HANOI UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

KNOWLEGDE

Student: Tran Duc Viet - 2101140080

Nguyen Thanh Vinh – 2101140081

Pham Tuan Phuong - 2201140068

**Tables of contents**

1. Terms 5

1.1. **Knowledge** 5

1.1.1. **Knowledge-Based Agents** 5

1.1.2. **Sentence** 5

1.1.3. **Example** 5

1.2. **Propositional Logic** 6

1.2.1. Propositions: Statements that are true or false. 6

1.2.2. Symbols: Letters representing propositions (e.g., P, Q, R). 6

1.2.3. Logic Connectives 6

1.2.4. Knowledge Base 7

1.2.5. **Model** 7

1.2.6. Entailment (⊨) 7

1.3. **Inference** 7

1.3.1. Process: Deriving new sentences from existing ones. 7

1.3.2. Model Checking: Enumerate all models to check entailment. 7

1.3.3. 7

1.4. **Inference Rules** 8

1.4.1. Modus Ponens: If P → Q and P, then Q. 8

1.4.2. And Elimination: From P ∧ Q, infer P or Q. 8

1.4.3. Double Negation: ¬¬P is equivalent to P. 8

1.4.4. Implication Elimination: P → Q is equivalent to ¬P ∨ Q. 8

1.4.5. Biconditional Elimination: P ↔ Q is equivalent to (P → Q) ∧ (Q → P). 8

1.4.6. De Morgan’s Laws: Transformations between And/Or with negations. 8

1.5. **Knowledge Engineering** 9

1.5.1. Process: Representing propositions and logic in AI. 9

1.5.2. Example 9

1.6. **Resolution** 10

1.6.1. Definition: Inference rule using complementary literals. 10

1.6.2. Application: Convert to Conjunctive Normal Form (CNF). 10

1.6.3. Example 10

1.7. **First Order Logic** 10

1.7.1. Symbols: Constant and Predicate symbols. 10

1.7.2. Quantification 10

2. Explain Code 11

2.1. **logic.py** 11

2.1.1. defines a **framework to represent logical sentences and perform logical inference** using basic components like AND, OR, NOT, and others 11

2.1.2. **Classes** 11

2.1.3. **model\_check(knowledge, query)** 13

* **Terms**
* **Knowledge**
* **Knowledge-Based Agents**

Knowledge-based agents represent a sophisticated class of artificial intelligence systems that employ internal knowledge representations to process information and make decisions. These agents are distinguished by their ability to maintain and utilize a structured repository of information about their domain of operation. The primary purpose of these agents is to draw meaningful conclusions from their existing knowledge base, enabling them to respond effectively to new situations and challenges..

* **Sentence**

At the core of knowledge-based systems lies the concept of sentences - formal assertions about the world expressed in a knowledge representation language. These sentences serve as the basic units of knowledge, allowing systems to encode complex information in a format suitable for logical manipulation and inference. Each sentence represents a discrete piece of information that contributes to the agent's overall understanding of its environment.

* **Example**

Consider the following logical scenario that demonstrates how knowledge-based agents process information:

Given the premises:

* "If there is no rain, I will learn AIN"
* "I will either learn AIN or play FIFA, but not both"
* "I play FIFA"

This example illustrates how knowledge-based agents can combine multiple pieces of information to derive new conclusions that are not explicitly stated in their knowledge base. The ability to make such inferences demonstrates the power of formal knowledge representation in artificial intelligence systems.

* **Propositional Logic**
* **Propositions**

A proposition is a declarative statement that is either true or false, but not both. It is the fundamental building block of propositional logic. Propositions must have several key characteristics:

* Must be a complete statement
* Must have a definite truth value
* Must be unambiguous
* Cannot be a question or command
* **Symbols**

In propositional logic, we use symbols (typically letters) to represent propositions:

* P, Q, R are most commonly used
* Lower case letters (p, q, r) may also be used
* Subscripts can be added when needed (P₁, P₂, P₃)
* **Logic Connectives**
* AND (∧)

The AND operator (∧) in propositional logic is a fundamental logical connective that combines two propositions to create a compound statement that is true only when both of its constituent propositions are true. For example, consider two propositions: P: "The color of the sky is red" and Q: "2+2 is 4". When combined with the AND operator, we get P ∧ Q: "The sky is red and 2+2 is 4". This compound proposition would be false because while Q is true (2+2 is indeed 4), P is false (the sky is not red), and the AND operator requires both propositions to be true for the compound statement to be true. This property makes the AND operator particularly useful in computer programming, logical reasoning, and system design where multiple conditions must be simultaneously satisfied, such as in security checks, database queries, or safety systems.

