Spring 5-19-2018

Automatic Document Summarization Using Knowledge Based System

Andrey Timofeyev

AUTOMATIC DOCUMENT SUMMARIZATION USING KNOWLEDGE BASED SYSTEM

by

Andrey Timofeyev, B.S., M.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements of the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

May 2018

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

MARCH 23, 2018
Date

We hereby recommend that the dissertation prepared under our supervision by

Andrey Timofeyev, B.S., M.S.

Entitled **Automatic document summarization using knowledge based system**

be accepted in partial fulfillment of the requirements for the Degree of

Doctor of Philosophy in Computational Analysis and Modeling

Supervisor of Dissertation Research

Head of Department

Department

Recommendation concurred in:

Advisory Committee

Approved:

Director of Graduate Studies

Dean of the College

Approved:

Dean of the Graduate School

GS Form 13
(8/10)

ABSTRACT

This dissertation describes a knowledge-based system to create abstractive summaries of documents by generalizing new concepts, detecting main topics and creating new sentences. The proposed system is built on the Cyc development platform that consists of the world's largest knowledge base and one of the most powerful inference engines. The system is unsupervised and domain independent. Its domain knowledge is provided by the comprehensive ontology of common sense knowledge contained in the Cyc knowledge base. The system described in this dissertation generates coherent and topically related new sentences as a summary for a given document. It uses syntactic structure and semantic features of the given documents to fuse information. It makes use of the knowledge base as a source of domain knowledge. Furthermore, it uses the reasoning engine to generalize novel information.

The proposed system consists of three main parts: knowledge acquisition, knowledge discovery, and knowledge representation. Knowledge acquisition derives syntactic structure of each sentence in the document and maps words and their syntactic relationships into Cyc knowledge base. Knowledge discovery abstracts novel concepts, not explicitly mentioned in the document by exploring the ontology of mapped concepts and derives main topics described in the document by clustering the concepts. Knowledge representation creates new English sentences to summarize main concepts and their relationships. The syntactic structure of the newly created sentences is extended beyond simple subject-predicate-object triplets by incorporating adjective and adverb modifiers. This structure allows the system to create sentences that are more complex. The proposed system was implemented and tested. Test results show that the system is capable of creating new sentences that include abstracted concepts not mentioned in the original document and is capable of combining information from different parts of the document text to compose a summary.

APPROVAL FOR SCHOLARLY DISSEMINATION

DEDICATION

This dissertation is dedicated to my beloved wife and my family. Without their love and unconditional support, this dissertation would not see the light.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my academic advisor Dr. Ben Choi and my committee members Dr. Pradeep Chowriappa, Dr. Weizhong Dai, Dr. Sumeet Dua, and Dr. Galen Turner. In addition, I would like to thank all of my Tech friends and colleagues for their encouragement and helpful advice.

CHAPTER 1

INTRODUCTION

Problems with information overload have drawn attention because of the exponential growth of information creation and distribution that has recently gained an incredible pace. Ninety percent of the entire world's recorded data has been generated in the past few years with two and a half million terabytes of data being created daily [1]. Around eighty percent of the data is unstructured and represented in the form of documents, web pages, images, and videos. This vast amount of data turns into a distraction and has a negative impact on human productivity and decision-making [2]. It is becoming harder for the public to navigate and comprehend information conveniently [3]. The issue of information overload raises a number of important questions – how to make this overwhelming amount of information accessible for users; how to find necessary information and to filter out the useless ones; and how to absorb and employ information effectively.

Information overload is very complex, and currently there is no known solution that can solve it all together, yet a number of approaches exist that try to address some of the issues. One of such approaches is text summarization. It aims to mitigate information overload specifically in the domain of unstructured data. Summarization process condenses text in a form of a summary while preserving the most important information, which ensures its high relevance. This drastically reduces the amount of information people would have to comprehend, thus decreasing the amount of time and effort spent on finding relevant information. Automatic text summarization is part of a broader field of natural language processing that combines advances in computer science, artificial intelligence and computational linguistics [4].

Automatic text summarization can be divided into two main approaches – extractive and abstractive. Extractive approach algorithms form a summary by choosing the most significant words, phrases or sentences in the text. Summaries created by such approach are highly relevant to the original text, but do not convey novel information. Extractive text summarization is a well-studied topic that has reached its potential [5]. Abstractive approach algorithms, in contrast, aim to create new phrases or sentences by analyzing the semantics of the text to form a summary. Such algorithms perform a synthesis of source text to derive knowledge that is more general. This branch of automatic text summarization is less studied and more complex. In order to create abstractive summary of a text, the algorithm has to obtain novel knowledge form original text and meaningfully combine information from different parts [6]. Summaries created by abstractive approach algorithms are more favorable, but inherently harder to achieve. The algorithm must use background knowledge of the subject matter to abstract new information. It must perform deep syntactic analysis of the input text to be capable of combining information from different parts appropriately. It must also use advances of natural language generation process to represent newly created knowledge in a way that is suitable for users to comprehend.

This dissertation provides the description of an abstractive text summarization algorithm that:

- Derives deep syntactic structure of the text;
- Generalizes new concepts based on the information derived from the text;
- Automatically discovers general topics described in the text;
- Identifies most informative subjects based on discovered topics;
- Creates new sentences for identified subjects combining information from different parts of the text to compose a summary.

Described algorithm uses Cyc development platform as a source of background knowledge. Cyc development platform consists of the world's largest ontology of commonsense knowledge and a reasoning engine [7]. Cyc ontology serves as a backbone for semantic analysis, knowledge generalization and natural language generation functionality of the algorithm. Deep syntactic analysis is performed by using capabilities of advanced natural language processing techniques. Combining both semantic knowledge and syntactic structure allows the algorithm to have domain knowledge of the subject matter and utilize relationships between words within given sentences. The following is the Knowledge Based System (KBS) algorithm, the details of which will be fleshed out in Chapters 3 and 4.

The KBS algorithm is composed of three main processes: knowledge acquisition, knowledge discovery, and knowledge representation. Knowledge acquisition process receives documents as an input and transforms them into syntactic representation. Then, it maps each word in the text to an appropriate Cyc concept and assigns the word's weight and the word's relationships to that concept. Knowledge discovery process finds the ancestor for each mapped Cyc concept, records ancestor-descendant relationships, and adds scaled descendant weight and descendant relationships to the ancestor concept. This process allows the algorithm to abstract novel concepts that are not mentioned directly in the original text. Then, the process identifies the main topics described in the text by clustering the mapped Cyc concepts. The knowledge representation process creates sentences in English for the most informative subjects identified in the main topics. This process allows the summary sentences to be composed by using the information from different parts of the text while preserving their coherence to the main topics. The workflow diagram of the algorithm is outlined in **Figure 1-1**.

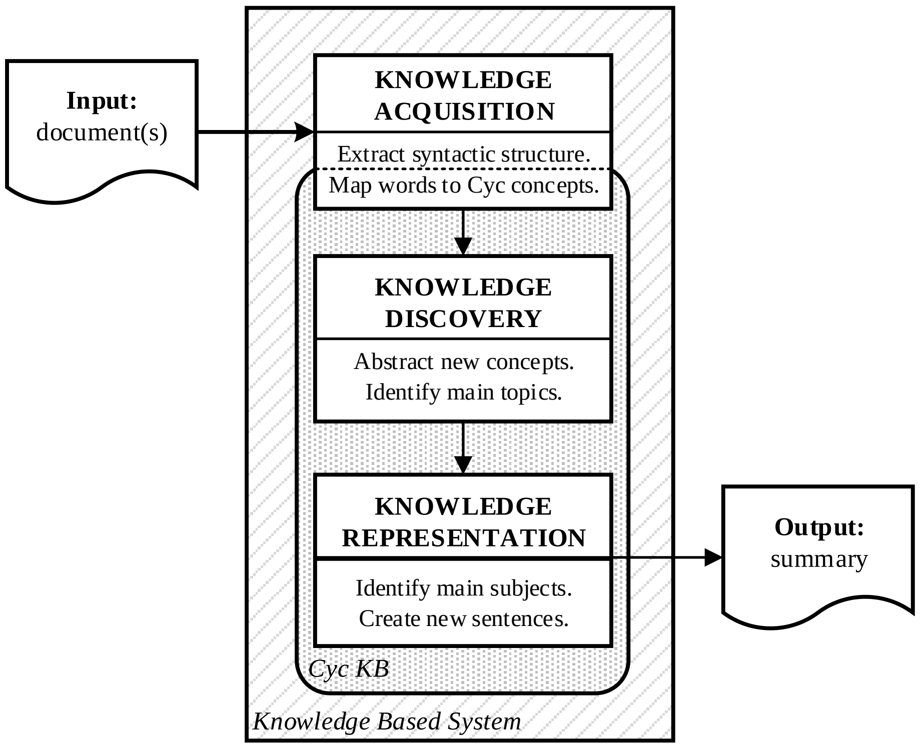


Figure 1-1: KBS algorithm workflow diagram.

An automated modular framework has been implemented to test the functionality of the proposed algorithm. Two sets of test experiments were conducted: first using synthetically created data and second using various documents and encyclopedia articles. Test results demonstrate that the algorithm is capable of generalizing concepts that are not mentioned explicitly in the original text, deriving general topics of the text and creating new sentences that combined information from different parts of the text to form an abstractive summary.

Main contributions of proposed algorithm are outlined as follows:

- We introduce a method to derive the main topics automatically and identify the most significant subjects based on the concepts clustering and syntactic structure of the text;
- We propose new sentence creation technique using semantic analysis and natural language generation capabilities of Cyc development platform. Proposed technique enhances the structure of newly created sentences by adding adjective and adverb modifiers to subject-predicate-object triplets;
- We propose a mechanism of combining information from different parts of the text to form a summary based on deep syntactic analysis of the text.

Proposed KBS algorithm falls into the intersection of text data mining, natural language processing and artificial intelligence domains. It gathers and analyzes text data, extracts deep syntactic structures of the text and generates new sentences as a summary. It utilizes Cyc development platform – world’s longest-lived artificial intelligence platform [7], as a backbone for the semantic reasoning.

The rest of the dissertation is organized as follows. Chapter 2 outlines previous work in the field of automatic text summarization and gives background of knowledge-based systems and advanced natural language processing techniques. The chapter provides the description of extractive and abstractive approaches, highlighting recent advances and gives an overview of Cyc development platform, its knowledge base and inference engine. Chapter 3 thoroughly describes the methodology of the proposed KBS algorithm. This chapter provides details of the knowledge acquisition, knowledge discovery and knowledge representation processes. Chapter 4 presents details of the implementation of the summarization system based on the proposed KBS algorithm. Chapter 5 discusses the results obtained by applying the implemented system to synthetically generated data and encyclopedia articles. Finally, Chapter 6 concludes the dissertation and provides discussion of directions for the future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we outline related work undertaken in the field of automatic text summarization. In addition, we provide an overview of the knowledge-based systems employed in the area, and give the background of the advanced natural language processing techniques used.

2.1 Automatic text summarization

Computational community has been studying automatic text summarization problem since the late 1950s. In literature, automatic text summarization is traditionally divided into two main areas, namely extractive and abstractive. The approaches in these two areas differ fundamentally by the way they compose the summary of the text.

Extractive methods create a summary by selecting the most informative phrases or sentences from the original text and filtering out those that do not convey useful information. Such methods generally vary by the different intermediate representations of the candidate phrases or sentences and different sentence scoring schemes [8]. The advantage of the extractive approach is that it does not require much semantic knowledge or deep syntactic analysis of the text because it is solely based on the statistics of word or phrase occurrences in the text. Summaries created by the extractive approach methods exhibit higher statistical correlation with the original text, which makes their performance easier to evaluate.

In contrast with the extractive approach, abstractive methods aim to create new sentences that carry novel knowledge or abstraction, not mentioned in the original text. Such methods involve generalization and aggregation of the information based on the content of the given text. New sentences are composed using natural language generation techniques by fusing the information that belongs to the same concept from different parts of the text. Summaries created by the abstractive approach methods tend to be more desirable because they have a higher correlation with the human expert created summaries [6]. At the same time, such summaries are harder to evaluate quantitatively since most of the metrics are based on the statistics that measure an overlap between the summary sentences and the sentences from the original text. Utilization of such metrics to evaluate the abstractive approach methods is impractical, since the main aim of the abstractive summarization is to deduce new information that was not explicitly mentioned in the original text.

2.1.1 Extractive approach methods for text summarization

In this subsection, we cover the most prominent methods used in extractive summarization. We progress through different intermediate representations of the features used by the methods, starting with a simple word frequency count based methods and progressing to more sophisticated graph representation of the text and machine learning applications.

2.1.1.1 *Frequency-driven approaches*

Methods based on the frequency counts are the simplest, oldest and most widely used in the area of extractive text summarization. These methods select the most representative sentences that contain significant words. The significance of the words is evaluated by the various frequency measures.

The first paper in the field of text summarization that was published in the late 1950s described the method based on raw frequency as a measure. The author concluded, however, that the raw frequency measure is not the best indicator, since some words could be frequent in many documents [9]. To take into account the length of the text to be summarized, word probability measure is introduced as an improvement on raw frequency counts [10], [11]. Another major improvement in frequency-based approach methods is the TF-IDF measure that is calculated by the product of term frequency (TF) and inverse document frequency (IDF) measures.

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D) \qquad \textbf{Eq. 2-1}$$

This measure was adopted from information retrieval domain. It favors the terms that are very frequent among a small number of documents in the corpus. In **Eq. 2-1**, *t* denotes the term, *d* denotes each document in the corpus, and *D* denotes the collection of all documents in the corpus. Selecting the sentences that contain terms with high TF-IDF score yields better extractive summaries [12], [13], [14]. A variation of TF-IDF score that uses the log-likelihood ratio test is introduced to identify topic signatures. Topic signature is the set of words that describes similar concept. The idea of this measure is similar to the TF-IDF in terms that it gives a higher score to the words frequently used in the input text and rare in the other texts, but it also provides a cutoff to include the words into topic signatures [15]. In the methods that use topic signature measure, the sentences are included in the summary by their significance that is computed by the number of topic signature words contained in the sentence [16], [17].

2.1.1.2 *Graph models for sentence importance*

Graph representation of the text aided the automatic text summarization area in many different ways. The main idea of such methods is to model a text as a graph, where the nodes are words, phrases, sentences or paragraphs, and the edges are weights that represent the similarity measure between text elements. Graph representation of an arbitrary text is illustrated in **Figure 2-1**. Informative sentences for the summary are selected based on the edges' weights by using graph traversal algorithms, such as the breadth-first search and the depth-first search.

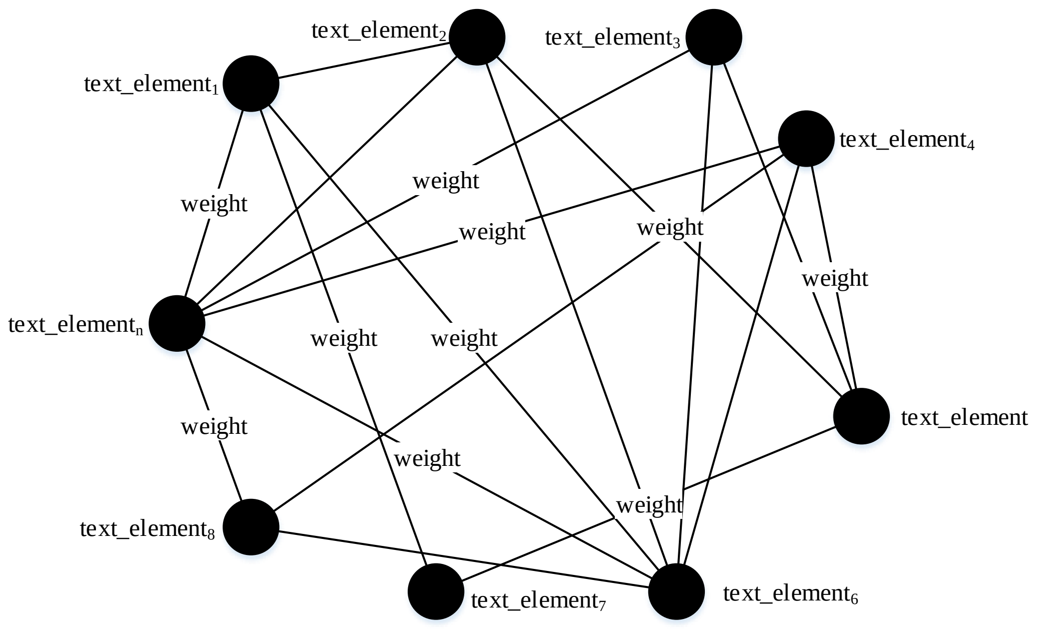


Figure 2-1: Graph representation of an arbitrary text.

TextRank approach proposed by [18] models input text as a graph, where nodes are represented as the words, phrases or sentences depending on the desired application. Edges between the nodes are expressed as a similarity measure weight based on the semantical or lexical relationships between the text elements or their contextual overlap. Nodes with the highest similarity weight are picked to form the final summary of the input text. The idea of graph ranking is exploited by [19] in the LexRank graph-based summarization approach. Their proposed method represents a document cluster as a graph where sentences are used as vertices, and the edges are defined as a degree of similarity between sentences. Summary of the text is then composed by the sentences that are chosen based on the number of links incident upon a node in the graph. Authors define sentence centrality in terms of similarity to other sentences. The sentences that are similar to many other sentences have higher centrality.

The idea of representing the document as a semantic graph is proposed by [20]. In the semantic graph text representation, nodes are modeled as noun phrases or verb phrases, and the edges connecting them are derived based on the syntactic relations analysis of the text elements. The authors trained Support Vector Machines (SVM) learning method on the described graph representation of the text using sets of various attributes, such as linguistics attributes, graph and document structure, to identify summary nodes and use them for extracting sentences that form a summary of the text. An affinity graph representation of the text is introduced by [21]. Affinity graph representation of the text expressed the semantic relations between sentences in terms of their content similarity. Candidate sentences for a summary are evaluated by two factors

– information richness and information novelty. These properties are computed based on the number of the informative neighbors the sentence is linked to.

2.1.1.3 *Machine learning and statistical applications*

Statistical methods and machine learning techniques showed great potential in scoring the candidate's sentences that are to be extracted to form a summary. The extractive approach methods based on such techniques improve state-of-the-art performance for the variety of tasks in the domain of text summarization. Majority of the methods utilize the idea of training a model using various sentence features to find most appropriate sentences for the extraction.

Sentence selection is approached as a simple classification problem in [22]. Their model, based on the Naïve Bayes classifier, estimates the probability of a given sentence to be included in the summary. The model is trained on the number of sentence features such as thematic words, fixed phrases' and proper names' inclusiveness, sentence length and sentence position in the paragraph. A similar set of features with a little variation is used in [23]. The authors propose to use the Hidden Markov model classification instead of the Naïve Bayes classifier, since some of the features used to train the model are violating the assumption of independence. Furthermore, they introduce the assumption that the probability of including the next sentence into the summary depends on the inclusion of the current sentence.

Another proposed method for the task of choosing sentences for summarization is to treat it as a sequence-labeling problem [24]. The objective of the summarization task is to label sentences as those that will be included in the summary and those that will not. The authors proposed the solution to this sequence-labeling problem by applying the conditional random field (CRF) method, which was state-of-the-art in sequence labelling at that time. Their approach also takes into consideration the sentence inclusion dependency. When a new sentence is added to the summary, one or more already chosen sentences might be deleted based on the calculated probability values. The sentence feature space used in this method is extended by more complex features like similarity of the sentence to its neighboring sentences, latent semantic analysis score and hyper-induced topic scores.

SVM classification methods showed promising results when applied to the sentence ranking problem for automatic text summarization. Methods based on the SVM use different set of sentence features to extract the most informative sentences to form a summary. Wide range of the semantic and the syntactic sentence features are used in a method proposed by [25]. Authors trained Mapping-Convergence (MC) version of the One-Class Support Vector Machine (OCSVM) classifier using following features: the position of the sentence in the document; the total number of sentences in the document, the total number of named entities found in the sentence; probabilities of the informative words contained in the sentence, the existence of discourse markers and the existence of particular words. Top ranked sentences extracted by a trained classifier are also checked for redundancy before being included into the final summary. One of the drawbacks of such supervised classification method is the need of large amount of labeled data for training, which is usually not feasible to obtain in the domain of automatic text summarization.

