transformation

Definizione 14.4 (Predicate Elimination)**.** Given an occurrence of a predicate symbol P of arity n applied to terms $t_1, \ldots, t_n$, we eliminate P by introducing:

- A fresh function symbol f_P (one for each predicate symbol $P \neq =$)
- A fresh constant symbol $\bullet$ (the same constant for all predicates)

The transformation replaces predicate applications with equality constraints:

$$P(t_1, \ldots, t_n) \mapsto f_P(t_1, \ldots, t_n) = \bullet$$
$$\neg P(t_1, \ldots, t_n) \mapsto f_P(t_1, \ldots, t_n) \neq \bullet$$

Intuition: The predicate $P(t_1, \ldots, t_n)$ is true if and only if the term $f_P(t_1, \ldots, t_n)$ evaluates to the distinguished constant $\bullet$.

14.5.2 Satisfiability Preservation

Teorema 14.1 (Satisfiability Preservation)**.** *Let F be a quantifier-free formula and let F' be the result of applying predicate elimination to F. Then F is satisfiable if and only if F' is satisfiable in models with cardinality greater than 1.*

Nota. The restriction to models with cardinality greater than 1 is necessary to avoid trivial models. In a model with domain $D = \{d\}$ (cardinality 1), all terms evaluate to the same element d, making all equalities trivially true and potentially invalidating the transformation.

14.5.3 Example: The Need for Cardinality Restriction

Consider the following two formulas:

Original formula:

$$S_1 = \{\forall x \, \forall y \, (x = y), \neg P(a)\}$$

This formula is $\mathcal{T}_E$-satisfiable. We can construct a model with domain $D = \{d_1, d_2\}$ where:

- $\Phi(a) = d_1$
- $\Phi(P) = \{d_2\}$ (so $d_1 \notin \Phi(P)$, making $\neg P(a)$ true)
- The first axiom $\forall x \, \forall y \, (x = y)$ is vacuously false in this model, but if we interpret it as requiring equality to be universal, we need a different construction

Actually, let's reconsider: $\{\forall x \, \forall y \, (x = y), \neg P(a)\}$ asserts that all elements are equal (domain has cardinality 1) and $P(a)$ is false. This is satisfiable in a model with $D = \{d\}$, $\Phi(a) = d$, and $\Phi(P) = \emptyset$.

Transformed formula:

$$S_2 = \{\forall x \, \forall y \, (x = y), f_P(a) \neq \bullet\}$$

In a model with cardinality 1, we have $D = \{d\}$, so $\Phi(f_P)(d) = d$ and $\Phi(\bullet) = d$. Therefore, $f_P(a) = \bullet$, contradicting $f_P(a) \neq \bullet$. Thus S_2 is unsatisfiable in models of cardinality 1.

This example demonstrates that the transformation preserves satisfiability only when we exclude trivial models with cardinality 1.

14.6 Treatment of Free Variables

In quantifier-free formulas, free variables may appear. For the purposes of satisfiability checking, free variables are treated analogously to constants.

Nota (Free Variables as Constants). To determine satisfiability of a formula F with free variables $x_1, \ldots, x_k$, we need to find an interpretation $\mathcal{I}$ and an assignment β such that $\mathcal{I} \models_\beta F$.

Equivalently, we can treat each free variable x_i as a fresh constant symbol c_i and check satisfiability of the resulting closed formula. The formula is satisfiable if and only if there exist domain elements that can be assigned to these constants to satisfy the formula.

This observation allows us to reduce the satisfiability problem for formulas with free variables to the satisfiability problem for closed formulas by a simple syntactic substitution.

15 The Congruence Closure Algorithm

15.1 Motivating Example

Consider the following conjunction of literals:

$$x = g(x) \land f(x, g(g(x))) \neq g(x) \land f(x, x) = g(g(x))$$

Question: Is this formula satisfiable or unsatisfiable?

Intuitive analysis: This formula appears to be unsatisfiable. From $x = g(x)$, we can derive $g(x) = g(g(x))$ by applying the function g to both sides (congruence). Then from $f(x, x) = g(g(x))$ and the derived equality $g(x) = g(g(x))$, we obtain $f(x, x) = g(x)$. But we also have $x = g(x)$, so by congruence on f, we get $f(x, x) = f(x, g(x))$. However, we need to check whether $f(x, g(g(x))) = g(x)$ follows from the positive equalities, which would contradict $f(x, g(g(x))) \neq g(x)$.