* OR (∨)

The OR operator (∨) in propositional logic serves as a logical connective that yields a true compound statement when at least one of its component propositions is true. Consider the example where P represents "The Sun is a small planet" (which is false, as the Sun is a star) and Q represents "Earth is a planet" (which is true). When these propositions are combined using the OR operator, we get P ∨ Q: "The Sun is a small planet or Earth is a planet." The resulting compound statement is true, even though one of its components (P) is false. This demonstrates a key property of the OR operator: it only requires one proposition to be true for the entire statement to be true. In this case, since Q is true (Earth is indeed a planet), the entire compound statement P ∨ Q is true, regardless of P being false. This makes the OR operator particularly useful in scenarios where we need to verify if at least one condition among several possibilities is satisfied, such as in computer programming, decision-making systems, or logical problem-solving.

* NOT (¬)

The NOT (¬) connective in logic negates a given proposition, reversing its truth value. For example, if P represents "It is raining," then ¬P means "It is not raining." If P is true (it is raining), then ¬P is false (it is not true that it isn't raining), and vice versa. This negation is crucial for constructing more complex logical statements, as it allows reasoning about both the presence and absence of conditions in logical systems.

* Implication (→)

The Implication (→) connective in logic establishes a conditional relationship between two propositions, where one proposition (the antecedent) leads to another (the consequent). In the example, P represents "Juan passes the exam," and Q represents "Juan's mother will give him a gift." The implication P → Q translates to "If Juan passes the exam, his mother will give him a gift." This means that whenever P is true (Juan passes the exam), Q must also be true (his mother gives him a gift). If P is false (Juan doesn't pass), the implication still holds true regardless of Q. Implication is crucial in expressing cause-and-effect relationships in logical reasoning.

* Biconditional (↔)

The Biconditional (↔) connective in logic expresses that two propositions are equivalent, meaning both must either be true or false simultaneously. In this example, P represents "A triangle is equilateral," and Q represents "The three sides of a triangle are equal." The biconditional P ↔ Q is interpreted as "A triangle is equilateral if and only if its three sides are equal." This means that if P is true (the triangle is equilateral), then Q must also be true (its three sides are equal), and vice versa. The truth value of P ↔ Q is true (V(P↔Q) = T) only when both P and Q share the same truth value, either both true or both false. Biconditional statements are vital for defining precise conditions in logical reasoning.

* **Knowledge Base**

A Knowledge Base is a collection of known sentences or statements. It serves as a repository of information that can be utilized to make inferences about the world. Essentially, it is the foundational data that an inference system relies on to derive new insights.

* **Model**

A Model is an assignment of truth values to propositions. It represents a particular way in which the world might be. By assigning truth values to each proposition, a model helps in understanding the possible states of the world.

* **Entailment (⊨)**

Entailment refers to the logical relationship between two propositions, where one proposition (β) is true in all models where another proposition (α) is true. If α ⊨ β, then β is true whenever α is true.

Example

α: "Today is a Monday in March."

β: "Today is in March."

In this case, α ⊨ β because if "Today is a Monday in March" is true, it necessarily implies that "Today is in March."

* **Inference**
* **Process**

Inference is the process of deriving new sentences or conclusions from existing ones. It involves logical reasoning to expand the knowledge base by adding new, logically deduced statements.

* **Model Checking**

Model Checking is a method used to verify entailment by enumerating all possible models. It involves systematically checking whether a given proposition is true in all models where the initial proposition is true.

* **Example**

It states that if it's sunny, Mary will go to the park. However, Mary also has the choice to either go to the park or go shopping, but not both. Given that Mary went shopping today, it can be deduced that she did not go to the park. Consequently, it can be inferred that today is not sunny, as the initial condition for Mary going to the park (sunny weather) is not met.

* **Inference Rules**
* **Modus Ponens**

Modus Ponens is a fundamental rule of inference that states if we have a conditional statement (P → Q) and the antecedent (P) is true, then the consequent (Q) must also be true.

* Premise 1: If I eat too much candy (P), then I'll get a sugar rush (Q).
* Premise 2: I ate too much candy (P).
* Conclusion: Therefore, I’m bouncing off the walls (Q).

Modus Ponens is widely used in everyday reasoning and formal proofs, serving as a reliable method to validate conclusions based on established conditions.

* **And Elimination**

And Elimination allows us to infer one component of a conjunction from the conjunction itself. If both P and Q are true (P ∧ Q), we can conclude that P or Q is true.

* Premise: I have a pen (P) and a notebook (Q).
* Conclusion: Therefore, I have a pen (P).

This rule is crucial for breaking down complex statements into simpler components, facilitating clearer reasoning and analysis.

