Notes for Logic (COMP0009)

Raphael Li

Sep 2025

Contents

1 Revision: Syntax and semantics of propositional and first-order logic

Formally, a *logic* consists of three components:

Component	Describes...
Syntax	The language and grammar for writing formulas
Semantics	How formulas are interpreted
Inference system (or proof system)	A syntactic device for proving true statements

Table 1: The three key components of a logic.

This module concerns algorithms that automatically parse and determine the validity of a formula.

1.1 Propositional logic

1.1.1 Syntax

Formulas are constructed by applying negation, conjunction and disjunction to propositions.

$$\begin{aligned} \text{proposition} &\coloneqq p \mid q \mid r \mid \cdots \\ \text{formula} &\coloneqq \text{proposition} \mid \neg\text{formula} \mid (\text{formula} \circ \text{formula}) \qquad \text{(where } \circ \text{ is } \wedge, \vee \text{ or } \rightarrow) \end{aligned}$$

A proposition or its negation is called a *literal*[1].

For any formula that isn't a proposition, the *main connective* is the one with the largest scope. In other words, it is not in the scope of any other connective.

$$((p \wedge q) \vee \neg(q \rightarrow r))$$

This is the connective with which evaluation begins. This is especially important when building parsers for algorithmically evaluating formulas.

Note that parsers working according to the above definition will recognise $(p \wedge q)$, but not $p \wedge q$, as a formula. Regardless, throughout this document we will use a looser definition where brackets may be ommitted in unambiguous cases.

1.1.2 Semantics

A valuation is a function v that maps each proposition to a truth value in $\{\top, \bot\}$.

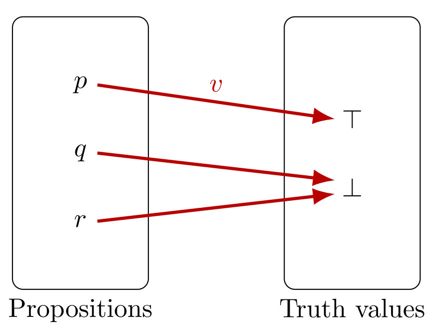

Figure 1: A valuation maps propositions to truth values.

[1] For example, p and $\neg p$ are both literals, but $\neg\neg q$ is not.

A valuation v can be extended to a unique *truth function* defined on all possible formulas. A truth function v' must satisfy

$$\begin{aligned}
v'(\neg\phi) = \top &\iff v'(\phi) = \bot \\
v'(\phi \vee \psi) = \top &\iff v'(\phi) = \top \text{ or } v'(\psi) = \top \\
v'(\phi \wedge \psi) = \top &\iff v'(\phi) = \top \text{ and } v'(\psi) = \top \\
v'(\phi \rightarrow \psi) = \top &\iff v'(\phi) = \bot \text{ or } v'(\psi) = \top \\
v'(\phi \leftrightarrow \psi) = \top &\iff v'(\phi) = v'(\psi)
\end{aligned}$$

for all formulas ϕ and ψ. From now on we use v to denote the more general truth function.

The result of applying a valuation v to a formula ϕ depends only on the propositional letters that occur in ϕ.

A formula ϕ is *valid* if $v(\phi) = \top$ for all valuations v, which we denote as $\models \phi$. A formula ϕ is *satisfiable* if $v(\phi) = \top$ for at least one valuation v. All valid formulas are satisfiable, but *not* vice versa.

Two formulas ϕ and ψ are *logically equivalent*, written as $\phi \equiv \psi$, if and only if for every valuation v we have $v(\phi) = v(\psi)$.

1.1.3 Truth tables

Consider the propositional formula $((p \vee \neg q) \wedge \neg(q \wedge r))$. We can check its validity and satisfiability by constructing its truth table.

p	q	r	$(p \vee \neg q)$	$\neg(q \wedge r)$	$((p \vee \neg q) \wedge \neg(q \wedge r))$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	0	0

Table 2: The truth table for the formula $((p \vee \neg q) \wedge \neg(q \wedge r))$.

In this case, the formula is satisfiable but not valid.

1.1.4 Parse trees

A parser interprets the semantics of a formula by breaking down its symbols into a *parse tree*, which shows the syntactic relation between symbols. For example, the formula $((p \vee \neg q) \wedge \neg(q \wedge r))$ can be broken down into the following parse tree.

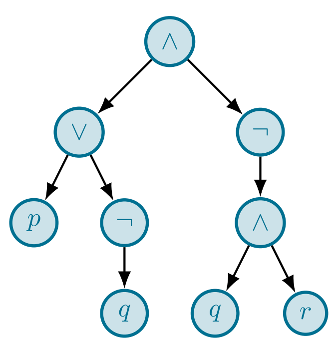

Figure 2: The parse tree for the formula $((p \vee \neg q) \wedge \neg(q \wedge r))$.

1.1.5 Disjunctive normal form (DNF)

A formula is said to be in *disjunctive normal form* (DNF) if it is a disjunction of one or more conjunctions of one or more literals.

$$\begin{aligned}
\text{proposition} &\coloneqq p \mid q \mid r \mid \cdots \\
\text{literal} &\coloneqq \text{proposition} \mid \neg\text{proposition} \\
\text{conjunctiveClause} &\coloneqq \text{literal} \mid \text{literal} \ \wedge \ \text{conjunctiveClause} \\
\text{DNF} &\coloneqq \text{conjunctiveClause} \mid \text{conjunctiveClause} \ \vee \ \text{DNF}
\end{aligned}$$

Below is an example of a formula in DNF.

$$\underbrace{(p \wedge \neg q \wedge \neg r)}_{\substack{\text{conjunctive}\\ \text{clause}}} \vee \underbrace{(\neg p \wedge \neg q \wedge r)}_{\substack{\text{conjunctive}\\ \text{clause}}} \vee \underbrace{(q \wedge \neg r)}_{\substack{\text{conjunctive}\\ \text{clause}}}$$

Any propositional formula has a DNF equivalent. For instance, the formula $(p \vee \neg q) \wedge \neg(q \wedge r)$ can be rewritten as follows.

$$\begin{aligned}
& (p \vee \neg q) \wedge \neg(q \wedge r) \\
\iff & (p \vee \neg q) \wedge (\neg q \vee \neg r) && \text{(De Morgan's law, to remove outer negation)} \\
\iff & ((p \vee \neg q) \wedge \neg q) \vee ((p \vee \neg q) \wedge \neg r) && \text{(distributing conjunctions over disjunctions)} \\
\iff & (p \wedge \neg q) \vee (\neg q \wedge \neg q) \vee (p \wedge \neg r) \vee (\neg q \wedge \neg r) && \text{(distributing conjunctions over disjunctions)} \\
\iff & (p \wedge \neg q) \vee \neg q \vee (p \wedge \neg r) \vee (\neg q \wedge \neg r)
\end{aligned}$$

Alternatively, this can also be achieved by referring to the truth table. From Table 2, we see that the formula can be written in DNF as

$$(\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge \neg q \wedge r) \vee (p \wedge q \wedge \neg r).$$

1.1.6 Conjunctive normal form (CNF)

A formula is said to be *conjunctive normal form* (CNF) if it is a conjunction of one or more disjunctions of one or more literals.

$$\begin{aligned}
\text{disjunctiveClause} &\coloneqq \text{literal} \mid \text{literal} \ \vee \ \text{disjunctiveClause} \\
\text{CNF} &\coloneqq \text{disjunctiveClause} \mid \text{disjunctiveClause} \ \wedge \ \text{CNF}
\end{aligned}$$

Below is a formula in CNF.

$$\underbrace{(p \vee \neg q \vee \neg r)}_{\substack{\text{conjunctive}\\ \text{clause}}} \wedge \underbrace{(\neg p \vee q \vee r)}_{\substack{\text{conjunctive}\\ \text{clause}}}$$

To find the CNF equivalent of a formula ϕ, we first express its negation $\neg\phi$ in DNF. Then, we negate it again to get $\neg\neg\phi$. Using De Morgan's law, the resultant formula will be in CNF.

For example, let ϕ be the formula $(p \vee \neg q) \wedge \neg(q \wedge r)$. To rewrite it in CNF, we start by constructing the truth table of its negation $\neg\phi$. This allows us to express $\neg\phi$ in DNF.

p	q	r	$((p \lor \neg q) \land \neg(q \land r))$	Negation of $((p \lor \neg q) \land \neg(q \land r))$
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Table 3: The truth table for the negation of $((p \lor \neg q) \land \neg(q \land r))$. This is obtained by flipping the results of Table 2.

Hence we have

$$\begin{aligned} \neg\phi &= (\neg p \land q) \lor (p \land q \land r) && \text{(DNF of } \neg\phi\text{)} \\ \neg\neg\phi &= \neg((\neg p \land q) \lor (p \land q \land r)) && \text{(negating both sides)} \\ \phi &= (p \lor \neg q) \land (\neg p \lor \neg q \lor \neg r) && \text{(double negation; De Morgan's laws)} \end{aligned}$$

which gives us ϕ in CNF.

1.2 First-order logic

1.2.1 Syntax

A first-order language $L(C, F, P)$ is determined by a set C of constant symbols, a set F of function symbols and a non-empty set P of predicate symbols. Each function symbol and predicate symbol has an associated *arity* $n \in \mathbb{N}$. We write f^n and p^n to represent an n-ary function symbol and an n-ary predicate symbol respectively. Moreover, let V be a countably infinite set of variable symbols.

$$\begin{aligned} \text{term} &\coloneqq c \mid v \mid f^n(\text{term}_0, \text{term}_1, \cdots, \text{term}_{n-1}) && \text{(where } c \in C,\ v \in V \text{ and } f^n \in F\text{)} \\ \text{atom} &\coloneqq p^n(\text{term}_0, \text{term}_1, \cdots, \text{term}_{n-1}) && \text{(where } p^n \in P\text{)} \\ \text{formula} &\coloneqq \text{atom} \mid \neg\text{formula} \mid (\text{formula}_0 \lor \text{formula}_1) \mid \exists v \text{ formula} && \text{(where } v \in V\text{)} \end{aligned}$$

This definition is functionally complete. Formulas involving universal quantifiers, implications and equivalence symbols can always be rewritten using only symbols defined above.

A *closed term* is a term with no variable symbols. A *sentence* is a formula with no free variables.

1.2.2 Semantics

For a first-order language $L(C, F, P)$, we may construct a corresponding first-order structure[2] $S = (D, I)$ where $I = (I_c, I_f, I_p)$.

$$S = (\underbrace{D}_{\text{non-empty domain}}, \overbrace{(I_c, I_f, I_p)}^{\text{interpretation } I})$$

Here,

- I_c maps each constant symbol in C to an element of D.
- I_f maps each n-ary function symbol in F to an n-ary function over D.
- I_p maps each n-ary predicate symbol $p \in P$ to an n-ary relation over D (i.e. a subset of D^n).

[2]Also known as an L-structure.