In order to address the lack of labeled data the semi-supervised SVM classification approach is proposed by [26]. The authors co-train SVM classifier on both labeled and unlabeled data combining various sentence features. Their semi-supervised method shows compatible performance while saving the time cost on labeling the data. The authors propose four different groups of sentence features: surface, content, event and relevance. The surface features consist of sentence position in the text and the length of the sentence. The content features measure the quantity of the indicative words, such as centroid words, signature terms and high frequency words. The event features are based on “person”, “location”, “organization” and “date” named entities contained in the text. Finally, the relevance features measure sentence relationships to other sentences in the text. The authors describe a co-training mechanism using the Probabilistic Support Vector Machine (PSVM) method for supervised training and the Naïve Bayes classification for semi-supervised training utilizing derived sentence features. The summary is then composed of the sentences extracted by the described co-training approach. The final order of the sentences is conditioned on the sentence length and its position in the text.

2.1.1.4 *Shallow semantic analysis methods*

Since statistical analysis is not capable of discovering the meaning of the words, and performing deep semantic analysis has high computational cost, the number of methods were proposed that leveraged parts of both approaches. Such methods are categorized as the shallow semantic analysis methods. Most prominent techniques used the idea of the lexical chains – sequences of related words; the concept lattice – document representation using concepts semantically linked to each other; and the Latent Semantic Analysis (LSA) – the process of clustering related words and sentences based on their semantics.

The idea of the lexical chains – representation of lexical cohesive structure of the text expressed by the sequence of related words, was first applied to the problem of automatic text summarization by [27]. The authors proposed the method for text summarization that does not require computing the full semantic representation of the text, but rather extracts significant sentences based on the strong lexical chains constructed for the input text. The summarization process starts with composing a set of candidate lexical chains. The construction process first selects a set of candidate words, then finds an appropriate chain for each word based on the similarity measure derived from the WordNet thesaurus and then updates the chain accordingly. After the set of candidate lexical chains is constructed, the strongest among them are selected by the ranking mechanism based on the scoring function. Finally, the significant sentences are extracted based on the distribution of the strongest lexical chains.

The idea of using lexical chains for the summarization task was later exploited by [28]. The authors propose improvements to the lexical chain construction process and a method to evaluate lexical chains as an intermediate representation of the input text. Their described approach uses scoring system based on the analysis of words relationships to assess the contribution of a candidate element to the chain. To evaluate if the lexical chains are a good representation of the text to use for the summarization task, the authors analyzed manually created summaries for the exclusiveness of words from the lexical chains. The results of the study shows great potential of the utilization of the lexical chains as a form of shallow semantic representation of the text as opposed to the single words and phrases frequencies.

Another type of shallow semantic representation of the text is a document concept lattice that is introduced by [29]. The concept lattice models the information contained in the text using the idea of linked concepts that cover the main facts and topics of the text. Such concepts are represented by the words that describe concrete or abstract entities together with their behavior. The process of concept lattice construction starts with the analysis of the input sentences parse trees to identify repeated concepts. Then the maximal common concepts are determined according to the concepts' frequency. The hierarchical representation of the concepts is then formed to serve as a structure for the document concept lattice. Final summary of the text is then composed by extracting an optimal set of the sentences by utilizing the derived document concept lattice representation as a basis. The advantage of the concept lattice representation method is in selecting the sentences that covered as many concepts as possible with the least amount of words.

Latent semantic analysis (LSA) is another shallow semantic analysis technique applied to the problem of identifying candidate sentences to be extracted from a given text to form a summary. LSA performs the singular value decomposition of the term by sentence matrix representation of the text to discover words or phrases that describe similar topic. This approach is driven by the assumption that the words that describe the same topics will generally appear in a similar context and will be mapped near to each other in the decomposed matrix. Such a decomposition allows to semantically group terms or sentences operating solely on the words or phrases frequencies. Text summarization method based on the shallow semantic representation of the text derived by LSA is described by [30] and [31]. In their proposed summarization methods, the input text is first decomposed into a term by sentence matrix representation based on various term frequency measures. Then the singular values decomposition technique is applied on the matrix to discover vector representation of the salient topics contained in the text. Finally, the sentences are extracted to form a summary based on the various vector relations between the sentence vector representation and the topic vector representation. Applying LSA method for text summarization allows extracting the sentences that are semantically related to the main topics of the text without performing the costly deep semantic analysis.

2.1.1.5 *Conclusion*

The described extractive text summarization methods suffer with the major drawback of inability to synthesize new information, being limited to the words and phrases comprised in the original text. The summaries produced by such methods tend to have high statistical correlation with the input documents, but do not convey any novel information.

2.1.2 Abstractive approach methods for text summarization

Abstractive text summarization methods are more desirable because they resemble the summarization process that the human experts undergo when they create the summaries, but such methods are inherently hard to develop and evaluate. Most of the methods in the area involve transforming the text into a graph representation, where the nodes denote text elements and the edges represent various relationships between these text elements. The final summary of the text is constructed by applying the graph transformation techniques, such as graph reduction, merging and compression.

2.1.2.1 *Graph reduction based methods*

The application of word graphs text representation for the purpose of the abstractive text summarization was investigated by [32] and illustrated by their multi-sentence compression algorithm. The algorithm is applied on a cluster of similar sentences to compose a single sentence as a summary. The algorithm starts by creating a word graph representation of a cluster using all words in the sentences. Such a graph is constructed iteratively by adding one sentence at a time. The nodes in the graph represent words, and the edges represent adjacency relation between words – carrying a weight, which expresses the frequency of the syntactic relation of the words. After the word graph representation of a cluster is built, the algorithm identifies the best path in the graph to assure high compression and informativeness. The best path is evaluated based on presence of the strong links and such a path has to follow through, what they refer to as the salient nodes [32]. Both of these criteria are identified by experimenting with the various weighting formulas. The path that has the lightest average edge weight is chosen as the summary sentence for the cluster of the input sentences.

The application of words graphs was extended to cover the whole document rather than a small cluster of sentences in [33]. The authors propose document-level representation of the text using the word graphs. Their method employs Dijkstra's algorithm to find the shortest path in the graph to accommodate for the sentence compression and to retain informative parts of the text. The algorithm that they describe generates a number of the candidate summary sentences and the final summary of the whole document is composed by choosing the most important ones, according to the heuristic rules. Methods based on the word graphs representation are capable of effectively combining information from different sentences, but lack the ability to produce novel information, not explicitly mentioned in the text.

Abstractive text summarization by the semantic graph text reduction technique was proposed by [34]. The authors introduce the idea of the rich semantic graph text representation, and enhancing graph nodes with the associative attributes derived from domain ontology. In the described graph, the nodes represent the verbs and nouns, and the edges represent the semantic and topological relationships among words. Such a rich semantic graph is constructed for the input document utilizing deep syntactic analysis. Initially, the sub-graphs are created for each sentence in the document and then merged together to derive a rich semantic graph of the whole document. On the next step, the graph is reduced according to the set of the heuristic rules. During the process, the nodes of the graph are combined, replaced or removed based on the additional semantic relationships derived from the WordNet thesaurus. Finally, the summary of the document is created from the reduced rich semantic graph using domain ontology. The method proposed by the authors uses the WordNet system to create a set of sentences with the synonyms of the words from the original document. The sentences to be included in the final summary are picked based on the frequency of the used words and the sentence discourse relations.

2.1.2.2 *Graph merging based methods*

Creating an abstractive summary of the text involves composing new sentences that combine the information from different parts of the text. The new sentence creation approach by the phrase selection and merging was proposed by [35]. The authors argue that using more fine-grained syntactic units such as the noun and verb phrases improves the process of the new sentence creation. Their described algorithm starts by extracting noun and verb phrases from each sentence dependency tree, and forming a set of the concepts and facts described in the input text. Then the salience score is calculated for each extracted phrase. This score incorporates the concept-based weight and the position-based frequency of the phrases. Next, new sentences are generated by identifying the most informative phrases and merging them while maximizing the salience and satisfying the predefined construction constraints. The structure of the composed sentences is based on the heuristic rules and the relations derived from the dependency trees, and follows the summarization requirements, such as the sentence length constrains, the avoidance of the redundancy and the utilization of the pronoun phrases. Finally, some of the post-processing steps are carried out to improve the order of the elements in the sentence and enhance the sentence readability.

The analysis of the discourse structure of the input text shows promising results in the area of abstractive summarization as reported by [36]. They propose an algorithm that creates a summary by using the discourse tree structure as an intermediate representation of a text. Such a representation illustrates how the text spans are connected and related to each other. The discourse trees of each sentence in the text are used to compose a directed graph that allows multiple connections between the two nodes. Such a graph is called the aspect rhetorical relation graph (ARRG). The nodes of ARRG represent the concepts derived from the text, and the edges represent specific relations between them, together with an importance weight. Their proposed algorithm starts the summarization process by extracting the sub-graphs containing the most informative concepts from the ARRG using the weighted page rank algorithm. Then the extracted sub-graphs are combined into the aspect hierarchical trees to be used by the abstract generation process implemented by natural language generation techniques such as the microplanning and the sentence realization.

Another type of graph text representation, namely Abstractive Meaning Representation (AMR), was applied to the problem of summarization by [37]. The AMR provides a semantic representation of each sentence in the text as the rooted, acyclic, directed graph. Their proposed approach performs the graph transformation that compresses the source graph into a summary graph and creates an abstractive summary based on it. The summarization process starts by transforming each sentence into AMR graph using the statistical semantic parser. Then the created graphs are merged and transformed into a single AMR graph that represents the whole document. This process involves pruning of the certain fragments of the graph and combining the parts of the graph that has the same labels. While merging subgraphs represent different sentences, every concept that is a root concept in the sentence graph is connected to new “ROOT” node to assure the connectedness of the final graph. Finally, additional edges are added to create a dense graph representation of the document. Such a representation is used to select the subset to represent a summary graph that is concise, contains important information and allows creating meaningful sentences. The final summary subgraph is selected by the integer linear programming technique. Since there is no automatic process to create natural language sentences from the AMR graphs, the authors propose a set of the heuristic rules to create the text from the final graph.

The sentence enhancement technique applied to the graph representation of the text to perform abstractive summarization was proposed by [38]. The novelty and advantage of the described approach is in allowing the conjunction of the syntactic dependency trees from any sentence of the input text. The event co-reference resolution algorithm controls correctness of such trees combination by using the distributional semantics approach. The summarization process is implemented in several steps. Initially, the algorithm finds the clusters of compatible sentences, ranks the clusters based on their salience, and picks the top ranked cluster to represent the core. Next, the algorithm composes sentence graph by merging similar vertices based on their syntactic features and the external information derived from the WordNet thesaurus. Then, the sentence graph is extended by adding the dependency trees of the sentences that were not the part of the core cluster, but still had been expressed by the similar features. Such an expanded sentence graph is pruned according to the defined heuristics. Finally, the summary dependency tree is extracted from the sentence graph by the integer linear programming techniques with the constraints for the salience, importance, grammatical correctness and length characteristics. The summary dependency tree is transformed into a final sequence of words with the help of the linearization technique.

2.1.2.3 *Conclusion*

Abstractive text summarization methods described above attempt to derive the latent semantic structure of the given text by transforming it into the graph representation and preserving various relationships among the text elements. While such techniques allow obtaining the shallow semantic features of the text and combining the information from different sentences, they lack the ability to generalize novel information that has not been mentioned in the input text, and only merge the information from the compatible sentences.

2.2 Knowledge based systems

A knowledge-based system (KBS) is a computer system that utilizes a combination of the data, information, and knowledge to allow solving complex problems with domain expertise capabilities. Such systems use artificial intelligence techniques in an attempt to understand the information related to the problem to provide a decision supported by the underlying knowledge. Regular information systems operate on data, but KBS exploit the knowledge contained in the information [39]. KBS generally consist of three main parts: a knowledge base for information storage and organization; an inference engine for the reasoning about the information stored in the knowledge base; and the user interface to allow system-user communication. Knowledge base (KB) resembles the idea of an intelligent database. Information is stored in the KB in an ontological form that grants performing the reasoning and deduction. Inference engine (IE) goes beyond simple search engine abilities by deducting new knowledge and utilizing existing information for the effective problem solving. IE can reason with the subjective fuzzy knowledge together with the explicit facts of established theories that resemble the human experts approach for the problem solving [40]. User interface allows users to communicate with KBS by providing access to the information contained in the knowledge base and to the capabilities of the inference engine.

The ability to derive underlined semantics and to reason about the knowledge comprised in the text are the crucial parts of the effective abstractive summarization algorithm. These factors distinguish the abstractive approaches from the extractive approaches in the area of text summarization. Achieving pure abstractive summary requires the algorithm to combine text from different parts of the input document to abstract and synthesize new knowledge based on the information contained in the document, and to utilize the common sense knowledge to compose the new sentences that represent the summary. Such a functionality is not feasible without taking the advantage of capabilities provided by the knowledge-based systems. Researchers attempting to tackle abstractive summarization problem used various knowledge based systems with WordNet, BabelNet, ConceptNet, and Cyc among the most noticeable.

2.2.1 WordNet lexical database

WordNet is a thesaurus that was developed with an aim to organize the lexical knowledge with regards of the word semantics, rather than the word forms. This is achieved by introducing the mappings between the word meaning and the word character representation. The vocabulary in WordNet is divided into four categories that correspond with the English language parts of speech: nouns, verbs, adjectives and adverbs. The nouns are organized as the topical hierarchies, the verbs represent various relationships, and the adjectives and adverbs serve as the modifiers for the nouns and verbs. The central idea of the semantic representation in WordNet is the grouping of words into synonym sets, known as “synsets”. The semantic relations are then defined as the pointers between different “synsets”.

There are four main categories of pointers between “synsets”: synonymy, antonymy, hyponymy, and meronymy. Synonymy and antonymy pointers form lexical relations between word forms, hyponymy and meronymy define semantic relations between word meanings. The latter two represent relations of a form “is-a” and “has-a” that are allowed to represent knowledge in a hierarchical form [41]. WordNet thesaurus showed promising potential in the area of abstractive text summarization providing a resource to enhance the algorithms with the semantic knowledge. However, the lack of the commonsense knowledge and the ability to reason about it is a major drawback of WordNet thesaurus to be widely applicable in the area of abstractive text summarization problems.

2.2.2 BabelNet encyclopedic dictionary

BabelNet is an encyclopedic dictionary that was created as an attempt to enhance WordNet thesaurus with the information from Wikipedia, a multilingual encyclopedic knowledge repository. The project resulted in multilingual semantic network providing the concepts and named entities connected by the numerous semantic relations. In BabelNet, the knowledge is encoded as a graph where the vertices are the concepts derived from Wikipedia and the edges are the semantic relations derived from WordNet. Such a network is populated automatically by retrieving the semantic information, such as the word senses and the semantic pointers from WordNet, and then merging it with the encyclopedic entries from Wikipedia pages. The linkage between the content to be merged is established by disambiguating the context in both Wikipedia pages and WordNet senses, and computing the conditional probabilities of the candidate contexts. The main advantage of BabelNet semantic network is adding more lexical structure to the encyclopedic knowledge by linking the information repository with the organized computation lexicon [42]. Although BabelNet enhanced WordNet with the world knowledge, it still lacked the commonsense reasoning capabilities that are crucial in the abstractive summarization domain.

2.2.3 ConceptNet semantic network

ConceptNet is a commonsense knowledgebase with the natural language processing capabilities. Inspired by the structure of WordNet knowledgebase, ConceptNet was developed with an aim to capture the content of a general world knowledge in a way that is more suitable for the natural language processing purposes. The main advantage of ConceptNet knowledgebase is in its emphasis on the contextual reasoning. The knowledgebase stores the information as a graph focusing on the semantically rich relationships represented as the edges and the complex concepts represented as the vertices. Such a graph is generated automatically by connecting over a million facts into a semantic network of three hundred thousand nodes.

The corpus of the English sentences from the Open Mind Common Sense project is taken as a basis for the semantic knowledge. The idea of WordNet graph knowledge representation is extended by the several enhancements. Vertices of ConceptNet semantic knowledge graph consist of the compound concepts, such as verb phrases rather than the atomic words. The edges in such a graph represent a wider variety of the semantic relationships between the concepts, including causality, affect, event hierarchy and location. Finally, the knowledge represented in ConceptNet is more casual, informal and applicable [43]. Although the aforementioned enhancements allow ConceptNet knowledgebase to be used for the applied reasoning over the raw text data, the amount of the knowledge captured and the types of the relationships between the concepts appear to be a major drawback when creating purely abstractive and domain independent summarization algorithm.

2.2.4 Cyc development platform

Cyc project started in the mid-1980s with an ambitious goal of encoding the commonsense knowledge of the whole world in the way that a computer can understand and be able to reason. To this date, Cyc contains more than 600,000 concepts, around 40,000 relationships connecting these concepts, and more than 7,000,000 of assertions about these concepts. The volume of the information captured in Cyc makes it the world's largest knowledge based system. The knowledge inside Cyc development platform is organized in a form of an ontology, and the powerful inference engine is provided to perform reasoning based on the knowledge. In order to formalize such an enormous amount of knowledge and ensure the machine readability and inference, the knowledge base is implemented in the CycL – flexible knowledge representation language. CycL syntax is a combination of the features from the first-order predicate calculus and Lisp high-level programming language. High expressiveness of CycL language allows the inference engine to perform the effective reasoning about the knowledge.

2.2.4.1 *Cyc knowledge base*

Cyc knowledge base arranges enormous volumes of common sense knowledge about the world such as the facts, rules of thumbs, concepts, and their interconnections, into a hierarchy that forms the knowledge ontology. The organization of the knowledge in Cyc ontology is illustrated in **Figure 2-2** [44]. The ontology can be viewed as a pyramid, where each layer is arranged by the level of the knowledge generalization. Elements of the ontology are connected by the generalization relationships of specialization or instantiation. Therefore, the knowledge can be propagated bottom-up by the specialization relation type or top-down by the instantiation relation type.

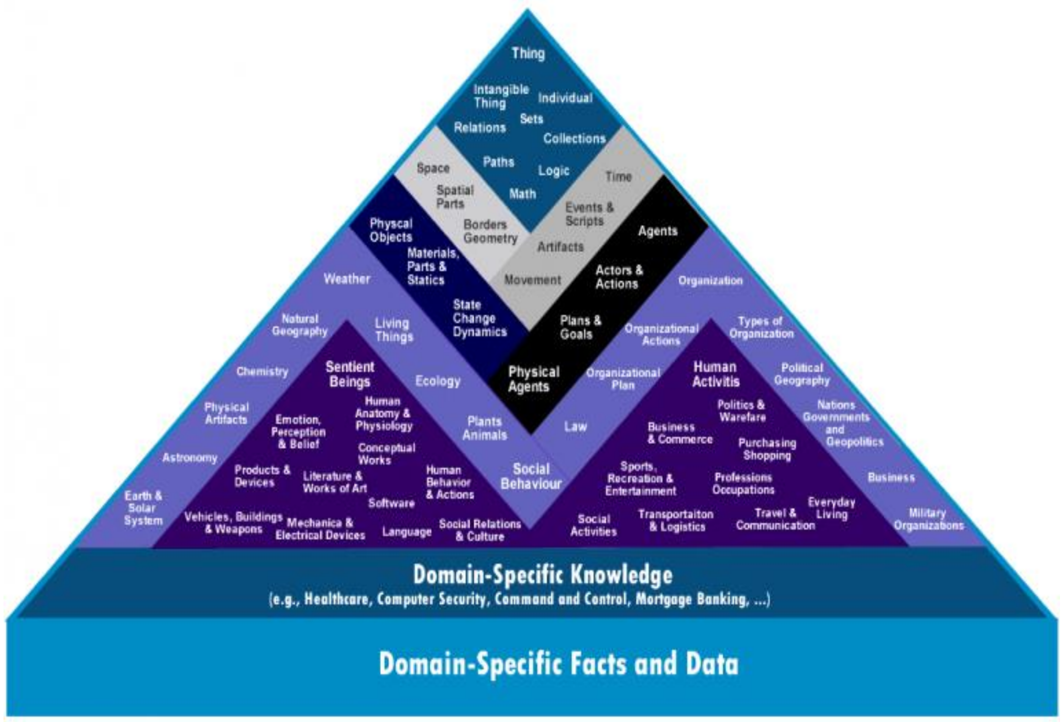


Figure 2-2: Cyc knowledge organization.

The peak of the pyramid constitutes the upper ontology that contains abstract concepts such as an idea of the event, individual, collection, temporal thing. Upper ontology also describes the relations between general concepts. At the very top of the upper ontology resides the most fundamental representation called A “Thing”. Every element in the knowledge base is an instance of the “Thing”. The next layer of the ontology is composed by the core theories that describe the space, time and causality relations. The rules described in the core theories build the fundament for the reasoning ability of the inference engine. The next layer is devoted to the domain-specific theories that cover the information about the broad number of diverse domains from banking and finance to healthcare and chemistry. This knowledge gives an inference engine the ability to perform the reasoning about the very specific domains of interest. The bottom layer of the pyramid consists of the domain-specific facts and data. This layer describes the specific ground level facts about the particular individuals or events and does not cover any theories.