* **Double Negation**

Double Negation asserts that negating a statement twice results in the original statement. In formal terms, ¬¬P is equivalent to P.

* Premise: It is not true that I don’t have a car (¬¬P).
* Conclusion: Therefore, I have a car (P).

This rule highlights the importance of clarity in logical expressions, demonstrating that affirmations can often be obscured by double negatives.

* **Implication Elimination**

Implication Elimination states that a conditional statement (P → Q) can be expressed as a disjunction (¬P ∨ Q). This transformation is useful in various logical proofs.

* Premise: If I wear my lucky socks (P), then my team will win (Q).
* Conclusion: Therefore, if my team loses (¬Q), I forgot my socks (¬P).

This rule emphasizes the interconnectedness of implication and disjunction, aiding in the reformatting of logical statements for easier manipulation.

* **Biconditional Elimination**

Biconditional Elimination establishes that a biconditional statement (P ↔ Q) can be decomposed into two implications: (P → Q) and (Q → P).

* Premise: I will go to the party (P) if and only if you go (Q).
* Conclusion: Therefore, if I go (P), you go (Q), and if you go (Q), I go (P).

This rule is instrumental in establishing equivalence between statements, providing a robust framework for reasoning about mutual conditions.

* **De Morgan’s Laws**

De Morgan’s Laws provide a method for transforming conjunctions and disjunctions under negation. Specifically, ¬(P ∧ Q) is equivalent to ¬P ∨ ¬Q, and ¬(P ∨ Q) is equivalent to ¬P ∧ ¬Q.

* Premise: It’s not true that I will both clean my room (P) and do my homework (Q).
* Conclusion: Therefore, I will either not clean my room (¬P) or not do my homework (¬Q).

De Morgan’s Laws are essential for simplifying complex logical expressions and understanding the relationships between negation and conjunction/disjunction.

* **Knowledge Engineering**
* **Process: Representing propositions and logic in AI.**
* **Example**









* **Resolution**
* **Definition**

Resolution is a powerful inference rule used in automated theorem proving and logic programming. It involves combining two clauses (disjunctions of literals) to produce a new clause. The resolution rule states that if we have two clauses P ∨ Q and ¬P ∨ R, we can infer the clause Q ∨ R. This rule is based on the principle that if one of P or Q is true, and one of ¬P or R is true, then either Q or R must be true.

* **Application: Convert to Conjunctive Normal Form (CNF).**
* **Example**
* (A→B)∧(¬C∨D) ➡ (¬A∨B)∧(¬C∨D)
* **First Order Logic**
* **Symbols:** Constant and Predicate symbols.

First-order logic, also known as predicate logic, is a more expressive language than propositional logic. It allows us to represent knowledge about objects, properties, and relationships between them. First-order logic includes:

* **Constant symbols:** Represent individual objects or entities, such as "John" or "the book."
* **Predicate symbols:** Represent properties or relationships between objects, such as "is tall," "is red," or "loves."
* **Quantification**

Quantifiers: Used to express the scope of statements.

* Universal quantifier (∀): Means "for all" or "for every."
* Existential quantifier (∃): Means "there exists" or "for some."
* **Explain Code**
* **logic.py**
* Defines a **framework to represent logical sentences and perform logical inference** using basic components like AND, OR, NOT, and others
* **Classes**

In the study of formal logic, constructing classes that represent logical sentences is a critical component. The base class Sentence serves as an abstract foundation for all logical sentences, featuring methods such as `validate(sentence)` to ensure the sentence's validity and `parenthesize(s)` to add parentheses when necessary. It also includes abstract methods like `evaluate(model)`, `formula () `, and `symbols ()`, which assess the logical value of the sentence based on a given model, return a string representation of the sentence, and provide a set of all symbols within the sentence. Subclasses like Symbol, Not, And, Or, Implication, and Biconditional extend from the Sentence class, each representing specific logical operations with corresponding methods for initialization, evaluation, string representation, and symbol extraction.

* **model check(knowledge, query)**

Notably, the method `model check (knowledge, query)` checks whether the knowledge base entails the query by utilizing `check all(knowledge, query, symbols, model)`, which recursively examines all possible models with true and false values for each symbol, ensuring that if the knowledge base is true in a model, the query must also be true. This structure not only provides a systematic approach to handling logical sentences but also supports the development of more complex logical systems.

* **clue.py, harry.py, mastermind.py, puzzle.py**
* Examples of using the logic.py framework
* Detail in **01\_puzzle\_explanation.ipynb, 02\_clue\_explanation.ipynb, 03\_harry\_explanation.ipynb, 04\_mastermind\_explanation.ipynb**.