- We may occasionally use I to denote a general interpretation function where

$$I(c) = I_c(c) \qquad \text{(for all } c \in C)$$
$$I(f) = I_f(f) \qquad \text{(for all } f \in F)$$
$$I(p) = I_p(p) \qquad \text{(for all } p \in P)$$

If P includes the equality symbol $=$, then it is always interpreted as the binary relation of true equality.

$$I_p(=) = \{(d,d) : d \in D\}$$

Given a structure $S = (D, I)$, a variable assignment A is a map from V to D. For any variable $v \in V$, two variable assignments A and A^* are said to be v-equivalent if $A(x) = A^*(x)$ for all $x \in V \setminus \{v\}$. In other words, two variable assignments are said to be v-equivalent if they are completely identical except possibly for the element in D assigned to v. This is written as $A \equiv_v A^*$.

Given a structure S and a variable assignment A, we may interpret any term as follows.

$$\begin{aligned} c^{S,A} &= I_c(c) \\ v^{S,A} &= A(v) \\ f^n(t_0, t_1, \cdots, t_{n-1})^{S,A} &= \underbrace{(I_f(f^n))}_{\substack{\text{interpreted} \\ \text{function}}}(t_0^{S,A}, t_1^{S,A}, \cdots, t_{n-1}^{S,A}) \end{aligned}$$

Formulas are evaluated as follows.

$$\begin{aligned} S \models_A p^n(t_0, t_1, \cdots, t_{n-1}) &\iff (t_0^{S,A}, t_1^{S,A}, \cdots, t_{n-1}^{S,A}) \in I_p(p^n) \\ S \models_A \neg\text{formula} &\iff S \not\models_A \text{formula} \\ S \models_A (\text{formula}_0 \vee \text{formula}_1) &\iff S \models_A \text{formula}_0 \text{ or } S \models_A \text{formula}_1 \\ S \models_A \exists v \text{ formula} &\iff S \models_{A[x \mapsto d]} \text{formula for some } d \in D \end{aligned}$$

Given a structure S and a formula ϕ, we say that

- ϕ is "valid in S" if $S \models_A \phi$ for every variable assignment A. This is written as $S \models \phi$.
- ϕ is "satisfiable in S" if $S \models_A \phi$ for some variable assignment A.
- ϕ is "valid" if ϕ is valid in all possible structures. This is written as $\models \phi$.
- ϕ is "satisfiable" if there exists some structure in which ϕ is satisfiable.

A formula ϕ is valid if and only if $\neg\phi$ is not satisfiable.

> **Proof.** Let $\neg\phi$ be a formula that is not satisfiable. Hence we have
>
> $$\begin{aligned} \neg\exists S\ \exists A \quad S \models_A \neg\phi &\iff \neg\exists S\ \exists A \quad S \not\models_A \phi \\ &\iff \forall S\ \neg\exists A \quad S \not\models_A \phi \\ &\iff \forall S\ \forall A \quad \neg S \not\models_A \phi \\ &\iff \forall S\ \forall A \quad S \models_A \phi \end{aligned}$$
>
> which means S is valid.

If ϕ is a sentence, then ϕ is valid in S if and only if it is also satisfiable in S.

1.2.3 Example: Arithmetic in the set of natural numbers

Consider the first-order language $L(C, F, P)$ defined as follows. Also assume a countably infinite set V of variable symbols.

$$\begin{aligned}
C &= 1, 2, 3, \cdots && \text{(constant symbols)} \\
F &= \{+, \times\} && \text{(function symbols, both binary)} \\
P &= \{=, <\} && \text{(predicate symbols, both binary)} \\
V &= \{x, y, z, \cdots\} && \text{(variable symbols)}
\end{aligned}$$

A term is a string of symbols that represents a "thing" or an "object" — this can be a constant, a variable, or a function output.

- x
- $1 + 3$
- $2 \times x + 1$

Of the terms shown above, only the second one is a closed terms because it has no variable symbols.

An atom is a string of symbols that represents the output of a predicate, which is a truth value.

- $1 = 2$
- $y < 3$
- $x + 1 < 2 \times z + 3$

Finally, a formula is constructed by applying negations, disjunctions, and existential quantifiers to atoms.

- $1 = 2 \ \wedge \ y < 3$
- $\neg \exists z \ \ x + 1 < 2 \times z + 3$

The latter example is a sentence because all of its variable symbols are bounded.

For this particular first-order language, we may use the structure of ordinary arithmetic[3], defined as $N = \{\mathbb{N}, \{I_c, I_f, I_p\}\}$ where

- I_c is a function that maps numerical symbols to the corresponding natural number.

$$\begin{aligned}
I_c(1) &= 1 \\
I_c(2) &= 2 \\
I_c(3) &= 3 \\
&\vdots
\end{aligned}$$

- I_f maps $+$ and $\times$ to the addition and multiplication operations in arithmetic respectively.
- I_p maps $=$ and $<$ to the following relations.

$$\begin{aligned}
I_p(=) &= \{(n, n) : n \in \mathbb{N}\} \\
I_p(<) &= \{(m, n) \in \mathbb{N}^2 : m < n\}
\end{aligned}$$

[3]There is also a similar structure $R = (\mathbb{R}, I)$ where the domain is the set of real numbers.

1.2.4 First-order structures and directed graphs

Consider a first-order language with only one binary predicate symbol p.

$$L(C, F, \{p\})$$

Any first-order structure $S = \{D, \{I_c, I_f, I_p\}\}$ for this language can be represented as a directed graph, where each vertex is an element of D and each directed edge represents an element of the relation $I_p(p)$.

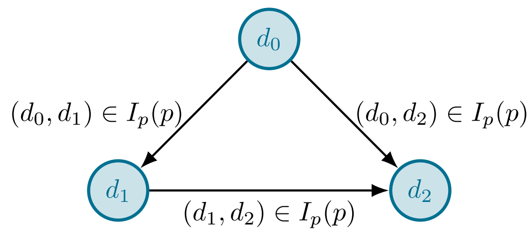

Figure 3: The first-order structure S can be visualised as a directed graph.

2 Axiomatic Proofs for Propositional Logic

A *proof system* is a system for determining the validity of formulas.

An obvious system would be to construct a truth table and check that all rows give a true result. However, this naive approach has an exponential time complexity[4], meaning that it will become increasingly impractical as more and more propositions are introduced.

To alleviate this issue, we shall introduce a different approach called a *Hilbert-style proof system.* This is an *axiomatic proof system* in which theorems are generated using axioms and inference rules.

2.1 Hilbert-style proof system

Firstly, we limit our propositional language to only use the connectives $\neg$ and $\rightarrow$. Double negations are prohibited.

Moreover, we will note some *axioms* that are known to be valid, and then try to derive other valid formulas from the axioms. Below we list three examples of *schemas*, from which axioms may be obtained by substituting any formulas in place of p, q and r.

I. $p \rightarrow (q \rightarrow p)$ (implication is true if consequent is true)

II. $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$ (implication chain as hypothetical syllogism)

III. $(\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$ (contrapositive)

Axioms on their own are insufficient in establishing a proof system. We also need *inference rules*, which stipulate how conclusions can be derived from premises. One of the main inference rules is *modus ponens*, which states that if you have proved both the formula ϕ and the implication $(\phi \rightarrow \psi)$, then you may deduce the conclusion ψ.

$$\frac{\phi \quad (\phi \rightarrow \psi)}{\psi} \qquad \text{(modus ponens)}$$

In this system, a *proof* is a sequence of formulas

$$\phi_0,\ \phi_1,\ \phi_2,\ \cdots,\ \phi_n$$

such that for each $i \leq n$, the formula ϕ_i is either

- an axiom; or
- obtained from two previous formulas ϕ_j and ϕ_k in the sequence via modus ponens (for some $j, k < i$).

If such a proof exists, then the final formula ϕ_n is called a *theorem* and we may write $\vdash \phi_n$.

[4] Using this system, checking the validity of a formula with n proposition symbols requires 2^n computations.

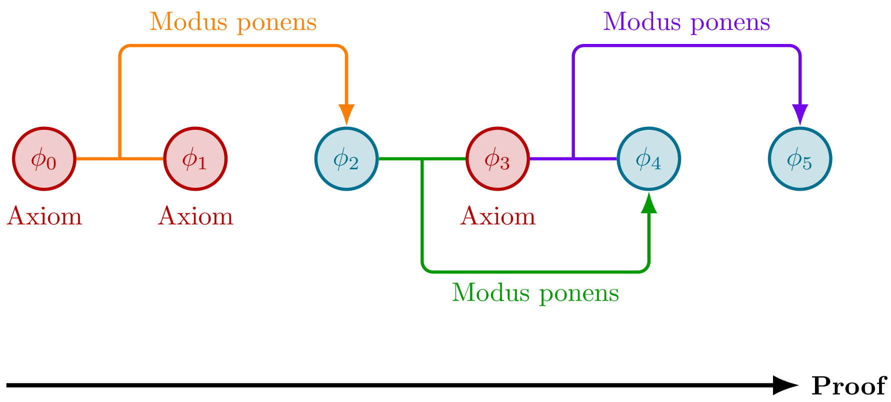

Figure 4: In a proof, every formula must be either an axiom, or derived from previous formulas via modus ponens.

For example, the theorem

$$\vdash (p \to p)$$

may be proved using the above proof system as follows.

1. $(p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p))$ (Axiom I, replacing p, q, r by $p, (p \to p), p$)
2. $p \to ((p \to p) \to p)$ (Axiom II, replacing p, q by $p, (p \to p)$)
3. $(p \to (p \to p)) \to (p \to p)$ (modus ponens, via 1 and 2)
4. $p \to (p \to p)$ (Axiom I, replacing p, q by p, p)
5. $p \to p$ (modus ponens, via 3 and 4)

To include double negations and other connectives like $\wedge$ and $\vee$, we may add more axioms to our proof system.

IV. $p \to \neg\neg p$ and $\neg\neg p \to p$ (double negation)

V. $(p \vee q) \to (\neg p \to q)$ and $(\neg p \to q) \to (p \vee q)$ (implication as disjunction)

VI. $(p \wedge q) \to \neg(p \to \neg q)$ and $\neg(p \to \neg q) \to (p \wedge q)$ (implication as conjunction)

2.2 Proofs with assumptions and the principle of explosion

Let Γ be a set of *assumptions*, i.e. formulas that are assumed to be true. Under these assumptions, a proof is defined as a sequence of formulas

$$\phi_0,\ \phi_1,\ \phi_2,\ \cdots \phi_n$$

such that for each $i \leq n$, the formula ϕ_i is either

- an axiom;
- an assumption $\phi_i \in \Gamma$; or
- obtained from two previous formulas ϕ_j and ϕ_k in the sequence via modus ponens (for some $j, k < i$).

If such a proof exists, then we may write $\Gamma \vdash \phi_n$.

For example, given the set of assumptions $\Gamma = \{p\}$, we may prove that $q \rightarrow p$ using the Hilbert-style proof system, as demonstrated below.

1. $p \rightarrow (q \rightarrow p)$ (Axiom I)
2. p (Assumption)
3. $q \rightarrow p$ (modus ponens, via 1 and 2)

Proving with assumptions can be quite tricky due to the *principle of explosion*[5], which states that any statement can be proven from a contradiction. In other words, it is possible to prove any given statement, true or false, using a proof system as long as at least one of the assumptions in Γ is false.

We shall illustrate this principle as follows. Let Γ be the set containing the invalid assumption $\neg(q \rightarrow q)$. We will use the Hilbert-style proof system to prove an arbitrary formula p under this assumption.