The knowledge, represented in the ontology, is divided into large number of collections of assertions called the micro theories. The assertions are split into the micro theories based on the shared topics, assumptions or sources. Some of the micro theories characterize certain domain of knowledge when others contain information about the certain period in history or describe certain geographical regions. Every assertion must fall into at least one micro theory. The main function of the micro theories is to maintain the local consistency of knowledge. Theories and facts may be contradictory across the micro theories, but within a single micro theory, the assertions must be mutually consistent. Such constraints allow the inference engine to perform the reasoning about the knowledge more efficiently in narrowing down the scope of the facts and rules to a particular micro theory of interest. Micro theories are also organized in a form of a hierarchy linked by the generalization relations. The most general micro theory is called "BaseKB" which holds the basic rules that describe the behavior of all micro theories.

2.2.4.2 *Cyc inference engine*

Cyc development platform allows performing the deductive reasoning about the vast amount of knowledge it comprises with the help of the inference engine. In general, the inference mechanism allows concluding new facts from existing facts and rules defined in the ontology. For example, if ontology contains the fact that "A" is an ancestor of “B” and “C”, then the fact that “B” and “C” are the relatives does not have to be included in the knowledge base, but instead can be deducted by the inference engine. Every deduction performed by Cyc inference engine is concluded in a context of the particular micro theory with all corresponding inheritances to reduce the search domain. Cyc inference engine functionality is based on the general logic deduction, such as the universal and existential qualification, mathematical reasoning, quality and temporal inference. Inference engine uses CycL language to perform the deduction effectively by manipulating the knowledge inside the ontology.

Such a robust and powerful inference engine gives the Cyc development platform an indisputable advantage over the other knowledge-based systems. It allows not only reasoning about the existent knowledge and deducting novel information, but it is also capable of performing the natural language generation tasks, such as deriving English language equivalents of the concepts contained in the knowledge base.

2.2.5 Conclusion

Cyc knowledge based system is chosen as a backbone for KBS algorithm described in this dissertation. Cyc surpasses WordNet, BabelNet and ConceptNet in a number of characteristics, such as the breadth and depth of the knowledge represented in the system, the variety of relations between concepts, and the capabilities of the inference engine that allows robust knowledge reasoning.

2.3 Advanced natural language processing techniques

Natural language processing (NLP) is a field of study that combines the ideas from the computer science, artificial intelligence and computational linguistics. NLP allows developing computer algorithms that can automatically process, analyze and represent human language [45]. NLP techniques range from simple word occurrence counting to complex analysis of the sentiment of a text passage. These techniques play a pivotal role during text the data preprocessing step, which is the process of transforming input data from the raw text to the format suitable for further interpretation and analysis.

Following are the main advanced NLP techniques that are frequently used to perform automatic text summarization:

- sentence segmentation;
- tokenization;
- lemmatization;
- part of speech tagging;
- dependency grammar analysis.

Sentence segmentation is a process of separating the text into individual sentences. Punctuation marks, such as a period or a question mark, are used to define sentence boundaries during the sentence segmentation process. Tokenization is a process of breaking up sentences into the separate words based on the primitive white space separator or more complex separator symbols. Tokenization is followed by the lemmatization, the process of reducing the inflectional and derivationally related word forms to a common form known as a lemma. Lemmatization performs the morphological analysis of the words derived by the tokenization to derive their base forms.

For example, words “dark”, “darker” and “darkest” are all lemmatized to the base form “dark”. Parts of speech tagging is a process of assigning a particular part of speech tag to a word in a sentence. There are four major parts of speech tags, also known as the open class tags: nouns, verbs, adjectives and adverbs. Sophisticated statistical methods are used to derive appropriate part of speech tags for the words in the text. The proper parts of speech tagging is crucial for the most of natural language processing techniques, including the lemmatization and syntactic parsing. There is a number of conventions used to denote parts of speech tags. In our research, we follow parts of speech tagging defined by the Universal Dependencies (UD) framework treebank for English language. Parts of speech tags with corresponding descriptions are provided in **Table 2-1**.

Table 2-1: Parts of speech tags from Universal Dependencies treebank.

Parts of speech tag	Description
ADJ	Adjective
ADP	Adposition
AUX	Adverb
CCONJ	Coordination conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numerical
PART	Particle
PRON	Pronoun
PROPN	Proper noun
PUNCT	Punctuation
SCONJ	Subordinating conjunction
SYM	Symbol
VERB	Verb
X	Other

Dependency grammar analysis derives the syntactic structure of the sentences based on the words and the grammatical relations that link these words. During the syntactic parsing, the sentence is being represented as a dependency tree. Such a tree structure has a root that states the head of the sentence and the nodes, represented by the words of the sentence. The nodes are connected by their syntactic relationships. For example, in the sentence, “I study computer science”, the verb “study” is the root of the dependency tree, the pronoun “I” is the subject of the verb “study”, the noun “science” is the object of the verb “study”, and the noun “computer” is a compound modifier of the noun “science” [46]. There is a number of conventions used to denote the dependency relation tags. In our research, we use dependency tags defined by the Universal Dependencies (UD) framework scheme for the English language. Descriptions of the dependency tags are provided in **Table 2-2**.

Table 2-2: Syntactic dependency relationships tags from Universal Depenencies scheme.

Dependency relation tag	Description
ACOMP	Adjectival complement
ADVMOD	Adverbial modifier
AMOD	Adjectival modifier
CSUBJ	Clausal subject
CSUBJPASS	Clausal subject (passive)
DOBJ	Direct object
IOBJ	Indirect object
NSUBJ	Nominal subject
NSUBJPASS	Nominal subject (passive)
OPRD	Object predicate
OBJ	Object
POBJ	Object of preposition

CHAPTER 3

ABSTRACTIVE TEXT SUMMARIZATION USING CYC DEVELOPMENT PLATFORM

This chapter provides a detailed description of the underlying methodology of the proposed algorithm for abstractive text summarization.

The KBS algorithm described in pages 3 and 4 attempts to bring the machines one-step closer to the comprehension of the knowledge comprised in the text. The algorithm performs text summarization in three principal steps: the knowledge acquisition, the knowledge discovery, and the knowledge representation. During the knowledge acquisition step, the algorithm receives text documents as an input, performs deep syntactic analysis, and maps the words with their syntactic relationships into the Cyc knowledge base. During the knowledge discovery step, the KBS algorithm performs a generalization of new concepts by propagating the concepts that were mapped into Cyc knowledge base by the knowledge acquisition step. It also performs the task of the identification of the main topics of the text based on the mapped and generalized concepts. Finally, during the knowledge representation step, the KBS algorithm generates new sentences using knowledge derived from the input text documents and the capabilities of Cyc inference engine. The subsections of this chapter describe the workflow of three steps of the KBS summarization algorithm.

3.1 Knowledge acquisition

The knowledge acquisition consists of two sub-processes. The first sub-process extracts the syntactic structures from the given documents. This sub-process serves as a data preprocessing and transformation step. It normalizes raw text data and transforms it into syntactic representation. The second sub-process maps words from syntactic representation of the text to Cyc concepts. Mapped Cyc concepts are utilized for reasoning during subsequent steps of the algorithm.

3.1.1 Syntactic structure extraction

The syntactic structure extraction sub-process starts by separating input text into individual sentences. Then it applies the process of tokenization to separate sentences into individual words and uses lemmatization to normalize word forms. Next, it assigns the appropriate parts of speech tag for each lemmatized word in the sentence. Parts of speech tags are required during the mapping process and help to address the disambiguation issue. Only open class parts of speech tags such as noun, verb, adjective, and adverb are used for the analysis.

Next, the sub-process applies the syntactic dependency parses to discover the relationships between the words in the sentences. Syntactic dependency relationships are recorded in the following format: (“word” “relationship type” “head”), where “word” is the dependent element in the relationship, “relationship type” is the type of the relationship, and “head” is the leading element in the relationship. For example, applying syntactic parser on sentence “Rottweiler rarely eats raw veal” produces the following relationships: (“Rottweiler” “nsubj” “eats”), (“veal” “dobj” “eats”), (“rarely” “advmod” “eats”), (“raw” “amod” “veal”). Syntactic dependency relationships of the example sentence are illustrated in **Figure 3-1**. Syntactic dependency relationships are crucial features for the new sentence generation sub-process of the knowledge representation step of the summarization algorithm.

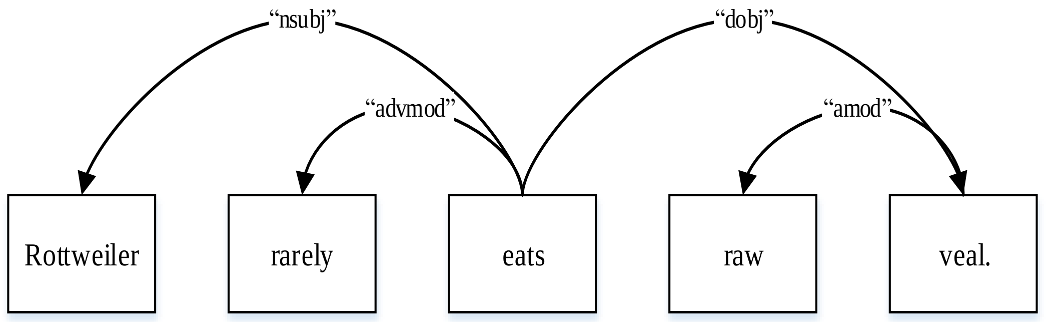


Figure 3-1: Example of syntactic dependency relationships in a sentence.

Finally, the sub-process counts and records frequencies of the word occurrences and their relationships. These frequencies are used as weights for corresponding Cyc concepts and their relationships during mapping sub-process of the knowledge acquisition step.

The syntactic structure extraction sub-process produces syntactic representation of the input text that consists of words, their frequencies, parts of speech tags, syntactic dependency relationships and their frequencies. Workflow diagram of the sub-process is outlined in **Figure 3-2**.

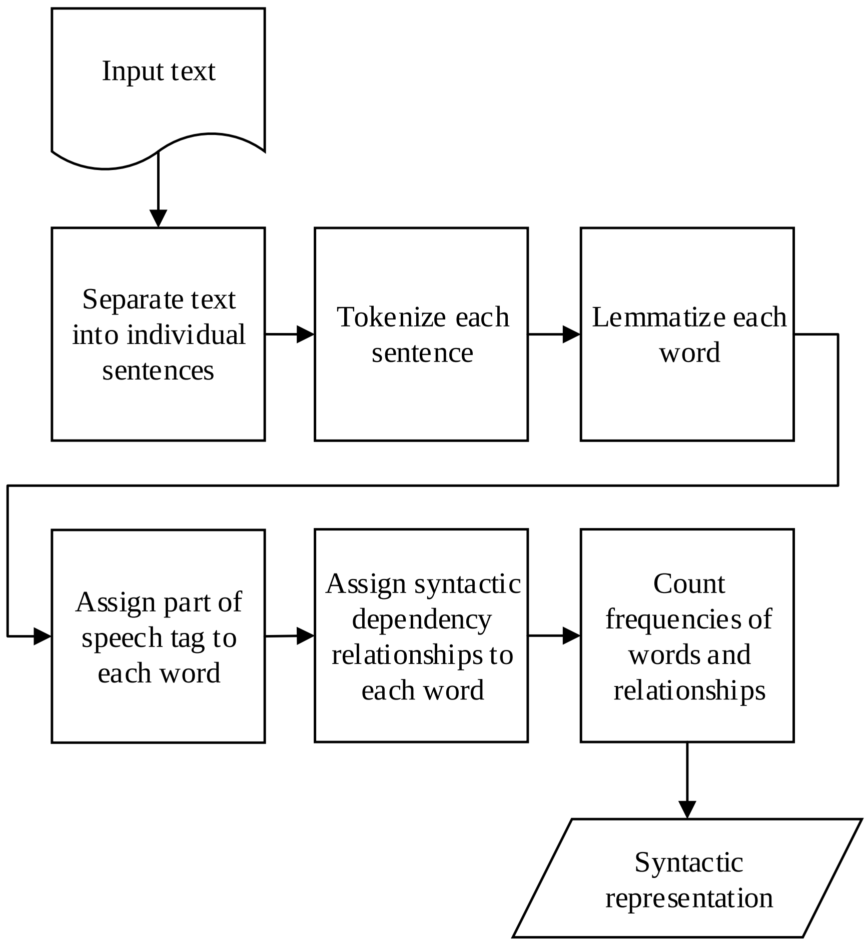

Input text
Separate text into individual sentences
Tokenize each sentence
Lemmatize each word
Assign part of speech tag to each word
Assign syntactic dependency relationships to each word
Count frequencies of words and relationships
Syntactic representation

Figure 3-2: Syntactic structure extraction sub-process workflow diagram.

3.1.2 Mapping words to Cyc concepts

The mapping words to Cyc concepts sub-process finds matching Cyc concept for each word in the input document. Once algorithm finds correspondent Cyc concept it assigns word’s weight, its syntactic dependency relationships and their weights to the Cyc concept. Word’s weight is a frequency, the number of times it is mentioned in the text. The dependency relationship is an association between two words in a sentence, derived by the syntactic dependency parser. Each dependency relationship has a weight associated with it that shows how frequently two words were used together in the text.

Higher weights represent stronger syntactic dependency relationships. Our algorithm enhances Cyc semantic knowledge about the concepts with the syntactic structures derived from the input text. The semantic knowledge and the syntactic structures are two crucial parts that make abstractive summary cohesive and meaningful. The steps of the mapping words to Cyc concepts sub-process are outlined as follows:

- For each word in the syntactic representation obtained by the syntactic structure extraction sub-process:
    - Map word to the corresponding Cyc concept;
    - Assign the word’s weight to the corresponding Cyc concept;
    - Map relationship head word to the corresponding Cyc concept;
    - Assign the word’s relationship and relationship’s weight to the corresponding Cyc concept.

Workflow diagram of the sub-process is illustrated in **Figure 3-3**.

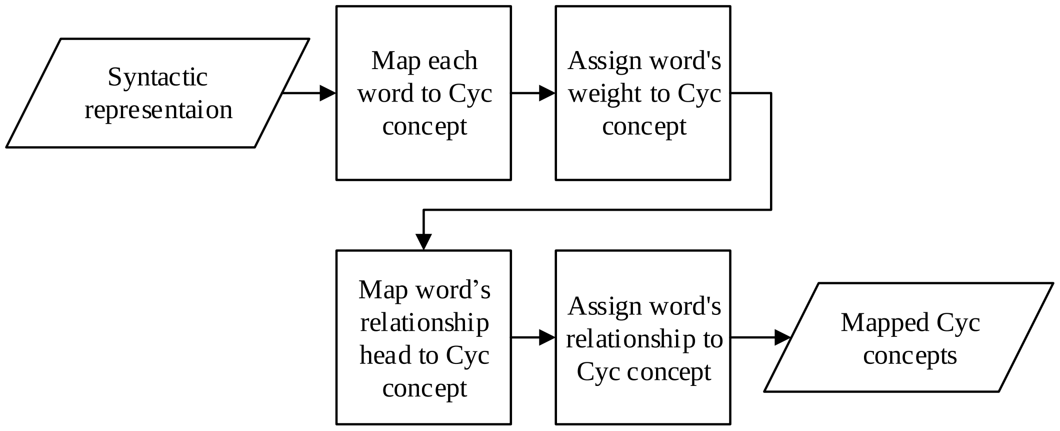

Syntactic representaion
Map each word to Cyc concept
Assign word's weight to Cyc concept
Map word’s relationship head to Cyc concept
Assign word's relationship to Cyc concept
Mapped Cyc concepts

Figure 3-3: Mapping words to Cyc concepts sub-process workflow diagram.

3.2 Knowledge discovery

The knowledge discovery step performs two tasks: it abstracts new concepts and identifies main topics described in the input text.

New concepts abstraction sub-process performs generalization of the information derived from the text. It finds the ancestors of mapped Cyc concepts and assigns the descendants' propagated weight and syntactic dependency relationships to the ancestors. It is an important part of abstractive summarization process as it allows deriving concepts that are not explicitly mentioned in the input text. For example, concepts like "cat", "tiger", "jaguar" and "lion" are generalized into more abstract "feline" concept. Another example of concepts propagation is illustrated in **Figure 3-4**. The relationship between descendant concepts "banana", "orange", "apple", "pear" and ancestor concept "edible fruit" in Cyc ontology is represented by the "#$isA" Cyc predicate.

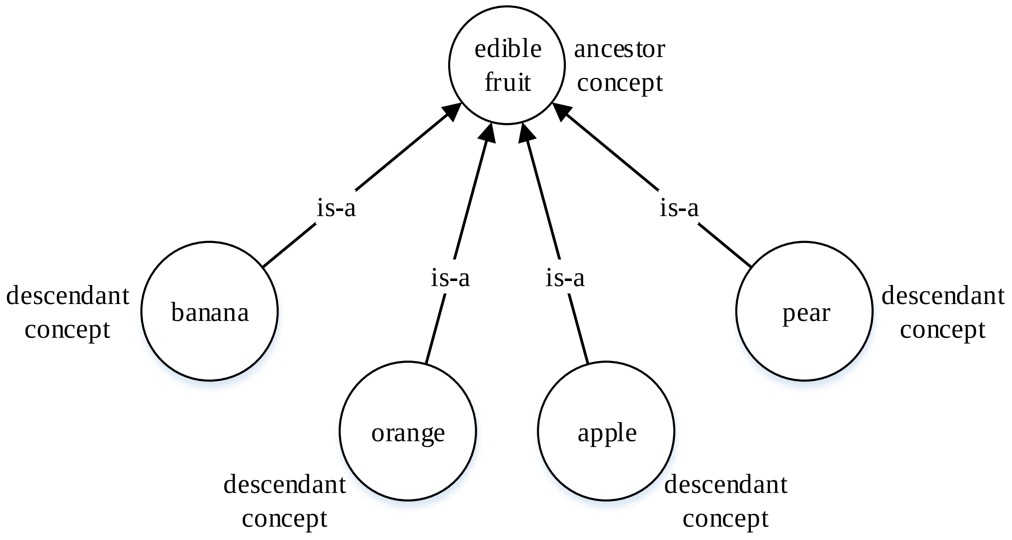


Figure 3-4: Upward concepts propagation in Cyc ontology.

The main topics identification sub-process detects topics described in the text with an assumption that they are represented by the most frequently used micro theories. Micro theories form the basis of knowledge organization in Cyc ontology being the clusters of Cyc concepts and facts, typically representing one specific domain of knowledge. For example, #$BiologyMt is a micro theory containing biological knowledge, and #$MathMt is a micro theory containing concepts and facts describing the field of mathematics. Each Cyc concept is defined within a micro theory.

3.2.1 New concepts abstraction

The new concepts abstraction sub-process consists of two steps: concepts propagation step and concepts' weight and relationships accumulation step. Concepts propagation derives an ancestor concept for each mapped Cyc concept. Concepts' weight and relationships accumulation adds the descendant concepts' accumulated weight and relationships to ancestor concept based on the generalization parameter.

The concepts propagation starts by finding the ancestor concept for each concept that was mapped to Cyc ontology during knowledge acquisition step. Then it records ancestor-descendant relationship, updates the number of ancestor's descendant concepts and accumulated descendant's weight. Accumulated descendant weight is scaled by the generalization parameter α. This step of the new concepts abstraction sub-process is described as follows:

- For each mapped Cyc concept:
    - Find the concept's ancestor;
    - Record the ancestor-descendant relation;
    - Update the ancestor's number of descendants;

- Update the ancestor's descendants accumulated weight;
  - Scale the descendant's weight by α.

Workflow diagram of the concepts propagation step is illustrated in **Figure 3-6**.

