Introduction to Artificial Intelligence

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 400

Subsection 1

Syntax

Syntax of Formulas

Syntax of Propositional Logic

Let $\mathrm{Op} = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ be the set of **logical operators**, Σ a set of symbols, called **propositional variables**, and $\{t, f\}$ the set of **truth symbols**, true and false. The sets Op, Σ and $\{t, f\}$ are pairwise disjoint. Σ is sometimes called the **signature**.

The set of **propositional logic formulas** is recursively defined:

- t and f are (atomic) formulas.
- All propositional variables in 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.

- Example: If $\Sigma = \{A, B, C\}$, the following are formulas:

$$A \wedge B, \quad A \wedge B \wedge C, \quad A \wedge A \wedge A, \quad C \wedge B \vee A,$$
$$(\neg A \wedge B) \Rightarrow (\neg C \vee A), \quad (((A)) \vee B).$$

Names of the Symbols and the Operators

Names of the Symbols and the Operators

We read the symbols and operators in the following way:

Formula	Reading	Formal Name
t	true	
f	false	
$\neg A$	not A	negation
$A \wedge B$	A and B	conjunction
$A \vee B$	A or B	disjunction
$A \Rightarrow B$	if A then B	implication
$A \Leftrightarrow B$	A if and only if B	equivalence

- The formulas defined in this way are so far purely syntactic constructions (strings of symbols) without assigned meaning.
- The meaning provides the semantics.

Subsection 2

Semantics

Interpretations

- In propositional logic there are two truth values: t for "true" and f for "false".
- Is a formula, such as $A \wedge B$ true? The answer depends on whether the variables A and B are true.
- Example: If A stands for "It is raining today" and B for "It is cold today" and these are both true, then $A \wedge B$ is true. If B represents "It is hot today" and this is false, then $A \wedge B$ is false.

Definition of Interpretation

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

- Since every propositional variable can take on two truth values, every propositional logic formula with n different variables has 2^n different interpretations.

Priorities of Connectives and Semantics of Operations

- For unparenthesized formulas, the **priorities** from strongest to weakest binding are:

$$\neg, \quad \wedge, \quad \vee, \quad \Rightarrow, \quad \Leftrightarrow .$$

- The **truth values for the basic operations** are defined for all possible interpretations by the following 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

Semantically Equivalent Formulas

- To clearly differentiate between the equivalence of formulas and syntactic equivalence ($\Leftrightarrow$), we define

Semantic Equivalence

Two formulas F and G are called **semantically equivalent** if they take on the same truth value for all interpretations. In this case, we write $F \equiv G$.

- Semantic equivalence serves in using English (the **meta-language**) to talk about logic (the **object language**).
- $A \equiv B$ is metalinguistic and means that *two formulas A and B are semantically equivalent*.
- In contrast, $A \Leftrightarrow B$ is a syntactic object of the object language of propositional logic.

Classification of Formulas and Models

- According to the number of interpretations in which a formula is true, we can divide formulas into various classes.

Classification of Formulas

A formula is called

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

- Another important concept is that of a model:

Definition of Model

An interpretation is a **model** of a formula if it satisfies the formula.

- The negation of every valid formula is unsatisfiable.
- The negation of a satisfiable, but not valid, formula F is satisfiable.

Using Truth Tables to Evaluate Formulas

- We explore some important equivalences of formulas:

Theorem (Important Equivalences)

The operations $\wedge$ and $\vee$ are commutative and associative. Moreover:

$$
\begin{array}{rcll}
\neg A \vee B & \Leftrightarrow & A \Rightarrow B & \text{(Implication)} \\
A \Rightarrow B & \Leftrightarrow & \neg B \Rightarrow \neg A & \text{(Contrapositive)} \\
(A \Rightarrow B) \wedge (B \Rightarrow A) & \Leftrightarrow & A \Leftrightarrow B & \text{(Equivalence)} \\
\neg(A \wedge B) & \Leftrightarrow & \neg A \vee \neg B & \text{(De Morgan's Law)} \\
\neg(A \vee B) & \Leftrightarrow & \neg A \wedge \neg B & \text{(De Morgan's Law)} \\
A \vee (B \wedge C) & \Leftrightarrow & (A \vee B) \wedge (A \vee C) & \text{(Distributive Law)} \\
A \wedge (B \vee C) & \Leftrightarrow & (A \wedge B) \vee (A \wedge C) & \text{(Distributive Law)} \\
A \vee \neg A & \Leftrightarrow & t & \text{(Tautology)} \\
A \wedge \neg A & \Leftrightarrow & f & \text{(Contradiction)} \\
A \vee f & \Leftrightarrow & A & \\
A \vee t & \Leftrightarrow & t & \\
A \wedge f & \Leftrightarrow & f & \\
A \wedge t & \Leftrightarrow & A &
\end{array}
$$

