**Syntax and Semantics of First-Order Logic in NLP**

***First-order logic (FOL)***, also known as first-order predicate logic, is a fundamental formal system used in mathematics, philosophy, computer science, and linguistics for expressing and reasoning about relationships between objects in a domain. Its syntax and semantics provide a robust framework for encoding information in a precise and structured manner, enabling AI systems to perform tasks such as automated reasoning, planning, and natural language understanding.

* [Syntax of First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/" \l "syntax-of-firstorder-logic)
* [Quantifiers in First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#quantifiers-in-firstorder-logic)
* [Well-Formed Formulas (WFFs) in First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#wellformed-formulas-wffs-in-firstorder-logic)
* [Semantics of First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#semantics)
* [Satisfaction in First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#satisfaction)
* [Validity in First-Order Logic](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#validity)
* [Applications of First-Order Logic in AI](https://www.geeksforgeeks.org/syntax-and-semantics-of-first-order-logic-in-ai/#applications-of-firstorder-logic-in-ai)

**Syntax of First-Order Logic**

The ***syntax of***[***first-order logic***](https://www.geeksforgeeks.org/first-order-logic-in-artificial-intelligence/)consists of symbols and rules for constructing well-formed formulas (WFFs), which are statements or formulas in the language of FOL. The syntax encompasses the language constructs used to express knowledge and relationships within a domain.

**Terms in First-Order Logic**

Terms represent objects or entities within the domain of discourse. In AI, terms can correspond to real-world entities, such as objects, individuals, or abstract concepts. They include:

* **Constants**: Specific entities, e.g., "John", "Apple".
* **Variables**: Placeholders for entities, e.g., "x", "y".
* **Functions**: Expressions applied to terms, e.g., "Age(John)", "Parent(x)".

**Predicates in First-Order Logic**

Predicates express properties, relations, or conditions that hold between objects. They describe the state of the world or assert facts about entities within the domain. Examples include:

* "IsHuman(x)"
* "IsParent(x, y)"

**Quantifiers in First-Order Logic**

Quantifiers in first-order logic allow for the specification of statements about the entirety or existence of objects within the domain.

* **Universal quantifiers (∀)**: Statements that hold for all objects.
* **Existential quantifiers (∃)**: Statements that hold for at least one object.

**Connectives in First-Order Logic**

Logical connectives such as conjunction (∧), disjunction (∨), implication (→), and negation (¬) enable the composition of complex statements from simpler ones. They facilitate the expression of logical relationships and constraints in AI knowledge representations.

**Connectives in First-Order Logic**

* **Conjunction (∧)**:
  + **Meaning**: Represents logical "and" between two propositions. The conjunction of two propositions is true only if both propositions are true.
  + **Example**: If P(x) represents "x is red" and Q(x) represents "x is round", then P(x)∧Q(x) represents "x is red and round".
* **Disjunction (∨)**:
  + **Meaning**: Represents logical "or" between two propositions. The disjunction of two propositions is true if at least one of the propositions is true.
  + **Example**: If P(x) represents "x is a cat" and Q(x) represents "x is a dog", then P(x)∨Q(x) represents "x is either a cat or a dog".
* **Implication (→)**:
  + **Meaning**: Represents logical "if-then" relationship between two propositions. The implication P→Q is true if either Q is true or if P is false.
  + **Example**: If P(x) represents "x is a mammal" and Q(x) represents "x produces milk", then P(x)→Q(x) represents "if x is a mammal, then it produces milk".
* **Negation (¬)**:
  + **Meaning**: Represents logical "not" or negation of a proposition. It reverses the truth value of the proposition.
  + **Example**: If P(x) represents "x is intelligent", then ¬P(x) represents "x is not intelligent".

**Quantifiers in First-Order Logic**

**Universal Quantifier (∀)**

* **Meaning**: Denotes that a statement holds for all objects in the domain.
* **Example**: ∀xP(x) means "for all x, P(x) is true", indicating that property P holds for all objects x in the domain.

