VNU-HUS MAT1206E/3508: Introduction to AI

Propositional Logic

Hoàng Anh Đức

Bộ môn Tin học, Khoa Toán-Cơ-Tin học
Đại học KHTN, ĐHQG Hà Nội
hoanganhduc@hus.edu.vn

Contents

Logic

In short, a *logic* can be viewed as a triple $(\mathcal{L}, \mathcal{S}, \mathcal{R})$ where

- $\mathcal{L}$, the logic's *language*, is a class of **well-formed (syntactically correct) sentences (formulas)**
- $\mathcal{S}$, the logic's *semantics*, is a formal specification of how to assign **meaning** in the "real world" to the elements of $\mathcal{L}$
- $\mathcal{R}$, the logic's *inference system*, is a set of formal **derivation rules** (or **inference rules**) over $\mathcal{L}$

Note

- We focus on *propositional logic* and *first-order logic*
- We will explain these concepts more precisely in the next slides

Logic

We mention some *fundamental concepts* of logical representation and reasoning without involving the technical details (See [Russell and Norvig 2010], Section 7.3). To illustrate these concepts, we use the ordinary arithmetic

Logic

We mention some *fundamental concepts* of logical representation and reasoning without involving the technical details (See [Russell and Norvig 2010], Section 7.3). To illustrate these concepts, we use the ordinary arithmetic

- A *sentence (formula)*[1] is expressed according to the *syntax* of the so-called *knowledge representation language*, which specifies all the sentences that are *well-formed (syntactically correct)*
  - "$x + y = 4$" is a well-formed sentence, while "$xy4+ =$" is not

[1] Here "sentence" is used as a technical term. It is related but not identical to the sentences of English and other natural languages

Logic

We mention some *fundamental concepts* of logical representation and reasoning without involving the technical details (See [Russell and Norvig 2010], Section 7.3). To illustrate these concepts, we use the ordinary arithmetic

- A *sentence (formula)*[1] is expressed according to the *syntax* of the so-called *knowledge representation language*, which specifies all the sentences that are *well-formed (syntactically correct)*
  - "$x + y = 4$" is a well-formed sentence, while "$xy4+ =$" is not
- The *semantics (meaning)* defines the *truth (true/false)* of each sentence with respect to each *possible world (assignment, interpretation)*
  - "$x + y = 4$" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1
  - In standard logics, *every sentence must be either true or false in each possible world*—there is no "in between."

[1] Here "sentence" is used as a technical term. It is related but not identical to the sentences of English and other natural languages

Logic

- If a sentence φ is true in a possible world (assignment, interpretation) m, we say that m *satisfies* φ or m is a *model* of φ. The notation $Mod(\varphi)$ is sometimes used to indicate the set of all models of φ
  - Any assignment of real numbers to the variables x and y such that $x + y = 4$ is a model of the sentence "$x + y = 4$"

Logic

- If a sentence φ is true in a possible world (assignment, interpretation) m, we say that m *satisfies* φ or m is a *model* of φ. The notation $Mod(\varphi)$ is sometimes used to indicate the set of all models of φ
  - Any assignment of real numbers to the variables x and y such that $x + y = 4$ is a model of the sentence "$x + y = 4$"
- *Logical reasoning* involves the relation of logical *entailment* between sentences (formulas)—the idea that *a sentence (formula) β follows logically from another sentence (formula) α*
  - In mathematical notion, we write $\alpha \models \beta$ to indicate that α *entails* β or β *logically follows from* α. That is, $\alpha \models \beta$ if and only if, *in every model in which α is true, β is also true*

$$\alpha \models \beta \text{ if and only if } Mod(\alpha) \subseteq Mod(\beta)$$

  - If $\alpha \models \beta$, then α is a *stronger* assertion than β
  - The sentence "$x = 0$" entails the sentence "$xy = 0$"

Logic

- An *inference algorithm* is one that carries out *logical inference*—the procedure of *deriving valid conclusions from a set KB of existing logical sentences* (e.g., a knowledge base)[2]
  - From the sentences "$x = 2$" and "$y = 2$", one can derive the conclusion "$x + y = 4$"

[2] One can think of KB as either a set of sentences or a single sentence that asserts all the individual sentences

Logic

- An *inference algorithm* is one that carries out *logical inference*—the procedure of *deriving valid conclusions from a set KB of existing logical sentences* (e.g., a knowledge base)[2]
  - From the sentences "$x = 2$" and "$y = 2$", one can derive the conclusion "$x + y = 4$"
- An inference algorithm to decide if $KB \models \alpha$ is the so-called *model checking*: *Enumerate all possible models* to check that α is true in all models in which KB is true. If so, return "yes". Otherwise, return "no". (Model checking works if the space of models is finite)

[2] One can think of KB as either a set of sentences or a single sentence that asserts all the individual sentences

Logic

- An *inference algorithm* is one that carries out *logical inference*—the procedure of *deriving valid conclusions from a set KB of existing logical sentences* (e.g., a knowledge base)[2]
  - From the sentences "$x = 2$" and "$y = 2$", one can derive the conclusion "$x + y = 4$"
- An inference algorithm to decide if $KB \models \alpha$ is the so-called *model checking*: *Enumerate all possible models* to check that α is true in all models in which KB is true. If so, return "yes". Otherwise, return "no". (Model checking works if the space of models is finite)
- A *sound (truth preserving)* inference algorithm *derives only entailed sentences*: if $KB \not\models \alpha$ then $KB \nvdash \alpha$

[2]One can think of KB as either a set of sentences or a single sentence that asserts all the individual sentences

Logic

- An *inference algorithm* is one that carries out *logical inference*—the procedure of *deriving valid conclusions from a set KB of existing logical sentences* (e.g., a knowledge base)[2]
  - From the sentences "$x = 2$" and "$y = 2$", one can derive the conclusion "$x + y = 4$"
- An inference algorithm to decide if $KB \models \alpha$ is the so-called *model checking*: *Enumerate all possible models* to check that α is true in all models in which KB is true. If so, return "yes". Otherwise, return "no". (Model checking works if the space of models is finite)
- A *sound (truth preserving)* inference algorithm *derives only entailed sentences*: if $KB \not\models \alpha$ then $KB \nvdash \alpha$
- A *complete* inference algorithm *derives all entailed sentences*: if $KB \models \alpha$ then $KB \vdash \alpha$

[2] One can think of KB as either a set of sentences or a single sentence that asserts all the individual sentences

Syntax

- Set of *logical operators* $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$
- Set of *symbols* Σ (called the *signature*) whose elements are *propositional variables*
- Set of *logical constants* $\{\mathbf{t}, \mathbf{f}\}$
- All the above sets are *pairwise disjoint*