5. $q \rightarrow q$ (proven previously)
6. $(q \rightarrow q) \rightarrow \neg\neg(q \rightarrow q)$ (Axiom IV, replacing p by q)
7. $\neg\neg(q \rightarrow q)$ (modus ponens, via 5 and 6)
8. $\neg\neg(q \rightarrow q) \rightarrow (\neg p \rightarrow \neg\neg(q \rightarrow q))$ (Axiom I, replacing p, q by $\neg\neg(q \rightarrow q), \neg p$)
9. $\neg p \rightarrow \neg\neg(q \rightarrow q)$ (modus ponens, via 7 and 8)
10. $(\neg p \rightarrow \neg\neg(q \rightarrow q)) \rightarrow (\neg(q \rightarrow q) \rightarrow p)$ (Axiom III, replacing p, q by $p, \neg\neg(q \rightarrow q)$)
11. $\neg(q \rightarrow q) \rightarrow p$ (modus ponens, via 9 and 10)
12. $\neg(q \rightarrow q)$ (assumption)
13. p (modus ponens, via 11 and 12)

2.3 Soundness, completeness and termination

A proof system is said to be *sound* if it can only prove valid theorems. In other words, anything proven using a sound system must be valid.

$$\underbrace{\vdash \phi}_{\text{proven}} \implies \underbrace{\models \phi}_{\text{valid}} \qquad \text{(soundness)}$$

Conversely, a proof system is said to be *complete* if it can prove any given valid theorem. In other words, if a formula is valid, it must be possible to prove it under a complete system.

$$\underbrace{\models \phi}_{\text{valid}} \implies \underbrace{\vdash \phi}_{\text{proven}} \qquad \text{(completeness)}$$

The main problem with the Hilbert-style proof system is that although it is relartively easy to check that a proof of a formula is correct, there is no systematic way for efficiently constructing proofs.

Moreover, even if a system is sound and complete, we don't know how long the proof for a given formula might be. Since it is impossible for us to check all the possibilities to see if a proof exists, testing the validity of a formula remains undecidable — there is no effective method for determining validity that terminates in finite time.

[5] This principle is sometimes referred to in Latin as *ex falso quodlibet*, which literally translates to "from falsehood, anything [follows]".

3 Propositional tableau

In view of the impracticality of Hilbert-style proof systems, we introduce below an easier and more implementable method for determining a formula's validity — *tableaus*.

Here is a brief overview of how a tableau works. Suppose we want to check the satisfiability of a formula ϕ. This formula will be placed at the root of a binary tree, called a tableau. We use a variety of expansion rules to grow the tree until it is complete. An *open* tableau indicates that ϕ is satisfiable, while a *closed* tableau indicates that ϕ is unsatisfiable.

To determine the validity of a formula, simply construct a tableau for $\neg\phi$. If the resultant tableau is open, then $\neg\phi$ is satisfiable, so ϕ is invalid. On the contrary, if the resultant tableau is closed, then $\neg\phi$ must be unsatisfiable, so ϕ is valid.

3.1 Constructing a tableau

In a tableau, every node is marked with a formula. To build a tableau for a formula ϕ, begin by placing ϕ at the root of a binary tree. Then, we repeat the following process:

1. Select a formula in the tree that has not been selected before. The formula must not be a literal.
2. Choose the expansion rule (see below) that applies to the selected formula.
3. For each leaf node, add new children nodes in accordance to the chosen expansion rule.
4. Place a tick beside the selected formula to make sure we don't expand it again.

There are two types of expansion rules:

- α-rules, which create one new child per leaf node; and
- β-rules, which create two new children per leaf node.

Figures 5 and 6 depict the α- and β rules respectively. Nodes that are newly created by each rule are highlighted in blue.

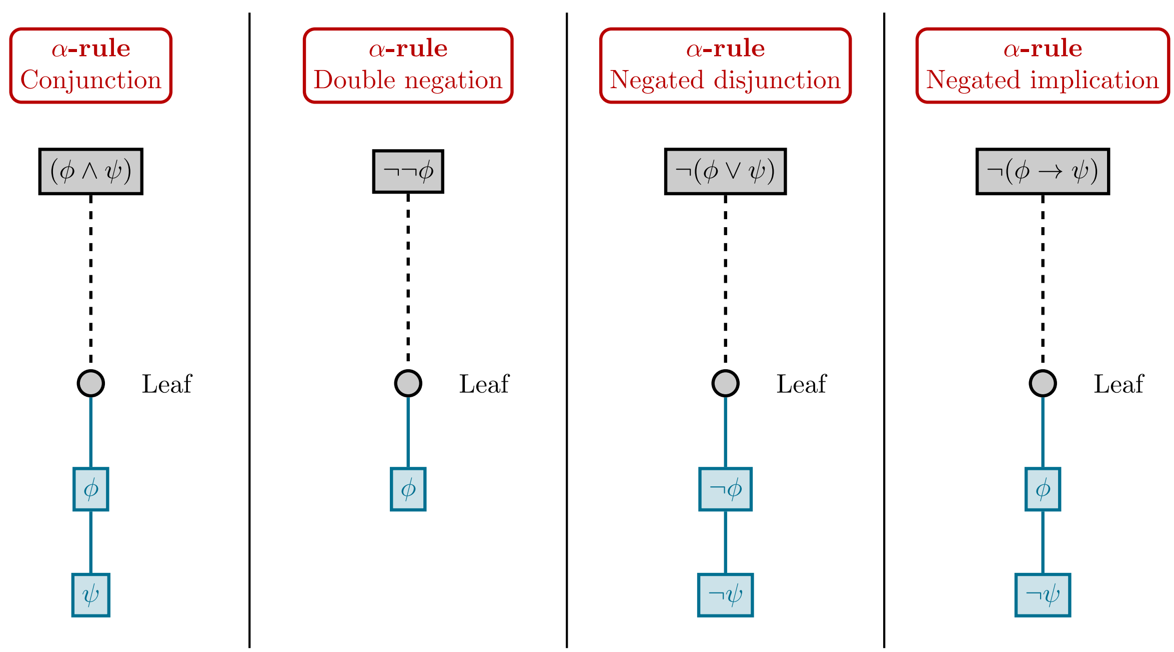

Figure 5: The four α-rules for constructing propositional tableaus.

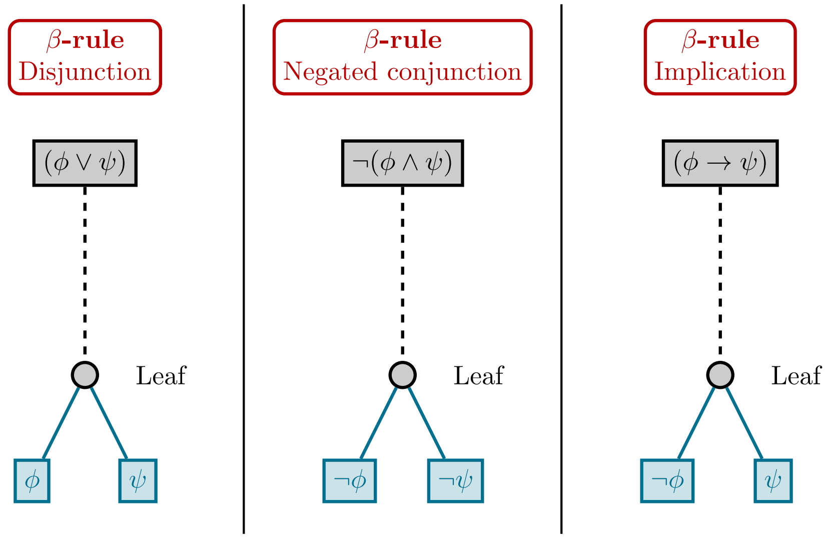

Figure 6: The three β-rules for constructing propositional tableaus.

In general, nodes located in the same branch[6] are considered in conjunction while the different branches are considered to be disjuncted. As a result, a tableau is a tree-like representation of a formula that is a disjunction of conjunctions, à la disjunctive normal form (DNF).

A tableau is considered *complete* if every node is either ticked (already expanded) or a literal. When a tableau is complete, we can determine the original formula's satisfiability as follows.

- A branch containing both a propositional letter and its negation (p and $\neg p$) is said to be *closed*, which we denote as $\oplus$. Otherwise, it is *open*.
- A tableau where all branches are closed is said to be *closed*, meaning that the formula at its root is unsatisfiable. Contrarily, a tableau with at least one open branch is said to be *open*, indicating that the formula is satisfiable.

3.2 Example of constructing a tableau and converting to DNF

To check if the formula

$$((p \vee q) \wedge (\neg p \to \neg q))$$

is satisfiable, we construct its tableau, as shown in figure 7.

Since only one of the four branches is closed, this formula is satisfiable. In fact, the literals in each open branch give a possible valuation that satisfies the given formula. For instance, the second branch from the left contains the literals p and $\neg q$. This indicates that the formula is true when p is true and q is false.

[6] A *branch* is defined as a path from the root of the tableau to one of its leaves.

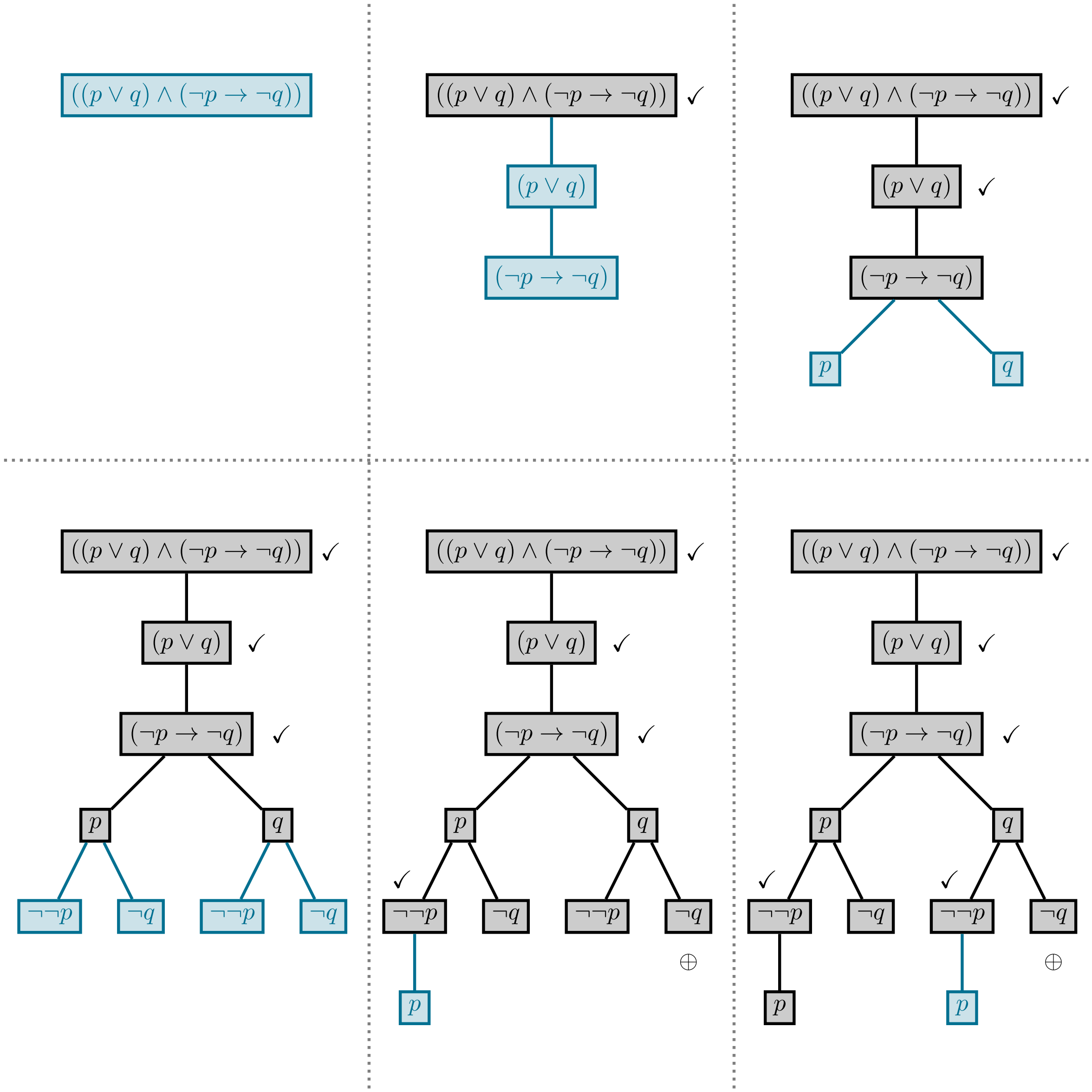

Figure 7: Constructing the tableau of $((p \vee q) \wedge (\neg p \to \neg q))$. Read from left to right and from top to bottom.