**Existential Quantifier (∃)**

* **Meaning**: Denotes that a statement holds for at least one object in the domain.
* **Example**: ∃xP(x) means "there exists an x such that P(x) is true", indicating that there is at least one object x in the domain for which property P holds.

**Well-Formed Formulas (WFFs) in First-Order Logic**

Well-formed formulas (WFFs) in first-order logic (FOL) are expressions constructed according to the syntactic rules of FOL, representing meaningful statements about the world. These formulas serve as the building blocks for encoding knowledge and reasoning in AI systems.

**Characteristics of WFF**

* **Syntax Compliance**: WFFs adhere to the syntax rules of first-order logic, which define how terms, predicates, quantifiers, and logical connectives can be combined to form valid expressions.
* **Symbolic Representation**: WFFs consist of symbols representing terms (constants, variables, and functions), predicates (relations), quantifiers (∀, ∃), and logical connectives (∧, ∨, →, ¬).
* **Quantifier Scope**: WFFs maintain clear quantifier scope, ensuring that quantifiers bind variables appropriately within the formula. The scope of quantifiers affects the interpretation and meaning of the formula.
* **Complexity and Nesting**: WFFs can range from simple atomic formulas to complex nested structures involving multiple quantifiers and connectives. Proper nesting and grouping of subformulas are essential for clarity and unambiguous interpretation.

**Importance of Well-Formed Formulas**

* **Knowledge Representation**: WFFs serve as a formal language for representing knowledge about the world in AI systems. They enable the encoding of facts, rules, constraints, and relationships in a structured and precise manner.
* **Automated Reasoning**: AI systems utilize WFFs for automated reasoning tasks such as deduction, inference, and logical decision-making. Well-formed formulas facilitate the application of formal logic principles to derive new information from existing knowledge.
* **Semantic Understanding**: Understanding the syntax and semantics of WFFs is crucial for natural language processing (NLP) systems to interpret and extract meaning from textual data. Mapping natural language statements to logical representations involves recognizing and constructing well-formed formulas.
* **Problem-Solving and Planning**: In AI planning and problem-solving domains, well-formed formulas play a key role in defining the initial state, goal state, and transition rules of a problem. They enable the formulation of logical constraints and objectives for automated planning algorithms.

**Semantics of First-Order Logic**

***Semantics in first-order logic*** deals with the interpretation of sentences and formulas within the framework of a mathematical model. It provides a way to assign meanings to the symbols and structures used in first-order logic.

**Key Elements of Semantics in First-Order Logic**

* **Variables**: These represent placeholders for objects or elements within a domain.
* **Constants**: These represent specific elements within the domain.
* **Predicates**: These are expressions that can be true or false depending on the objects they're applied to.
* **Functions**: These map elements from the domain to other elements in the domain.
* **Quantifiers**: Such as "for all" (∀) and "exists" (∃), used to express universal and existential quantification, respectively.

**Interpretation in First-Order Logic**

The semantics of first-order logic involve defining what makes a formula true or false in a given interpretation (also called a model). An interpretation consists of:

* **A domain of discourse**: This is the set of objects over which the variables range.
* **Interpretations of constants**: Each constant is mapped to an element in the domain.
* **Interpretations of predicates**: These are mappings that determine whether a predicate holds true for a particular tuple of objects from the domain.
* **Interpretations of functions**: These mappings assign values to functions based on the values of their arguments.

The truth of a sentence or formula in first-order logic is determined by evaluating it within a specific interpretation. This is done recursively, where the truth of atomic formulas (predicates applied to terms) is determined based on the interpretation of predicates and the values of the terms.

* **Universal quantification (∀)** asserts that a statement holds true for all objects in the domain.
* **Existential quantification (∃)** asserts that there exists at least one object for which the statement is true.

Overall, semantics in first-order logic provides a formal framework for understanding how to assign meaning to expressions and how to determine their truth values within a given interpretation.

**Satisfaction in First-Order Logic**

* **Definition**: A formula is said to be satisfied by an interpretation if, under that interpretation, the formula evaluates to true.
* **Symbolic Notation**: M⊨ϕ, where M is an interpretation and ϕ is a formula.