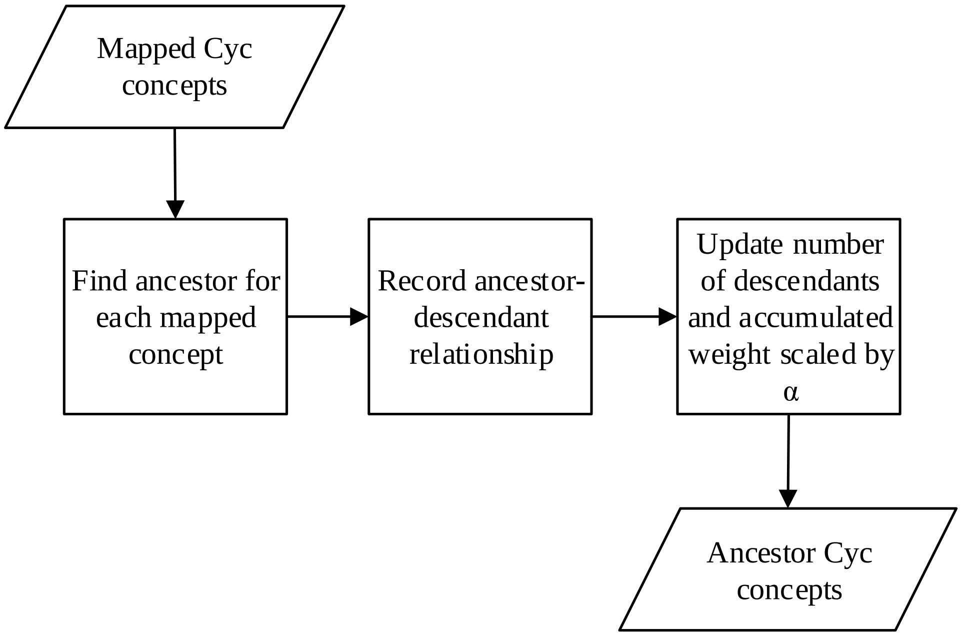

Mapped Cyc concepts
Find ancestor for each mapped concept
Record ancestor-descendant relationship
Update number of descendants and accumulated weight scaled by α
Ancestor Cyc concepts

Figure 3-5: Concepts propagation step workflow diagram.

The concepts' weight and relationships accumulation step starts by calculating the descendant-ratio – the number of mapped descendants divided by the number of all descendants of a concept.

$$desc\_ratio = \frac{\#\ mapped\ descendants}{\#\ of\ all\ descendants} \qquad \textbf{Eq. 3-1}$$

Next, if the descendant-ratio is higher than the defined generalization parameter β, then the descendants' weight and descendants' relationships are added to the ancestor concept. Parameters α and β regulate the desired level of generalization. Higher α and lower β yield greater level of generalization giving more emphasis to ancestor concepts.

Concept’s weight and relationships accumulation step of the new concepts abstraction sub-process is described as follows:

- For each ancestor Cyc concept:
    - Find the number of concept’s mapped descendants;
    - Find the number of all concept’s descendants;
    - Calculate descendants’ ratio;
    - If descendant-ratio is larger than the defined threshold β:
        - Add descendants’ accumulated weight to the ancestor’s weight;
        - Add descendants’ relationships to the ancestor’s relationships;
            - Scale descendant’s relationship weight by α.

Workflow diagram of the concepts’ weight and relationships accumulation step is illustrated in **Figure 3-6**.

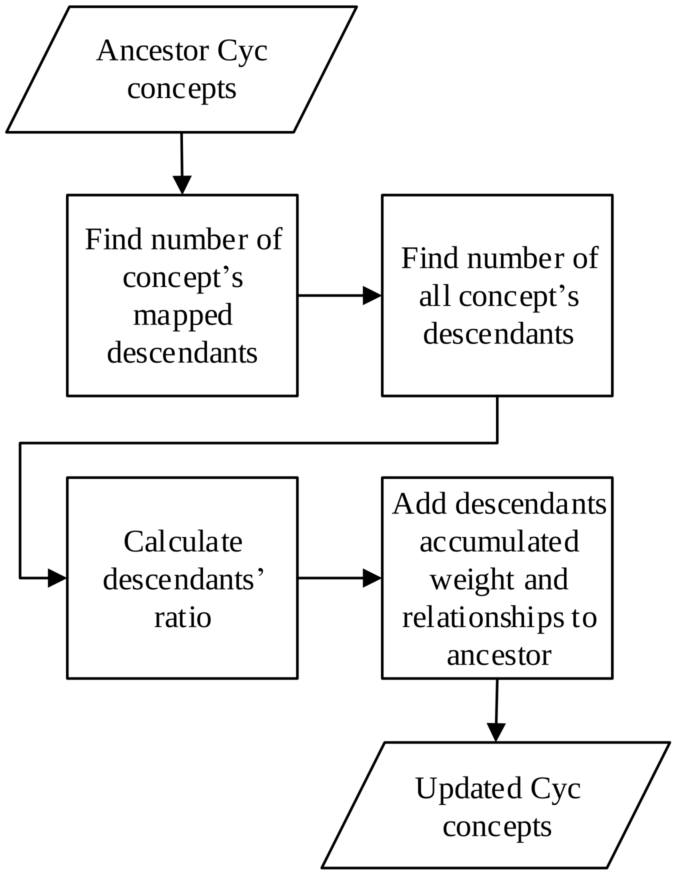

Ancestor Cyc concepts
Find number of concept's mapped descendants
Find number of all concept's descendants
Calculate descendants' ratio
Add descendants accumulated weight and relationships to ancestor
Updated Cyc concepts

Figure 3-6: Concepts' weight and relationships accumulation step workflow diagram.

3.2.2 Main topics identification

The main topics of the input text are identified by the most frequent micro theories derived from the updated mapped Cyc concepts. The sub-process starts by finding defining micro theory for each mapped Cyc concept. Next, it counts frequencies of discovered micro theories. Then, it picks the top-n micro theories with the highest frequencies that will represent the main topics of the input text.

The main topics identification sub-process is described as follows:

- For each mapped Cyc concept:
    - Find defining micro theories.
- Count the frequencies of discovered micro theories;
- Pick the top-n micro theories with the highest frequencies.

Workflow diagram of the sub-process is illustrated in **Figure 3-7**.

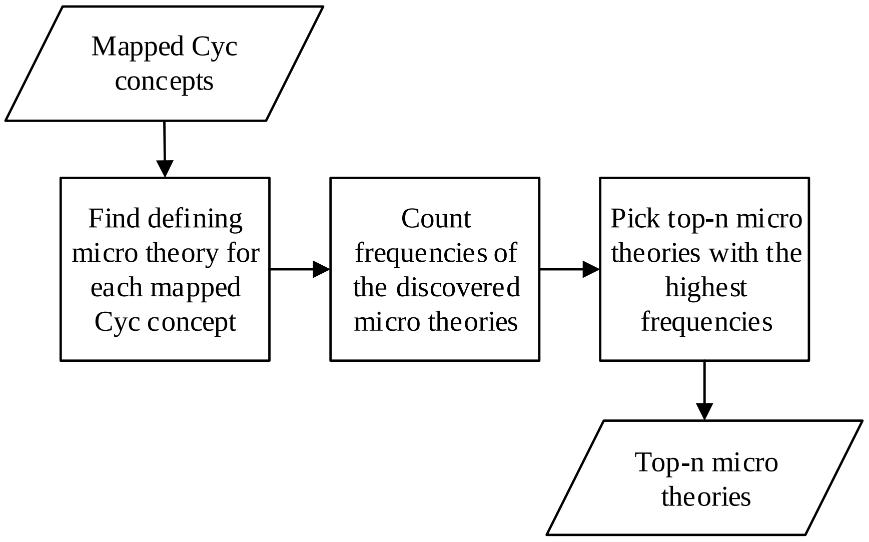

Mapped Cyc concepts
Find defining micro theory for each mapped Cyc concept
Count frequencies of the discovered micro theories
Pick top-n micro theories with the highest frequencies
Top-n micro theories

Figure 3-7: Main topics identification sub-process workflow diagram.

3.3 Knowledge representation

The knowledge representation utilizes powerful capabilities of the Cyc inference engine to generate new sentences based on the information discovered during knowledge acquisition and knowledge discovery steps. This step uses mapped and generalized Cyc concepts, their syntactic dependency relationships, and most frequent micro theories as inputs. Knowledge representation step consists of two sub-processes – candidate subjects discovery and new sentences generation. Candidate subjects discovery sub-process identifies significant subject concepts out of all the mapped and generalized Cyc concepts. New sentences generation sub-process composes new sentences for each of the identified candidate subject concept. Generated sentences serve as a final summary of the input text.

3.3.1 Candidate subjects discovery

The candidate subjects discovery sub-process starts by finding all mapped Cyc concepts in each main topic derived during knowledge discovery process. Then it calculates the subjectivity ratio of each of the found Cyc concepts. Subjectivity ratio is defined as the number of concept's relationships labelled as subject relationship divided by the total number of all concept's relationships. This ratio allows identifying concepts that have more subject relationships and helps distinguish concepts with a stronger subject role in the input text.

$$subj\_ratio = \frac{\#\ of\ subject\ associations}{\#\ of\ all\ associations} \qquad \textbf{Eq. 3-2}$$

Next, it calculates subjectivity rank for each found subject concepts. Subjectivity rank is defined as a product of concept weight and concept subjectivity ratio. Subjectivity rank scales the weight of the concept by the subjectivity ratio, which allows choosing subjects that are more semantically meaningful in the context of the given text.

$$subj\_rank = concept\_weight * subj\_ratio \qquad \textbf{Eq. 3-3}$$

Finally, concepts with the highest subjectivity rank are chosen as the candidate subject concepts and new sentences are being created for each of them during new sentence generation sub-process.

The candidate subjects discovery sub-process is described as follows:

- For each top-n micro theory:
    - For each concept mapped from the text:
        - Find the number of subject associations;
        - Find the number of all associations;
        - Calculate subjectivity ratio;
        - Calculate subjectivity rank;
    - Pick the top-n subjects with the highest subjectivity rank.

Workflow diagram of the sub-process is outlined in **Figure 3-8**.

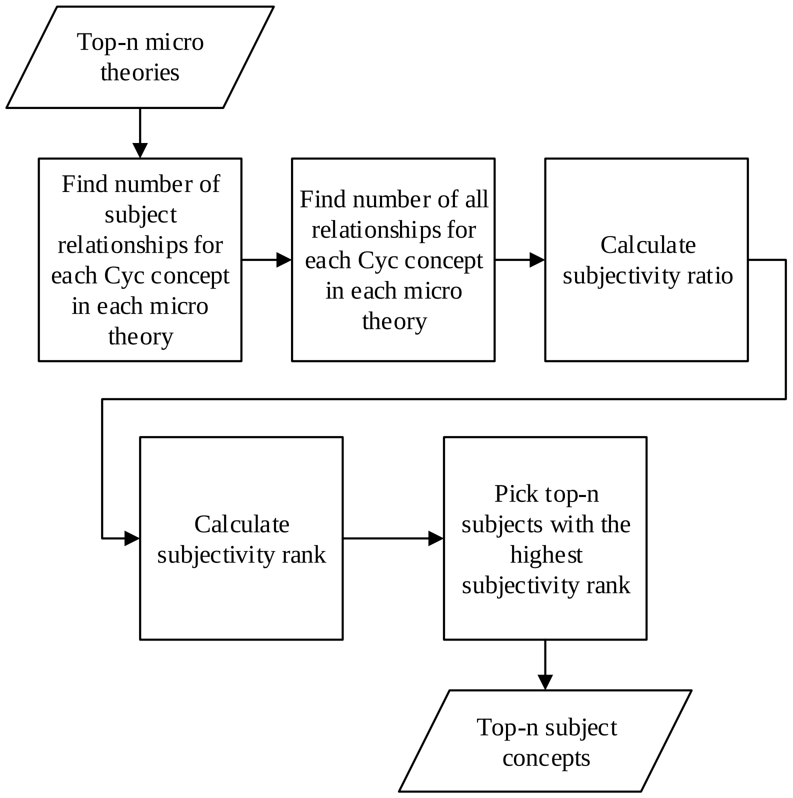

Top-n micro theories
Find number of subject relationships for each Cyc concept in each micro theory
Find number of all relationships for each Cyc concept in each micro theory
Calculate subjectivity ratio
Calculate subjectivity rank
Pick top-n subjects with the highest subjectivity rank
Top-n subject concepts

Figure 3-8: Candidate subjects discovery sub-process workflow diagram.

3.3.2 New sentences generation

The new sentences generation sub-process uses subject concepts identified during the candidate subjects discovery sub-process and their syntactic dependency relationships discovered during the knowledge acquisition process. This sub-process creates new English sentences for each candidate subject concept to generate a summary of the input text based on the discovered knowledge. The basic structure of newly created sentences follows the shallow triplet model, where each sentence has subject, predicate and object elements. Such basic triplet structure is enhanced by the adjective modifiers for the subject and object elements and by the adverb modifiers for the predicate elements when available. Subject, predicate and object elements of the sentences are mandatory while adjective and adverb modifiers are optional. **Figure 3-9** illustrates the enhanced structure of newly created sentences.

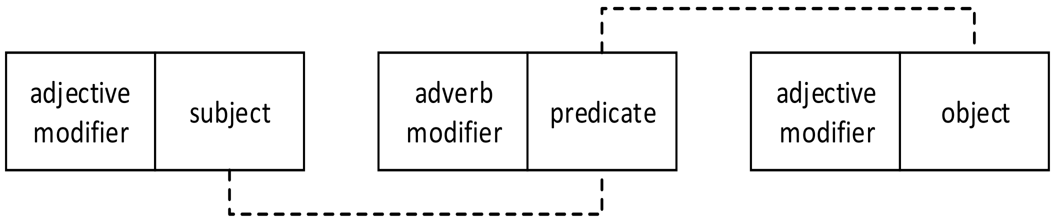


Figure 3-9: Enhanced structure of newly created sentence.

Described sentence structure enhancement allows creating new sentences with a more complex structure that goes beyond simple subject-predicate-object model. Sentence creation process starts by identification of the corresponding predicate and object elements for each candidate subject based on the weights of the subject-predicate, predicate-object and subject-object syntactic dependency relationships. Then it proceeds by deriving the appropriate adjective and adverb modifiers for subject, predicate and object elements, based on the weights of subject-adjective, predicate-adverb and object-adjective syntactic dependency relationships.

Subject, predicate, object, adverb, and adjective elements of new sentences are derived from Cyc knowledge base as Cyc concepts that are expressed in a particular format having a “#$” prefix. For example, dog is expressed as a “#$Dog” concept in Cyc knowledge base. New sentence generation sub-process uses natural language generation capabilities of Cyc inference engine to derive English language representations of Cyc concepts. Cyc command “generate-phrase” allows retrieving natural language word or phrase equivalent of a Cyc concept. As an example, applying “generate-phrase” Cyc command to "#$EatingEvent" Cyc concept produces the string "eat" as an output and applying it to "#$Coyote-Animal" produces the string "coyote". This powerful natural language generation functionality of Cyc inference engine is another advantage of using Cyc development platform as a backbone.

The new sentence generation sub-process is outlined as follows:

- For each candidate subject:
    - Convert subject Cyc concept to natural language representation;
    - Pick the adjective with the highest subject-adjective relationship weight;
    - Convert adjective Cyc concept to natural language representation;
    - Pick the top-n predicates with the highest subject-predicate relationship weights;
    - For each predicate in the top-n predicates:

- Convert predicate Cyc concept to natural language representation;
- Pick the adverb with the highest predicate-adverb relationship weight;
- Convert adverb Cyc concept to natural language representation;
- Pick the top-n objects with the highest product of subject-object and predicate-object relationships weights;
- For each object in the top-n objects:
    - Convert object Cyc concept to natural language representation;
    - Pick the adjective with highest object-adjective relationship weight;
    - Convert adjective Cyc concept to natural language representation;
    - Compose the new sentence using subject, subject-adjective, predicate, predicate-adverb, object, and object-adjective natural language representations.

Workflow diagram of the sub-process is outlined in **Figure 3-10**.

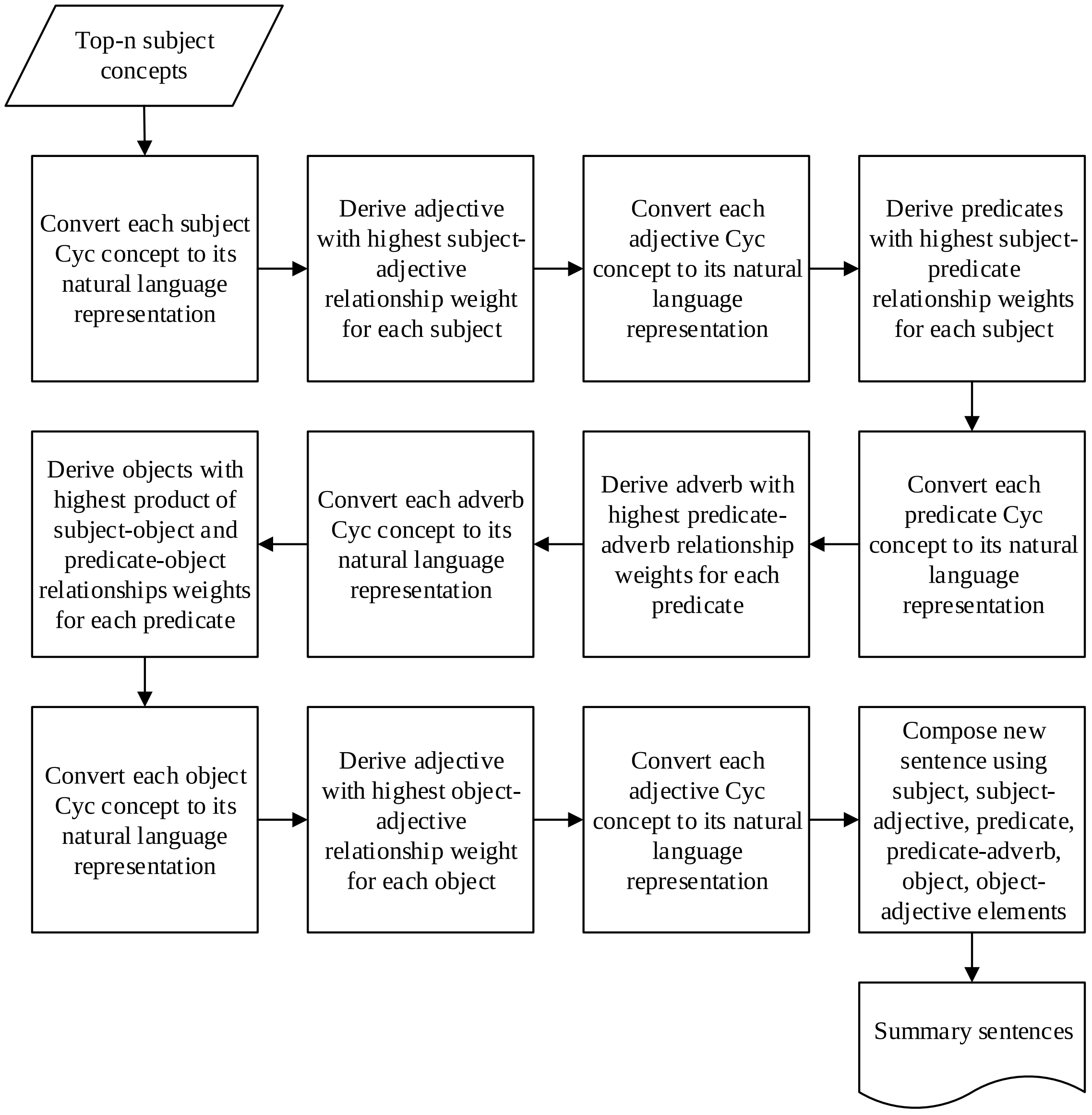

Top-n subject concepts
Convert each subject Cyc concept to its natural language representation
Derive adjective with highest subject-adjective relationship weight for each subject
Convert each adjective Cyc concept to its natural language representation
Derive predicates with highest subject-predicate relationship weights for each subject
Convert each predicate Cyc concept to its natural language representation
Derive adverb with highest predicate-adverb relationship weights for each predicate
Convert each adverb Cyc concept to its natural language representation
Derive objects with highest product of subject-object and predicate-object relationships weights for each predicate
Convert each object Cyc concept to its natural language representation
Derive adjective with highest object-adjective relationship weight for each object
Convert each adjective Cyc concept to its natural language representation
Compose new sentence using subject, subject-adjective, predicate, predicate-adverb, object, object-adjective elements
Summary sentences

Figure 3-10: New sentences generation sub-process workflow diagram.

CHAPTER 4

IMPLEMENTATION OF THE ABSTRACTIVE TEXT SUMMARIZATION SYSTEM

KBS algorithm was implemented as an abstractive text summarization system. This chapter provides description of the system design and the technical details of the system implementation.

The system was implemented using Python programming language. Python was a natural choice because of the advanced Natural Language Processing tools and libraries supplied by the language. Sentence segmentation, tokenization, lemmatization, parts of speech tagging and dependency grammar analysis were implemented with the help of SpaCy – Python library for advanced natural language processing. This library is the fastest in the world with the accuracy within one percent of the current state of the art systems for parts of speech tagging and dependency grammar analysis [47].

4.1 Cyc development platform integration

Our system uses Cyc knowledge base and its inference engine as a backbone for the semantic analysis. Cyc development platform supports communications with the knowledge base and utilization of the inference engine through the application programming interfaces (APIs) implemented in Java. We utilize Java-Python wrapper supported by JPype Python library to allow our system using Cyc Java API packages. JPype library provides a code written in Python convenient access to Java class libraries.