15.2 The Congruence Closure Decision Procedure

The **congruence closure algorithm** is a decision procedure for satisfiability in the quantifier-free fragment of the theory of equality. The algorithm operates as follows:

1. **Build the smallest congruence**: Construct the smallest congruence relation that satisfies all positive equalities in the input formula
2. **Check negative equalities**: For each negative equality $s \neq t$ in the input, check whether s and t belong to the same equivalence class in the congruence
3. **Determine satisfiability**:
   - If any pair (s, t) with $s \neq t$ belongs to the same equivalence class, the formula is **unsatisfiable**
   - Otherwise, the congruence provides a model, and the formula is **satisfiable**

The key insight is that the smallest congruence containing all positive equalities is included in every congruence that satisfies those equalities. Therefore, if a negative equality is violated in the smallest congruence, it must be violated in every model.

15.3 Equivalence Relations and Partitions

Before describing the algorithm in detail, we recall fundamental properties of equivalence relations.

Definizione 15.1 (Equivalence Relation)**.** An **equivalence relation** R on a set S is a binary relation that is:

- **Reflexive**: $\forall s \in S,\ s\, R\, s$
- **Symmetric**: $\forall s, t \in S,\ s\, R\, t \implies t\, R\, s$
- **Transitive**: $\forall s, t, u \in S,\ (s\, R\, t \land t\, R\, u) \implies s\, R\, u$

Definizione 15.2 (Equivalence Class)**.** Given an equivalence relation R on set S, the **equivalence class** of an element $s \in S$ is:

$$[s]_R = \{s' \in S \mid s'\, R\, s\}$$

Definizione 15.3 (Quotient Set)**.** The **quotient set** S/R is the set of all equivalence classes:

$$S/R = \{[s]_R \mid s \in S\}$$

Proposizione 15.1 (Partition Property)**.** *An equivalence relation R on S induces a partition of S. That is, for any two distinct equivalence classes $[s]_R$ and $[p]_R$:*

$$s \not\sim_R p \quad \Longrightarrow \quad [s]_R \cap [p]_R = \emptyset$$

where $s \sim_R p$ denotes $s\,R\,p$.

Conversely, every element belongs to exactly one equivalence class, and the union of all equivalence classes equals S.

This partition property is fundamental to the congruence closure algorithm, as it allows us to represent the congruence relation efficiently using a union-find data structure.

Parte IV

Decision Procedures for Quantifier-Free Theories

16 The Congruence Closure Algorithm

16.1 Computing Congruence Closure

We now describe how to compute the congruence closure for the quantifier-free fragment of the theory of equality $\mathcal{T}_E$.

16.1.1 Input Problem Structure

Consider an input formula F consisting of a conjunction of equality and disequality literals:

$$F = \underbrace{\{s_1 = t_1, \ldots, s_n = t_n\}}_{F^+} \cup \underbrace{\{s_{n+1} \neq t_{n+1}, \ldots, s_{n+m} \neq t_{n+m}\}}_{F^-}$$

where:

- F^+ denotes the set of **positive equalities** (equations)
- F^- denotes the set of **negative equalities** (disequations)

The congruence closure algorithm will construct the smallest congruence containing all equalities in F^+, then check whether any disequality in F^- is violated.

16.1.2 Equivalence Relations and Representatives

Given a binary equivalence relation $R \subseteq S \times S$ on a set S, we can construct its quotient:

$$S/R = \{[s]_R \mid s \in S\}$$

where the equivalence class of s is:

$$[s]_R = \{p \in S \mid p \, R \, s\}$$

The element s is called a **representative** of the equivalence class $[s]_R$.

16.1.3 Disjointness of Equivalence Classes

Proposizione 16.1 (Disjoint Classes)**.** *For any two equivalence classes $[s_1]_R$ and $[s_2]_R$, either they are identical or they are disjoint:*

$$[s_1]_R \cap [s_2]_R \neq \emptyset \quad \Longrightarrow \quad [s_1]_R = [s_2]_R$$

Dimostrazione. Suppose there exists $p \in [s_1]_R \cap [s_2]_R$. Then:

- $p \, R \, s_1$ (since $p \in [s_1]_R$)
- $p \, R \, s_2$ (since $p \in [s_2]_R$)

By symmetry and transitivity of R, we have $s_1 \, R \, s_2$. Therefore, for any $q \in [s_1]_R$, we have $q \, R \, s_1$ and $s_1 \, R \, s_2$, so by transitivity $q \, R \, s_2$, which means $q \in [s_2]_R$. Similarly, $[s_2]_R \subseteq [s_1]_R$. Thus $[s_1]_R = [s_2]_R$. □