It follows that given the tableau of a formula, its DNF equivalent can be expressed as

$$\bigvee_{\text{open branch } \Theta} \left(\bigwedge \{\text{literals in } \Theta\} \right).$$

As always, the CNF of a formula can be obtained by negating the DNF form of its negation.

4 Predicate tableau

In first-order logic, a *literal* is an atom or its negation, i.e.

$$r^n(t_1, t_2, \cdots, t_n)$$

or

$$\neg r^n(t_1, t_2, \cdots, t_n)$$

where r^n is an n-ary predicate and t_i is a term.

The method for tableau construction in first-order logic is identical to that in propositional logic, but with a few extra expansion rules for dealing with quantifiers.

4.1 Expansion rules

In addition to α- and β-rules, we also require δ- and γ-rules, as depicted in Figures 8 and 9.

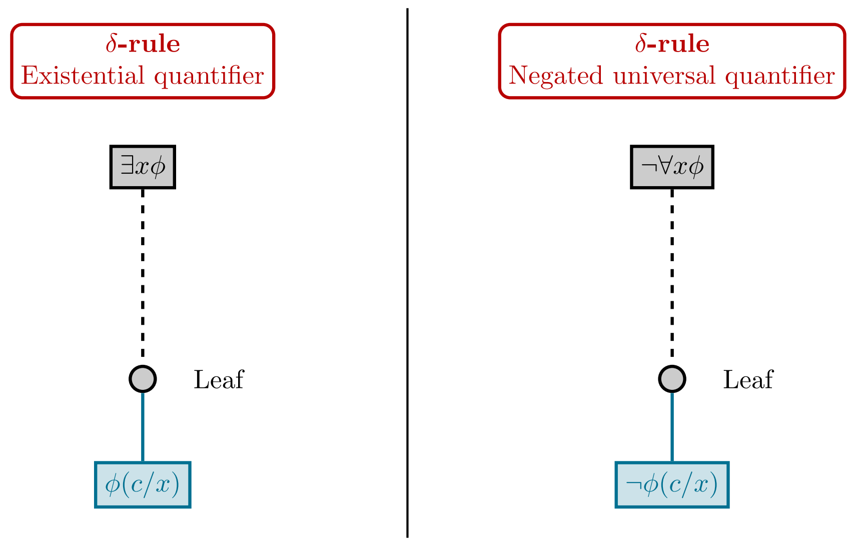

Figure 8: The two δ-rules for constructing predicate tableaus. In both rules, c should be a new constant that has not been used in the tableau before.

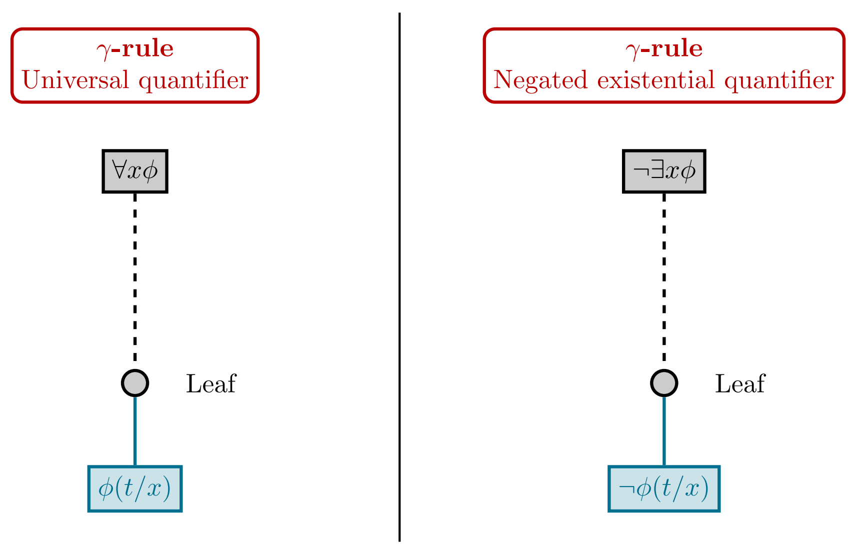

Figure 9: The two γ-rules for constructing predicate tableaus. In both rules, t is a closed term. Formulas should **not** be ticked following a γ-rule expansion.

When applying a δ-rule, make sure to introduce a new constant symbol that is not used anywhere before in the tableau. This new constant acts as a *witness*[7] for the existential statement.

Compared to the other rules, γ-rules are usually applied last. When applying a γ-rule, instantiate x with a closed term that appeared earlier in the current branch[8]. Formulas expanded via a γ-rule should **not** be ticked.

4.2 Termination

Similar to propositional tableaus, a predicate tableau's branch is closed if it contains both a literal $P(t_1, t_2, \cdots, t_n)$ and its negation $\neg P(t_1, t_2, \cdots, t_n)$. Otherwise, it is open.

The tableau terminates when:

- Every branch is closed. This shows that the root formula is unsatisfiable.
- All formulas are fully expanded and no further rules can be applied. If at least one branch remains open and cannot be further expanded, the tableau is open, indicating the root formula's satisfiability.

4.3 Example

Suppose we want to check whether the formula

$$(\forall x\ \neg p(x) \rightarrow \neg \exists y\ p(y))$$

is valid. To do this, we place its negation at the root of our tableau.

[7] Or: *Skolem witness*.

[8] This closed term should **not** be new.

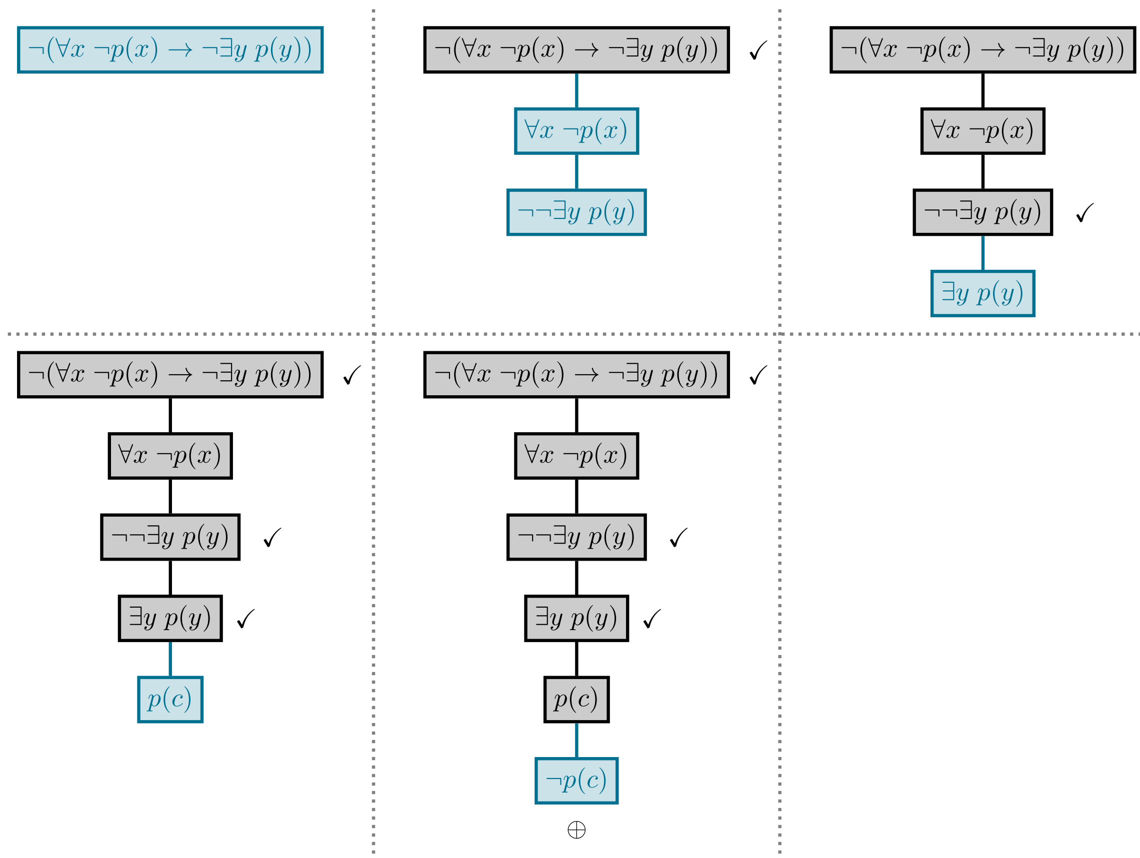

Figure 10: Constructing the tableau of $\neg(\forall x\ \neg p(x) \rightarrow \neg\exists y\ p(y))$. Read from left to right and from top to bottom. In the fourth step (bottom left), the existential formula $\exists y\ p(y)$ is expanded via a δ-rule by introducing a new constant c. In the last step (bottom middle), the universal formula $\forall x\ \neg p(x)$ is expanded via a γ-rule by replacing all bounded instances of x with the closed term c from earlier in the current branch, thereby producing both $p(c)$ and $\neg p(c)$ in the same branch. This results in a closed branch and hence a closed tableau, indicating that the formula at the root is unsatisfiable.

As shown, the negation $\neg(\forall x\ \neg p(x) \rightarrow \neg\exists y\ p(y))$ is unsatisfiable. This means that our original formula must be valid.

4.4 Non-termination

Predicate tableaus may not always terminate.

For instance, if a tableau unendingly generates nodes that require expansion via δ-rules, more and more constants would be introduced, and the number of γ-rule applications required would increase dramatically. This may result in non-termination.

Before we elaborate on this non-terminating scenario, we must note that in order to systematically handle possible infinite expansions, we should adopt a fair application strategy. A tableau construction is *fair* if

- Every formula that can still be expanded eventually will be, and
- Every formula that falls under a γ-rule will eventually be instantiated via that rule using all closed terms that appear in its branch.