It is essentially an interface at a basic level of virtual machines. Such wrapper allows using Java API calls provided by Cyc development platform inside our system, which is developed in Python. JPype library requires starting Java Virtual Machine before Java packages or classes can be used within the Python code. Then any packages, methods or classes are accessible given an appropriate path to their jar file implementation [48]. Communication between our system and Cyc development platform is illustrated in **Figure 4-1**. To the best of our knowledge, our summarization system is the first Python-based system that allows communication with Cyc development platform.

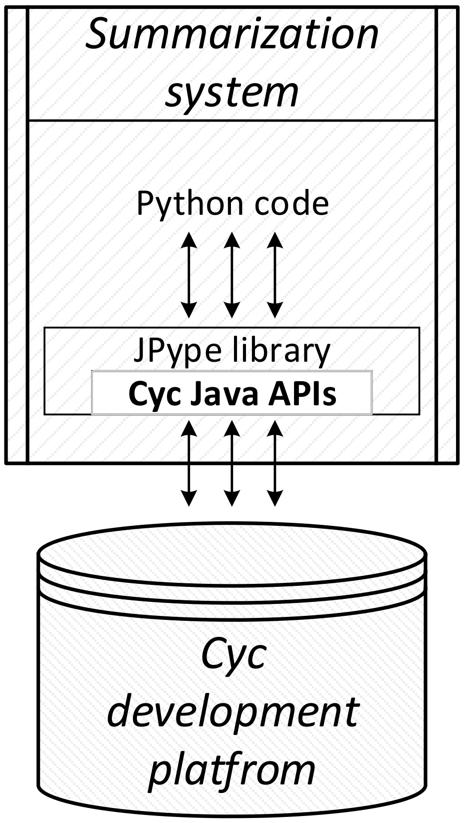


Figure 4-1: Communication between summarization system and Cyc development platform.

4.2 Summarization system's design

We designed our abstractive summarization system as a modular and pipelined data-mining framework. Modularity provides the ability to conveniently maintain parts of the system and to add new functionality as needed. Pipelined design of the system allows comprehensible data flow between different modules.

The system consists of seven modules:

A. Syntactic structure extraction;

B. Mapping words to Cyc concepts;

C. Concepts propagation;

D. Concepts' weight and relationships accumulation;

E. Main topics identification;

F. Candidate subjects discovery;

G. New sentences generation.

Modules A and B together constitute the knowledge acquisition step of the summarization algorithm. Modules C, D and E together make up the knowledge discovery step of the summarization algorithm. Modules F and G together form knowledge representation step of the summarization algorithm. Each module is implemented as a separate function with defined input parameters and generated outputs. Modular system's design is illustrated in **Figure 4-2**. The rest of the chapter provides the description of system's modules.

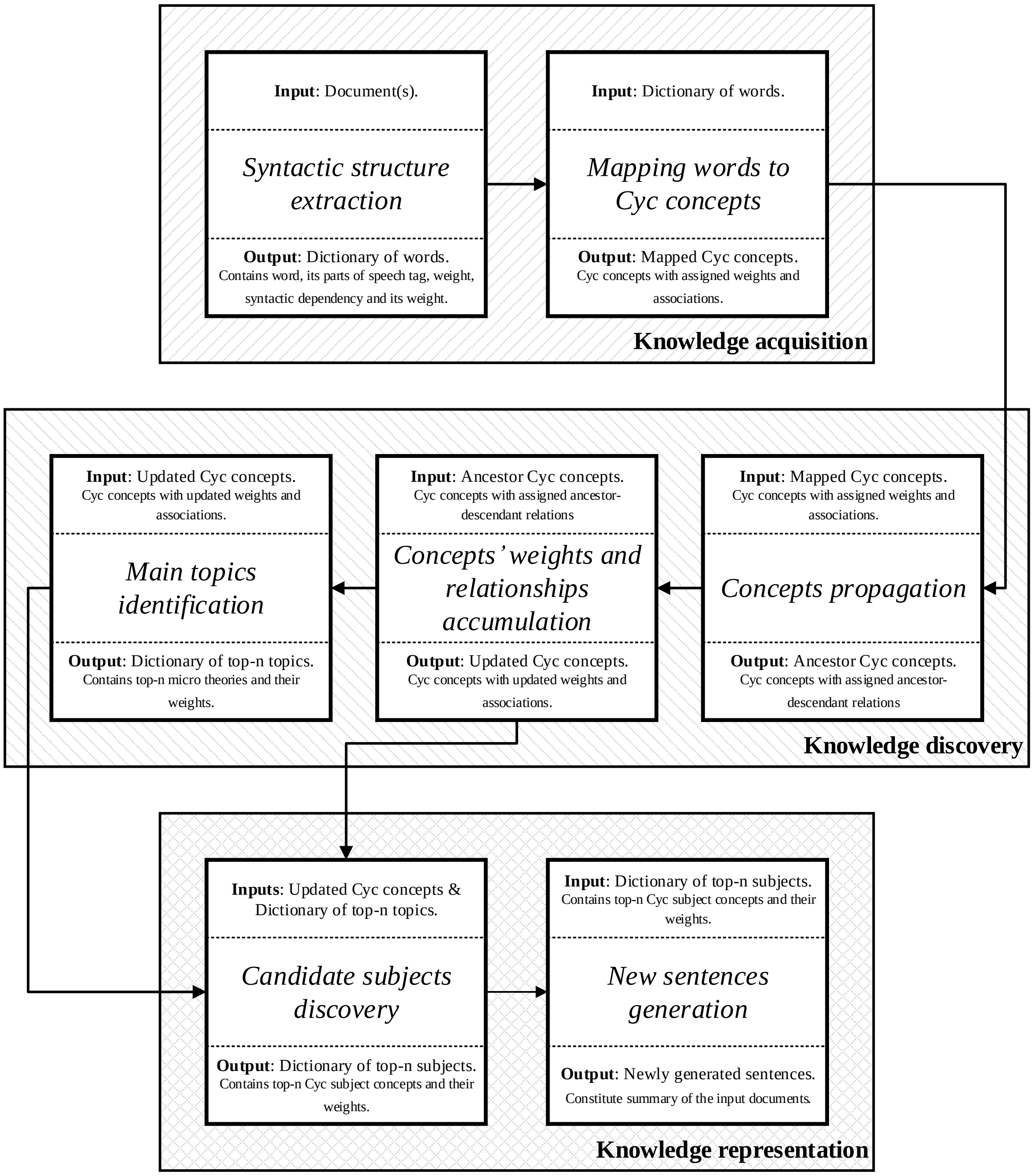

Input: Document(s).
Syntactic structure extraction
Output: Dictionary of words.
Contains word, its parts of speech tag, weight, syntactic dependency and its weight.
Input: Dictionary of words.
Mapping words to Cyc concepts
Output: Mapped Cyc concepts.
Cyc concepts with assigned weights and associations.
Knowledge acquisition
Input: Updated Cyc concepts.
Cyc concepts with updated weights and associations.
Main topics identification
Output: Dictionary of top-n topics.
Contains top-n micro theories and their weights.
Input: Ancestor Cyc concepts.
Cyc concepts with assigned ancestor-descendant relations
Concepts' weights and relationships accumulation
Output: Updated Cyc concepts.
Cyc concepts with updated weights and associations.
Input: Mapped Cyc concepts.
Cyc concepts with assigned weights and associations.
Concepts propagation
Output: Ancestor Cyc concepts.
Cyc concepts with assigned ancestor-descendant relations
Knowledge discovery
Inputs: Updated Cyc concepts & Dictionary of top-n topics.
Candidate subjects discovery
Output: Dictionary of top-n subjects.
Contains top-n Cyc subject concepts and their weights.
Input: Dictionary of top-n subjects.
Contains top-n Cyc subject concepts and their weights.
New sentences generation
Output: Newly generated sentences.
Constitute summary of the input documents.
Knowledge representation

Figure 4-2: Modular design of the system.

4.2.1 <u>“Syntactic structure extraction” module</u>

The “Syntactic structure extraction” module is implemented using SpaCy – Python library for advanced natural language processing. This module operates outside of the Cyc development platform. The output of the module is a dictionary that contains words, their part of speech tags, weights and syntactic dependencies. This dictionary serves as an input for the “Mapping words to Cyc concepts” module. Source code of the module implementation is provided in A.1

4.2.2 “Mapping words to Cyc concepts” module

The “Mapping words to Cyc concepts” module communicates with Cyc development platform and updates weight and syntactic dependency relationships of Cyc concepts. The output of the module are mapped Cyc concepts with assigned weights and syntactic dependency relationships. The mapped Cyc concepts serve as an input for “Concepts propagation” module. “Syntactic structure extraction” and “Mapping words to Cyc concepts” modules together constitute the knowledge acquisition step of the summarization process. **Table 4-1** provides description of Cyc commands used to map word to Cyc concept (a), assign the word’s weight (b), the word’s syntactic relationship and syntactic relationship’s weight (c) to the Cyc concept. Source code of the module implementation is provided in A.2.

Table 4-1: Description of Cyc commands used by "Mapping words to Cyc concepts" module.

ID	Cyc command	Description
(a)	(#$and (#$denotation ?Word ?POS ?Num ?Concept) (#$word-Forms ?Word ?WordForm "word") (#$genls ?POS ?POSTag))	Command uses built-in "#$denotation" Cyc predicate to relate a "word", its part of speech tag (?POS), and a sense number (?Num) to concept (?Concept). It also uses "#$wordForms" and "#$genls" predicates to accommodate for all variations of word's lexical forms.
(b)	(#$conceptWeight ?Concept ?Weight)	Command uses user-defined "#$conceptWeight" Cyc predicate that assigns the weight (?Weight) to the concept (?Concept).
(c)	(#$conceptAssociation ?Concept ?Type ?HeadConcept ?Weight)	Command uses user-defined "#$conceptAssociation" Cyc predicate that assigns a specific type (?Type) of a syntactic dependency association, the leading element (?HeadConcept) and the weight (?Weight) to the concept (?Concept).

4.2.3 "Concepts propagation" module

The "Concepts propagation" module communicates with Cyc development platform to derive all mapped Cyc concepts (a), find closest ancestor concepts (b) and update ancestor concepts' relations (c, d). The output of the module are ancestor Cyc concepts with assigned descendant concepts' weights and counts and ancestor-descendant relations. The ancestor Cyc concepts are used by the "Concepts' weight and relationships accumulation" module. Cyc commands used by the "Concepts propagation" module are described in **Table 4-2**. Source code of the module implementation is provided in A.3.

Table 4-2: Description of Cyc commands used by "Concepts propagation" module.

ID	Cyc command	Description
(a)	(#$conceptWeight ?Concept ?Weight)	Command uses user-defined "#$conceptWeight" Cyc predicate to retrieve concepts (?Concept) that have as-signed weights (?Weight).
(b)	(#$min-genls ?Concept)	Command uses built-in "min-genls" Cyc predicate to retrieve the closest ancestor concept for the given concept (?Concept).
(c)	(#$conceptDescendants ?Concept ?Weight ?Count)	Command uses user-defined "#$conceptDescendants" Cyc predicate to record the number of descendants (?Count) and their weight (?Weight) to the ancestor concept (?Concept).
(d)	(#$conceptAncestorOf ?Concept ?Descendant)	Command uses user-defined "#$conceptAncestorOf" predicate to assign ancestor-descendant relation between the ancestor concept (?Concept) and the descendant concept (?Descendant).

4.2.4 "Concepts' weight and relationships accumulation" module

The "Concepts' weight and relationships accumulation" module communicates with Cyc development platform to derive all ancestor Cyc concepts (a), find the number of ancestor's mapped descendants (b), find the number of all ancestor's descendants (c) and update ancestor's weight and relations (d, e). The output of the module are the Cyc concepts with updated weights and syntactic dependency relationships. Updated Cyc concepts are used by the "Main topics identification" and the "Candidate subjects discovery" modules. Cyc commands used by the "Concepts' weight and relationships accumulation" module are described in **Table 4-3**. Source code of the module implementation is provided in A.4.

Table 4-3: Description of Cyc commands used by "Concepts weight and relationships accumulation" module.

ID	Cyc command	Description
(a)	(#$conceptDescendants ?Concept ?Weight ?Count)	Command uses user-defined "#$conceptDescendants" Cyc predicate to retrieve all concepts (?Concept) that have descendants.
(b)	(#$conceptAncestorOf ?AncConcept ?MappedDesc)	Command uses user-defined "#$conceptAncestorOf" predicate to retrieve mapped descendant concepts (?MappedDesc) of the given ancestor concept (?AncConcept).
(c)	(#$genls ?AncConcept ?DescConcept)	Command uses built-in "#$genls" Cyc predicate to retrieve all descendant concepts (?DescConcept) of the given ancestor concept (?AncConcept).
(d)	(#$conceptWeight ?AncConcept ?DescWeight)	Command uses user-defined "#$conceptWeight" Cyc predicate to assigns the descendant concepts' propagated weight (?DescWeight) to the ancestor concept (?AncConcept).

(e)	(and (#$conceptAncestorOf ?AncConcept ?DescConcept) (#$conceptAssociation ?DescConcept ?Type ?Head-Concept ?Weight))	Command uses user-defined “#$conceptAncestorOf” and “#$conceptAssociation” Cyc predicates to assign descendant’s association (?DescConcept) and its propagated weight (?Weight) to the ancestor concept (?AncConcept).

4.2.5 “Main topics identification” module

The “Main topics identification” module communicates with Cyc development platform to derive defining micro theory for each mapped Cyc concept (a). Calculation of the derived micro theories’ frequencies is handled outside of the Cyc development platform. The output of the module is the micro theories dictionary that contains top-n micro theories with the highest weights. This dictionary serves as an input for the “Candidate subjects discovery” module. The “Concepts propagation”, the “Concepts’ weight and relationships accumulation” and the “Main topics identification” modules together constitute knowledge discovery step of the summarization process. **Table 4-4** provides the description of Cyc command used by the “Main topics identification” module. Source code of the module implementation is provided in A.5.

Table 4-4: Description of Cyc command used by “Main topic identification” module.

ID	Cyc command	Description
(a)	(#$and (#$conceptWeight ?Concept ?Weight) (#$definingMt ?Concept ?MicroTheory))	Command uses user-defined “#$conceptWeight” Cyc predicate and built-in “definingMt” Cyc predicate to derive defining micro theory (?MicroTheory) for each concept (?Concept) that have assigned weight (?Weight).

4.2.6 “Candidate subjects discovery” module

The “Candidate subjects discovery” module communicates with Cyc development platform to derive mapped Cyc concepts for each defining micro theory in the input dictionary (a) and to find the number of the concept’s syntactic dependency associations labelled as “subject” relation (b) and the number of all syntactic dependency associations of the concept (c). Calculations of the subjectivity ratio and the subjectivity rank are handled outside of the Cyc development platform. The output of the module is the dictionary that contains top-n subjects with the highest subjectivity rank. This dictionary serves as an input for the “New sentences generation” module. **Table 4-5** provides the description of Cyc commands used by the “Candidate subjects discovery” module. Source code of the module implementation is provided in A.6.

Table 4-5: Description of Cyc commands used by “Candidate subjects identification” module.

ID	Cyc command	Description
(a)	(#$and (#$definingMt ?Concept ?MicroTheory) (#$conceptWeight ?Concept ?Weight))	Command uses built-in “#$definingMt” Cyc predicate and user-defined “conceptWeight” Cyc predicate to derive concepts (?Concept) that have assigned weight (?Weight) for each micro theory (?MicroTheory) in micro theories dictionary.
(b)	(#$conceptAssociation ?Concept "nsubj" ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “nsubj” parameter to derive the concept’s (?Concept) syntactic dependency associations labelled as “subject” relations.
(c)	(#$conceptAssociation ?Concept ?Type ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with no parameter specified (?Type) to derive all concept’s (?Concept) syntactic dependency associations.

4.2.7 “New sentences generation” module

The “New sentences generation” module communicates with Cyc development platform to derive appropriate Cyc concepts for each sentence element based on the weights of their syntactic dependency relationships (a, b, c, d, e) and to derive their natural language representations (f). New sentences are composed outside of the Cyc development platform and serve as an output for the module and the whole summarization system. The “Candidate subjects identification” and the “New sentences generation” modules together constitute the knowledge representation step of the summarization process. **Table 4-6** provides the description of Cyc commands used by the “New sentences generation” module. Source code of the module implementation is provided in A.7.

Table 4-6: Description of Cyc commands used by “New sentences generation” module.

ID	Cyc command	Description
(a)	(#$conceptAssociation ?Concept "amod" ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “amod” parameter to derive Cyc concept (?Concept) associations labelled as adjective modifier syntactic dependency relation.
(b)	(#$conceptAssociation ?Concept "pred" ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “pred” parameter to derive Cyc concept (?Concept) associations labelled as predicate syntactic dependency relation.
(c)	(#$conceptAssociation ?Concept "advmod" ?Head-Concept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “advmod” parameter to derive Cyc concept (?Concept) associations labelled as adverb modifier syntactic dependency relation.
(d)	(#$conceptAssociation ?Concept "obj" ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “obj” parameter to derive Cyc

		concept (?Concept) associations labelled as object syntactic dependency relation.
(e)	(#$conceptAssociation ?Concept "subj-obj" ?HeadConcept ?Weight)	Command uses user-defined “#$conceptAssociation” Cyc predicate with “subj-obj” parameter to derive Cyc concept (?Concept) associations labelled as subject-object syntactic dependency relation.
(f)	(#$generate-phrase ?Concept)	Command uses built-in “#$generate-phrase” Cyc predicate to retrieve corresponding natural language representation for a Cyc concept (?Concept).

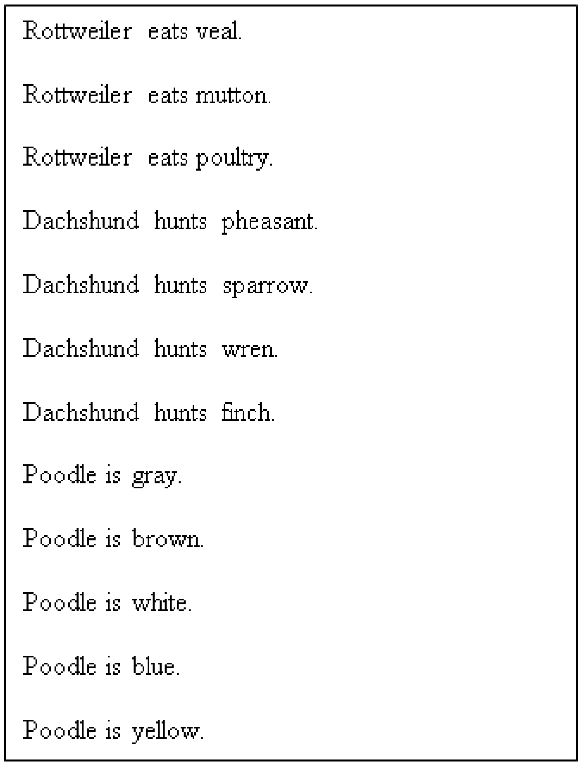
CHAPTER 5

EXPERIMENT AND RESULTS

Several experiments were conducted to highlight different capabilities of proposed abstractive summarization system. The first experiment was performed using artificially generated sentences to illustrate the process of concepts generalization. Other experiments were conducted using real world data parsed from encyclopedia articles that described concepts from various domains.

5.1 Experiments conducted on artificially generated data

Two sets of sentences were created to perform experiments with an artificial data. The first set consisted of simple sentences, only containing subject, predicate and object elements. The sentences are listed in **Figure 5-1**.



Rottweiler eats veal.

Rottweiler eats mutton.

Rottweiler eats poultry.

Dachshund hunts pheasant.

Dachshund hunts sparrow.

Dachshund hunts wren.

Dachshund hunts finch.

Poodle is gray.

Poodle is brown.

Poodle is white.

Poodle is blue.

Poodle is yellow.

Figure 5-1: Artificial sentences with simple structure used for testing.

The results of applying summarization system to the set of described sentences are illustrated in **Table 5-1**.

Table 5-1: Summarization results of applying system to the first set of artifical data.

Sentences expressed by Cyc concepts	Natural language representation
#$Dog #$eatingEvent #$Meat	Dog eating meat
#$Dog #$being #$coloredThing	Dog being colored
#$Dog #$huntingEvent #$Bird	Dog hunting bird

The results highlight the process of concepts generalization. Word "dog" represented by Cyc concept "#$Dog" has not been mentioned in the input text implicitly and has been generalized as an ancestor concept from "Rottweiler", "Dachshund" and

“Poodle” descendant concepts, all being types of dog breeds. **Figure 5-2** illustrates described ancestor-descendant relationships.