Syntax

- Set of *logical operators* $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$
- Set of *symbols* Σ (called the *signature*) whose elements are *propositional variables*
- Set of *logical constants* $\{\mathbf{t}, \mathbf{f}\}$
- All the above sets are *pairwise disjoint*

Propositional Logic Formulas

- $\mathbf{t}$ and $\mathbf{f}$ are (atomic) formulas
- All propositional variables (e.g., members of Σ) are (atomic) formulas
- If A and B are formulas, then $\neg A$, (A), $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ are also formulas

Syntax

$\mathbf{t}$	"true"
$\mathbf{f}$	"false"
$\neg A$	"not A"
$A \wedge B$	"A and B"
$A \vee B$	"A or B"
$A \Rightarrow B$	"if A then B" (implication)
$A \Leftrightarrow B$	"A if and only if B" (equivalence)

Syntax

$\mathbf{t}$	"true"
$\mathbf{f}$	"false"
$\neg A$	"not A"
$A \wedge B$	"A and B"
$A \vee B$	"A or B"
$A \Rightarrow B$	"if A then B" (implication)
$A \Leftrightarrow B$	"A if and only if B" (equivalence)

Example 1

Given $\Sigma = \{A, B, C\}$. Then

- $A \wedge B$
- $A \wedge B \wedge C$
- $A \wedge A \wedge A$
- $C \wedge B \vee A$
- $(\neg A \wedge B) \Rightarrow (\neg C \vee A)$
- $(((A)) \vee B)$

are (syntactically correct) formulas

Syntax

Backus-Naur Form (BNF)

- Developed by John Backus and Peter Naur (1960).
- A formal notation to describe the syntax of a given language.
- Meta-symbols of BNF
  - `::=` meaning "is defined as"
  - `|` meaning "or"
  - `<>` angle brackets used to surround category names
- Many authors have introduced some slight extensions of BNF for the ease of use.

Example 2 (BNF for numbers)

```
<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

which can be read as: "a number is a digit, or any number followed by an extra digit"

Example 3 (BNF for sentences)