16.1.4 Example: Equivalence Closure

Consider the binary relation:

$$R = \{(a, b), (b, c), (d, d)\}$$

We can visualize this as a directed graph:

$$a \longrightarrow b \longrightarrow c \qquad\qquad d \circlearrowright$$

The **equivalence closure** R^* of R must include:

- **Reflexivity**: $(a, a), (b, b), (c, c), (d, d)$
- **Symmetry**: $(b, a), (c, b)$
- **Transitivity**: $(a, c), (c, a)$

Thus:

$$R^* = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c), (d, d)\}$$

The quotient set is:

$$S/R^* = \{\{a, b, c\}, \{d\}\}$$

Definizione 16.1 (Equivalence Closure)**.** Given a binary relation R on set S, the **equivalence closure** of R, denoted R^*, is the $\subseteq$-smallest equivalence relation such that:

1. R^* is an equivalence relation
2. $R \subseteq R^*$

Equivalently, R^* is the intersection of all equivalence relations containing R.

The equivalence closure can be computed by iteratively applying reflexivity, symmetry, and transitivity rules until a fixed point is reached.

16.1.5 Understanding the $\subseteq$-Smallest Property

Proposizione 16.2 (Minimality of Equivalence Closure)**.** *Let R be a binary relation and R^* be its equivalence closure. For any equivalence relation P such that $R \subseteq P$, we have $R^* \subseteq P$.*

This means that R^* is the **smallest** equivalence relation containing R with respect to the subset ordering $\subseteq$.

Definizione 16.2 (Congruence Closure)**.** Given a binary relation R, the **congruence closure** of R, denoted R^*_{cong}, is the $\subseteq$-smallest relation such that:

1. R^*_{cong} is a congruence relation
2. $R \subseteq R^*_{\text{cong}}$

A congruence relation is an equivalence relation that is preserved under function application: if $s_1 \sim t_1, \ldots, s_n \sim t_n$, then $f(s_1, \ldots, s_n) \sim f(t_1, \ldots, t_n)$ for all function symbols f.

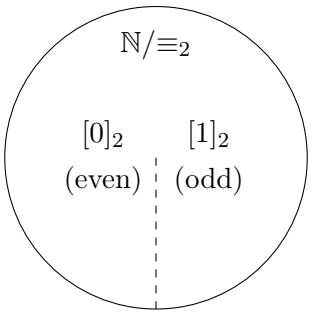
16.1.6 Example: Refinement of Equivalence Relations

To illustrate the concept of one equivalence relation being smaller (finer) than another, consider the following example on the natural numbers $\mathbb{N}$.

Equivalence modulo 2: Define the relation $\equiv_2 \subseteq \mathbb{N} \times \mathbb{N}$ by:

$$n \equiv_2 m \quad \text{iff} \quad n \equiv m \pmod 2$$

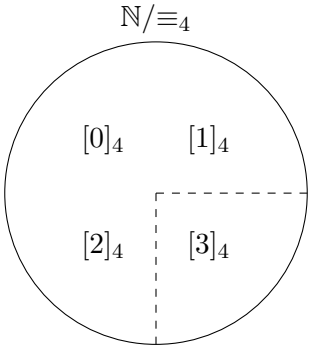
This relation partitions $\mathbb{N}$ into two equivalence classes:



Equivalence modulo 4: Define the relation $\equiv_4 \subseteq \mathbb{N} \times \mathbb{N}$ by:

$$n \equiv_4 m \quad \text{iff} \quad n \equiv m \pmod 4$$

This relation partitions $\mathbb{N}$ into four equivalence classes:



Proposizione 16.3 (Refinement)**.** *The relation $\equiv_4$ is a **refinement** of $\equiv_2$. That is:*

$$n \equiv_4 m \quad \Longrightarrow \quad n \equiv_2 m$$

However, the converse does not hold:

$$n \equiv_2 m \quad \not\Rightarrow \quad n \equiv_4 m$$

Dimostrazione. If $n \equiv_4 m$, then $n \equiv m \pmod 4$, which means $n - m = 4k$ for some $k \in \mathbb{Z}$. Therefore, $n - m = 2(2k)$, so $n \equiv m \pmod 2$, i.e., $n \equiv_2 m$.