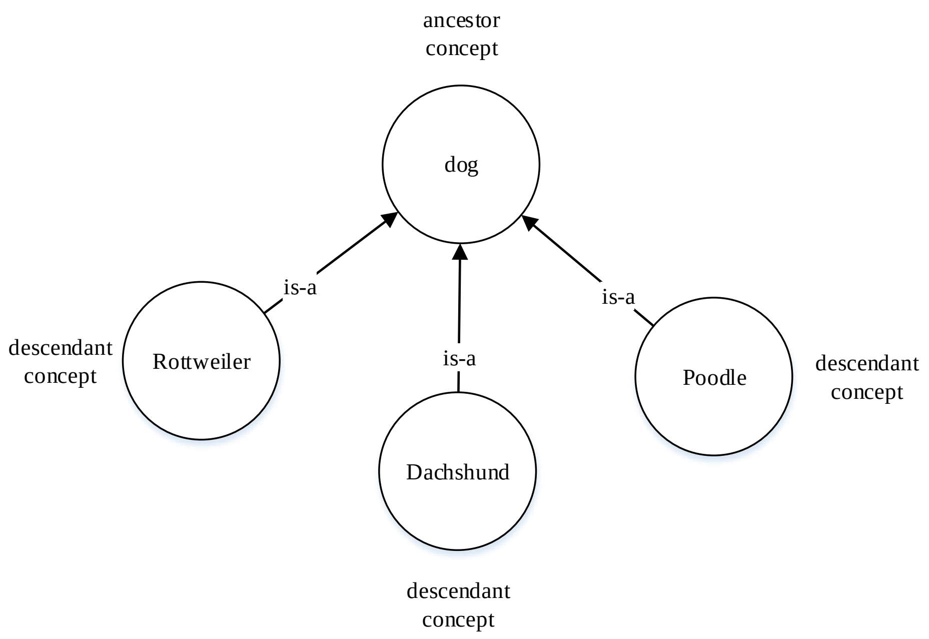


Figure 5-2: “Dog” concept ancestor-descendant relationships in Cyc ontology.

Following this analogy, the word “meat” represented by Cyc concept “#$Meat” was generalized from “veal”, “mutton” and “poultry” descendant concepts, all being types of meats. **Figure 5-3** illustrates described ancestor-descendant relationships.

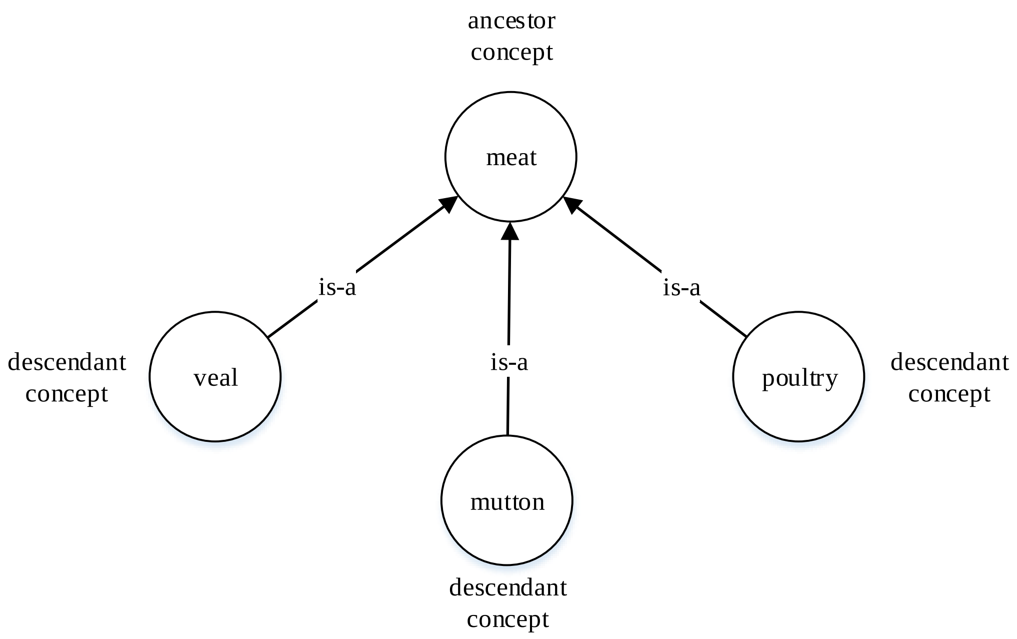


Figure 5-3: “Meat” concept ancestor-descendant relationships in Cyc ontology.

The word “bird” represented by Cyc concept “#$Bird” was generalized from “pheasant”, “sparrow”, “wren” and “finch” descendant concepts, all being types of birds. **Figure 5-4** illustrates described ancestor-descendant relationships.

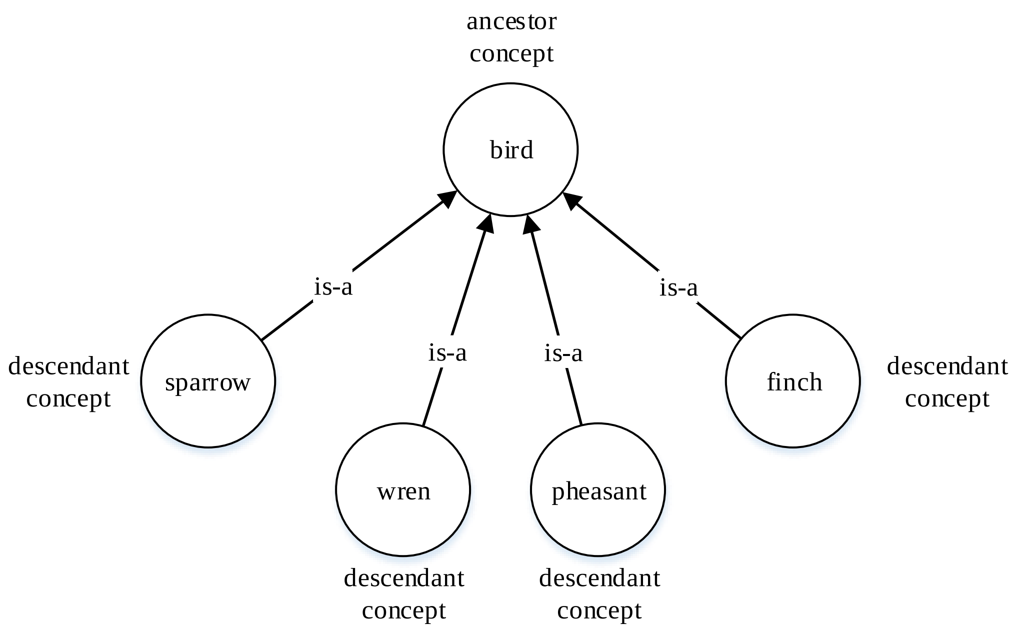


Figure 5-4: “Bird” concept ancestor-descendant relationships in Cyc ontology.

The word “colored” represented by Cyc concept “#$coloredThing” was generalized from “grey”, “white”, “brown”, “blue” and “yellow” descendant concepts, all being different colors. **Figure 5-5** illustrates described ancestor-descendant relationships.

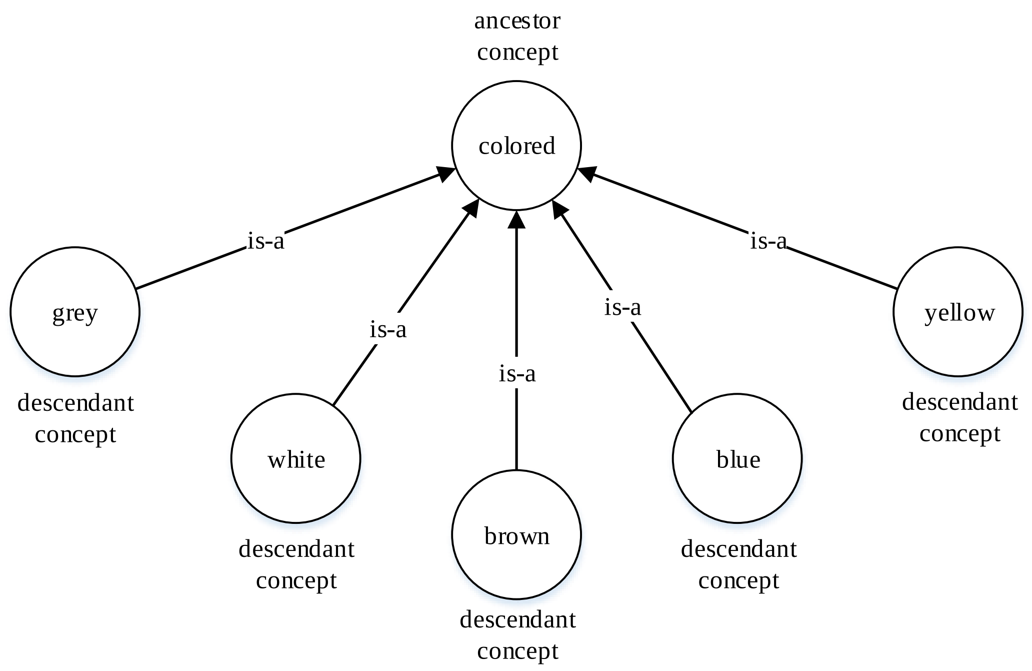
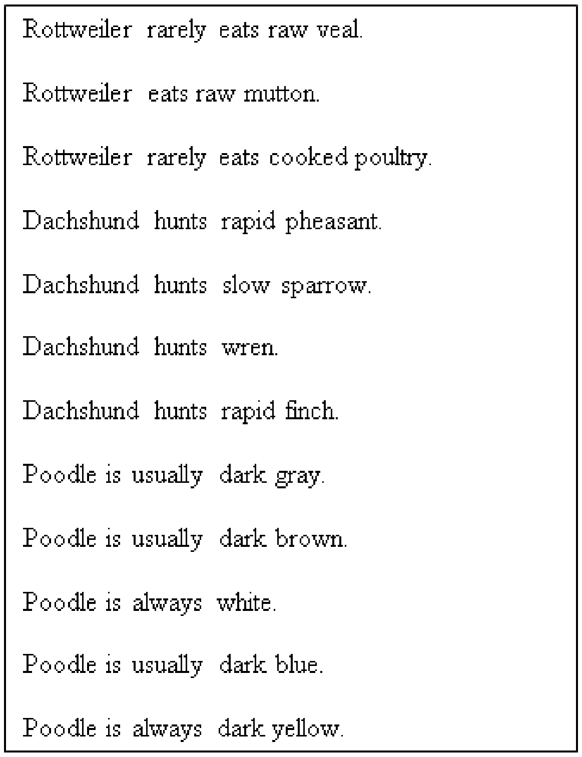


Figure 5-5: “Colored” concept ancestor-descendant relationships in Cyc ontology.

The second set of artificial data consisted of more complex sentences that were composed using adjective and adverb modifiers. Sentences are listed in **Figure 5-6**.



Rottweiler rarely eats raw veal.

Rottweiler eats raw mutton.

Rottweiler rarely eats cooked poultry.

Dachshund hunts rapid pheasant.

Dachshund hunts slow sparrow.

Dachshund hunts wren.

Dachshund hunts rapid finch.

Poodle is usually dark gray.

Poodle is usually dark brown.

Poodle is always white.

Poodle is usually dark blue.

Poodle is always dark yellow.

Figure 5-6: Artificial sentences with complex structure used for testing.

The results of applying summarization system to the set of described sentences are illustrated in **Table 5-2**.

Table 5-2: Summarization results of applying system to the second set of artifical data.

Sentences expressed by Cyc concepts	**Natural language representation**
#$Dog #$rarity #$eatingEvent #$rawThing #$Meat	"Dog rarely eating raw meat"
#$Dog #$normalThing #$being #$darkness #$coloredThing	"Dog normally being dark colored"
#$Dog #$huntingEvent #$highRateEvent #$Bird	"Dog hunting rapid bird"

In addition to exhibiting generalization capabilities ("dog", "meat", "bird" and "colored" concepts), the presented results show that the system is able to create sentences with the structure that extends beyond simple subject-predicate-object triplets utilizing adjective and adverb modifiers (“rarely”, “raw”, “normally”, “dark” and “rapid” concepts).

5.2 Experiments conducted on encyclopedia articles

Several experiments were conducted using real world text data parsed from encyclopedia articles describing various topics.

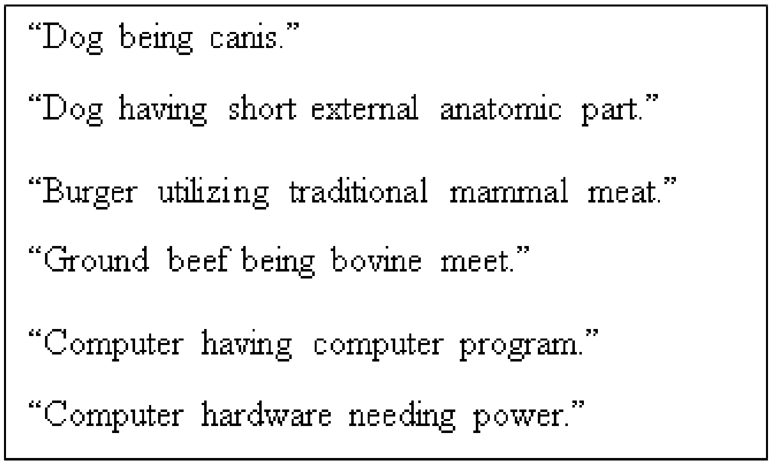
First, the system was applied to Wikipedia articles representing information from different domains and describing domestic dog, personal computer and hamburger. Original articles are illustrated in **Figure B-1**, **Figure B-2**, and **Figure B-3**. Concepts and main topics derived from analyzed articles are summarized in **Table 5-3**.

Table 5-3: Concepts and main topics derived from Wikipedia articles describing various topics.

Article name	Topics		Concepts	
	Cyc micro theory	**Description**	**Cyc concept**	**Natural language**
Dog	#$BiologyMt	Micro theory that describes concepts and relationships related to the field of Biology.	#$Dog	Dog
			#$CanisGenus	Canine
			#$Person	Person
			#$BiologicalSubspecies	Subspecies
	#$NaivePhysicsMt	Micro theory that describes concepts and relationships represented as Naïve physics beliefs and practices.	#$Breeder	Breeder
Hamburger	#$HumanFoodGMt	Micro theory that describes concepts and relationships related to the topic of food normally consumed by humans.	#$Food	Food
			#$Burger	Burger
			#$HamburgerSandwich	Hamburger
			#$GroundBeef	Ground beef
			#$Cheese	Cheese
	#$ProductGMt	Micro theory that describes concepts and relationships related to the broader field of		

		various commodities.		
Computer	#$InformationTerminologyMt	Micro theory that describes concepts and relationships used to describe terminology related to the information technology field.	#$Computer	Computer
			#$ComputerProgrammer	Programmer
			#$outputs	Outputs
			#$ComputerHardwareItem	Computer hardware
			#$ControlDevice	Controller
	#$HumanSocialLifeMt	Micro theory that describes concepts and relationships used to describe various aspects of human social life.		

Some of the new sentences generated by the summarization process are presented in **Figure 5-7**. The structure of each sentence consists of at least subject-predicate-object elements. In addition, auxiliary adjective and adverb modifiers enhance the structure of some sentences. Such enhancement is possible when subject, predicate or object sentence elements have strong subject-adjective, object-adjective and predicate-adverb relationships.


"Dog being canis."

"Dog having short external anatomic part."

"Burger utilizing traditional mammal meat."

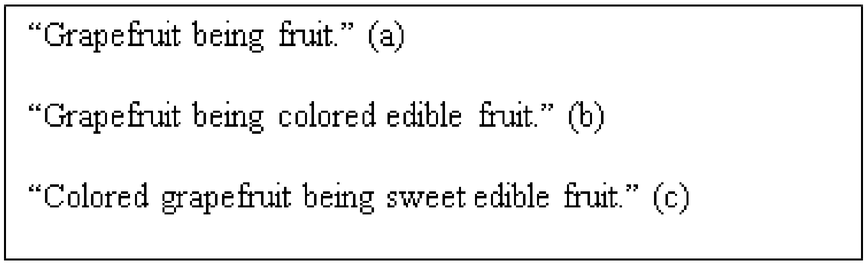
"Ground beef being bovine meet."

"Computer having computer program."

"Computer hardware needing power."

Figure 5-7: New sentences created for Wikipedia articles describing various topics.

Next, an experiment was conducted using multiple encyclopedia articles describing grapefruit. The experiment consisted of three stages, where the number of analyzed articles was increased during each stage. Original articles are illustrated in **Figure B-4**, **Figure B-5**, and **Figure B-6**. Results of this experiment highlight the system's ability to improve summarization results by creating sentences that are more complex when additional data is provided. New sentences created by the system are demonstrated in **Figure 5-8**. The results exhibit the progression of newly created sentences' structure complexity which form simple subject-predicate-object triplet when only a single article was provided as an input (part (a)) to more complex structure extended by the adjective and adverb modifiers when more articles were processed by the algorithm (part (b) and part (c)).


"Grapefruit being fruit." (a)

"Grapefruit being colored edible fruit." (b)

"Colored grapefruit being sweet edible fruit." (c)

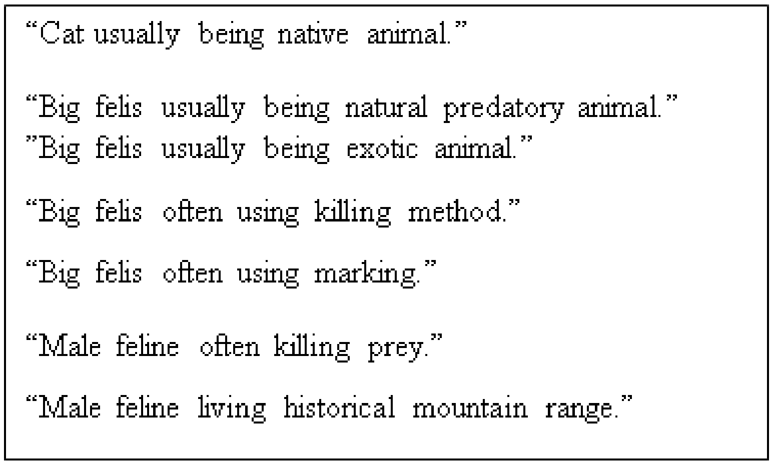
Figure 5-8: Test results of new sentences created for multiple articles about grapefruit; (a) – single article, (b) – two articles, (c) – three articles.

Finally, the system was applied to multiple Wikipedia articles describing different types of felines: cat, tiger, cougar, jaguar and lion. Original articles are illustrated in **Figure B-7**, **Figure B-8**, **Figure B-9**, **Figure B-10**, and **Figure B-11**. **Table 5-4** outlines the main topics and concepts obtained from the analyzed articles.

Table 5-4: Concepts and main topics derived from Wikipedia articles describing felines.

Topics		**Concepts**	
Cyc MT	**Description**	**Cyc term**	**Natural language**
#$BiologyMt	Micro theory that describes concepts and relationships related to the field of Biology.	#$Cat	Cat
		#$DomesticCat	Domestic cat
		#$FelisGenus	Felis
		#$FelidaeFamily	Feline
		#$Animal	Animal
#$HumanSocialLifeMt	Micro theory that describes concepts and relationships used to describe various aspects of human social life.		

Figure 5-9 shows new sentences created by the system as a summary of the analyzed articles. Concepts like “canis”, “mammal meat” and “felis” were generalized by the abstraction process and were not mentioned in the original text. The results of the final experiment illustrate the system’s capability to derive main topics and concepts described in the text and to create new sentences that contain generalized concepts combining information from various parts of the input text.


“Cat usually being native animal.”

“Big felis usually being natural predatory animal.”
”Big felis usually being exotic animal.”

“Big felis often using killing method.”

“Big felis often using marking.”

“Male feline often killing prey.”

“Male feline living historical mountain range.”

Figure 5-9: New sentences created as a summary for multiple articles about felines.

The algorithm proposed in this dissertation yields better results compared to the results reported by [49]. New sentences created by the algorithm have more complex syntactic structure and contain the information fused from different parts of the text. These peculiar properties allow the summary of the text to be more abstractive, informative, and meaningful.

5.3 System performance

The computational complexity of our proposed system is upper bounded by the polynomial expression in the size of the vocabulary of the input documents and therefore, the system is considered to be of the polynomial time complexity. Vocabulary of the document is the number of the unique lemmas contained in the document.

Table 5-5 illustrates the performance of the system when applied to the encyclopedia articles. The experiments were conducted on a machine with 2.0 GHz Intel Xeon E5-2620 CPU and 32 GB of RAM.