Proving Equivalences

- We create truth tables to ascertain the truth values of formulas and prove equivalences.
- To show, e.g., $(\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$, we calculate the truth table for $\neg A \vee B$ and $A \Rightarrow B$ and see that the truth values for both formulas are the same for all interpretations. The formulas are therefore equivalent, and thus all the values of the last column are t's.

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

- The proofs for the other equivalences of the preceding theorem are similar.

Subsection 3

Proof Systems

Entailment

- In AI we are interested in processing existing knowledge and deriving new knowledge or answering questions.
- In propositional logic this translates to showing that a query formula Q "follows" from or is "entailed" by a knowledge base KB.

Definition of Entailment

A formula KB **entails** a formula Q (or Q **follows from** KB) if every model of KB is also a model of Q. In this case, we write KB $\models Q$.

- Equivalently, in every interpretation in which KB is true, Q is also true.
- Since interpretations of variables are in play, entailment is a semantic concept.
- Tautologies, such as $A \vee \neg A$, are true in all interpretations. So for every tautology T, it is the case that $\models T$.
- There is an important connection between the semantic concept of entailment and the syntactic concept of implication.

The Deduction Theorem

The Deduction Theorem

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

- Form the truth table for $A \Rightarrow B$:

A	B	$A \Rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

Observe $A \Rightarrow B$ is true except when A is true and B is false.

- If $A \models B$ holds, for every interpretation that makes A true, B is also true. Thus, the second row of the truth table does not even apply in this case. Therefore, $A \Rightarrow B$ is true.
- If, conversely, $A \Rightarrow B$ holds, the second row of the truth table again does not apply. Thus, every model of A is then also a model of B. Therefore, $A \models B$ holds.

Truth Table Method

- If we wish to show that KB entails Q, we can use the truth table method to show that KB $\Rightarrow Q$ is a tautology.
- This provides a proof system for propositional logic, which is easily automated.
- The disadvantage of the method is the very long computation time in the worst case.
  - Specifically, in the worst case with n propositional variables, the formula KB $\Rightarrow Q$ must be evaluated for all 2^n interpretations of the variables.
  - Therefore, the computation time grows exponentially with the number of variables.

Proof by Contradiction

Theorem (Proof by Contradiction)

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

- If a formula KB entails a formula Q, then by the Deduction Theorem $KB \Rightarrow Q$ is a tautology. Therefore the negation $\neg(KB \Rightarrow Q)$ is unsatisfiable. Now note $\neg(KB \Rightarrow Q) \equiv \neg(\neg KB \vee Q) \equiv KB \wedge \neg Q$.
- To show that the query Q follows from the knowledge base KB, we can
  - add the negated query $\neg Q$ to the knowledge base and
  - derive a contradiction.
- This procedure, which is frequently used in mathematics, is used in various automatic proof calculi, such as the resolution calculus and in the processing of PROLOG programs.

Calculi: Soundness and Completeness

- To avoid the truth table method we can syntactically manipulate the formulas KB and Q by application of **inference rules** to simplify them.
- The goal is to be able, in the end, to instantly see that $\text{KB} \models Q$.
- We call this syntactic process **derivation** and write $\text{KB} \vdash Q$.
- Such syntactic proof systems are called **calculi**.

Definition of Soundness and Completeness

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

Sound and Complete Calculi

- The soundness of a calculus ensures that all derived formulas are in fact semantic consequences of the knowledge base. The calculus does not produce any "false consequences".
- The completeness of a calculus, on the other hand, ensures that the calculus does not overlook anything. A complete calculus always finds a proof if the formula to be proved follows from the knowledge base.

$$\begin{array}{rccccc} \text{Syntax:} & & \text{KB} & \vdash & Q & \\ & \text{Mod} & \downarrow & & \downarrow & \text{Mod} \\ \text{Semantics:} & & \text{KB} & \models & Q & \end{array}$$

- If a calculus is sound and complete, then syntactic derivation and semantic entailment are identical relations.