This ensures that if the tableau can close, it will close after finitely many steps. We won't miss a contradiction because we ignored a rule.

Now, assuming a fair application strategy,

- If a branch keeps repeating the same configuration of formulas over and over with no new information, it is effectively saturated. This branch is then considered open, meaning that the root formula is satisfiable. This is because we may construct an infinite model for the root formula by reading off literals in the limit of the infinitely "looping" branch in the same way as we did for propositional tableaus. See Figure 11 for an example.
- If a branch runs indefinitely without closure, the satisfiability of the root formula is **undecided** and **inconclusive**.

(1) $\neg(\forall x \neg q(x) \vee \exists x \forall y \neg(x < y))$ ✓

$\alpha(1)$

(2) $\neg\forall x \neg q(x)$ ✓

(3) $\neg\exists x \forall y \neg(x < y)$

$\delta(2, c)$

(4) $\neg\neg q(c)$ ✓

$\alpha(4)$

(5) $q(c)$

$\gamma(3, c)$

(6) $\neg\forall y \neg(c < y)$ ✓

$\delta(6, d)$

(7) $\neg\neg(c < d)$ ✓

$\alpha(7)$

(8) $(c < d)$

$\gamma(3, d)$

(9) $\neg\forall y \neg(d < y)$ ✓

$\delta(9, e)$

(10) $\neg\neg(d < e)$ ✓

$\alpha(10)$

(11) $(d < e)$

...

Open tableau — the tableau will never close, hence the **root formula is satisfiable and the original formula is not valid.**

Figure 11: A non-terminating tableau where the root formula is satisfiable.

4.5 Free variables

Predicate tableaus are predominantly designed to work on sentences, where free variables are not allowed. To prove the validity of a formula with free variables, we may prefix it with an appropriate universal quantifier. For instance, if we want to show that

$$x < 5$$

is valid, where x is a free variable. Notice that this is equivalent to showing the validity of

$$\forall x\ x < 5$$

which uses a universal quantifier to remove the free variable. Consequently, we can simply construct a tableau with

$$\neg\forall x\ x < 5$$

at its root and check its satisfiability as usual.

4.6 More on fairness

Note that when applying expansion rules in non-terminating predicate tableaus, it is always possible to find a fair application strategy.

To see why this is, consider a countably infinite set of processes $P = \{P_1, P_2, \cdots, P_i, \cdots\}$, each awaiting some input. When a process receives an input, this may result in the creation of a new process, which is subsequently added to P. We want to find some fair schedule where if any process P_i is awaiting input at time t, then eventually at some time $t' > t$ it will receive some input.

Since the set P always remains countable even when a new process is created and added to it, such a schedule must exist.

5 More on tableaus, their representations and their properties

5.1 Preliminaries: On subsets and subformulas

In this subsection, we present two lemmas on subsets and subformulas. Given a formula ϕ, a subformula is a substring of ϕ that is also a formula.

Lemma. *A set S of cardinality n has 2^n subsets.*

Proof. To construct a subset $S' \subseteq S$, each element of S can either appear or not appear in S'. This involves a total of n independent binary choices. Hence, there are 2^n possible subsets. $\square$

Lemma. *A formula ϕ has at most $|\phi|$ subformulas.*

Proof. Consider the parse tree of ϕ, where every node contains exactly one symbol and is the root of a subtree that represents a subformula of ϕ. Hence we have

$$\begin{aligned}
&\text{Number of subformulas of } \phi \\
=\ &\text{Number of nodes in parse tree of } \phi \\
=\ &\text{Number of symbols that appear in parse tree of } \phi \\
\leq\ &\text{Number of symbols in } \phi && \text{(as brackets do not appear in parse trees)} \\
=\ &|\phi|. && \square
\end{aligned}$$

5.2 Tableaus as lists of theories

While tableaus can be visualised as trees, they can also be represented as lists. To see how this works, let us first review the α- and β- rules. As shown in tables 4 and 5, each α-rule produces at most two new nodes α_1 and α_2, while each β-rule produces at most two new nodes β_1 and β_2.

α	α_1	α_2
$(A \wedge B)$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$
$\neg(A \to B)$	A	$\neg B$
$\neg\neg A$	A	-

Table 4: The α-rules tabulated.

β	β_1	β_2
$(A \vee B)$	A	B
$(A \to B)$	$\neg A$	B
$\neg(A \wedge B)$	$\neg A$	$\neg B$

Table 5: The β-rules tabulated.

Instead of a tree, we represent a tableau as a list of *theories*, where each theory is a set of unticked formulas in a branch that has not yet closed. Figure 12, based on Figure 7, shows the construction of a propositional tableau in both tree and list form.

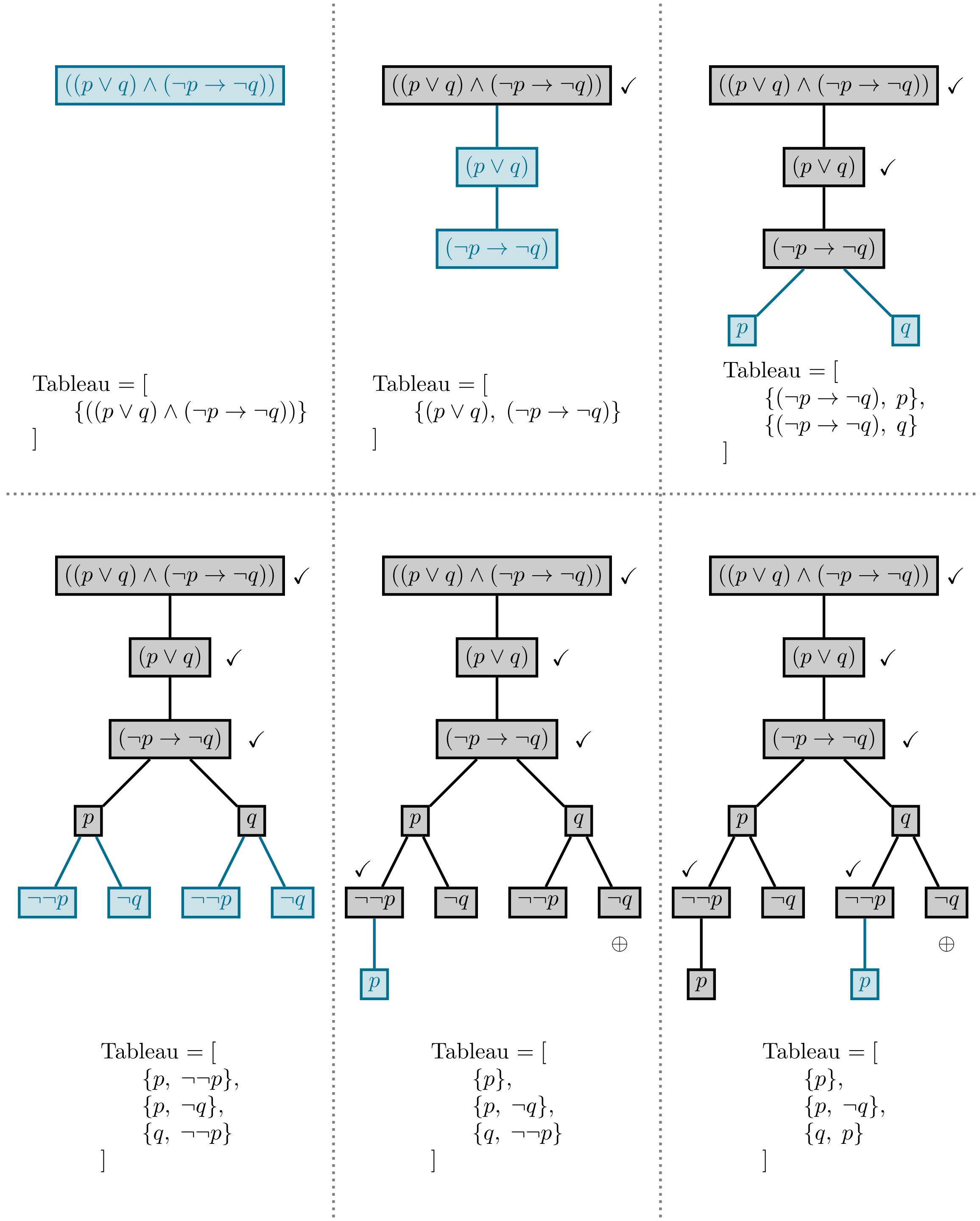

Figure 12: Constructing the tableau of $((p \vee q) \wedge (\neg p \to \neg q))$ as a tree and as a list. Read from left to right and from top to bottom.

Below is a pseudocode snippet outlining how the propositional tableau method can be implemented programmatically using the list representation. Here, `Tableau` is initialised as a queue of theories. The variables α_1, α_2, β_1 and β_2 refer to the ones labelled in Tables 4 and 5.

```
def is_satisfiable(φ):
    Tableau = Queue()
    Tableau.enqueue(φ)

    while Tableau is not empty:
        # Dequeue a theory Σ from the tableau
        Σ = Tableau.dequeue()

        if Σ is fully expanded and has no contradictory literals:
            return True
        else:
            fairly select a non-literal ψ from Σ

            if α-rule is applicable to ψ:
                Σ = Σ with ψ replaced by α1 and α2
                if Σ has no contradictory literals and is not in Tableau:
                    Tableau.enqueue(Σ)

            elif β-rule is applicable to ψ:
                Σ1 = Σ with ψ replaced by β1
                if Σ1 has no contradictory literals and is not in Tableau:
                    Tableau.enqueue(Σ1)

                Σ2 = Σ with ψ replaced by β2
                if Σ2 has no contradictory literals and is not in Tableau:
                    Tableau.enqueue(Σ2)

    # Empty queue in Tableau
    return False
```

We can easily modify this algorithm to represent predicate tableaus by adding the following cases to the innermost `if-elif` statement.

```
elif δ-rule is applicable to ψ:
    if ψ = ∃x θ(x):
        Σ = Σ with ψ replaced by θ(c) for some new constant c
    elif ψ = ¬∀x θ(x):
        Σ = Σ with ψ replaced by ¬θ(c) for some new constant c

    if Σ has no contradictory literals and is not in Tableau:
        Tableau.enqueue(Σ)

elif γ-rule is applicable to ψ:
    if ψ = ∀x θ(x):
        fairly select a closed term t from Σ
        Σ = Σ with θ(t) added
    elif ψ = ¬∃x θ(x):
        fairly select a closed term t from Σ
        Σ = Σ with ¬θ(t) added

    if Σ has no contradictory literals and is not in Tableau:
        Tableau.enqueue(Σ)
```

5.3 Proving the termination and soundness of tableaus

Here we will use the list representation of tableaus to prove several of their properties.

Theorem. *The propositional tableau algorithm must terminate for any root formula ϕ.*

Proof. Let X be the set of subformulas of ϕ and negations thereof. Double negations of subformulas are excluded. Since ϕ has at most $|\phi|$ subformulas, the cardinality of X cannot exceed $2|\phi|$.