Table 5-5: System performance scores using encyclopedia articles.

# of articles	Article name(s)	Source(s)	Vocabulary size (Lemmas)	CPU Time (Seconds)
1	"Dog"	Wikipedia	2087	2751
1	"Computer"	Wikipedia	1604	2245
1	"Hamburger"	Wikipedia	1348	1887
3	"Grapefruit"	Wikipedia, Morton, New World Encyclopedia	1988	2608
5	"Cat" "Tiger" "Cougar" "Jaguar" "Lion"	Wikipedia	5812	6974

CHAPTER 6

CONCLUSION AND FUTURE WORK

This dissertation describes a novel algorithm for creating an abstractive text summary. The task of producing purely abstractive summary of a given text is still considered challenging for people and therefore even more so for the machines. Human experts use the generalization and synthesis of information together with the domain competence to compose abstractive summary of a text. They rephrase the sentences and reformulate the information based on the knowledge deducted from the text. Such a summary becomes more informative and useful since it presents an aggregation and analysis of a given text to distill and provide the knowledge that is more general or not mentioned explicitly [6]. Described aggregation and generalization of the information is not feasible without analyzing the semantics of the text and utilizing the domain knowledge expertise. the analysis of the syntactic structure of the text also takes a significant part in the process of abstractive summarization as it allows representing the derived knowledge as grammatically correct sentences for the user convenience. KBS algorithm described in this dissertation uses Cyc knowledge base and its reasoning engine as a backbone to accommodate these capabilities. Employing the semantic features and the syntactic structure of the text together with the world's largest knowledge base system shows great potential in creating abstractive summaries. The algorithm creates a summary of a given text by composing new sentences that contain the information aggregated from the various parts of the text. The structure of the summary sentences is enhanced from simple subject-predicate-object triplets to a more complex structure by adding the adjective and adverb modifiers. The appropriate modifiers are derived by the analysis of the syntactic relationships of the subjects, predicates and objects in the sentences of the original text.

The contributions of the described algorithm can be summarized as follows:

- Automatically derives main concepts and topics that describe the text;
- Generalizes and synthesizes information derived from the text;
- Creates new sentences using syntactic relations and aggregating information from various parts of the text;
- Enhances the structure of newly created summary sentences to include adjective and adverbs modifiers;
- Uses the world's largest ontology of commonsense knowledge and reasoning engine as a backbone for semantic analysis.

The proposed algorithm has been implemented as a modular pipelined system developed in Python programming language for the testing purposes. The experimental results showed that the algorithm is able to abstract new concepts not mentioned in the text, automatically identify main topics described in the text, and create new sentences that combine the information from different parts of the text. Information synthesis and complex structure of newly created sentences allows the described algorithm to yield better results than the algorithm presented by [49] that is the closest in terms of the functionality.

The algorithm described in this dissertation showed promising results that open a number of the future directions in the area of the knowledge based abstractive text summarization. The first direction is to enhance the domain knowledge representation since the semantic knowledge and reasoning are only limited to functionality and performance of Cyc development platform. At this moment, the algorithm is as powerful as the capabilities of the Cyc knowledge base, which is the largest ontology of commonsense knowledge. For future improvement, the algorithm could use the information derived from the whole World Wide Web as a domain knowledge. This would possess challenging research questions such as information inconsistency and sense disambiguation. In addition, a robust inference engine would be required to process the information correctly and in a timely fashion.

The second future research direction could involve the improvement of the syntactic structure of newly created sentences. Proposed algorithm uses subject-predicate-object triplets enhanced by adjective and adverb modifiers. Although such structure is more complex than the one used in previous research, it still does not resemble the structure of the sentences created by people. Structure of newly created sentences could be improved by using more sophisticated representation of syntactic structure of the sentence. As an example, graph representation of the sentence could capture and preserve more complex relations among words or phrases in a sentence. Using the graph structure as a basis for new sentence creation could yield sentences that have syntactic structures that are more complex.

The third direction for future research could be related to the problem of summary sentences connectedness. At this moment, sentences created by the algorithm as a summary of the text are not conceptually connected to each other. Therefore, the summary overall does not look like a concise abstract of the text. Analyzing the relations and interactions of the main concepts of the text on the document level could help in preserving coherency of the sentences created as a summary. This problem could be approached by representing the whole document as a graph of connected concepts with various relationships among them and then creating new sentences based on these relationships.

The fourth future research direction could be the investigating of the parallelizability of the proposed summarization algorithm. Since algorithm operates on the enormous amounts of data comprised in Cyc knowledge base, its performance could benefit from allowing the algorithm to run on parallel and distributed computing platforms.

Finally, the fifth future research direction could be in developing a universal merit for the evaluation of purely abstractive text summarization algorithms. This improvement is not related directly to the proposed algorithm, but rather to the problem of abstractive text summarization in general. Currently, there is a number of merits that are used to statistically evaluate the performance of extractive summarization algorithms. Abstractive summarization algorithms in contrast are inherently more challenging to evaluate, since they tend to generalize and aggregate information in a given text, thus producing the summary that might not overlap much with the original text. Most of the abstractive summarization approaches try to compare their results to human experts created summaries, which are not always available or costly and time consuming to produce. Thus, developing an automatic and universal merit to evaluate the results of abstractive text summarization algorithms is an interesting and challenging area of future research in the abstractive text summarization.

APPENDIX A

SOURCE CODE

A.1 “Syntactic structure extraction” function

```
def preprocessing(dir):
    import spacy
    nlp = spacy.load('en_core_web_md')
    nouns = []
    nouns_dep = []
    verbs = []
    verbs_dep = []
    adverbs = []
    adverbs_dep = []
    adjectives = []
    adjectives_dep = []
    for filename in os.listdir(dir):
        with open(filename) as file:
            doc = nlp(file.read())
            subj_obj = []
            # preprocess text, attach POS and dependency to each word
            for sent in doc.sents:
                subjects = []
                objects = []
                for word in nlp(sent.text):
                    if word.dep_ == 'nsubj':
                        subjects.append((word.lemma_, word.pos_))
                        assoc = 'nsubj'
                    elif word.dep_ in ['acomp', 'ccomp', 'xcomp', 'dobj', 'iobj',
'pobj', 'attr', 'oprd']:
                        objects.append((word.lemma_, word.pos_))
                        assoc = 'obj'
                    else:
                        assoc = word.dep_
                    if word.pos_ in ['NOUN', 'PROPN']:
                        nouns.append("'"+word.lemma_+"'")
                        nouns_dep.append(("'"+word.lemma_+"'",
("'"+assoc+"'", "'"+word.head.lemma_+"'", "'"+word.head.pos_+"'")))
                    elif word.pos_ == 'VERB':
                        verbs.append("'"+word.lemma_+"'")
                        verbs_dep.append(("'"+word.lemma_+"'",
("'"+assoc+"'", "'"+word.head.lemma_+"'", "'"+word.head.pos_+"'")))
                    elif word.pos_ == 'ADV':
                        adverbs.append("'"+word.lemma_+"'")
                        adverbs_dep.append(("'"+word.lemma_+"'",
("'"+assoc+"'", "'"+word.head.lemma_+"'", "'"+word.head.pos_+"'")))
                    elif word.pos_ == 'ADJ':
                        adjectives.append("'"+word.lemma_+"'")
                        adjectives_dep.append(("'"+word.lemma_+"'",
("'"+assoc+"'", "'"+word.head.lemma_+"'", "'"+word.head.pos_+"'")))
                for sub in subjects:
                    for obj in objects:
                        if sub[1] in ['NOUN', 'PROPN']:
                            subj_obj.append(("'"+sub[0]+"'", ("'subj-
obj'", "'"+obj[0]+"'", "'"+obj[1]+"'")))
            nouns_dep_tot = nouns_dep + subj_obj
    # create a dictionary for each POS counting word and dependency frequencies
    noun_dict = defaultdict(set)
```

```
        for word, deps in collections.Counter(nouns_dep_tot).items():
                noun_dict[word[0]].add(word[1]+(deps,))
        for k, v in collections.Counter(nouns).items():
                noun_dict[k].add(v)
        verb_dict = defaultdict(set)
        for word, deps in collections.Counter(verbs_dep).items():
                verb_dict[word[0]].add(word[1]+(deps,))
        for k, v in collections.Counter(verbs).items():
                verb_dict[k].add(v)
        adj_dict = defaultdict(set)
        for word, deps in collections.Counter(adjectives_dep).items():
                adj_dict[word[0]].add(word[1]+(deps,))
        for k, v in collections.Counter(adjectives).items():
                adj_dict[k].add(v)
        adv_dict = defaultdict(set)
        for word, deps in collections.Counter(adverbs_dep).items():
                adv_dict[word[0]].add(word[1]+(deps,))
        for k, v in collections.Counter(adverbs).items():
                adv_dict[k].add(v)
        # create a dictionary for the whole text, organizing the words by POS and record dependencies
        doc_dict = defaultdict(dict)
        for k, v in noun_dict.items():
                doc_dict['Noun'].update({k : {'weight': [i for i in list(v) if type(i) == int], 'deps': [i for i in
list(v) if type(i) == tuple]}})
        for k, v in verb_dict.items():
                doc_dict['Verb'].update({k : {'weight': [i for i in list(v) if type(i) == int], 'deps': [i for i in
list(v) if type(i) == tuple]}})
        for k, v in adj_dict.items():
                doc_dict['Adjective'].update({k : {'weight': [i for i in list(v) if type(i) == int], 'deps': [i for
i in list(v) if type(i) == tuple]}})
        for k, v in adv_dict.items():
                doc_dict['Adverb'].update({k : {'weight': [i for i in list(v) if type(i) == int], 'deps': [i for i
in list(v) if type(i) == tuple]}})
        return doc_dict
```

A.2 “Mapping words to Cyc concepts” function

```
def mapping(inp_dict):
    from jpype import *
    # packages, classes and method from Java CYC Api
    client = JPackage("com.cyc.kb.client")
    base = JPackage("com.cyc.base")
    fact_impl = client.FactImpl
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
        # for each key (word) and value (frequency count) in input dictionary:
    # use key in a query to map word to CYC concept
    # use value to assign weight to a concept
    for global_POS, global_values in inp_dict.iteritems():
        for word, attributes in global_values.iteritems():
            # keep track of words part-of-speech tags to use them in "denotation" function
            if global_POS == 'Noun':
                global_string = "nounStrings"
            elif global_POS == 'Verb':
                global_string = "verbStrings"
            elif global_POS == 'Adjective':
                global_string = "adjStrings"
            else:
                global_string = "adverbStrings"
            # construct query to map word to CYC concept through "denotation" function
            try:
                denotation_terms = access.converse().converseObject("(query-variable '?TERM '(#$and
(#$denotation ?WORD ?POS ?NUM ?TERM) (#$wordForms ?WORD #${2} {0}) (#$genls ?POS #${1}))
#$InferencePSC)".format(word, global_POS, global_string))
            except:
                print "CYC api error was raised, while mapping word: {0}".format(word)
            if str(denotation_terms) != "NIL":
                # go through each item in result set derived from a query
                for term in set(denotation_terms):
                    # accumulate all weights of the mapped concept in case any words were mapped to it before
                    c_weight = 0
                    try:
                        if '(' in str(term):
                            initial_w = str(access.converse().converseObject("(query-variable '?IWEIGHT
'(#$conceptWeight {0} ?IWEIGHT) #$InferencePSC)".format(str(term).replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$'))))
                        else:
                            initial_w = str((access.converse().converseObject("(query-variable '?IWEIGHT
'(#$conceptWeight #${0} ?IWEIGHT) #$InferencePSC)".format(term))))
                    except:
                        initial_w = "NIL"
                    try:
                        if initial_w != "NIL":
                            c_weight = sum(map(lambda x: float(x), initial_w.strip('()').split()))
                            for j in initial_w.strip('()').split():
                                fact_impl.findOrCreate("(conceptWeight {0} {1})".format(term, j), "BaseKB").delete()
                        fact_impl.findOrCreate("(conceptWeight {0} {1})".format(term, str(attributes['weight'][0] +
float(c_weight))), "BaseKB")
                    except:
                        print "CYC api error was raised, while updating weight for term {0}.".format(term)
                    # map dependency words to CYC concepts
                    # keep track of words part-of-speech tags to use them in "denotation" function
                    for dep_attributes in attributes['deps']:
                        # record only subject, predicate, object and modifier associations types
                        if dep_attributes[0] in ['"nsubj"', '"obj"', '"subj-obj"', '"amod"', '"advmod"']:
                            if dep_attributes[2] in ['"NOUN"', '"PROPN"']:
                                head_string = "nounStrings"
                                head_pos = 'Noun'
                            elif dep_attributes[2] == '"VERB"':
                                head_string = "verbStrings"
                                head_pos = 'Verb'
                            elif dep_attributes[2] == '"ADJ"':
                                head_string = "adjStrings"
                                head_pos = 'Adjective'
                            elif dep_attributes[2] == '"ADV"':
                                head_string = "adverbStrings"
                                head_pos = 'Adverb'
                            # construct query to map word from dependency to CYC concept through "denotation"
function
```

```
                        head_denotation_terms = access.converse().converseObject("(query-variable '?HTERM
'(#$and (#$denotation ?HWORD ?HPOS ?HNUM ?HTERM) (#$wordForms ?HWORD #${2} {0})
(#$genls ?HPOS #${1})) #$InferencePSC)".format(dep_attributes[1], head_pos, head_string))
                        # check if denotation head word is mapped to Cyc Concept
                        if str(head_denotation_terms) != "NIL":
                            # go through each item in result set derived from a query
                            for head_term in set(head_denotation_terms):
                                assoc_weight = 0
                                try:
                                    if '(' in str(term) and '(' in str(head_term):
                                        assoc_init_w = str(access.converse().converseObject("(query-variable '?W
'(#$conceptAssociation {0} {1} {2} ?W) #$InferencePSC)".format(str(term).replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$'), dep_attributes[0], str(head_term).replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$'))))
                                    elif '(' in str(term) and '(' not in str(head_term):
                                        assoc_init_w = str(access.converse().converseObject("(query-variable '?W
'(#$conceptAssociation {0} {1} #${2} ?W) #$InferencePSC)".format(str(term).replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$'), dep_attributes[0], head_term)))
                                    elif '(' not in str(term) and '(' in str(head_term):
                                        assoc_init_w = str(access.converse().converseObject("(query-variable '?W
'(#$conceptAssociation #${0} {1} {2} ?W) #$InferencePSC)".format(term, dep_attributes[0],
str(head_term).replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$'))))
                                    else:
                                        assoc_init_w = str(access.converse().converseObject("(query-variable '?W
'(#$conceptAssociation #${0} {1} #${2} ?W) #$InferencePSC)".format(term, dep_attributes[0],
head_term)))
                                except:
                                    assoc_init_w = "NIL"
                                if assoc_init_w != "NIL":
                                    assoc_weight = sum(map(lambda x: float(x), assoc_init_w.strip('()').split()))
                                    for i in assoc_init_w.strip('()').split():
                                        fact_impl.findOrCreate("(conceptAssociation {0} {1} {2} {3})".format(term,
dep_attributes[0], head_term, i), "BaseKB").delete()
                                total_mapped_weight = (assoc_weight + dep_attributes[3])
                                # use TERM as a parameter to assign dependencies to mapped CYC concept
                                try:
                                    fact_impl.findOrCreate("(conceptAssociation {0} {1} {2} {3})".format(term,
dep_attributes[0], head_term, str(total_mapped_weight)), "BaseKB")
                                except:
                                    print "Association cannot be created in current microtheory."
    return
```

A.3 “Concepts propagation” function

```
def propagation():
    from jpype import *
    # packages, classes and method from Java CYC Api
    query = JPackage("com.cyc.query")
    client = JPackage("com.cyc.kb.client")
    kb = JPackage("com.cyc.kb")
    base = JPackage("com.cyc.base")
    query_factory = query.QueryFactory
    fact_impl = client.FactImpl
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
    # query for CYC concepts that have assigned weights
    q_weight = query_factory.getQuery("(conceptWeight ?TERM1 ?CWEIGHT)")
    res_weight = q_weight.getResultSet()
    while res_weight.next():
        # filter TERM and CWEIGHT variables from query results output
        # TERM - CYC concept to be propagated
        # CWEIGHT - weight of CYC concept to be propagated
        term3 = str(res_weight.getKBObject("?TERM1", kb.KBIndividual))
        cweight = str(res_weight.getKBObject("?CWEIGHT", kb.KBIndividual))
        # generalization step
        # use "min-genls" CYC command to find closest parent of CYC concept to be generalized
        try:
            # use formatting scheme in case CYC concept is composite
            if '(' in term3:
                min_genls = access.converse().converseCycObject("(min-genls '{0})".format(term3.replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$')))
            else:
                min_genls = access.converse().converseCycObject("(min-genls #${0})".format(term3))
        except:
            print "CYC Api error - constant: {0} was not found".format(term3)
        # check if CYC concept was successfully generalized
        if len(min_genls) != 0:
            for i in range(len(min_genls)):
                # output generalized CYC concept
                print "1st level generalized term: {0}".format(min_genls[i])
                d_count = 0
                d_weight = 0
                q_gen_weight = query_factory.getQuery('(conceptDescendants {0} ?WEIGHT
?COUNT)'.format(min_genls[i]))
                res_sum_q_gen = q_gen_weight.getResultSet()
                while res_sum_q_gen.next():
                    try:
                        d_weight = str(res_sum_q_gen.getKBObject("?WEIGHT", kb.KBIndividual))
                        d_count = str(res_sum_q_gen.getKBObject("?COUNT", kb.KBIndividual))
                        fact_impl.findOrCreate("(conceptDescendants {0} {1} {2})".format(min_genls[i],
str(d_weight), str(d_count)), "BaseKB").delete()
                    except:
                        print "CYC Api error while propagating: {0}".format(min_genls[i])
                total_weight = (float(cweight) * 0.1 + float(d_weight))
                total_count = float(d_count) + 1
                # assign accumulated weight of generalized CYC concept (initial weight + propagated weight)
                fact_impl.findOrCreate("(conceptDescendants {0} {1} {2})".format(min_genls[i],
str(total_weight), str(total_count)), "BaseKB")
                # record ancestor-descendant relation
                fact_impl.findOrCreate("(conceptAncestorOf {0} {1})".format(min_genls[i], term3'), "BaseKB")
    return
```