For the converse, consider $n = 0$ and $m = 2$. Then $n \equiv_2 m$ (both are even), but $n \not\equiv_4 m$ (since $0 \equiv 0 \pmod 4$ and $2 \equiv 2 \pmod 4$). □

Set-theoretic interpretation: This means that $(n, m) \in \equiv_4 \implies (n, m) \in \equiv_2$, so:

$$\equiv_4 \subseteq \equiv_2$$

In this sense, $\equiv_4$ is **strictly smaller** than $\equiv_2$ (it is a proper subset), and provides a finer partition of $\mathbb{N}$.

16.2 The Congruence Closure Decision Procedure

We now return to the main problem: deciding satisfiability for a quantifier-free formula in the theory of equality.

16.2.1 Algorithm Description

Recall that the input formula F has the form:

$$F = \underbrace{\{s_1 = t_1, \ldots, s_n = t_n\}}_{F^+} \cup \underbrace{\{s_{n+1} \neq t_{n+1}, \ldots, s_{n+m} \neq t_{n+m}\}}_{F^-}$$

The congruence closure algorithm proceeds as follows:

Algorithm 1 Congruence Closure Decision Procedure

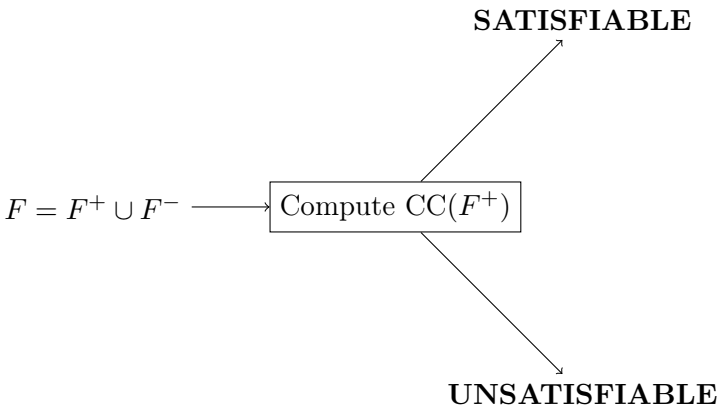
```
1: Input: Formula F = F⁺ ∪ F⁻
2: Compute the congruence closure CC(F⁺) of the positive equalities F⁺
3: for i = n + 1 to n + m do
4:    if (s_i, t_i) ∈ CC(F⁺) then
5:       return UNSATISFIABLE
6:    end if
7: end for
8: return SATISFIABLE
```

Step 1: Compute $\mathrm{CC}(F^+)$, the smallest congruence relation containing all pairs (s_i, t_i) for $i = 1, \ldots, n$.

Step 2: For each disequality $s_i \neq t_i$ in F^- (where $n + 1 \leq i \leq n + m$), check whether (s_i, t_i) belongs to the same equivalence class in $\mathrm{CC}(F^+)$.

Step 3:

- If $(s_j, t_j) \in \mathrm{CC}(F^+)$ for some j with $n + 1 \leq j \leq n + m$, then F is **unsatisfiable**
- Otherwise, F is **satisfiable**



16.2.2 Correctness of the Algorithm

Teorema 16.4 (Soundness and Completeness)**.** *The congruence closure algorithm is sound and complete for deciding satisfiability in the quantifier-free fragment of* $\mathcal{T}_E$.

Proof sketch. **Soundness (Unsatisfiability)**: Suppose there exists j with $n + 1 \leq j \leq n + m$ such that $(s_j, t_j) \in \mathrm{CC}(F^+)$.

Since $\mathrm{CC}(F^+)$ is the smallest congruence relation containing F^+, the pair (s_j, t_j) must belong to *every* congruence relation that contains F^+.

Therefore, in any model $\mathcal{I}$ that satisfies all equalities in F^+, we must have $\mathcal{I} \models s_j = t_j$. But F also contains $s_j \neq t_j$, so $\mathcal{I} \not\models s_j \neq t_j$. Thus, no model can satisfy both F^+ and F^-, making F unsatisfiable.

Completeness (Satisfiability): If $(s_i, t_i) \notin \mathrm{CC}(F^+)$ for all i with $n+1 \leq i \leq n+m$, then we can construct a model by interpreting the equivalence classes of $\mathrm{CC}(F^+)$ as domain elements. This model satisfies all equalities in F^+ (by construction) and all disequalities in F^- (since the terms belong to different equivalence classes). □

The key insight is that the congruence closure captures exactly those equalities that are *forced* by the positive equalities in F^+ together with the congruence axioms of equality.