```
<sentence> ::= <noun_phrase> <verb>
<noun_phrase> ::= <article> <noun>
<noun> ::= "horse" | "dog" | "hamster"
<article> ::= "" | "a" | "the"
<verb> ::= "stands" | "walks" | "jumps"
```

Some strings defined from this grammar:

- the horse jumps
- a dog walks
- hamster jumps

Exercise 1 ([Ertel 2025], Exercise 2.1, p. 38)

Give a Backus-Naur form grammar for the syntax of propositional logic.

Semantics

In propositional logic, there are two *truth values*: t for “true” and f for “false”

- The logical constants $\mathbf{t}$ and $\mathbf{f}$ are always assigned the truth values t and f, respectively

Semantics

In propositional logic, there are two *truth values*: t for "true" and f for "false"

- The logical constants $\mathbf{t}$ and $\mathbf{f}$ are always assigned the truth values t and f, respectively

Truth assignment

A mapping $I : \Sigma \to \{t, f\}$, which assigns a truth value to every propositional variable, is called an *assignment* or *interpretation*

Exercise 2

How many assignments a propositional logic formula with n different variables can have?

Note

In the textbook [Ertel 2025], the author *uses t and f for both logical constants and truth values*

Semantics

- A *truth table* for a formula φ is a table describing *all possible assignments* of φ
- Definition of the logical operators by *truth table*

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

- The *empty formula* is true for all assignments
- **Operator priorities:** $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Semantics

Semantically Equivalent Formulas

Two formulas F and G are called *semantically equivalent* if they take on the same truth value for all assignments. We write $F \equiv G$

- $F \equiv G$ if and only if $F \Leftrightarrow G$ is true for all assignments

Semantics

Classification of Formulas

A formula is called

- *Satisfiable* if it is *true* for *at least one assignment*
- *Logically valid* or simply *valid* if it is *true* for *all assignment*. True formulas are also called *tautologies*
- *Unsatisfiable* if it is *not true* for *any assignment*

Every assignment that satisfies a formula is called a *model* of the formula

Semantics

Classification of Formulas

A formula is called

- *Satisfiable* if it is *true* for *at least one assignment*
- *Logically valid* or simply *valid* if it is *true* for *all assignment*. True formulas are also called *tautologies*
- *Unsatisfiable* if it is *not true* for *any assignment*

Every assignment that satisfies a formula is called a *model* of the formula

- The *negation* of any *generally valid* formula is *unsatisfiable*
- The *negation* of a *satisfiable*, but *not generally valid* formula is *satisfiable*

Semantics

Classification of Formulas

A formula is called

- *Satisfiable* if it is *true* for *at least one assignment*
- *Logically valid* or simply *valid* if it is *true* for *all assignment*. True formulas are also called *tautologies*
- *Unsatisfiable* if it is *not true* for *any assignment*

Every assignment that satisfies a formula is called a *model* of the formula

- The *negation* of any *generally valid* formula is *unsatisfiable*
- The *negation* of a *satisfiable*, but *not generally valid* formula is *satisfiable*

Exercise 3 ([Ertel 2025], Exercise 2.4, p. 37)

Check the following statements for satisfiability or validity

(a) $(\text{play\_lottery} \wedge \text{six\_right}) \Rightarrow \text{winner}$
(b) $(\text{play\_lottery} \wedge \text{six\_right} \wedge (\text{six\_right} \Rightarrow \text{win})) \Rightarrow \text{win}$
(c) $\neg(\neg\text{gas\_in\_tank} \wedge (\text{gas\_in\_tank} \vee \neg\text{car\_starts}) \Rightarrow \neg\text{car\_starts})$

Semantics

Theorem 1

The operations $\wedge, \vee$ are commutative and associative, and the following equivalences are generally valid:

$$\neg A \vee B \Leftrightarrow A \Rightarrow B \qquad \textit{(implication)}$$

$$A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A \qquad \textit{(contraposition)}$$

$$(A \Rightarrow B) \wedge (B \Rightarrow A) \Leftrightarrow (A \Leftrightarrow B) \qquad \textit{(equivalence)}$$

$$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B \qquad \textit{(De Morgan's law)}$$

$$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$$

$$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C) \qquad \textit{(distributive law)}$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$$

$$A \vee \neg A \Leftrightarrow \mathbf{t} \qquad \textit{(tautology)}$$

$$A \wedge \neg A \Leftrightarrow \mathbf{f} \qquad \textit{(contradiction)}$$

$$A \vee \mathbf{f} \Leftrightarrow A$$

$$A \vee \mathbf{t} \Leftrightarrow \mathbf{t}$$

$$A \wedge \mathbf{f} \Leftrightarrow \mathbf{f}$$

$$A \wedge \mathbf{t} \Leftrightarrow A$$

Proof of Theorem 1.

We prove only the first formula

A	B	$\neg A$	$\neg A \vee B$	$A \Rightarrow B$	$(\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$
t	t	f	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
f	f	t	t	t	t

□

Proof of Theorem 1.

We prove only the first formula

A	B	$\neg A$	$\neg A \lor B$	$A \Rightarrow B$	$(\neg A \lor B) \Leftrightarrow (A \Rightarrow B)$
t	t	f	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
f	f	t	t	t	t

□

Exercise 4

Prove all remaining formulas in Theorem 1 using truth table

Exercise 5 ([Ertel 2025], Exercise 2.2, p. 38)

Show that the following formulas are tautologies

(a) $\neg(A \land B) \Leftrightarrow \neg A \lor \neg B$

(b) $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$

(c) $((A \Rightarrow B) \land (B \Rightarrow A)) \Leftrightarrow (A \Leftrightarrow B)$

(d) $(A \lor B) \land (\neg B \lor C) \Rightarrow A \lor C$

Semantics

Exercise 6 ([Ertel 2025], Exercise 2.5, p. 39)

Using the programming language of your choice, program a theorem prover for propositional logic using the truth table method for formulas in conjunctive normal form. To avoid a costly syntax check of the formulas, you may represent clauses as lists or sets of literals, and the formulas as lists or sets of clauses. The program should indicate whether the formula is unsatisfiable, satisfiable, or true, and output the number of different interpretations and models.

Proof Systems

In AI, we are interested in deriving new knowledge or answering questions using an existing *knowledge base* – KB.

Question

Does a formula (query) Q follow from the knowledge base KB?

Proof Systems

In AI, we are interested in deriving new knowledge or answering questions using an existing *knowledge base* – KB.

Question

Does a formula (query) Q follow from the knowledge base KB?

Entailment

A formula KB *entails* a formula Q (or Q *follows* from KB) if every model of KB is also a model of Q. We write $KB \models Q$

- **Recall:** Any assignment satisfying a formula F is a *model* of F
- Thus, $KB \models Q$ if in any assignment in which KB is true, Q is also true