**Atomic Formulas**

An atomic formula P(t₁, t₂, ..., tₙ) is satisfied by an interpretation if the objects assigned to the terms make the predicate P true.

**Complex Formulas**

The satisfaction of complex formulas is determined recursively based on the satisfaction of their constituent parts, considering logical connectives and quantifiers. For example, a conjunction ϕ∧ψ is satisfied if both ϕ and ψ are satisfied.

**Quantifiers**

* A universally quantified formula ∀xϕ(x) is satisfied if ϕ(x) is satisfied for all objects in the domain.
* An existentially quantified formula ∃xϕ(x) is satisfied if ϕ(x) is satisfied for at least one object in the domain.

**Validity in First-Order Logic**

* **Definition**: A formula is considered valid if it is satisfied by every interpretation, meaning it holds true universally.
* **Symbolic Notation**: ⊨ϕ, meaning ϕ is valid.

**Examples**

1. ∀x(P(x)→Q(x)) is valid if, under every interpretation, whenever P(x) holds true, Q(x) also holds true.
2. ∃xP(x) is satisfied if there exists at least one object in the domain for which P(x) holds true.

**Relationship between Satisfaction and Validity**

* A formula is valid if and only if its negation is unsatisfiable. In other words, a formula is valid if there is no interpretation that makes it false.
* If a formula is valid, it is satisfied by every interpretation.
* If a formula is satisfied by a specific interpretation, it does not necessarily mean it is valid unless it holds true under all possible interpretations.

**Applications of First-Order Logic in AI**

First-order logic (FOL) plays a pivotal role in various AI domains by providing a structured and formal framework for representing and reasoning about knowledge. Here are some key applications:

**1. Automated Reasoning**

* **Deduction**: AI systems use FOL to perform logical deductions, deriving new information from existing knowledge bases.
* **Theorem Proving**: FOL underpins automated theorem provers that can verify mathematical theorems and logical assertions.

**2.**[Knowledge Representation](https://www.geeksforgeeks.org/knowledge-representation-in-first-order-logic/)

* **Ontology Engineering**: FOL is used to create and manage ontologies that define the relationships between different concepts within a domain.
* **Expert Systems**: AI systems encode domain-specific knowledge using FOL, enabling them to make informed decisions and provide expert advice.

**3. Natural Language Processing (NLP)**

* **Semantic Parsing**: FOL helps in parsing natural language sentences into logical forms that AI systems can process and understand.
* **Information Extraction**: AI systems use FOL to extract structured information from unstructured text.

**4. Planning and Problem Solving**

* **Automated Planning**: FOL defines the initial state, goal state, and transition rules, allowing AI systems to devise plans to achieve specific objectives.
* **Constraint Satisfaction Problems (CSPs)**: FOL represents constraints and conditions that AI systems must satisfy to find viable solutions to complex problems.

**5. Robotics**

* **Perception and Action**: FOL is used to represent the relationships between objects and actions in a robot’s environment, facilitating autonomous decision-making and navigation.
* **Task Planning**: Robots use FOL to plan and execute sequences of actions to accomplish tasks.

**Syntax-Driven Semantic Analysis in NLP**

**Overview**

**Semantic analysis** means checking the text for meaningfulness. It is defined as drawing the exact or the dictionary meaning from a piece of text. One should not confuse semantic analysis with lexical analysis. Lexical analysis is based on smaller tokens, but on the other side, semantic analysis focuses on larger chunks. In **Natural Language Processing** or NLP, semantic analysis plays a very important role. This article revolves around the syntax-driven semantic analysis in NLP.

**Introduction**

One of the prerequisites of this article is a good knowledge of grammar in NLP.

**Syntax-driven semantic analysis** is the process of assigning representations based on the meaning that depends solely on static knowledge from the lexicon and the grammar. This provides a representation that is **"both context-independent and inference free"**.

Syntax-driven semantic analysis is based on the **principle of composability**.

**Definition:**  
The meaning of a sentence is the sum of the meanings of its constituent elements.

However, this idea should not be interpreted. The meaning of a sentence is not just based on the meaning of the words that make it up but also on the grouping, ordering, and relations among the words in the sentence.