16.3 Examples: Manual Execution

We now illustrate the congruence closure algorithm with concrete examples.

16.3.1 Notation and Setup

For a formula F of the form:

$$F = \{s_1 = t_1, \ldots, s_n = t_n\} \cup \{s_{n+1} \neq t_{n+1}, \ldots, s_{n+m} \neq t_{n+m}\}$$

we define S_F as the **set of all terms** that appear in F.

16.3.2 Example 1

Consider the formula:

$$F = \{x = g(x),\ f(x,x) = g(g(x)),\ f(x, g(g(x))) \neq g(x)\}$$

The set of terms is:

$$S_F = \{x,\ g(x),\ f(x,x),\ g(g(x)),\ f(x, g(g(x)))\}$$

Notation:

- x is a **free variable symbol**
- f is a **binary function symbol** (arity 2)
- g is a **unary function symbol** (arity 1)

Positive equalities: $F^+ = \{x = g(x),\ f(x,x) = g(g(x))\}$

Negative equalities: $F^- = \{f(x, g(g(x))) \neq g(x)\}$

16.3.3 Example 2

Consider the formula:

$$F = \{f(a,b) = a,\ f(f(a,b),b) \neq a\}$$

The set of terms is:

$$S_F = \{f(a,b),\ a,\ b,\ f(f(a,b),b)\}$$

Notation:

- a, b are **constant symbols** (by convention)

- f is a **binary function symbol**

Positive equalities: $F^+ = \{f(a,b) = a\}$

Negative equalities: $F^- = \{f(f(a,b),b) \neq a\}$

Nota (Free Variables vs. Constants). In the quantifier-free fragment, all variables are free. From a semantic perspective, free variables and constants can be used interchangeably: both are interpreted as arbitrary elements of the domain. The distinction is purely syntactic.

16.3.4 Step-by-Step Execution for Example 1

We now execute the congruence closure algorithm on Example 1:

$$F = \{x = g(x),\ f(x,x) = g(g(x)),\ f(x,g(g(x))) \neq g(x)\}$$

Step 1: Initialize the partition

Create a partition of S_F where every term is in its own singleton equivalence class:

$$\{\{x\},\{g(x)\},\{f(x,x)\},\{g(g(x))\},\{f(x,g(g(x)))\}\}$$

Step 2: Process positive equalities

For each equality $(s_i = t_i) \in F^+$ (for all i with $1 \leq i \leq n$):

- Unite the equivalence class of s_i with the equivalence class of t_i
- Unite the classes of all terms in S_F that become congruent as a consequence of $s_i = t_i$

Processing $x = g(x)$:

Merge the classes $\{x\}$ and $\{g(x)\}$:

$$\{\{x,g(x)\},\{f(x,x)\},\{g(g(x))\},\{f(x,g(g(x)))\}\}$$

Since $x = g(x)$, by congruence we derive $g(x) = g(g(x))$ (applying g to both sides). Merge these classes:

$$\{\{x,g(x),g(g(x))\},\{f(x,x)\},\{f(x,g(g(x)))\}\}$$

Processing $f(x,x) = g(g(x))$:

Since $g(g(x)) \in \{x,g(x),g(g(x))\}$ and $f(x,x) \in \{f(x,x)\}$, merge these classes:

$$\{\{x,g(x),g(g(x)),f(x,x)\},\{f(x,g(g(x)))\}\}$$

Now check for congruence: since $x \sim g(g(x))$ (both in the same class), we have:

$$f(x,x) \sim f(x,g(g(x)))$$

by the congruence property (both arguments are in the same equivalence classes). Merge these classes:

$$\{\{x,g(x),g(g(x)),f(x,x),f(x,g(g(x)))\}\}$$

Step 3: Check negative equalities

Check whether $(s_i,t_i) \in \mathrm{CC}(F^+)$ for all i with $n+1 \leq i \leq n+m$.

For the disequality $f(x,g(g(x))) \neq g(x)$:

- $f(x, g(g(x))) \in \{x, g(x), g(g(x)), f(x, x), f(x, g(g(x)))\}$
- $g(x) \in \{x, g(x), g(g(x)), f(x, x), f(x, g(g(x)))\}$

Both terms are in the same equivalence class, so $(f(x, g(g(x))), g(x)) \in \mathrm{CC}(F^+)$.

Conclusion: The formula F is **UNSATISFIABLE**.