Proof Systems

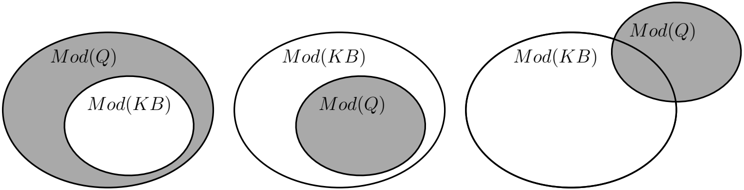
In AI, we are interested in deriving new knowledge or answering questions using an existing *knowledge base* – KB.

Question

Does a formula (query) Q follow from the knowledge base KB?

Entailment

A formula KB *entails* a formula Q (or Q *follows* from KB) if every model of KB is also a model of Q. We write $KB \models Q$



$KB \models Q$ $\qquad$ $KB \not\models Q$ $\qquad$ $KB \not\models Q$

Proof Systems

- Every formula chooses a subset of the set of all assignments as its model
  - Tautologies, such as $A \vee \neg A$, do not restrict the number of satisfying assignments
- Recall that the empty formula is true for all assignments. Then, $\emptyset \models T$ *for any tautology* T. For short, we write $\models T$

Proof Systems

- Every formula chooses a subset of the set of all assignments as its model
  - Tautologies, such as $A \vee \neg A$, do not restrict the number of satisfying assignments
- Recall that the empty formula is true for all assignments. Then, $\emptyset \models T$ *for any tautology* T. For short, we write $\models T$

Theorem 2 (Deduction theorem)

$A \models B$ *if and only if* $\models A \Rightarrow B$

Proof.

- **If $A \models B$ then $A \Rightarrow B$ is a tautology.**

- **If $A \Rightarrow B$ holds then $A \models B$.**

Proof Systems

- Every formula chooses a subset of the set of all assignments as its model
  - Tautologies, such as $A \vee \neg A$, do not restrict the number of satisfying assignments
- Recall that the empty formula is true for all assignments. Then, $\emptyset \models T$ *for any tautology* T. For short, we write $\models T$

Theorem 2 (Deduction theorem)

$A \models B$ *if and only if* $\models A \Rightarrow B$

Proof.

- **If** $A \models B$ **then** $A \Rightarrow B$ **is a tautology.** Since $A \models B$, in any assignment that makes A true, B is also true. In other words, it cannot happen that $A \mapsto t$ and $B \mapsto f$. Therefore, $A \Rightarrow B$ is always true
- **If** $A \Rightarrow B$ **holds then** $A \models B$**.** Since $A \Rightarrow B$ holds, it never happens that $A \mapsto t$ and $B \mapsto f$. Therefore, any assignment that makes A true also makes B true, i.e., $A \models B$

Proof Systems

- To *show* $KB \models Q$, we can construct a *truth table* to *prove* $KB \Rightarrow Q$ *is a tautology*. This is the first *proof system* for propositional logic
  - **Remind:** This is indeed proof by *model checking*
- **Disadvantage:** Very large computing in the worst case (2^n assignments for n propositional variables)

Proof Systems

A simple, but important consequence of Theorem 2 (Deduction theorem):

Theorem 3 (Proof by contradiction)

$KB \models Q$ *if and only if* $KB \land \neg Q$ *is unsatisfiable*

Proof Systems

A simple, but important consequence of Theorem 2 (Deduction theorem):

Theorem 3 (Proof by contradiction)

$KB \models Q$ *if and only if* $KB \wedge \neg Q$ *is unsatisfiable*

Proof.

- **If** $KB \models Q$ **then** $KB \wedge \neg Q$ **is unsatisfiable.**

- **If** $KB \wedge \neg Q$ **is unsatisfiable, then** $KB \models Q$**.**

Proof Systems

A simple, but important consequence of Theorem 2 (Deduction theorem):

Theorem 3 (Proof by contradiction)

$KB \models Q$ if and only if $KB \land \neg Q$ is unsatisfiable

Proof.

- **If $KB \models Q$ then $KB \land \neg Q$ is unsatisfiable.** By Theorem 2, $KB \Rightarrow Q$ is a tautology. Thus,

$$\neg(KB \Rightarrow Q) \equiv \neg(\neg KB \lor Q) \equiv KB \land \neg Q$$

  is unsatisfiable

- **If $KB \land \neg Q$ is unsatisfiable, then $KB \models Q$.** Since $KB \land \neg Q$ is unsatisfiable, we have

$$\neg(KB \land \neg Q) \equiv \neg KB \lor Q \equiv KB \Rightarrow Q$$

  is a tautology. Thus $KB \models Q$

Proof Systems

Using Theorem 3, we can also prove $KB \models Q$ as follows:

- To *show* $KB \models Q$, we can *add the negated query* $\neg Q$ *to the knowledge base* KB and *derive a contradiction*
- By Theorem 1, *a contradiction is unsatisfiable* $A \wedge \neg A \Leftrightarrow \mathbf{f}$

Proof Systems

Using Theorem 3, we can also prove $KB \models Q$ as follows:

- To *show $KB \models Q$*, we can *add the negated query $\neg Q$ to the knowledge base KB* and *derive a contradiction*
- By Theorem 1, *a contradiction is unsatisfiable* $A \wedge \neg A \Leftrightarrow \mathbf{f}$

Question

But what does "derive a contradiction" mean?

Proof Systems

Using Theorem 3, we can also prove $KB \models Q$ as follows:

- To *show* $KB \models Q$, we can *add the negated query* $\neg Q$ *to the knowledge base* KB and *derive a contradiction*
- By Theorem 1, *a contradiction is unsatisfiable* $A \wedge \neg A \Leftrightarrow \mathbf{f}$

Question

But what does "derive a contradiction" mean?

Derivation $KB \vdash Q$

is the *syntactic manipulation* of *the formulas* KB *and* Q by *applying the inference rules* in order to greatly simplifying them, such that *in the end we can instantly see that* $KB \models Q$

- We will mention the "inference rules" later

Proof Systems

Using Theorem 3, we can also prove $KB \models Q$ as follows:

- To *show* $KB \models Q$, we can *add the negated query* $\neg Q$ *to the knowledge base* KB and *derive a contradiction*
- By Theorem 1, *a contradiction is unsatisfiable* $A \wedge \neg A \Leftrightarrow \mathbf{f}$

Question

But what does "derive a contradiction" mean?

Derivation $KB \vdash Q$

is the *syntactic manipulation* of *the formulas KB and Q* by *applying the inference rules* in order to greatly simplifying them, such that *in the end we can instantly see that $KB \models Q$*

- We will mention the "inference rules" later
- $KB \vdash Q$ means "Q follows from KB *syntactically*" and $KB \models Q$ means "Q follows from KB *semantically*"

Proof Systems

Using Theorem 3, we can also prove $KB \models Q$ as follows:

- To *show* $KB \models Q$, we can *add the negated query* $\neg Q$ *to the knowledge base* KB and *derive a contradiction*
- By Theorem 1, *a contradiction is unsatisfiable* $A \land \neg A \Leftrightarrow \mathbf{f}$

Question

But what does "derive a contradiction" mean?

Derivation $KB \vdash Q$

is the *syntactic manipulation* of *the formulas KB and Q* by *applying the inference rules* in order to greatly simplifying them, such that *in the end we can instantly see that $KB \models Q$*

- We will mention the "inference rules" later
- $KB \vdash Q$ means "Q follows from KB *syntactically*" and $KB \models Q$ means "Q follows from KB *semantically*"
- This is another *proof system* for propositional logic, which we call a *calculus*

Proof Systems

In general, to ensure that a calculus does not generate errors, we define its two fundamental properties:

Properties of a calculus

- A calculus is called *sound* if *every derived proposition follows semantically*. That is, for two formulas KB and Q: *if* $KB \vdash Q$ *then* $KB \models Q$