Conjunctive Normal Form (CNF)

- Automatic proof systems usually operate on formulas in conjunctive normal form.

Definition of CNF

- A **literal** is a variable (**positive literal**) or a negated variable (**negative literal**).
- A **clause** K_i consists of a disjunction $L_{i1} \vee L_{i2} \vee \cdots \vee L_{in_i}$ of literals.
- 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.

- Example: The formula

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

is in conjunctive normal form.

The CNF Theorem

The CNF Theorem

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

- Example: Transform $A \vee B \Rightarrow C \wedge D$ into conjunctive normal form.

$$
\begin{aligned}
& A \vee B \Rightarrow C \wedge D \\
\equiv \quad & \neg(A \vee B) \vee (C \wedge D) && \text{(Implication)} \\
\equiv \quad & (\neg A \wedge \neg B) \vee (C \wedge D) && \text{(De Morgan)} \\
\equiv \quad & (\neg A \vee (C \wedge D)) \wedge (\neg B \vee (C \wedge D)) && \text{(Distributive)} \\
\equiv \quad & ((\neg A \vee C) \wedge (\neg A \vee D)) \wedge ((\neg B \vee C) \wedge (\neg B \vee D)) && \text{(Distributive)} \\
\equiv \quad & (\neg A \vee C) \wedge (\neg A \vee D) \wedge (\neg B \vee C) \wedge (\neg B \vee D) && \text{(Associative)}
\end{aligned}
$$

Modus Ponens and Resolution

- We develop a calculus for syntactic proofs of propositional logic formulas.
- We start with the **modus ponens**, which allows the derivation of B from the validity of A and $A \Rightarrow B$, written $\frac{A, A \Rightarrow B}{B}$.
- Modus ponens, as a rule by itself, is sound but not complete.
- By adding more rules we can create a complete calculus, but we do not do that here.
- The **resolution rule** is $\frac{A \vee B, \neg B \vee C}{A \vee C}$. $A \vee C$ is called the **resolvent**.
- An equivalent form is $\frac{A \vee B, B \Rightarrow C}{A \vee C}$. If we set A to f, we see that the resolution rule is a generalization of the modus ponens.
- The resolution rule can also be used if C is missing or if both A and C are missing. In the latter case the empty clause can be derived from the contradiction $B \wedge \neg B$.

Subsection 4

Resolution

The General Resolution Rule

- To allow clauses with an arbitrary number of literals, we introduce the **General Resolution Rule**.
- If $A_1, \ldots, A_m$, B, $C_1, \ldots, C_n$ are literals, then

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

- The literals B and $\neg B$ are called **complementary**.
- The resolution rule has the effect of deleting a pair of complementary literals from the two clauses and combines the rest of the literals into a new clause.

Adding Factorization

- To prove that from a knowledge base KB, a query Q follows, we carry out a proof by contradiction.
- We must show that a contradiction can be derived from $\text{KB} \wedge \neg Q$.
- In CNF, a contradiction appears in the form of two clauses (A) and $(\neg A)$, which lead to the empty clause as their resolvent.
- To ensure this process really works, we need a complete calculus, which forces a new addition.
- Example: Let $\text{KB} = (A \vee A)$. Suppose we want to show $Q = (A \wedge A)$ using resolution.
  We form $\text{KB} \wedge \neg Q = (A \vee A) \wedge \neg(A \wedge A) \equiv (A \vee A) \wedge (\neg A \vee \neg A)$.
  With the resolution rule alone, this is impossible.
- Factorization allows deletion of copies of literals from clauses. We get

$$\dfrac{\dfrac{(A \vee A), (\neg A \vee \neg A)}{(A), (\neg A)}}{()} \quad \begin{matrix} \text{(Factorization)} \\ \text{(Resolution)} \end{matrix}$$

Resolution Calculus

Theorem (Soundness and Completeness of Resolution Calculus)

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

- Since the resolution calculus must derive a contradiction from $KB \wedge \neg Q$, the knowledge base KB itself must be consistent:

Definition of Consistent Formula

A formula KB is called **consistent** if it is impossible to derive from it a contradiction, i.e., a formula of the form $\varphi \wedge \neg\varphi$.

- If KB is not consistent, anything can be derived from KB.
- Resolution has only two inference rules, and it works with formulas in conjunctive normal form. This leads to a simpler implementation.
- Another advantage is the relatively small number of possibilities for the application of inference rules in every step of the proof. This reduces the size of the search space.

