University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Natural Language Processing
معالجة لغات طبيعية

Dr. Hiba Basim Alwan
د. هبة باسم علوان

cs.uotechnology.edu.iq

Table of Contents

Experiment One: Number of Characters in a Word

Introduction

Text processing is an important task of computational linguistics and AI. Prolog, a good logic programming language, support different built-in predicates for managing text-based computations. One such task includes identifying the number of characters in a given word, which is essential in tasks like natural language processing and text analysis. This experiment demonstrates how to compute the length of a word in Prolog using a built-in predicate, showcasing Prolog's efficiency in declarative programming.

The Aim of the Experiment

To understand and implement how to find the number of characters in a given word.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to compute the number of characters in a given word.
3. Save the file.
4. Run test queries to check the number of characters in different words.
5. Observe the output to verify that the correct length is returned.

Experiment Two: Number of Words in a String

Introduction

Text processing plays a significant role in NLP and AI. Prolog, a declarative programming language, supports built-in predicates to handle and analyze text. Such an operation is identifying the number of words in a given string. This is useful in different computational linguistics tasks. This experiment illustrates how to compute the number of words in a string using Prolog. Prolog represents strings as lists of characters or atoms. The built-in predicates can be utilized to segment a string into words by identifying delimiters like spaces. By counting the number of elements in the resulting list, we can compute the number of words in the string.

The Aim of the Experiment

To understand and implement how to find the number of words in a given string.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to compute the number of words in a given string.
3. Save the file.
4. Run test queries to check the number of words in different strings.
5. Observe the output to verify that the correct length is returned.

Experiment Three: Dictionary Building

Introduction

A dictionary is an important data structure used in NLP and knowledge representation. Prolog is a declarative language, that permits efficient representation and querying of dictionaries utilizing facts and rules. In Prolog, a dictionary can be represented as a collection of facts where each fact associates a word with its meaning. Queries can be made to retrieve meanings, check for the existence of words, and add new words dynamically. This experiment illustrates how to build and query a simple dictionary in Prolog and how to manage structured knowledge representation and retrieval.

The Aim of the Experiment

To generate a dictionary in Prolog that keeps word-meaning pairs and permits efficient lookup of definitions. The program should correctly return the meaning of words kept in the generated dictionary and permit adding new words in the generated dictionary.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to build and query a dictionary.
3. Save the file.
4. Run test queries to check word meaning.
5. Dynamically add a new word.
6. Check that the new word is added by querying it.

Experiment Four: Context-Free Grammar

Introduction

Context-Free Grammars (CFGs) are a type of formal grammar that includes a set of production rules utilized to create strings in a given language. Prolog, a logic programming language, supports a good instrument known as Definite Clause Grammars (DCGs) to define and process CFGs efficiently. In Prolog, CFGs can be programmed utilizing DCGs, which support a concise way to represent rules and parse input. NLP plays an essential role in AI and computational linguistics. One way to analyze and create sentences computationally is through utilizing CFGs. A CFG consists of:

1. Terminals: The actual words or symbols.
2. Non-terminals: Categories like sentences, noun phrases, verb phrases, etc.
3. Production Rules: Rules defining how non-terminals expand into other non-terminals and terminals.

Example Grammar:

S → NP VP

NP → Det N

VP → V NP

Det → the | a

N → cat | dog

V → chases | sees

The Aim of the Experiment

To understand and implement CFGs utilizing DCGs in Prolog to parse and create sentences.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to write the above CFG rules in it.
3. Save the file.
4. Run test queries by parsing a given sentence.
5. Check the given result.

Experiment Five: Morphology Analysis

Introduction

Morphological analysis is a fundamental task in NLP that includes segmenting words into their morphemes—the smallest meaningful units which are affixes (prefixes, suffixes, and infixes) and roots. This process helps in understanding the structure of words and is broadly utilized in applications like spell-checking, stemming, and machine translation. Prolog, with its pattern-matching and logical inference abilities, supports a suitable platform for implementing morphological analysis. Words can be broken down as follows:

1. **Root Words:** The main part of a word that carries meaning.
2. **Prefixes:** Morphemes added at the beginning of a word (e.g., “un-” in “undo”).
3. **Suffixes:** Morphemes added at the end of a word (e.g., “-ing” in “running”).
4. **Infixes:** Morphemes inserted within a word (less common in English but found in some languages like Germany).

The Aim of the Experiment

To understand and implement morphological analysis using Prolog by segmenting words into their component morphemes and recognizing word structures.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to write the morphology rules in it.
3. Save the file.
4. Run test queries by parsing a given word.
5. Check the given result.

Experiment Seven: Top – Down Parsing

Introduction

Top – down parsing is a fundamental approach in syntactic analysis, mainly utilized in NLP and compiler design. It includes starting from the highest-level grammar rule and recursively expanding non-terminals until the input string is derived. Prolog, with its built-in backtracking and pattern-matching capabilities, is well-suited for implementing top – down parsers utilizing DCGs. Top – down parsing works by:

1. **Starting from the root (start symbol)** of the grammar.
2. **Expanding non-terminals** recursively using production rules.
3. **Matching terminals** against the input sentence.
4. **Backtracking when necessary** to find valid derivations.

The Aim of the Experiment

To understand and implement top – down parsing in Prolog by defining a grammar and parsing given input sentences using recursive descent.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to write the CFG rules in experiment four.
3. Save the file.
4. Run test queries by checking if a given sentence is valid.
5. Check the given result.

Experiment Eight: Bottom – Up Parsing

Introduction

Bottom – up parsing is a fundamental approach in syntactic analysis, commonly utilized in NLP and compiler design. Unlike top – down parsing, bottom – up parsing starts with the input symbols and attempts to build the parse tree from the leaves to the root. Prolog, with its built-in pattern-matching and backtracking capabilities, is well-suited for implementing bottom-up parsers utilizing DCGs. Bottom – up parsing works by:

1. **Starting from the input tokens (terminals).**
2. **Combining adjacent symbols** to form higher-level non-terminals based on production rules.
3. **Continuing reductions** until the start symbol (the root of the grammar) is derived.
4. **Using backtracking** to explore alternative derivations if necessary.

The Aim of the Experiment

To understand and implement bottom – up parsing in Prolog by defining a grammar and parsing given input sentences using shift-reduce techniques.

The Procedure

1. Open a text editor and create a new Prolog file.
2. Write Prolog code to write the CFG rules in experiment four.
3. Save the file.
4. Run test queries by checking if a given sentence is valid.
5. Check the given result.