- A calculus is called *complete* if *all semantic consequences can be derived*. That is, for two formulas KB and Q: *if* $KB \models Q$ *then* $KB \vdash Q$

Proof Systems

If a calculus is *both sound and complete*, then syntactic derivation and semantic entailment are two *equivalent relations*

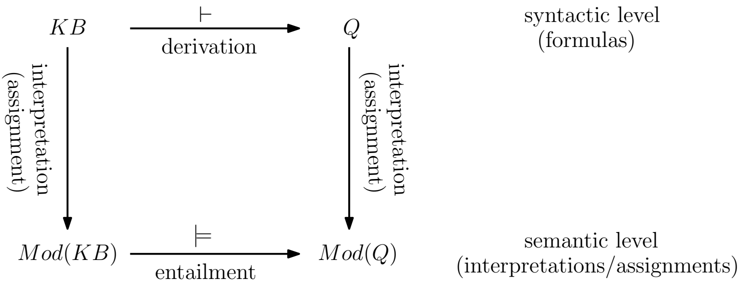


Figure: Syntactic derivation and semantic entailment. $Mod(X)$ denotes the set of models of a formula X

Proof Systems

Proof systems are usually made to *operate on formulas in conjunctive normal form*

Conjunctive Normal Form (CNF)

A formula is in *conjunctive normal form (CNF)* if and only if it consists of a *conjunction*

$$K_1 \wedge K_2 \wedge \cdots \wedge K_m$$

of clauses. Each clause K_i $(1 \leq i \leq m)$ consists of a *disjunction*

$$L_{i1} \vee L_{i2} \vee \ldots L_{in_i}$$

of literals. Finally, a *literal* is either a variable (positive literal) or a negated variable (negative literal)

Example 4 (Conjunctive Normal Form)

- The formula $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ is in conjunctive normal form
  - **Variables:** A, B, C
  - **Literals:** A, $\neg A$, B, $\neg B$, C, $\neg C$
  - **Clauses:** $A \vee B \vee \neg C$, $A \vee B$, and $\neg B \vee \neg C$

Proof Systems

Example 4 (Conjunctive Normal Form)

- The formula $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ is in conjunctive normal form
  - **Variables:** A, B, C
  - **Literals:** A, $\neg A$, B, $\neg B$, C, $\neg C$
  - **Clauses:** $A \vee B \vee \neg C$, $A \vee B$, and $\neg B \vee \neg C$

Theorem 4

Every propositional logic formula can be transformed into an equivalent conjunctive normal form

Transforming a formula to CNF

- Eliminate $\Rightarrow$, $\Leftrightarrow$ using known logic equivalences
- Reduce the scope of signs through De Morgan's laws and the double negation
- Convert to CNF using the associative and distributive laws

Example 5 (Transforming a formula to CNF)

$A \vee B \Rightarrow C \wedge D$	
$\equiv \neg(A \vee B) \vee (C \wedge D)$	implication
$\equiv (\neg A \wedge \neg B) \vee (C \wedge D)$	De Morgan
$\equiv (\neg A \vee (C \wedge D)) \wedge (\neg B \vee (C \wedge D))$	distributive law
$\equiv ((\neg A \vee C) \wedge (\neg A \vee D)) \wedge ((\neg B \vee C) \wedge (\neg B \vee D))$	distributive law
$\equiv (\neg A \vee C) \wedge (\neg A \vee D) \wedge (\neg B \vee C) \wedge (\neg B \vee D)$	associative law

Exercise 7 ([Ertel 2025], Exercise 2.3, p. 39)

Transform the following formulas into conjunctive normal form:

(a) $A \Leftrightarrow B$
(b) $A \wedge B \Leftrightarrow A \vee B$
(c) $A \wedge (A \Rightarrow B) \Rightarrow B$

Proof Systems

A Quick Recap

- **Goal:** showing $KB \models Q$ by *adding the negated query $\neg Q$ to the knowledge base KB* and *derive a contradiction*
  - It suffices to work only with *conjunctive normal forms*
  - To carry out the (syntactic) derivation, **we need a calculus** that is *both sound and complete for the proof of unsatisfiability of formulas*. Which *inference rules* should we use in such a calculus?

Proof Systems

We now focus on the *inference rules*—*the rules applied to a set of formulas to finally derive some conclusions*

- The *Modus Ponens*[3] (or *Implication-Elimination*) *rule*

$$\frac{A, A \Rightarrow B}{B} \quad \text{or} \quad A \wedge (A \Rightarrow B) \vdash B \quad \text{or} \quad \{A, A \Rightarrow B\} \vdash B$$

  - The set of formulas above the line is the *premise*
  - The formula below the line is the *conclusion*
  - One can derive the conclusion from the premise

[3] Latin for "mode that affirms"

Proof Systems

We now focus on the *inference rules*—*the rules applied to a set of formulas to finally derive some conclusions*

- The *Modus Ponens*[3] (or *Implication-Elimination*) *rule*

$$\frac{A, A \Rightarrow B}{B} \quad \text{or} \quad A \wedge (A \Rightarrow B) \vdash B \quad \text{or} \quad \{A, A \Rightarrow B\} \vdash B$$

  - The set of formulas above the line is the *premise*
  - The formula below the line is the *conclusion*
  - One can derive the conclusion from the premise
- Modus Ponens is *sound*
  - If both two formulas in the premise are true, the conclusion is also true [Why? Verify this by a truth table]

[3] Latin for "mode that affirms"

Proof Systems

We now focus on the *inference rules*—*the rules applied to a set of formulas to finally derive some conclusions*

- The *Modus Ponens*[3] (or *Implication-Elimination*) *rule*

$$\frac{A, A \Rightarrow B}{B} \quad \text{or} \quad A \wedge (A \Rightarrow B) \vdash B \quad \text{or} \quad \{A, A \Rightarrow B\} \vdash B$$

  - The set of formulas above the line is the *premise*
  - The formula below the line is the *conclusion*
  - One can derive the conclusion from the premise
- Modus Ponens is *sound*
  - If both two formulas in the premise are true, the conclusion is also true [Why? Verify this by a truth table]
- Modus Ponens is *not complete*
  - $\{A \Rightarrow B, \neg B\} \models \neg A$ [Why? Verify this by a truth table] but $\{A \Rightarrow B, \neg B\} \nvdash \neg A$ *using just Modus Ponens repeatedly*
  - To create a complete calculus as required, we can add more rules. But, we will consider another inference rule instead

[3] Latin for "mode that affirms"

Proof Systems

- The *resolution rule*

$$\frac{A \vee B, \neg B \vee C}{A \vee C} \quad \text{or} \quad \frac{A \vee B, B \Rightarrow C}{A \vee C}$$

  - The derived formula is also called a *resolvent*
  - The resolution rule *delete a pair of complement literals* (B and $\neg B$) from the two clauses and *combines the rest* of the literals into a new clause
  - Two clauses may have more than one resolvent
    - Take C to be $\neg A$
    - In this case, these resolvents are tautologies [Why?]

Proof Systems

- The *resolution rule*

$$\frac{A \lor B, \neg B \lor C}{A \lor C} \quad \text{or} \quad \frac{A \lor B, B \Rightarrow C}{A \lor C}$$

  - The derived formula is also called a *resolvent*
  - The resolution rule *delete a pair of complement literals* (B and $\neg B$) from the two clauses and *combines the rest* of the literals into a new clause
  - Two clauses may have more than one resolvent
    - Take C to be $\neg A$
    - In this case, these resolvents are tautologies [Why?]
- The resolution rule is a *generalization of the Modus Ponens*
  - Set A to be $\mathbf{f}$

Proof Systems

- The *resolution rule*

$$\frac{A \vee B, \neg B \vee C}{A \vee C} \quad \text{or} \quad \frac{A \vee B, B \Rightarrow C}{A \vee C}$$

  - The derived formula is also called a *resolvent*
  - The resolution rule *delete a pair of complement literals* (B and $\neg B$) from the two clauses and *combines the rest* of the literals into a new clause
  - Two clauses may have more than one resolvent
    - Take C to be $\neg A$
    - In this case, these resolvents are tautologies [Why?]