The English Family Puzzle

- A native German speaker picked up in Bavaria three hitchhikers, a father, mother, and daughter, and realized that they were English and only spoke English. At each of the sentences they spoke he wavered between two possible interpretations:

  Father: "We are going to Spain" or "We are from Newcastle"
  Mother: "We are not going to Spain and are from Newcastle" or "We stopped in Paris and are not going to Spain"
  Daughter: "We are not from Newcastle" or "We stopped in Paris"

  What can be concluded about the English family?
- We work in three steps:
  - Formalization,
  - Transformation into normal form,
  - Proof.
- Formalization is the most difficult step because it is easy to make mistakes or forget small details.

Untangling the Puzzle

- Formalization: Introduce the variables: S for "We are going to Spain", N for "We are from Newcastle", P for "We stopped in Paris".

  Father: $(S \vee N)$.
  Mother: $(\neg S \wedge N) \vee (P \wedge \neg S)$.
  Daughter: $(\neg N \vee P)$.

- Transformation into Normal Form:
  Original Sentence: $(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P)$.
  Factoring out $\neg S$ in the middle sub-formula yields
  KB: $(S \vee N) \wedge (\neg S) \wedge (P \vee N) \wedge (\neg N \vee P)$.
- Proof:

$$\frac{\frac{(S \vee N), (\neg S), (P \vee N), (\neg N \vee P)}{(N), (P \vee N), (\neg N \vee P)}}{(N), (P)}$$

The English family comes from Newcastle, stopped in Paris, but is not going to Spain.

The High Jump Bets Puzzle

- Three girls practice high jump. The bar is set to 1.20 meters.

  Girl 1 to Girl 2: I bet I will make it over iff you don't.
  Girl 2 to Girl 3: I bet I will make it over iff you don't.
  Girl 3 to Girl 1: I bet I will make it over iff you don't.

  Would it be possible for all three to win their bets?

- Formalization: A: Girl 1 succeeds, B: Girl 2 succeeds, C: Girl 3 succeeds.

  Girl 1 to Girl 2: $(A \Leftrightarrow \neg B)$.
  Girl 2 to Girl 3: $(B \Leftrightarrow \neg C)$.
  Girl 3 to Girl 1: $(C \Leftrightarrow \neg A)$.

  Not all three can win their bets:
  $Q \equiv \neg((A \Leftrightarrow \neg B) \wedge (B \Leftrightarrow \neg C) \wedge (C \Leftrightarrow \neg A))$.
  We must show that $\neg Q$ is unsatisfiable.

The High Jump Bets Puzzle (Cont'd)

- Transformation into CNF:
  Note that

$$\begin{aligned} \neg Q &\equiv (A \Leftrightarrow \neg B) \wedge \cdots \\ &\equiv (A \Rightarrow \neg B) \wedge (\neg B \Rightarrow A) \wedge \cdots \\ &\equiv (\neg A \vee \neg B) \wedge (A \vee B) \wedge \cdots . \end{aligned}$$

  Thus,
  $\neg Q \equiv (\neg A \vee \neg B) \wedge (A \vee B) \wedge (\neg B \vee \neg C) \wedge (B \vee C) \wedge (\neg C \vee \neg A) \wedge (C \vee A).$

- Proof:

$$\dfrac{\dfrac{\dfrac{(\neg A \vee \neg B), (A \vee B), (\neg B \vee \neg C), (B \vee C), (\neg C \vee \neg A), (C \vee A)}{(C \vee \neg B), (B \vee C), (B \vee \neg C), (\neg B \vee \neg C)}}{(C), (\neg C)}}{()}$$

  Therefore, $\neg Q$ is unsatisfiable, showing that Q must hold.

Subsection 5

Horn Clauses

Definite Clauses

- A clause in conjunctive normal form may contain positive and negative literals:

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n),$$

  where $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$ are variables.
- An equivalent form is $A_1 \wedge \cdots \wedge A_m \Rightarrow B_1 \vee \cdots \vee B_n$.
- Example: "If the weather is nice and there is snow on the ground, I will go skiing or I will work" is a proposition of this form.
- A clearer statement of intention would be "If the weather is nice and there is snow on the ground, I will go skiing".
- Since the receiver of the latter statement knows definitively, clauses with exactly one positive literal are called **definite clauses**.

Horn Clauses

Definition of Horn Clauses

Clauses with at most one positive literal of the form

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

or, equivalently,

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

are named **Horn clauses**.
A clause of the last kind (just a single positive literal) is called a **fact**.
In clauses of the first kind (with negative and one positive literal), the positive literal is called the **head**.