**What is Syntax?**

**Syntax** refers to the set of rules, principles, and processes involving the structure of sentences in a natural language.

The syntactic categories of a natural language are as follows:

* Sentence(S)
* Noun Phrase(NP)
* Determiner(Det)
* Verb Phrase(VP)
* Prepositional Phrase(PP)
* Verb(V)
* Noun(N)

Syntax is how different words, such as Subjects, Verbs, Nouns, Noun Phrases, etc., are sequenced in a sentence.

**Semantic Errors**

A **semantic error** is a text which is grammatically correct but doesn’t make any sense.

Consider the statement in the C language:

int a = 1.25;

1.25 is not an integer literal, and there is no implicit conversion from 1.25 to int, so this statement does not make sense. But it is grammatically correct.

Semantic errors are also called **malapropisms**. This means replacing a word with another existing word similar in letter composition and/or sound but semantically incompatible with the context.

**Attribute Grammar**

**Attribute Grammar** is a special type of context-free grammar. To provide context-sensitive information, some additional information (attributes) is appended to one or more of its non-terminals.

In other words, attribute grammar provides semantics to context-free grammar. Attribute grammar, when viewed as a parse tree, can pass values or information among the nodes of a tree.

**Example**

S-> S + E { S.value = S.value + E.value }

The right part of the CFG contains the semantic rules that signify how the grammar should be interpreted. Here, the values of non-terminals S and E are added together and the result is copied to the non-terminal S.

**Steps in Semantic Representation**

**Mapping of a Parse Tree to Semantic Representation**

**Semantic parsing** is the process of mapping natural language sentences to formal meaning representations. Semantic parsing techniques can be performed on various natural languages as well as task-specific representations of meaning.

Semantic Analysis is related to creating representations for the meaning of linguistic inputs. It deals with how to determine the meaning of the sentence from the meaning of its parts. So, it generates a logical query which is the input of the Database Query Generator.

* The task of the Database Query Generator is to map the elements of the logical query to the corresponding elements of the user databases.
* The query generator uses four routines, each of which manipulates only one specific part of the query.
* The first routine selects the part query that corresponds to the appropriate DML command with the attribute's names (i.e. SELECT \* clause).
* The second routine selects the part of the query that would be mapped to a table's name or a group of tables' names to construct the FROM clause.
* The third routine selects the part of the query that would be mapped to the WHERE clause (condition).
* The fourth routine selects the part of the natural language query that corresponds to the order of displaying the result (ORDER BY clause with the name of the column).

The purpose of this system is to get the correct result from the database. It executes the query on the database and produces the results required by the user.

**Lambda Calculus**

**Lambda calculus** is a notation for describing mathematical functions and programs. It is a mathematical system for studying the interaction of functional abstraction and functional application. It captures some of the essential, common features of a wide variety of programming languages. As it directly supports abstraction, it is a more natural model of universal computation than a Turing machine.

**Syntax**

**A λ-calculus term is:**

* a variable x∈Var, where Var is a countably infinite set of variables;
* an **application**, a function e0 applied to an argument e1, usually written e0e1*e*0*e*1 or e0(e1)*e*0(*e*1), or
* a **lambda abstraction**, an expression λx.e representing a function with input parameter x and body e. Where a mathematician would write x ↦ x2, or an SML programmer would write fnx=>x∗x*fnx*=>*x*∗*x*, in the λ-calculus, we write λx.x2.

**Ambiguity Resolution**

**POS Tagger**

Language cannot elude ambiguity. Not only could a sentence be written in different ways and still convey the same meaning, but even lemmas — a concept that is supposed to be far less ambiguous — can carry different meanings.

Consider the following sentences:

* The batsmen had a good play today.
* I am learning to play the guitar.

The word **"play"** has two completely different meanings in both sentences. The first one is a noun, and the second is a verb. Assigning the correct grammatical label to each token is called PoS (Part of Speech) tagging, and it’s not a piece of cake.

The ambiguity in POS tagging can be resolved using expert.ai.