- The resolution rule is a *generalization of the Modus Ponens*
  - Set A to be $\mathbf{f}$
- The resolution rule is equally usable if C is missing or if A and C are missing

Proof Systems

Exercise 8 ([Ertel 2025], Exercise 2.6, p. 39)

(a) Show that Modus Ponens is a valid inference rule by showing that $A \wedge (A \Rightarrow B) \models B$

(b) Show that the resolution rule is a valid inference rule

Exercise 9 ([Ertel 2025], Exercise 2.7, p. 39)

Show by application of the resolution rule that, in conjunctive normal form, the empty clause is equivalent to the false statement (**Hint:** Using the resolution rule, how to derive the empty clause $()$? What about the false statement $\mathbf{f}$?)

Exercise 10

(a) Does $(A \vee A) \wedge (\neg A \vee \neg A) \models \mathbf{f}$ hold? Can you derive $\mathbf{f}$ from $(A \vee A) \wedge (\neg A \vee \neg A)$ using only the resolution rule?

(b) From (a), what can you say about the completeness of a calculus containing only the resolution rule?

Resolution

- The *generalized resolution rule*

$$\frac{(A_1 \lor A_2 \lor \cdots \lor A_m \lor B), (\neg B \lor C_1 \lor C_2 \cdots \lor C_n)}{(A_1 \lor A_2 \lor \cdots \lor A_m \lor C_1 \lor C_2 \cdots \lor C_n)}$$

Resolution

- The *generalized resolution rule*

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_m \vee B), (\neg B \vee C_1 \vee C_2 \cdots \vee C_n)}{(A_1 \vee A_2 \vee \cdots \vee A_m \vee C_1 \vee C_2 \cdots \vee C_n)}$$

Exercise 11

(a) Let $\mathcal{C}_1 = A_1 \vee \neg A_2 \vee A_3$ and $\mathcal{C}_2 = A_2 \vee \neg A_3 \vee A_4$ be two clauses. Show that one can resolve $\mathcal{C}_1$ and $\mathcal{C}_2$ in more than one way

(b) Are the resolvents from (a) tautologies?

(c) Prove the generalized statement: if two clauses can be resolved in more than one way then all those resolvents are tautologies

Resolution

- The *generalized resolution rule*

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_m \vee B), (\neg B \vee C_1 \vee C_2 \cdots \vee C_n)}{(A_1 \vee A_2 \vee \cdots \vee A_m \vee C_1 \vee C_2 \cdots \vee C_n)}$$

- With the resolution rule alone, we *cannot* build a complete calculus as desired (e.g., see Exercise 10). With *factorization*, which *allows deletion of copies of literals from clauses*, this problem is eliminated

Resolution

- The *generalized resolution rule*

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_m \vee B), (\neg B \vee C_1 \vee C_2 \cdots \vee C_n)}{(A_1 \vee A_2 \vee \cdots \vee A_m \vee C_1 \vee C_2 \cdots \vee C_n)}$$

- With the resolution rule alone, we *cannot* build a complete calculus as desired (e.g., see Exercise 10). With *factorization*, which *allows deletion of copies of literals from clauses*, this problem is eliminated

Recap: Resolution Calculus

- Transform $KB \wedge \neg Q$ into CNF
- Repeatedly apply the resolution and factorization rules until there is no resolvable pair of clauses
- Every time the resolution rule is applied, add the resolvent to KB if it has not yet been included
- If the empty clause is derived, conclude $KB \models Q$. Otherwise, if there is no more resolvable pair of clauses (and the empty clause is not derived), conclude $KB \not\models Q$

Resolution

- The *generalized resolution rule*

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_m \vee B), (\neg B \vee C_1 \vee C_2 \cdots \vee C_n)}{(A_1 \vee A_2 \vee \cdots \vee A_m \vee C_1 \vee C_2 \cdots \vee C_n)}$$

- With the resolution rule alone, we *cannot* build a complete calculus as desired (e.g., see Exercise 10). With *factorization*, which *allows deletion of copies of literals from clauses*, this problem is eliminated

Theorem 5

The resolution calculus for the proof of unsatisfiability of formulas in conjunctive normal form is sound and complete

Note

- The resolution calculus is *not complete in general*
  - $\{A, B\} \models A \vee B$ but $\{A, B\} \nvdash A \vee B$ *using resolution rule*
- For our purpose, it suffices that *for any unsatisfiable formula F, we have $F \vdash ()$*. (The resolution calculus satisfies this, and thus it is *refutation-complete*)

Resolution

Because it is the job of the resolution calculus to derive a contradiction from $KB \land \neg Q$, it is very important that the knowledge base KB is *consistent*:

Consistent formulas

A formula KB is called *consistent* if it is *impossible to derive from it a contradiction*, that is, a formula of the form $\phi \land \neg\phi$.

- If KB *is not consistent* then *anything can be derived*

Exercise 11 ([Ertel 2025], Exercise 2.8, p. 39)

Show that, with resolution, one can "derive" any arbitrary clause from a knowledge base that contains a contradiction (**Hint:** What can be derived from a contradiction?)

Resolution

We start with a simple logic puzzle to illustrate the *important steps of a resolution proof*

Example 6

Let's solve the following logic puzzle from [Berrondo 1989]

A charming English family

Despite studying English for seven long years with brilliant success, I must admit that when I hear English people speaking English I'm totally perplexed. Recently, moved by noble feelings, I picked up three hitchhikers, a father, mother, and daughter, who I quickly realized were English and only spoke English. At each of the sentences that follow I wavered between two possible interpretations. They told me the following (the second possible meaning is in parentheses): The father: "We are going to Spain (we are from Newcastle)." The mother: "We are not going to Spain and are from Newcastle (we stopped in Paris and are not going to Spain)." The daughter: "We are not from Newcastle (we stopped in Paris)." What about this charming English family?

Resolution

- **Step 1: Formalization (easy to make mistake or forget small details)**

- **Step 2: Transformation into CNF**

- **Step 3: Proof (often very difficult)**

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P)$
- **Step 2: Transformation into CNF**
- **Step 3: Proof**

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \vee N)_1 \wedge (\neg S)_2 \wedge (P \vee N)_3 \wedge (\neg N \vee P)_4$
- **Step 3: Proof**

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \lor N) \land [(\neg S \land N) \lor (P \land \neg S)] \land (\neg N \lor P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \lor N)_1 \land (\neg S)_2 \land (P \lor N)_3 \land (\neg N \lor P)_4$
- **Step 3: Proof**
  - We write *Res*$(m, n) : \langle$*clause*$\rangle_k$ to indicate that $\langle$clause$\rangle$ is obtained by resolution of clause m and clause n and is numbered k

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \lor N) \land [(\neg S \land N) \lor (P \land \neg S)] \land (\neg N \lor P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \lor N)_1 \land (\neg S)_2 \land (P \lor N)_3 \land (\neg N \lor P)_4$
- **Step 3: Proof**