- Horn clauses are easier to handle both in daily life and in formal reasoning.

Reasoning with Horn Clauses

- Let the knowledge base consist of the following clauses

  (nice_weather)
  (snowfall)
  (snowfall⇒ snow)
  (nice_weather ∧ snow⇒ skiing)

  Suppose we now want to know whether (skiing) holds.
- A slightly generalized modus ponens suffices as an inference rule:

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

- The proof proceeds as follows:

$$\frac{\dfrac{(\text{snowfall}), (\text{snowfall} \Rightarrow \text{snow})}{(\text{snow}),} \qquad (\text{nice\_weather}), (\text{nice\_weather} \wedge \text{snow} \Rightarrow \text{skiing})}{(\text{skiing})}$$

- Modus ponens constitutes a complete calculus for Horn clauses.

Selection Rule Driven Linear Resolution (SLD)

- Since modus ponens may derive many unnecessary formulas, in many cases it is better to use a calculus that starts with the query and works backward until the facts are reached.
- For backward chaining of Horn clauses, SLD resolution is used.
- The initials stand for "Selection rule driven Linear resolution for Definite clauses".
- Example: Augment the previous knowledge base with (skiing$\Rightarrow$ f):

$$\text{(nice\_weather)}, \quad \text{(snowfall)}, \quad \text{(snowfall}\Rightarrow \text{snow)}$$
$$\text{(nice\_weather} \wedge \text{snow}\Rightarrow \text{skiing)}, \quad \text{(skiing}\Rightarrow f\text{)}$$

We work as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\text{skiing}\Rightarrow f),\ (\text{nice\_weather} \wedge \text{snow}\Rightarrow \text{skiing})}{(\text{nice\_weather} \wedge \text{snow} \Rightarrow f),\quad (\text{nice\_weather})}}{(\text{snow} \Rightarrow f),\quad (\text{snowfall} \Rightarrow \text{snow})}}{(\text{snowfall} \Rightarrow f),\quad (\text{snowfall})}}{()}$$

SLD Features

- We can easily see
  - Linearity: Processing is always done on the currently derived clause.
  - Selection Rule Driven: Literals of the current clause are always processed in a fixed order, e.g., left-to-right.
- The literals of the current clause are called **subgoals**.
- The literals of the negated query are the **goals**.
- The inference rule for one step reads $\frac{A_1 \wedge \cdots \wedge A_m \Rightarrow B_1, B_1 \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow f}{A_1 \wedge \cdots \wedge A_m \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow f}$. Before application $B_1, B_2, \ldots, B_n$ must be proved. After application, B_1 is replaced by the new subgoal $A_1 \wedge \cdots \wedge A_m$. This process continues until the list of subgoals of the current clauses (the **goal stack**) is empty. With that, a contradiction has been found.
- 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. The query is, then, unprovable.

Subsection 6

Complexity and Limitations

Complexity Issues

- The truth table method:
  - Algorithm that can determine all models of a formula in finite time.
  - Sets of unsatisfiable, satisfiable, and valid formulas are **decidable**.
  - The computation time of the truth table method for satisfiability grows in the worst case exponentially with the number n of variables.
  - An optimization, the method of semantic trees, avoids looking at variables that do not occur in clauses, and thus saves computation time in many cases, but in the worst case it is still exponential.
- The resolution method:
  - In the worst case the number of derived clauses grows exponentially with the number of initial clauses.
- To decide between the two processes, the rule of thumb is that
  - in the case of many clauses with few variables, the truth table method is preferable;
  - in the case of few clauses with many variables, resolution will probably finish faster.

Applications and Limitations

- Theorem provers for propositional logic are part of the developer's toolbox in digital technology.
  - The verification of digital circuits and the generation of test patterns for testing of microprocessors are some of these tasks.
- Special proof systems that work with binary decision diagrams (BDD) are used as a data structure for processing propositional formulas.
- In AI, propositional logic is employed in simple applications.
  - Simple expert systems can certainly work with propositional logic. The variables must all be discrete, with only a few values, and cross-relations between variables are not allowed.
- Predicate logic elegantly expresses logical connections.
- Probabilistic logic is a combination of propositional logic and probabilistic computation that allows modeling of uncertain knowledge.
- Fuzzy logic allows infinitely many truth values.