**Code Implementation:**

Let's import the library first.

from expertai.nlapi.cloud.client import ExpertAiClient

client = ExpertAiClient()

For this code example, we will take two sentences with the same word(lemma) "key".

*# Two sentences in which the same word, "key", has a different grammatical label*

key\_as\_noun = "The key broke in the lock."

key\_as\_adjective = "The key problem was not one of quality but of quantity."

The first "key" is a noun and the second one is an adjective.

To analyze each sentence we need to create a request to NL API: the most important parameters — shown in the code below as well — are the text to analyze, the language, and the analysis we are requesting, represented by the resource parameter.

**Creating the first request from the api:**

*# Requesting for the disambiguation of the first sentence, key\_as\_noun*

*# Notice: the parameter for resource specifies the kind of exploration we want to perform on the documents.*

document = client.specific\_resource\_analysis(

body={"document": {"text": key\_as\_noun}},

params={'language': 'en', 'resource': 'disambiguation'})

Let's see the results:

*# Producing and printing PoS tagging of the first sentence*

*# Notice: to retrieve the textual form of the element, we use a document. content with slicing on element start and end chars*

print(f'Parts of speech for "{key\_as\_noun}"\n')

for tokens in the document.tokens:

print(f'{document.content[token.start:token.end]:{15}}\tPOS: {token.pos}')

**Output:**

Part of speech for "The key broke in the lock."

The POS: DET

key POS: NOUN

broke in POS: VERB

the POS: DET

lock POS: NOUN

. POS: PUNCT

The model is correct! The first sentence's "key" is a noun.

Let's check the next sentence:

*# Requesting for the disambiguation of the second sentence, key\_as\_adj*

document = client.specific\_resource\_analysis(

body={"document": {"text": key\_as\_adjective}},

params={'language': 'en', 'resource': 'disambiguation'})

*# Producing and printing PoS tagging of the first sentence*

*# Notice: to retrieve the textual form of the element we use a document. content with slicing on element start and end chars*

print(f'Part of speech for "{key\_as\_adjective}"\n')

for tokens in the document.tokens:

print(f'{document.content[token.start:token.end]:{15}}\tPOS: {token.pos}')

**Output:-**

Part of speech for "The key problem was not one of quality but of quantity."

The POS: DET

key POS: ADJ

problem POS: NOUN

was POS: AUX

not POS: PART

one POS: NUM

of POS: ADP

quality POS: NOUN

but POS: CCONJ

of POS: ADP

quantity POS: NOUN

. POS: PUNCT

Correct again! This time the lemma "key" is tagged as an adjective.

**WSD Mechanism**

**WSD** stands for **Word Sense Disambiguation**. It is the ability to determine which meaning of the word is activated by the use of the word in a particular context. WSD is used in the problem of resolving semantic ambiguity.

For example, consider the two examples of the distinct sense that exist for the word “bass” −

* I can hear the bass sound.
* He likes to eat grilled bass.

The first "bass" is an adjective and the second one is a noun.

Let's see some of the methods for word sense disambiguation(WSD):

* **Dictionary-based or Knowledge-based Methods:**  
  These methods primarily rely on dictionaries, treasures, and lexical knowledge base. They do not use corpora evidence for disambiguation.
* **Supervised Methods:**  
  In these methods, machine learning models make use of sense-annotated corpora to train. The models assume that the context can provide enough evidence on its own to disambiguate the sense. The context is represented as a set of “features” of the words. It includes information about the surrounding words also. However, the words knowledge and reasoning are deemed unnecessary. Support vector machine and memory-based learning are the most successful supervised learning approaches to WSD.
* **Semi-supervised Methods:**  
  Word sense disambiguation algorithms use semi-supervised learning methods due to a lack of training corpus. Semi-supervised methods use both labeled as well as unlabeled data. These methods require a very small amount of annotated text and a large amount of plain unannotated text. The technique that we use by semisupervised methods is bootstrapping from seed data.
* **Unsupervised methods:**  
  These methods have no labelled data. They assume that similar senses occur in a similar context. they work on the principle that senses can be induced from the text by clustering word occurrences by using some measure of similarity of the context. They are also called word sense induction or discrimination.