  - We write *Res*$(m, n) : \langle$*clause*$\rangle_k$ to indicate that $\langle$clause$\rangle$ is obtained by resolution of clause m and clause n and is numbered k
  - Res$(1, 2) : (N)_5$, Res$(3, 4) : (P)_6$, and Res$(1, 4) : (S \lor P)_7$

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \vee N)_1 \wedge (\neg S)_2 \wedge (P \vee N)_3 \wedge (\neg N \vee P)_4$
- **Step 3: Proof**
  - We write *Res*$(m, n) : \langle$*clause*$\rangle_k$ to indicate that $\langle$clause$\rangle$ is obtained by resolution of clause m and clause n and is numbered k
  - Res$(1, 2) : (N)_5$, Res$(3, 4) : (P)_6$, and Res$(1, 4) : (S \vee P)_7$
  - P is also derived from Res$(4, 5)$ and Res$(2, 7)$. Every further resolution step would lead to the derivation of clauses that are already available [Verify this claim]

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \vee N)_1 \wedge (\neg S)_2 \wedge (P \vee N)_3 \wedge (\neg N \vee P)_4$
- **Step 3: Proof**
  - We write *Res*(m, n) : $\langle$*clause*$\rangle_k$ to indicate that $\langle$clause$\rangle$ is obtained by resolution of clause m and clause n and is numbered k
  - Res$(1, 2) : (N)_5$, Res$(3, 4) : (P)_6$, and Res$(1, 4) : (S \vee P)_7$
  - P is also derived from Res$(4, 5)$ and Res$(2, 7)$. Every further resolution step would lead to the derivation of clauses that are already available [Verify this claim]
  - Finally, to show that $\neg S$ holds [Why?], we add $(S)_8$ to the KB as a negated query and derive $()_9$ by Res$(2, 8)$

Resolution

- **Step 1: Formalization**
  - We set the variables $S =$ "We are going to Spain", $N =$ "We are from Newcastle", and $P =$ "We stopped in Paris"
  - From three propositions of *father*, *mother*, and *daughter*, we obtain $(S \lor N) \land [(\neg S \land N) \lor (P \land \neg S)] \land (\neg N \lor P)$
- **Step 2: Transformation into CNF**
  Converting the above formula into CNF gives us the following knowledge base (clauses are numbered by the subscripted indices) $KB \equiv (S \lor N)_1 \land (\neg S)_2 \land (P \lor N)_3 \land (\neg N \lor P)_4$
- **Step 3: Proof**
  - We write *Res*(m, n) : $\langle$*clause*$\rangle_k$ to indicate that $\langle$clause$\rangle$ is obtained by resolution of clause m and clause n and is numbered k
  - Res$(1, 2) : (N)_5$, Res$(3, 4) : (P)_6$, and Res$(1, 4) : (S \lor P)_7$
  - P is also derived from Res$(4, 5)$ and Res$(2, 7)$. Every further resolution step would lead to the derivation of clauses that are already available [Verify this claim]
  - Finally, to show that $\neg S$ holds [Why?], we add $(S)_8$ to the KB as a negated query and derive $()_9$ by Res$(2, 8)$
  - Thus, we obtain $\neg S \land N \land P$. The family comes from Newcastle, stopped in Paris, but is not going to Spain

Resolution

Example 7

Another logic puzzle from [Berrondo 1989]

The High Jump

Three girls practice high jump for their physical education final exam. The bar is set to 1.20 meters. "I bet", says the first girl to the second, "that I will make it over if, and only if, you don't". If the second girl said the same to the third, who in turn said the same to the first, would it be possible for all three to win their bets?

Resolution

- **Step 1: Formalization**
  - We set the variables $A =$ "The first girl's jump succeeds", $B =$ "The second girl's jump succeeds", and $C =$ "The third girl's jump succeeds"
  - The first girl's bet is $A \Leftrightarrow \neg B$, the second girl's bet is $B \Leftrightarrow \neg C$, and the third girl's bet is $C \Leftrightarrow \neg A$
  - We show that they cannot all win the bet, that is, $Q \equiv \neg((A \Leftrightarrow \neg B) \wedge (B \Leftrightarrow \neg C) \wedge (C \Leftrightarrow \neg A))$ *holds*. In other words, we need to *show by resolution that $\neg Q$ is unsatisfiable, i.e., $\neg Q \vdash ()$*
- **Step 2: Transformation into CNF**
  $\neg Q \equiv (\neg A \vee \neg B)_1 \wedge (A \vee B)_2 \wedge (\neg B \vee \neg C)_3 \wedge (B \vee C)_4 \wedge (\neg C \vee \neg A)_5 \wedge (C \vee A)_6$
- **Step 3: Proof**
  - Res$(1, 6) : (\neg B \vee C)_7$
  - Res$(4, 7) : (C)_8$
  - Res$(2, 5) : (B \vee \neg C)_9$
  - Res$(3, 9) : (\neg C)_{10}$
  - Res$(8, 10) : ()$

Resolution

Exercise 12 ([Ertel 2025], Exercise 2.9, p. 39)

Formalize the following logical functions with the logical operators and show that your formula is valid. Present the result in CNF

(a) The XOR operation (exclusive or) between two variables (Recall that p XOR q is true if and only if exactly one of p and q is true)

(b) The statement "at least two of the three variables A, B, C are true"

Exercise 13 ([Ertel 2025], Exercise 2.10, p. 39)

Solve the following case with the help of a resolution proof:

> If the criminal had an accomplice, then he came in a car. The criminal had no accomplice and did not have the key, or he had the key and an accomplice. The criminal had the key. Did the criminal come in a car or not?

Exercise 14 ([Ertel 2025], Exercise 2.11, p. 40)

Show by resolution that the formula

(a) $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$ is a tautology

(b) $\neg(\neg$gas_in_tank $\wedge$ (gas_in_tank $\vee \neg$car_starts) $\Rightarrow \neg$car_starts) is unsatisfiable

Horn Clauses

- The (refutation-)completeness of *resolution calculus* makes it a *very important inference method*.
- In many practical situations, however, *the full power of resolution is not needed.* Some real-world knowledge bases *satisfy certain restrictions on the form of formulas (sentences)* they contain, which enables them to *use a more restricted and efficient inference algorithm*

Horn Clauses

Horn Clauses

are the clauses of the form (having *at most one positive literal*)

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \quad \text{or} \quad (\neg A_1 \vee \cdots \vee \neg A_m) \quad \text{or} \quad B$$

or (equivalently)

$$A_1 \wedge \cdots \wedge A_m \Rightarrow B \quad \text{or} \quad A_1 \wedge \cdots \wedge A_m \Rightarrow \mathbf{f} \quad \text{or} \quad B$$

Example 8

- Set $A_1 =$ "The weather is nice", $A_2 =$ "There is snow on the ground", $B =$ "I will go skiing", $C =$ "I will work"
- The sentence "If the weather is nice and there is snow on the ground, I will go skiing or I will work" $(A_1 \wedge A_2 \Rightarrow B \vee C)$ is not a Horn clause
- The sentence "If the weather is nice and there is snow on the ground, I will go skiing" $(A_1 \wedge A_2 \Rightarrow B)$ is a Horn clause

Horn Clauses

- Horn clauses are *distinctly simpler to interpret* than the general clauses
- A Horn clause with a single positive literal is a *fact*. In Horn clauses with negative and one positive literal, the positive literal is called the *head*, and the rest is called the *body*

Exercise 15 ([Ertel 2025], Exercise 2.12, p. 40)

Prove the following equivalences, which are important for working with Horn clauses:

(a) $(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \equiv A_1 \wedge \cdots \wedge A_m \Rightarrow B$