Notice that any theory in the tableau of ϕ must be a subset of X. Since each theory can only be enqueued to the tableau at most once, the number of enqueued theories must not exceed $2^{2|\phi|}$. Therefore, the algorithm must terminate in no more than $2^{2|\phi|}$ steps. □

Theorem. *The propositional tableau algorithm is sound.*

Proof. To prove soundness, we must show that $\vdash \phi \implies \models \phi$, i.e.

$$\text{Tableau of } \neg\phi \text{ is closed} \implies \neg\phi \text{ is unsatisfiable.}$$

Taking the contrapositive and renaming our variables, we see that this is equivalent to showing that

$$\phi \text{ is satisfiable} \implies \text{tableau of } \phi \text{ never closes.}$$

Assume ϕ is satisfiable. This means there is some truth function v for which $v(\phi) = \top$. We want to prove by induction that the following statement $P(n)$ holds for any $n \in \mathbb{N}$.

Statement. After executing n iterations of the `while` loop, there exists a theory Σ in the tableau where $\theta \in \Sigma \to v(\theta) = \top$.

Base case. When $n = 0$, the tableau is given by $[\{\phi\}]$. The base case holds trivially by taking $\Sigma = \{\phi\}$ and noting $v(\phi) = \top$.

Step case. Assume $P(n)$ holds for some $n \in \mathbb{N}$. This means that after n iterations there exists some Σ in the tableau where $\theta \in \Sigma \to v(\theta) = \top$. For $P(n+1)$, consider executing an additional iteration.

- If any theory other than Σ is dequeued, Σ will still remain in the tableau unchanged by the end of the iteration. Therefore, $P(n+1)$ holds.
- If Σ is dequeued and a non-literal $\psi \in \Sigma$ is selected, we have $v(\psi) = \top$ (by induction hypothesis).
  - If an α-rule is applicable to ψ, then ψ will be replaced by two formulas α_1 and α_2 which — according to properties of truth functions — satisfy $v(\alpha_1) = v(\alpha_2) = \top$. Therefore, the statement $\theta \in \Sigma[\psi/\{\alpha_1, \alpha_2\}] \to v(\theta) = \top$ is still true.
  - If a β-rule is applicable to ψ, then we enqueue two new theories: Σ_1 where ψ is replaced by β_1; and Σ_2 where ψ is replaced by β_2. According to properties of truth functions, at least one of $v(\beta_1) = \top$ and $v(\beta_2) = \top$ is true. Therefore, we have either $\theta \in \Sigma_1 \to v(\theta) = \top$ or $\theta \in \Sigma_2 \to v(\theta) = \top$.

Hence proved. □

Theorem. *The predicate tableau algorithm is sound.*

Proof. Similar to the above, we want to show that

$$\phi \text{ is satisfiable} \implies \text{tableau of } \phi \text{ never closes.}$$

Assume ϕ is satisfiable. This means there is some first-order structure S and variable assignment A for which $S \models_A \phi$. We want to prove by induction that the following statement $P(n)$ holds for any $n \in \mathbb{N}$.

Statement. After executing n iterations of the `while` loop, there exists a theory Σ in the tableau where $\theta \in \Sigma \to S \models_A \theta$ for some structure S and variable assignment A.

Base case. When $n = 0$, the tableau is given by $[\{\phi\}]$. The base case holds trivially by taking $\Sigma = \{\phi\}$ and noting $S \models_A \phi$.

Step case. Assume $P(n)$ holds for some $n \in \mathbb{N}$. This means that after n iterations there exists some Σ in the tableau where $\theta \in \Sigma \to S \models_A \phi$ for some structure S and variable assignment A. For $P(n+1)$, consider executing an additional iteration.

- If any theory other than Σ is dequeued, Σ will still remain in the tableau unchanged by the end of the iteration. Therefore, $P(n+1)$ holds.

- If Σ is dequeued and a non-literal $\psi \in \Sigma$ is selected, we have $S \models_A \psi$ (by induction hypothesis).

  - If an α-rule is applicable to ψ, then ψ will be replaced by two formulas α_1 and α_2 which — according to properties of truth functions — satisfy $S \models_A \alpha_1$ and $S \models_A \alpha_2$. Therefore, the statement $\theta \in \Sigma[\psi/\{\alpha_1, \alpha_2\}] \to S \models_A \theta$ is still true.

  - If a β-rule is applicable to ψ, then we enqueue two new theories: Σ_1 where ψ is replaced by β_1; and Σ_2 where ψ is replaced by β_2. According to properties of truth functions, at least one of $S \models_A \beta_1$ and $S \models_A \beta_2$ is true. Therefore, we have either $\theta \in \Sigma_1 \to S \models_A \theta$ or $\theta \in \Sigma_2 \to S \models_A \theta$.

  - If a δ-rule is applicable to ψ, then it is either of the form $\exists x\ \theta(x)$ or $\neg\forall x\ \theta(x)$.

    For $\exists x\ \theta(x)$, we know by induction hypothesis that $S \models_A \exists x\ \theta(x)$. This means there exists some s within the domain of S such that $S \models_{A[x \to s]} \theta(x)$. The δ-rule replaces ψ with $\theta(c)$ where c is a new constant. Let S' be a first-order structure identical to S except $I(c) = s$. Then $S' \models_A \Sigma[\exists x\ \theta(x)/\theta(c)]$ holds.

    A similar argument can be made for $\neg\forall x\ \theta(x)$, but is omitted here for brevity.

  - If a γ-rule is applicable to ψ, then it is either of the form $\forall x\ \theta(x)$ or $\neg\exists x\ \theta(x)$.

    For $\forall x\ \theta(x)$, we know by induction hypothesis that $S \models_A \forall x\ \theta(x)$. It follows that $S \models_A \theta(t)$ for any closed term t. Therefore, the statement $\theta \in \Sigma[\psi/\theta(t)]$ is still true.

    A similar argument can be made for $\neg\exists x\ \theta(x)$, but is omitted here for brevity.

    Note that unlike in the δ-rule case, this case does not involve the modification of S or A.

Hence proved. $\square$

5.4 Hypotheses

Similar to how assumptions can be added to axiomatic proof systems, we can use tableaus to prove from *hypotheses*. Suppose we want to show that the formula ϕ is valid under a set of hypotheses $\Gamma = \{\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_{n-1}\}$. There are several ways of doing this via tableaus.

- **As a tree:** Place $\neg\phi$ at the root of a tableau. Continue to construct the tableau as usual, but with the additional rule that at any stage we may select some hypothesis $\gamma_i \in \Gamma$ and add a node labelled γ_i at any leaf.

- **As a tree, assuming a finite set of hypotheses:** Place $\neg\phi$ along with all hypotheses $\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_{n-1}$ all in a single tableau branch. Continue to construct the tableau as usual.
- **As a list/queue:** Initialise the tableau as $[\Gamma \cup \{\neg\phi\}]$. Continue to construct the tableau as usual.

If the tableau eventually closes (or becomes empty, in the case of the list/queue representation), we may write

$$\Gamma \vdash \phi$$

to denote that ϕ is valid under the hypotheses Γ.

5.5 Equality rules

Recall that in predicate logic, the equality symbol "=" is always interpreted as true equality in predicate logic.

Therefore, when constructing a tableau for a formula containing an equality symbol, we must also assume the following equality rules.

- If in some branch we have both $A(t)$ and $t = s$, then we may add $A(s)$ to its leaf.
- If in some branch we have both $A(t)$ and $s = t$, then we may add $A(s)$ to its leaf.
- If a branch contains a formula in the form $\neg(t = t)$, that branch is closed.

For example, suppose we want to prove that under the hypothesis $s = t$, the formula $t = s$ is valid, i.e.

$$s = t \vdash t = s.$$

We set up a tableau containing the hypothesis, followed by the formula's negation. We then complete the tableau as normal. See Figure 13.

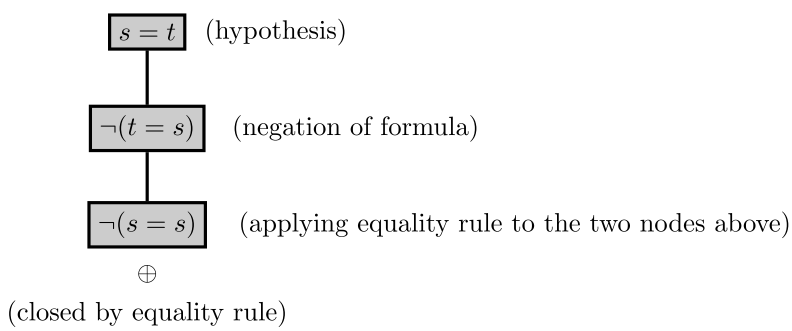

Figure 13: Proving the validity of $s = t \vdash t = s$ by constructing a predicate tableau using equality rules.

5.6 Parents and ancestors

If a theory Σ in the tableau is dequeued and a new theory Σ_1 (and possibly Σ_2) is subsequently enqueued, then Σ is a *parent* of Σ_1 and Σ_2. We denote this relationship using the function P, defined as follows.

$$\begin{aligned} P(\Sigma) &= \Sigma' \qquad \text{if the parent of } \Sigma \text{ is } \Sigma' \\ P^0(\Sigma) &= \Sigma \\ P^{n+1}(\Sigma) &= P(P^n(\Sigma)) \end{aligned}$$

We say that Σ' is an *ancestor* of Σ' if

$$P^n(\Sigma) = \Sigma'$$

for some $n \in \mathbb{N}$. For example, if a tableau is initialised with only one tableau, then that tableau is an ancestor of every theory in the tableau, including itself.

5.7 Proving the completeness of propositional tableaus

Theorem. *The propositional tableau algorithm is complete.*

Proof. To prove completeness, we must show that $\models \phi \implies \vdash \phi$, i.e.

$$\neg\phi \text{ is unsatisfiable} \implies \text{tableau of } \neg\phi \text{ is closed.}$$

Taking the contrapositive and renaming our variables, we see that this is equivalent to showing that

$$\text{Tableau of } \phi \text{ does not close} \implies \phi \text{ is satisfiable.}$$

Assume that the tableau of ϕ does not close. This means that there is some theory in the tableau that, when dequeued, is found to be fully expanded and have no contradictory literals, thus causing `is_satisfiable(`ϕ`)` to return `True`. We denote this theory by Σ.

Let v be a valuation where for each propositional letter p, we have

$$v(p) = \top \iff p \in \Sigma.$$

Extending v to a truth function, it follows that

$$\phi \in \Sigma \implies v(\phi) = \top \tag{*}$$

for all formulas ϕ.

We shall now prove by induction that the statement

$$\phi \in P^n(\Sigma) \implies v(\phi) = \top$$

is true for all $n \in \mathbb{N}$.

Base case. For $n = 0$, we want to show that $\phi \in P^0(\Sigma) \implies v(\phi) = \top$. This was already established by equation (*).

Step case. Assume for some $n \in \mathbb{N}$ that the theory $P^n(\Sigma)$ satisfies $\phi \in P^n(\Sigma) \implies v(\phi) = \top$. Now consider its parent $P^{n+1}(\Sigma)$. We know that some expansion rule — α or β — is used to replace some formula in the parent theory $P^{n+1}(\Sigma)$ with a new formula to form the child theory $P^n(\Sigma)$.

- If this is an α-rule, then both formulas α_1 and α_2 will be present in the child theory $P^n(\Sigma)$. By the induction hypothesis, we have $v(\alpha_1) = v(\alpha_2) = \top$, so $v(\alpha) = \top$.

- If this is an β-rule, then one of the formulas β_1 and β_2 will be present in the child theory $P^n(\Sigma)$. By the induction hypothesis, we have either $v(\beta_1) = \top$ or $v(\beta_2) = \top$. In either case we have $v(\beta) = \top$.

Since no other formulas are changed when generating $P^n(\Sigma)$ from $P^{n+1}(\Sigma)$, we have $\phi \in P^{n+1}(\Sigma) \implies v(\phi) = \top$, establishing the step case.

By the principles of induction, we have

$$\phi \in P^n(\Sigma) \implies v(\phi) = \top$$

for all $n \in \mathbb{N}$. Since the initial theory $\{\phi\}$ is the ancestor of all theories in the tableau, we have $v(\phi) = \top$, implying satisfiability. □

5.8 Herbrand structures

A closed term contains only constant symbols and function symbols, with no free variables. A *Herbrand structure* is a first-order structure $H = (D, I)$ where

- the domain D is defined as the set of closed terms; and
- the interpretation $I = (I_c, I_f, I_p)$ is such that

$$I_c(c) = c \qquad \text{(interpret each constant symbol as the symbol itself)}$$
$$I_f(f^n(d_1, d_2, \cdots, d_n)) = f^n(d_1, d_2, \cdots, d_n) \qquad \text{(interpret each function as the string itself)}$$

  and I_p can be chosen freely.

It follows that for any closed term t and any variable assignment[9] A, we have $[t]^{H,A} = t$.

Herbrand's theorem, which we will not prove here, states that if ϕ does not contain the equality symbol "=" but is satisfiable in some structure S and variable assignment A, then it must also be satisfiable in some Herbrand structure H and variable assignment B.

5.9 Ranks

We define the *rank* of a first-order formula ϕ, denoted as $\mathrm{Rk}(\phi)$, as follows.

$$\begin{aligned}
\mathrm{Rk}(P(t_0, t_1, \cdots, t_{k-1})) &= 1 \\
\mathrm{Rk}(\neg\phi) &= \mathrm{Rk}(\phi) + 1 \\
\mathrm{Rk}(\phi \circ \psi) &= \mathrm{Rk}(\phi) + \mathrm{Rk}(\psi) + 1 && \text{(where $\circ$ is a binary connective)} \\
\mathrm{Rk}(\exists x\ \phi) &= \mathrm{Rk}(\phi) + 1 \\
\mathrm{Rk}(\forall x\ \phi) &= \mathrm{Rk}(\phi) + 1
\end{aligned}$$

In other words, $\mathrm{Rk}(\phi)$ is number of nodes in the parse tree of ϕ.

5.10 Proving the completeness of the predicate tableaus

Lemma (König's Tree Lemma)**.** *Let T be a tree where each node has only finitely many immediate successors. If each branch is of finite length, then the number of nodes in the tree is finite.*

Proof. We prove the contrapositive of the lemma: Assuming that T has infinitely many nodes, it must contain an infinite branch.

Assume T has infinitely many nodes. Let P be a path starting at the root of T. We say that a node n in T is *bottomless* if there are infinitely many nodes below n in the tree. It follows that the root of T is bottomless.

Recall that each node has only finitely many immediate successors. Therefore, the root must only have finitely many (immediate) children. If all of these children are not bottomless, then they must all have finitely many children, which contradicts the fact that the root is bottomless and has infinitely many nodes beneath it[10]. Hence, the root has at least one bottomless child. Select any one of these bottomless children and append it to the path P.

This process of selecting a bottomless child of the last node of P and then adding it to P can be repeated indefinitely to create an infinite branch. □

Theorem. *Predicate tableaus are complete.*

[9] The variable assignment here is irrelevant since t is a closed term.
[10] This is because the sum of a finite number of finite numbers is finite.

Proof. To prove completeness, we must show that $\models \phi \implies \vdash \phi$, i.e.

$$\neg\phi \text{ is unsatisfiable } \implies \text{ tableau of } \neg\phi \text{ is closed.}$$

Taking the contrapositive and renaming our variables, we see that this is equivalent to showing that

$$\text{Tableau of } \phi \text{ never closes under fair expansion schedule } \implies \phi \text{ is satisfiable.}$$

Assume the tableau of a formula ϕ never closes under a fair expansion schedule. By the contrapositive of König's Tree Lemma, there must exist an infinite sequence of theories $\Sigma_0, \Sigma_1, \Sigma_2, \cdots$ from the tableau where $\Sigma_n = P(\Sigma_{n+1})$. Let $\Sigma = \cup_{n \in \mathbb{N}} \Sigma_n$. Since a fair schedule is used, we have

$$\begin{aligned}
\alpha \in \Sigma &\implies \alpha_1 \in \Sigma \text{ and } \alpha_2 \in \Sigma \\
\beta \in \Sigma &\implies \beta_1 \in \Sigma \text{ or } \beta_2 \in \Sigma \\
\exists x\, \theta(x) \in \Sigma &\implies \theta(c) \in \Sigma \quad \text{for some } c \\
\neg\forall x\, \theta(x) \in \Sigma &\implies \neg\theta(c) \in \Sigma \quad \text{for some } c \\
\forall x\, \theta(x) \in \Sigma &\implies \theta(t) \in \Sigma \quad \text{for all closed terms } t \\
\neg\exists x\, \theta(x) \in \Sigma &\implies \neg\theta(t) \in \Sigma \quad \text{for all closed terms } t. \qquad (*)
\end{aligned}$$

Let H be a Herbrand structure where

- the domain is the set of closed terms in Σ;
- $I(t) = t$ for all closed terms t; and
- for each n-ary predicate R^n, let

$$(t_0, t_1, \cdots, t_{n-1}) \in I(R^n) \iff R^n(t_0, t_1, \cdots, t_{n-1}) \in \Sigma.$$

We now prove by complete induction on $\mathrm{Rk}(\theta)$ that for any sentence θ, we have

$$\theta \in \Sigma \implies H \models \theta.$$

Strong induction does not require a base case.

Step case. Assume that for some $n \in \mathbb{N}$, we have

$$\theta \in \Sigma \implies H \models \theta$$

for all formulas θ with $\mathrm{Rk}(\theta) < n$. Now consider a new formula with rank n. By equations (*) and our induction hypothesis, the expansions of this new formula — all of which must have rank strictly less than n — are valid in H. It follows that this new formula is also valid in H, completing the step case.

This shows that $\theta \in \Sigma \implies H \models \theta$. Since $\phi \in \Sigma$, we have $H \models \phi$, so ϕ is satisfiable. $\square$

6 Axiomatic proofs for Predicate Logic

6.1 Recap: Axiomatic proof system for propositional logic

Recall that axiomatic proofs for propositional logic are constructed using the following axiom schemas:

I. $p \to (q \to p)$ (implication is true if consequent is true)

II. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$ (implication chain as hypothetical syllogism)

III. $(\neg p \to \neg q) \to (q \to p)$ (contrapositive)

IV. $p \to \neg\neg p$ and $\neg\neg p \to p$ (double negation)

V. $(\neg p \to q) \leftrightarrow (p \vee q)$ (implication as disjunction)

VI. $\neg(p \to \neg q) \leftrightarrow (p \wedge q)$ (implication as conjunction)

alongside the inference rule modus ponens.

$$\frac{A \quad (A \to B)}{B} \qquad \text{(modus ponens)}$$

For the sake of convenience, we may sometimes want to make use of the deduction theorem.

$$A \vdash B \iff \vdash (A \to B) \qquad \text{(deduction theorem)}$$

We may also want to incorporate extra inference rules, as listed below.

$$\frac{(A \to B) \quad (B \to C)}{A \to C} \qquad \text{(hypothetical syllogism)}$$

$$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B} \qquad \frac{A \quad B}{A \wedge B} \qquad \frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \text{(etc.)}$$

6.2 Creating an axiomatic proof system for predicate logic

To adopt this proof system for predicate logic, we must add seven more axiom schemas, as listed below. An instance of an axiom is obtained by replacing $A, B, C, \cdots$ by arbitrary formulas. Axioms VII through IX are quantifier axioms, whereas Axioms X through XIII are equality axioms.

VII. $\forall x\, \neg A \leftrightarrow \neg\exists x\, A$ (negated existential statement)

VIII. $\forall x\, A(x) \to A(t/x)$ if t is substitutable for x in A. (universal statement)

IX. $\forall x\, (A \to B) \to (\forall x\, A \to \forall x\, B)$ (universal implication)

X. $x = x$ (equality is reflexive)

XI. $(x = y) \to (y = x)$ (equality is symmetric)

XII. $(x = y) \to (t(x) = t(y/x))$ (term is unchanged by substitution)

XIII. $(x = y) \to (A(x) \to A(y/x))$ if y is substitutable for x in A.

(predicate's truth value is unchanged by substitution)

A term t is *substitutable* for x in A if no variable in t becomes bound after replacing x in A by t. For instance, if we have

$$t = \text{“}f(y)\text{”}$$
$$A = \text{“}\forall x\, \exists y\, (f(x) = y)\text{”}$$

then we **may not** use Axiom VIII and modus ponens to deduce

$$\exists y(f(f(y) = y)).$$

This is because the originally free instance of y in t (in blue) becomes bound by the existential quantifier in A (in red) after the substitution, which is not allowed.

The axiomatic proof system for predicate logic is made complete by a new inference rule called *universal generalisation*.

$$\frac{A(x)}{\forall x\ A(x)} \qquad \text{(universal generalisation)}$$

This inference rule states that if $A(x)$ is valid, then $\forall x\ A(x)$ is also valid[11].

We summarise our description of this axiomatic proof system as follows.

Axiomatic proof system for predicate logic

Axiom schemas.

I. $p \to (q \to p)$

II. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$

III. $(\neg p \to \neg q) \to (q \to p)$

IV. $p \to \neg\neg p$ and $\neg\neg p \to p$

V. $(\neg p \to q) \leftrightarrow (p \vee q)$

VI. $\neg(p \to \neg q) \leftrightarrow (p \wedge q)$

VII. $\forall x\ \neg A \leftrightarrow \neg\exists x\ A$

VIII. $\forall x\ A(x) \to A(t/x)$ if t is substitutable for x in A.

IX. $\forall x\ (A \to B) \to (\forall x\ A \to \forall x\ B)$

X. $x = x$

XI. $(x = y) \to (y = x)$

XII. $(x = y) \to (t(x) = t(y/x))$

XIII. $(x = y) \to (A(x) \to A(y/x))$ if y is substitutable for x in A.

Inference rules.

- Modus ponens.

$$\frac{A \quad (A \to B)}{B}$$

- Universal generalisation.

$$\frac{A(x)}{\forall x\ A(x)}$$

[11] Note that while this rule is sound, $A(x) \implies \forall x\ A(x)$ is not an axiom.

Additional inference rules for convenience.

- Hypothetical syllogism.

$$\frac{(A \to B) \quad (B \to C)}{A \to C}$$

- Nature of AND and OR connectives.

$$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B} \qquad \frac{A \quad B}{A \wedge B} \qquad \frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \text{(etc.)}$$

Deduction theorem.

$$A \vdash B \iff \vdash (A \to B)$$

Similar to in propositional logic, we define a *proof* to be a finite sequence of formulas

$$\phi_0,\ \phi_1,\ \phi_2,\ \cdots,\ \phi_n$$

such that for each $i \leq n$, the formula ϕ_i is either

- an axiom; or
- obtained from one or two previous formulas — ϕ_j and possibly ϕ_k — in the sequence via an inference rule (for some $j, k < i$).