(b) $(\neg A_1 \vee \cdots \vee \neg A_m) \equiv A_1 \wedge \cdots \wedge A_m \Rightarrow \mathbf{f}$

(c) $A \equiv \mathbf{t} \Rightarrow A$

Horn Clauses

- Horn clauses are easier to handle not only in daily life, but also in formal reasoning
- The *generalized Modus Ponens rule*

$$\frac{A_1 \wedge \cdots \wedge A_m, A_1 \wedge \cdots \wedge A_m \Rightarrow B}{B}$$

Horn Clauses

- Horn clauses are easier to handle not only in daily life, but also in formal reasoning
- The *generalized Modus Ponens rule*

$$\frac{A_1 \wedge \cdots \wedge A_m, A_1 \wedge \cdots \wedge A_m \Rightarrow B}{B}$$

Example 9

Let the knowledge base consist of the following clauses

- $(\text{nice\_weather})_1$
- $(\text{snowfall})_2$
- $(\text{snowfall} \Rightarrow \text{snow})_3$
- $(\text{nice\_weather} \wedge \text{snow} \Rightarrow \text{skiing})_4$

Does "skiing" hold?

Horn Clauses

- Horn clauses are easier to handle not only in daily life, but also in formal reasoning
- The *generalized Modus Ponens rule*

$$\frac{A_1 \wedge \cdots \wedge A_m, A_1 \wedge \cdots \wedge A_m \Rightarrow B}{B}$$

Example 9

Let the knowledge base consist of the following clauses

- (nice_weather)$_1$
- (snowfall)$_2$
- (snowfall $\Rightarrow$ snow)$_3$
- (nice_weather $\wedge$ snow $\Rightarrow$ skiing)$_4$

Does “skiing” hold? YES (use Modus Ponens (MP))

- MP$(2, 3)$: (snow)$_5$
- MP$(1, 5, 4)$: (skiing)$_6$

Horn Clauses

- With *Modus Ponens* we obtain a *complete* calculus for *formulas that consist of propositional logic Horn clauses*
  - If KB contains only Horn clauses and $KB \models Q$, then $KB \vdash Q$ using just Modus Ponens
  - Horn clauses are closed under resolution (and therefore Modus Ponens): if you resolve two Horn clauses, you get back a Horn clause
- Modus Ponens can be used with *forward chaining* or *backward chaining* algorithms
  - **Forward Chaining:** starts with facts and finally derives the query (as in Example 9)
    - In the case of large knowledge bases, however, Modus Ponens may derive many unnecessary formulas if one begins with the wrong clauses
  - **Backward Chaining:** starts with the query and works backwards until the facts are reached
  - Both algorithms are very natural and run in time that is *linear in the size of the knowledge base*

Horn Clauses

- For backward chaining, *SLD resolution* is often used instead of Modus Ponens
- Inference rule:

$$\frac{A_1 \wedge \cdots \wedge A_m \Rightarrow B_1, B_1 \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow \mathbf{f}}{A_1 \wedge \cdots \wedge A_m \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow \mathbf{f}}$$

Example 10

- Let's come back to Example 9 and now add the negated query $(\text{skiing} \Rightarrow \mathbf{f})_5$ to the knowledge base
  - $(\text{nice\_weather})_1$
  - $(\text{snowfall})_2$
  - $(\text{snowfall} \Rightarrow \text{snow})_3$
  - $(\text{nice\_weather} \wedge \text{snow} \Rightarrow \text{skiing})_4$
  - $(\text{skiing} \Rightarrow \mathbf{f})_5$
- We carry out SLD resolution beginning with the resolution steps that follow from this clause
  - $\text{Res}(5,4) : (\text{nice\_weather} \wedge \text{snow} \Rightarrow \mathbf{f})_6$
  - $\text{Res}(6,1) : (\text{snow} \Rightarrow \mathbf{f})_7$
  - $\text{Res}(7,3) : (\text{snowfall} \Rightarrow \mathbf{f})_8$
  - $\text{Res}(8,2) : ()$

Horn Clauses

With *SLD Resolution* ("Selection rule driven linear resolution for definite clauses")

- **"linear resolution":** further processing is always done on the currently derived clause
- The search space is reduced
- The literals of the current clause are always processed in a fixed order (for example, from right to left) (**"Selection rule driven"**)
- The literals of the *current* clause are called *subgoals*. The literals of the *negated query* are the *goals*
- The proof (contradiction) is found, if the list of subgoals of the current clauses (the so-called *goal stack*) is empty
- If, for a subgoal $\neg B_i$, there is no clause with the complementary literal B_i as its clause head, the proof terminates and no contradiction can be found

PROLOG programs consist of predicate logic Horn clauses. Their processing is achieved by means of SLD resolution

Horn Clauses

Exercise 16 ([Ertel 2025], Exercise 2.13, p. 40)

Show by SLD resolution that the following Horn clause set is unsatisfiable

- $(A)_1$
- $(B)_2$
- $(C)_3$
- $(D)_4$
- $(E)_5$
- $(A \wedge B \wedge C \Rightarrow F)_6$
- $(A \wedge D \Rightarrow G)_7$
- $(C \wedge F \wedge E \Rightarrow H)_8$
- $(H \Rightarrow \mathbf{f})_9$

Computability and Complexity

- The truth table method determines every model of any formula in finite time
- The sets of unsatisfiable, satisfiable, and valid formulas are decidable
- The worst case running time is $O(2^n)$ where n is the number of variables
- Optimization: *semantic tree*, grows exponentially in the worst case.
- In resolution, in the worst case, the number of derived clauses grows exponentially with the number of clauses

Computability and Complexity

Question

Can proof in propositional logic go faster? Are there better algorithms?

- **Answer:** Probably not
  - **Cook-Levin Theorem ([Cook 1971]; [Levin 1973]):** The 3-SAT problem is NP-complete
- For Horn clauses, however, there is an algorithm in which the computation time for testing satisfiability grows only linearly as the number of literals in the formula increases

Exercise 17 ([Ertel 2025], Exercise 2.14, p. 40)

In Sect. 2.6, it says: "Thus it is clear that there is probably (modulo the P/NP problem) no polynomial algorithm for 3-SAT, and thus probably not a general one either." Justify the "probably" in this sentence.

Applications and Limitations

- Theorem provers for propositional logic are part of the developer's everyday toolset in *digital technology*
  - *Verification of digital circuits*
  - *Generation of test patterns* for testing of microprocessors in fabrication
  - Special proof systems that work with binary decision diagrams (BDD) are also employed as a data structure for *processing propositional logic formulas*
- *Simple AI applications:* simple expert systems can work with discrete variables, few values, no cross-relations between variables
- *Probabilistic logic* uses propositional logic and probabilistic computation to model uncertainty

References

- Ertel, Wolfgang (2025). *Introduction to Artificial Intelligence*. 3rd. Springer. DOI: 10.1007/978-3-658-43102-0.
- Russell, Stuart J. and Peter Norvig (2010). *Artificial Intelligence: A Modern Approach*. 3rd. Pearson.
- Berrondo, Marie (1989). *Fallgruben für Kopffüssler*. 8703.
- Levin, Leonid Anatolevich (1973). “Universal sequential search problems.” In: *Problemy Peredachi Informatsii* 9.3, pp. 115–116.
- Cook, Stephen A. (1971). “The complexity of theorem proving procedure.” In: *Proc. 3rd Ann. ACM Symp. Theory of Computing*, pp. 151–158.