If such a proof exists, then the final formula ϕ_n is called a *theorem* and we may write $\vdash \phi_n$.

So far, we have only proved the validity of formulas over arbitrary models. If we want to demonstrate validity in a particular model (or type of model), we may add a set of hypotheses Γ. If there is a sequence

$$\phi_0,\ \phi_1,\ \phi_2,\ \cdots,\ \phi_n$$

such that for each $i \leq n$, the formula ϕ_i is either

- an axiom;
- obtained from one or two previous formulas — ϕ_j and possibly ϕ_k — in the sequence via an inference rule (for some $j, k < i$); or
- a hypothesis in Γ,

then we may write $\Gamma \vdash \phi$.

For instance, suppose we want to prove a formula's validity in a *linearly ordered model.* We thus assume the following hypotheses.

- $\forall x\ \forall y\ (x < y \vee y < x \vee x = y)$ (totality)
- $\forall x\ \neg(x < x)$ (irreflexivity)
- $\forall x\ \forall y\ \forall z\ ((x < y \wedge y < z) \to (x < z))$ (transitivity)

Below shows an example of this, where we prove the validity of the formula

$$\forall x\ \forall y\ \neg(x < y \wedge y < x)$$

in a linearly ordered model.

1. $\forall x\, \forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow (x < z))$ (hypothesis)
2. $\forall x\, \forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow (x < z)) \rightarrow \forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow (x < z))$

   (Axiom VIII, with x as t)
3. $\forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow (x < z))$ (modus ponens, via 1 and 2)
4. $\forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow (x < z)) \rightarrow \forall z\, ((x < y \wedge y < z) \rightarrow (x < z))$ (Axiom VIII, with y as t)
5. $\forall z\, ((x < y \wedge y < z) \rightarrow (x < z))$ (modus ponens, via 3 and 4)
6. $\forall z\, ((x < y \wedge y < z) \rightarrow (x < z)) \rightarrow ((x < y \wedge y < x) \rightarrow (x < x))$ (Axiom VIII, with x as t)
7. $(x < y \wedge y < x) \rightarrow (x < x)$ (modus ponens, via 5 and 6)
8. $((x < y \wedge y < x) \rightarrow (x < x)) \rightarrow (\neg(x < x) \rightarrow \neg(x < y \wedge y < x))$

   (instance of $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$, provable in propositional logic)
9. $\neg(x < x) \rightarrow \neg(x < y \wedge y < x)$ (modus ponens, via 7 and 8)
10. $\forall x\, \neg(x < x)$ (hypothesis)
11. $\forall x\, \neg(x < x) \rightarrow \neg(x < x)$ (Axiom VIII, with x as t)
12. $\neg(x < x)$ (modus ponens, via 10 and 11)
13. $\neg(x < y \wedge y < x)$ (modus ponens, via 9 and 12)
14. $\forall y\, \neg(x < y \wedge y < x)$ (universal generalisation of y, from 13)
15. $\forall x\, \forall y\, \neg(x < y \wedge y < x)$ (universal generalisation of x, from 14)

7 Entailment, recursive languages, and recursively enumerable languages

7.1 The deduction theorem in more detail

Theorem (Deduction theorem)**.** *Let A and B be sentences. For any (possibly infinite) set of assumptions Σ, a formula B is deducible from the assumptions $\Sigma \cup \{A\}$ if and only if the implication $A \to B$ is deducible from the assumptions Σ. In symbols, we have*

$$\Sigma \cup \{A\} \vdash B \iff \Sigma \vdash (A \to B).$$

Proof. ($\Leftarrow$): Assuming that $\Sigma \vdash (A \to B)$, there must exist a proof

$$\phi_0, \phi_1, \phi_2, \cdots, \phi_n$$

where $\phi_n = A \to B$, and each ϕ_i is an axiom, an element of Σ, or derived from two previous formulas via modus ponens.

We extend this proof as follows.

$$\phi_0, \phi_1, \phi_2, \cdots, \phi_n, A, B$$

Note that this is an acceptable proof of B under the assumptions $\Sigma \cup \{A\}$.

- Each of $\phi_0, \phi_1, \phi_2, \cdots, \phi_n$ is an axiom, an element of Σ, or derived from modus ponens.
- A is an assumption from $\Sigma \cup \{A\}$.
- B is derived from ϕ_n (i.e. $A \to B$) and A via modus ponens.

Hence we have $\Sigma \vdash (A \to B) \implies \Sigma \cup \{A\} \vdash B$.

($\Rightarrow$): Consider the following axiom schematas.

I. $p \to (q \to p)$

II. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$

Assuming that $\Sigma \cup \{A\} \vdash B$, there must exist a proof

$$\phi_0, \phi_1, \phi_2, \cdots, \phi_n$$

where $\phi_n = B$, and each ϕ_i is an axiom, an element of Σ, the additional assumption A, or derived from two previous formulas via modus ponens.

We will transform this sequence into a proof from Σ of $A \to B$. To do this, we will show by strong induction[12] that for each index i ($0 \leq i \leq n$), we can construct a proof of the implication $A \to \phi_i$ under the assumptions in Σ.

Induction hypothesis. For all natural numbers $k < i$, the formula $A \to \phi_k$ is a theorem under the assumptions Σ.

Step case. We want to construct a proof of $A \to \phi_i$ under the assumptions Σ.

- If ϕ_i is an axiom or an element of Σ, then we have the following proof.
  1. $\phi_i \to (A \to \phi_i)$ (Axiom I)
  2. ϕ_i (Axiom/Assumption)

[12]Recall that strong (or complete) induction does not require a base case.

3. $A \to \phi_i$ (modus ponens, from 1 and 2)

- If $\phi_i = A$, then we have the following proof.

  1. $A \to ((A \to A) \to A)$ (Axiom I)
  2. $(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$ (Axiom II)
  3. $(A \to (A \to A)) \to (A \to A)$ (modus ponens, from 1 and 2)
  4. $(A \to (A \to A))$ (Axiom I)
  5. $A \to A$ (modus ponens, from 3 and 4)

- Suppose ϕ_i is derived via modus ponens by two previous formulas ϕ_j and $\phi_j \to \phi_i$. By the induction hypothesis, we have $\Sigma \vdash A \to \phi_j$ and $\Sigma \vdash A \to (\phi_j \to \phi_i)$. Now consider the axiom

$$(A \to (\phi_j \to \phi_i)) \to ((A \to \phi_j) \to (A \to \phi_i)). \quad \text{(Axiom II)}$$

  Since both $\Sigma \vdash A \to \phi_j$ and $\Sigma \vdash A \to (\phi_j \to \phi_i)$ are theorems, we may obtain the theorem $(A \to \phi_i)$.

This completes the induction proof, giving us $\Sigma \vdash A \to \phi_n$, which can be rewritten as $\Sigma \vdash A \to B$. Hence we have $\Sigma \cup \{A\} \vdash B \implies \Sigma \vdash (A \to B)$. □

7.2 What is entailment?

Let Γ be a set of sentences and let S be a first-order structure (L-structure). We say that S is a *model* of Γ if for each sentence $\phi \in \Gamma$ we have $S \models \phi$. This is denoted as $S \models \Gamma$.

Furthermore, we say that $\Gamma \models \psi$ for some sentence ψ if every model of Γ is also a model of ψ, i.e. $S \models \Gamma \implies S \models \psi$. This is written as $\Gamma \models \psi$.

It's worth taking a moment to review the many roles that the symbol "$\models$" takes on in first-order logic. This is summarised in table 6.

Expression	Meaning
$(D, I) \models_A \phi$	ϕ is true in the structure (D, I) under the variable assignment A.
$(D, I) \models \phi$	ϕ is valid in the structure (D, I).
$\models \phi$	ϕ is valid.
$\mathcal{K} \models \phi$	ϕ is valid in all structures $(D, I) \in \mathcal{K}$.
$(D, I) \models \Sigma$	For all $\phi \in \Sigma$ we have $(D, I) \models \phi$.
$\Sigma \models \phi$	Σ entails ϕ. (Every model of Σ is a model of ϕ.)

Table 6: The various meanings of the symbol $\models$ in first-order logic. Here, (D, I) is a first-order structure with domain D and interpretation I; A is a variable assignment; ϕ is a formula; $\mathcal{K}$ is a set of structures; and Σ is a set of sentences.

From this we can define the following properties.

$$\Gamma \vdash \phi \implies \Gamma \models \phi \quad \text{(Soundness)}$$
$$\Gamma \models \phi \implies \Gamma \vdash \phi \quad \text{(Strong completeness)}$$

7.3 Recursive and recursively enumerable languages

A *language* refers to a set of strings over a finite alphabet Σ.

A language L is said to be *recursive* (also called *decidable* or *computable*) if there exists a computer program that

- takes an arbitrary string $s \in \Sigma^*$ as input;
- correctly outputs `yes` if $s \in L$ and `no` otherwise; and
- always terminates for any input.

For example, the set of all formulas of first-order logic is recursive, since it is possible to write a parser that decides whether a string is a well-formed formula. Meanwhile, the set of valid first-order statements form a language, but it is not recursive.

Moreover, a language L is *recursively enumerable* (abbreviated *r.e.*) if there exists a computer program that outputs strings from L, only strings from L, and will eventually output any given string from L.

Theorem. *The set of valid statements in first-order logic is recursively enumerable.*

Proof 1 — using Hilbert-style axiomatic proof systems. Let c be an injective function that encodes any Hilbert-style first-order logic proof

$$\overline{\phi} = (\phi_0, \phi_1, \phi_2, \cdots, \phi_{k-1})$$

as a number $c(\overline{\phi})$. Then, the following program will eventually output any first-order theorem in finite time.

```
for (i = 0, i++, forever):
    if i is the code of a proof: # i = c(φ̄)
        output the proved formula
```

Hence proved. □

Proof 2 — using predicate tableaus. Let $\phi_0, \phi_1, \cdots$ be an enumeration of all formulas. Consider the following program.

```
tableaus = []
for (i = 0, i++, forever):
    create a new tableau T_i with ¬φ_i at root
    tableaus.append(T_i)

    for each j ≤ i:
        if the j-th tableau T_j can be expanded:
            expand T_j once with fair schedule
            if T_j becomes closed:
                output φ_j
```

For any formula ϕ_k,

- if ϕ_k is not valid, then T_k never closes (by soundness); and
- if ϕ_k is valid, then T_k will eventually close, resulting in ϕ_k being outputted in finite time.

Therefore, this program only outputs valid formulas, and any valid formula will eventually be outputted. □

Theorem. *All recursive sets are recursively enumerable.*

Proof. Let L be a recursive language. Let A be a terminating algorithm such that

$$A(s) = \begin{cases} 1 & \text{if } s \in L \\ 0 & \text{otherwise} \end{cases}$$

for any input string s. We then construct the following program.

```
for each string s (sorted first by increasing length, then alphabetically):
    if A(s) = 1:
        output s
```

This program only outputs strings from L, and each $s \in L$ is eventually output. Therefore, L is recursively enumerable. $\square$

Note that the converse of the above theorem does not hold. While all recursive sets are recursively enumerable, not all recursively enumerable sets are recursive.