A.4 “Concepts’ weight and relationships accumulation” function

```
def accumulate_descendants():
    from jpype import *
    # packages, classes and method from Java CYC Api
    query = JPackage("com.cyc.query")
    client = JPackage("com.cyc.kb.client")
    kb = JPackage("com.cyc.kb")
    base = JPackage("com.cyc.base")
    query_factory = query.QueryFactory
    fact_impl = client.FactImpl
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
    # query for CYC concepts that have descendants
    concept_descendants_q = query_factory.getQuery("(conceptDescendants ?ANCTERM ?PROPWEIGHT
?DCOUNT)")
    concept_descendants = concept_descendants_q.getResultSet()
    while concept_descendants.next():
        ancestor_concept = str(concept_descendants.getKBObject("?ANCTERM", kb.KBIndividual))
        desc_weight = str(concept_descendants.getKBObject("?PROPWEIGHT", kb.KBIndividual))
        # calculate "descendants percentage" measure = # of concept descendants with weight / total # of
concept descendants
        try:
            if '(' in ancestor_concept:
                ancestor_mapped_desc = access.converse().converseObject("(query-variable '?M
'(#$conceptAncestorOf {0} ?M) #$InferencePSC)".format(ancestor_concept.replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$')))
                ancestor_total_desc = access.converse().converseObject("(query-variable '?T '(#$genls ?T {0})
#$InferencePSC)".format(ancestor_concept.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
            else:
                ancestor_mapped_desc = access.converse().converseObject("(query-variable '?M
'(#$conceptAncestorOf #${0} ?M) #$InferencePSC)".format(ancestor_concept))
                ancestor_total_desc = access.converse().converseObject("(query-variable '?T '(#$genls ?T #${0})
#$InferencePSC)".format(ancestor_concept))
            desc_percentage = float(len(ancestor_mapped_desc)) / float(len(ancestor_total_desc))
        except:
            print "CYC Api error while retrieving descendants for concept: {0}\n".format(ancestor_concept)
            ancestor_mapped_desc = 0
            ancestor_total_desc = 0
            desc_percentage = 0
        # if "descendants percentage" is higher than a threshold then add propagated descendants weight to
initial concept weight
        if desc_percentage > 0.5:
            # query for parent's initial concept weight
            try:
                if '(' in ancestor_concept:
                    init_weight = str(access.converse().converseObject("(query-variable '?WEIGHT
'(#$conceptWeight ({0}) ?WEIGHT) #$InferencePSC '(:max-number
1))".format(ancestor_concept.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))).strip('()')
                else:
                    init_weight = str(access.converse().converseObject("(query-variable '?WEIGHT
'(#$conceptWeight #${0} ?WEIGHT) #$InferencePSC '(:max-number
1))".format(ancestor_concept))).strip('()')
            except:
                print "CYC Api error while retrieving weight for concept: {0}\n".format(ancestor_concept)
                init_weight = "NIL"
            # if parent has concept weight then accumulate it with its descendant propagated weight
            if init_weight != "NIL":
                total_dweight = float(init_weight) + float(desc_weight)
                fact_impl.findOrCreate("(conceptWeight {0} {1})".format(ancestor_concept, str(init_weight)),
"BaseKB").delete()
                fact_impl.findOrCreate("(conceptWeight {0} {1})".format(ancestor_concept, total_dweight),
"BaseKB")
            # if parent does not have concept weight then use its descendants propagated weight
            else:
```

```
            total_dweight = desc_weight
            fact_impl.findOrCreate("(conceptWeight {0} {1})".format(ancestor_concept, str(total_dweight)),
"BaseKB")
        # adding direct associations to propagated ancestors
        q_accum = query_factory.getQuery('(and (conceptAncestorOf {0} ?DESC) (conceptAssociation
?DESC ?ATYPE ?AHEAD ?DESW))'.format(ancestor_concept))
        res_q_accum = q_accum.getResultSet()
        while res_q_accum.next():
            desc_concept = str(res_q_accum.getKBObject("?DESC", kb.KBIndividual))
            desc_level = str(res_q_accum.getKBObject("?LEVEL", kb.KBIndividual))
            a_type = str(res_q_accum.getKBObject("?ATYPE", kb.KBIndividual))
            a_head = str(res_q_accum.getKBObject("?AHEAD", kb.KBIndividual))
            desc_a_weight = str(res_q_accum.getKBObject("?DESW", kb.KBIndividual))
            association_w = 0
            try:
                # handles multi-member concepts
                if '(' in ancestor_concept and '(' in a_head:
                    anc_association_w = str(access.converse().converseObject("(query-variable '?ANCW
'(#$conceptAssociation {0} \"{1}\" {2} ?ANCW) #$InferencePSC)".format(ancestor_concept.replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$'), a_type, a_head.replace(' (', '(').replace(' ', ' #$').replace('(', '
(#$')))).strip('()')
                elif '(' in ancestor_concept and '(' not in a_head:
                    anc_association_w = str(access.converse().converseObject("(query-variable '?ANCW
'(#$conceptAssociation {0} \"{1}\" #${2} ?ANCW) #$InferencePSC)".format(ancestor_concept.replace('
(', '(').replace(' ', ' #$').replace('(', ' (#$'), a_type, a_head))).strip('()')
                elif '(' not in ancestor_concept and '(' in a_head:
                    anc_association_w = str(access.converse().converseObject("(query-variable '?ANCW
'(#$conceptAssociation #${0} \"{1}\" {2} ?ANCW) #$InferencePSC)".format(ancestor_concept, a_type,
a_head.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))).strip('()')
                else:
                    anc_association_w = str(access.converse().converseObject("(query-variable '?ANCW
'(#$conceptAssociation #${0} \"{1}\" #${2} ?ANCW) #$InferencePSC)".format(ancestor_concept, a_type,
a_head))).strip('()')
                if anc_association_w != "NIL":
                    association_w = anc_association_w
                    fact_impl.findOrCreate('(conceptAssociation {0} "{1}" {2} {3})'.format(ancestor_concept,
a_type, a_head, anc_association_w), "BaseKB").delete()
                # use 0.1 scaling for propagation
                p_prop_weight = float(association_w) + 0.1 * float(desc_a_weight)
                # assign propagated weight to parent association
                fact_impl.findOrCreate(
                    '(conceptAssociation {0} "{1}" {2} {3})'.format(ancestor_concept, a_type, a_head,
str(p_prop_weight)), "BaseKB")
            except:
                print "CYC Api error while mapping concept: {0}".format(ancestor_concept)
        # adding indirect associations to propagated ancestors
        q_m_accum = query_factory.getQuery('(and (conceptAncestorOf {0} ?MDESC)
(conceptAssociation ?MTERM ?MATYPE ?MDESC ?MDESW))'.format(ancestor_concept))
        res_q_m_accum = q_m_accum.getResultSet()
        while res_q_m_accum.next():
            m_desc_concept = str(res_q_m_accum.getKBObject("?MDESC", kb.KBIndividual))
            m_desc_level = str(res_q_m_accum.getKBObject("?MLEVEL", kb.KBIndividual))
            m_a_type = str(res_q_m_accum.getKBObject("?MATYPE", kb.KBIndividual))
            m_a_term = str(res_q_m_accum.getKBObject("?MTERM", kb.KBIndividual))
            m_desc_a_weight = str(res_q_m_accum.getKBObject("?MDESW", kb.KBIndividual))
            m_association_w = 0
            try:
                # handles multi-member concepts
                if '(' in ancestor_concept and '(' in m_a_term:
                    m_anc_association_w = str(access.converse().converseObject("(query-variable '?MANCW
'(#$conceptAssociation {2} \"{1}\" {0} ?MANCW) #$InferencePSC)".format(ancestor_concept.replace('
(', '(').replace(' ', ' #$').replace('(', ' (#$'), m_a_type, m_a_term.replace(' (', '(').replace(' ', ' #$').replace('(', '
(#$')))).strip('()')
                elif '(' in ancestor_concept and '(' not in m_a_term:
                    m_anc_association_w = str(access.converse().converseObject("(query-variable '?MANCW
'(#$conceptAssociation #${2} \"{1}\" {0} ?MANCW)
#$InferencePSC)".format(ancestor_concept.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$'), m_a_type,
m_a_term))).strip('()')
                elif '(' not in ancestor_concept and '(' in m_a_term:
                    m_anc_association_w = str(access.converse().converseObject("(query-variable '?MANCW
'(#$conceptAssociation {2} \"{1}\" #${0} ?MANCW) #$InferencePSC)".format(ancestor_concept,
m_a_type, m_a_term.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))).strip('()')
                else:
                    m_anc_association_w = str(access.converse().converseObject("(query-variable '?MANCW
'(#$conceptAssociation #${2} \"{1}\" #${0} ?MANCW) #$InferencePSC)".format(ancestor_concept,
m_a_type, m_a_term))).strip('()')
                if m_anc_association_w != "NIL":
                    m_association_w = m_anc_association_w
                    fact_impl.findOrCreate('(conceptAssociation {2} "{1}" {0} {3})'.format(ancestor_concept,
m_a_type, m_a_term, m_anc_association_w), "BaseKB").delete()
                # use 0.1 scaling for propagation
                m_p_prop_weight = float(m_association_w) + 0.1 * float(m_desc_a_weight)
                # assign propagated weight to parent association
                fact_impl.findOrCreate('(conceptAssociation {2} "{1}" {0} {3})'.format(ancestor_concept,
m_a_type, m_a_term, str(m_p_prop_weight)), "BaseKB")
            except:
                print "CYC Api error while mapping concept: {0}".format(m_a_term)
    return
```

A.5 “Main topics identification” function

```
def top_mts(n):
    from jpype import *
    # packages, classes and method from Java CYC Api
    base = JPackage("com.cyc.base")
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
    mts_list = []
    terms = access.converse().converseObject("(new-cyc-query '(#$and (#$conceptWeight ?T ?W)
(#$definingMt ?T ?MT)) #$InferencePSC)")
    for i in range(len(terms)):
        mts_list.append(str(terms[i][2][1]))
    mtc_dict = defaultdict(set)
    for mt, mtc in Counter(mts_list).items():
        mtc_dict[mt] = mtc
    mts_count = OrderedDict(sorted(mtc_dict.iteritems(), key=operator.itemgetter(1), reverse=True)[:n])
    return mts_count
```

A.6 “Candidate subjects discovery” function

```
def top_subjects(mts, s):
    from jpype import *
    # packages, classes and method from Java CYC Api
    base = JPackage("com.cyc.base")
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
    term_dict = {}
    for mt in mts:
        terms = access.converse().converseObject("(new-cyc-query '(#$and (#$definingMt ?T #${0})
(#$conceptWeight ?T ?W)) #$InferencePSC)".format(mt))
        for t in terms:
            term = str(t[0][1])
            weight = str(t[1][1])
            if term not in term_dict.keys():
                if '(' in term:
                    try:
                        subj_associations = access.converse().converseObject("(cyc-query '(#$conceptAssociation
{0} \"nsubj\" ?SAHEAD ?SAWEIGHT) #$InferencePSC)".format(term.replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$')))
                        tot_associations = access.converse().converseObject("(cyc-query '(#$conceptAssociation
{0} ?ATYPE ?SAHEAD ?SAWEIGHT) #$InferencePSC)".format(term.replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$')))
                    except:
                                                                    subj_associations = 0
                        tot_associations = 0
                else:
                    try:
                        subj_associations = access.converse().converseObject("(cyc-query '(#$conceptAssociation
#${0} \"nsubj\" ?SAHEAD ?SAWEIGHT) #$InferencePSC)".format(term))
                        tot_associations = access.converse().converseObject("(cyc-query '(#$conceptAssociation
#${0} ?ATYPE ?SAHEAD ?SAWEIGHT) #$InferencePSC)".format(term))
                    except:
                        subj_associations = 0
                                                                    tot_associations = 0
                    subj_ratio = float(len(subj_associations)) / float(len(tot_associations))
                rank = (float(weight) * subj_ratio)
                term_dict[term] = rank
    subject_terms = OrderedDict(sorted(term_dict.iteritems(), key=operator.itemgetter(1), reverse=True)[:s])
    return subject_terms
```

A.7 "New sentences generation" function

```
def summarization(path, subjects):
    from jpype import *
    # packages, classes and method from Java CYC Api
    query = JPackage("com.cyc.query")
    kb = JPackage("com.cyc.kb")
    base = JPackage("com.cyc.base")
    query_factory = query.QueryFactory
    cyc_access_mgr = base.CycAccessManager
    access = cyc_access_mgr.getCurrentAccess()
    # clear output file
    open(path, 'w').close()
    # empty dictionary to serve as a final summary
    summary = {}
    # SUBJECT
    # go through subject CYC concepts
    for k, v in subjects.iteritems():
        # find subject CYC concept natural language phrase
        try:
            if '(' in k:
                subj_nl = access.converse().converseString("(generate-phrase '{0})".format(k.replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$')))
            else:
                subj_nl = access.converse().converseString('(generate-phrase #${0})'.format(k))
        except:
            print "CYC Api error when retrieving NL phrase for subject: {0}".format(k)
            subj_nl = ''
        # SUBJECT-ADJECTIVE
                adj_count = {}
                # find all adjective associated with subject/object CYC concepts
                # query for CYC concepts with "amod" dependency type
                if '(' in term:
                        # use formatting scheme in case CYC concept is composite
                        adj_term = query_factory.getQuery('(conceptAssociation ?ADJTERM "amod"
{0} ?ADJW)'.format(term.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
                else:
                        adj_term = query_factory.getQuery('(conceptAssociation ?ADJTERM "amod"
#${0} ?ADJW)'.format(term))
                try:
                        adj_term_res = adj_term.getResultSet()
                except:
                        print 'CYC Api error when finding adjective for term: {0}'.format(term)
                while adj_term_res.next():
                        # filter TERM1 and W1 variables from query results output
                        # TERM1 - adjective CYC concept
                        # W1 - adjective dependency weight
                        adj = str(adj_term_res.getKBObject("?ADJTERM", kb.KBIndividual))
                        adj_dep_w = str(adj_term_res.getKBObject("?ADJW", kb.KBIndividual))
                        # record adjective weight times its dependency weight
                        adj_count[adj] = float(adj_dep_w)
                if len(adj_count) != 0:
                        top_adjective = dict(sorted(adj_count.iteritems(), key=operator.itemgetter(1),
reverse=True)[:1])
                        subj_adj_term = top_adjective.keys()[0]
                        subj_adj_weight = top_adjective.values()[0]
                        # derive natural language phrase of adjective CYC concept
                        try:
                                if '(' in subj_adj_term:
                                        subj_adj_nl = access.converse().converseString("(generate-
phrase '{0})".format(subj_adj_term. replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
                                else:
                                        subj_adj_nl = access.converse().converseString('(generate-
phrase #${0})'.format(subj_adj_term))
                        except:
                                print "CYC Api error when retrieving NL phrase for adjective:
{0}".format(subj_adj_term)
                                subj_adj_nl = ''
                else:
                        subj_adj_weight = 0
                        subj_adj_term = None
                        subj_adj_nl = ''
        # PREDICATE
        # query for CYC concepts with "nsubj" dependency type
        pred_count = {}
        if '(' in k:
          try:
            # use formatting scheme in case CYC concept is composite
            pred_term_query = query_factory.getQuery('(conceptAssociation {0} "nsubj" ?PTERM
?PW)'.format(k.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
          except:
            print "CYC Api error when finding term: {0}".format(k)
            pred_term_query = 'NIL'
        else:
          pred_term_query = query_factory.getQuery('(conceptAssociation #${0} "nsubj" ?PTERM
?PW)'.format(k))
        pred_term_res = pred_term_query.getResultSet()
        while pred_term_res.next():
          # filter TERM1 and W1 variables from query results output
          # TERM1 - predicate CYC concept
          # W1 - predicate dependency weight
          pred = str(pred_term_res.getKBObject("?PTERM", kb.KBIndividual))
          pred_dep_w = str(pred_term_res.getKBObject("?PW", kb.KBIndividual))
          # record predicate weight times its dependency weight
          pred_count[pred] = float(pred_dep_w)
        top_predicate = OrderedDict(sorted(pred_count.iteritems(), key=operator.itemgetter(1),
reverse=True)[:5])
        for pred_keys, pred_values in top_predicate.iteritems():
          # generate natural language phrase for predicate with strongest (highest weight) relation
          if '(' in pred_keys:
            predicate_nl = access.converse().converseString("(generate-phrase
'{0})".format(pred_keys.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
          else:
            predicate_nl = access.converse().converseString('(generate-phrase #${0})'.format(pred_keys))
          # PREDICATE-ADVERB
          # find adverb CYC concepts assotiated with predicates concepts
                        if '(' in pred_keys:
                                adv_query = query_factory.getQuery('(conceptAssociation
?ADVTERM "advmod" {0} ?ADVW)'.format(pred_keys.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
                        else:
                                adv_query = query_factory.getQuery('(conceptAssociation
?ADVTERM "advmod" #${0} ?ADVW)'.format(pred_keys))
                        adv_query_res = adv_query.getResultSet()
                        adv_count = {}
                        while adv_query_res.next():
                                # filter TERM1 and W1 variables from query results output
                                # TERM1 - adverb CYC concept
                                # W1 - adverb dependency weight
                                adv = str(adv_query_res.getKBObject("?ADVTERM",
kb.KBIndividual))
                                adv_dep_w = str(adv_query_res.getKBObject("?ADVW",
kb.KBIndividual))
                                # record adverb weight times its dependency weight
                                adv_count[adv] = float(adv_dep_w)
                        if len(adv_count) != 0:
                                top_adverb = dict(sorted(adv_count.iteritems(),
key=operator.itemgetter(1), reverse=True)[:1])
                                pred_adv_term = top_adverb.keys()[0]
```

```
                                        pred_adv_weight = top_adverb.values()[0]
                                        try:
                                                if '(' in pred_adv_term:
                                                        pred_adv_nl =
access.converse().converseString("(generate-phrase '{0})".format(pred_adv_term.replace(' (', '(').replace(' ',
' #$').replace('(', ' (#$')))
                                                else:
                                                        pred_adv_nl =
access.converse().converseString('(generate-phrase #${0})'.format(pred_adv_term))
                                        except:
                                                print "Natural language word for adverb '{0}' cannot be
derived.".format(pred_adv_term)
                                                pred_adv_nl = ''
                                else:
                                        print "No adverb was found."
                                        pred_adv_weight = 0
                                        pred_adv_term = None
                                        pred_adv_nl = ''
        # OBJECT
        # check all possible object associations
        obj_count = {}
        # find objects concepts associated with predicates
        if '(' in pred_keys:
           try:
              # use formatting scheme in case CYC concept is composite
              q_obj = query_factory.getQuery('(conceptAssociation ?OTERM "obj" {0}
?OW)'.format(pred_keys.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
           except:
              print "CYC Api error when finding object for term: {0} via 'dobj'.".format(pred_keys)
              q_obj = 'NIL'
        else:
           q_obj = query_factory.getQuery('(conceptAssociation ?OTERM "obj" #${0}
?OW)'.format(pred_keys))
        q_obj_res = q_obj.getResultSet()
        # keep track of all objects associated with predicates
        while q_obj_res.next():
           obj = str(q_obj_res.getKBObject("?OTERM", kb.KBIndividual))
           obj_dep_w = str(q_obj_res.getKBObject("?OW", kb.KBIndividual))
           # find subject-object relation weight
           try:
              if '(' in k and '(' in obj:
                 subj_obj_w = str(access.converse().converseObject("(query-variable
'?SOW'(#$conceptAssociation {0} \"subj-obj\" {1} ?SOW) #$InferencePSC)".format(str(k).replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$'), str(obj).replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))).strip('()')
              elif '(' in k and '(' not in obj:
                 subj_obj_w = str(access.converse().converseObject("(query-variable
'?SOW'(#$conceptAssociation {0} \"subj-obj\" #${1} ?SOW) #$InferencePSC)".format(str(k).replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$'), obj))).strip('()')
              elif '(' not in k and '(' in obj:
                 subj_obj_w = str(access.converse().converseObject("(query-variable
'?SOW'(#$conceptAssociation #${0} \"subj-obj\" {1} ?SOW) #$InferencePSC)".format(k, str(obj).replace('
(', '(').replace(' ', ' #$').replace('(', ' (#$')))).strip('()')
              else:
                 subj_obj_w = str(access.converse().converseObject("(query-variable
'?SOW'(#$conceptAssociation #${0} \"subj-obj\" #${1} ?SOW) #$InferencePSC)".format(k,
obj))).strip('()')
```

```
            except:
                subj_obj_w = 0
            if subj_obj_w != "NIL":
                obj_rank = ((sum(map(lambda x: float(x), str(subj_obj_w).split()))) + float(obj_dep_w))
            else:
                obj_rank = float(obj_obj_rankdep_w)
                                                obj_count[obj] = float(obj_rank)
        if len(obj_count) != 0:
            top_object = OrderedDict(sorted(obj_count.iteritems(), key=operator.itemgetter(1),
reverse=True)[:5])
            for obj_keys, obj_values in top_object.iteritems():
                try:
                    if '(' in obj_keys:
                        object_nl = access.converse().converseString("(generate-phrase
'{0})".format(obj_keys.replace(' (', '(').replace(' ', ' #$').replace('(', ' (#$')))
                    else:
                        object_nl = access.converse().converseString('(generate-phrase #${0})'.format(obj_keys))
                except:
                    print "CYC Api error when retrieving NL phrase for object: {0}".format(obj_keys)
                    object_nl = ''
                                                # OBJECT-ADJECTIVE
                adj_count = {}
                                                # find all adjective associated with subject/object CYC
concepts
                                                # query for CYC concepts with "amod" dependency type
                                                if '(' in term:
                                                        # use formatting scheme in case CYC concept is
composite
                                                        adj_term =
query_factory.getQuery('(conceptAssociation ?ADJTERM "amod" {0} ?ADJW)'.format(term.replace(' (',
'(').replace(' ', ' #$').replace('(', ' (#$')))
                                                else:
                                                        adj_term =
query_factory.getQuery('(conceptAssociation ?ADJTERM "amod" #${0} ?ADJW)'.format(term))
                                                try:
                                                        adj_term_res = adj_term.getResultSet()
                                                except:
                                                        print 'CYC Api error when finding adjective for term:
{0}'.format(term)
                                                while adj_term_res.next():
                                                        # filter TERM1 and W1 variables from query results
output
                                                        # TERM1 - adjective CYC concept
                                                        # W1 - adjective dependency weight
                                                        adj = str(adj_term_res.getKBObject("?ADJTERM",
kb.KBIndividual))
                                                        adj_dep_w =
str(adj_term_res.getKBObject("?ADJW", kb.KBIndividual))
                                                        # record adjective weight times its dependency
weight
                                                        adj_count[adj] = float(adj_dep_w)
                                                if len(adj_count) != 0:
                                                        top_adjective = dict(sorted(adj_count.iteritems(),
key=operator.itemgetter(1), reverse=True)[:1])
                                                        obj_adj_term = top_adjective.keys()[0]
                                                        obj_adj_weight = top_adjective.values()[0]
```

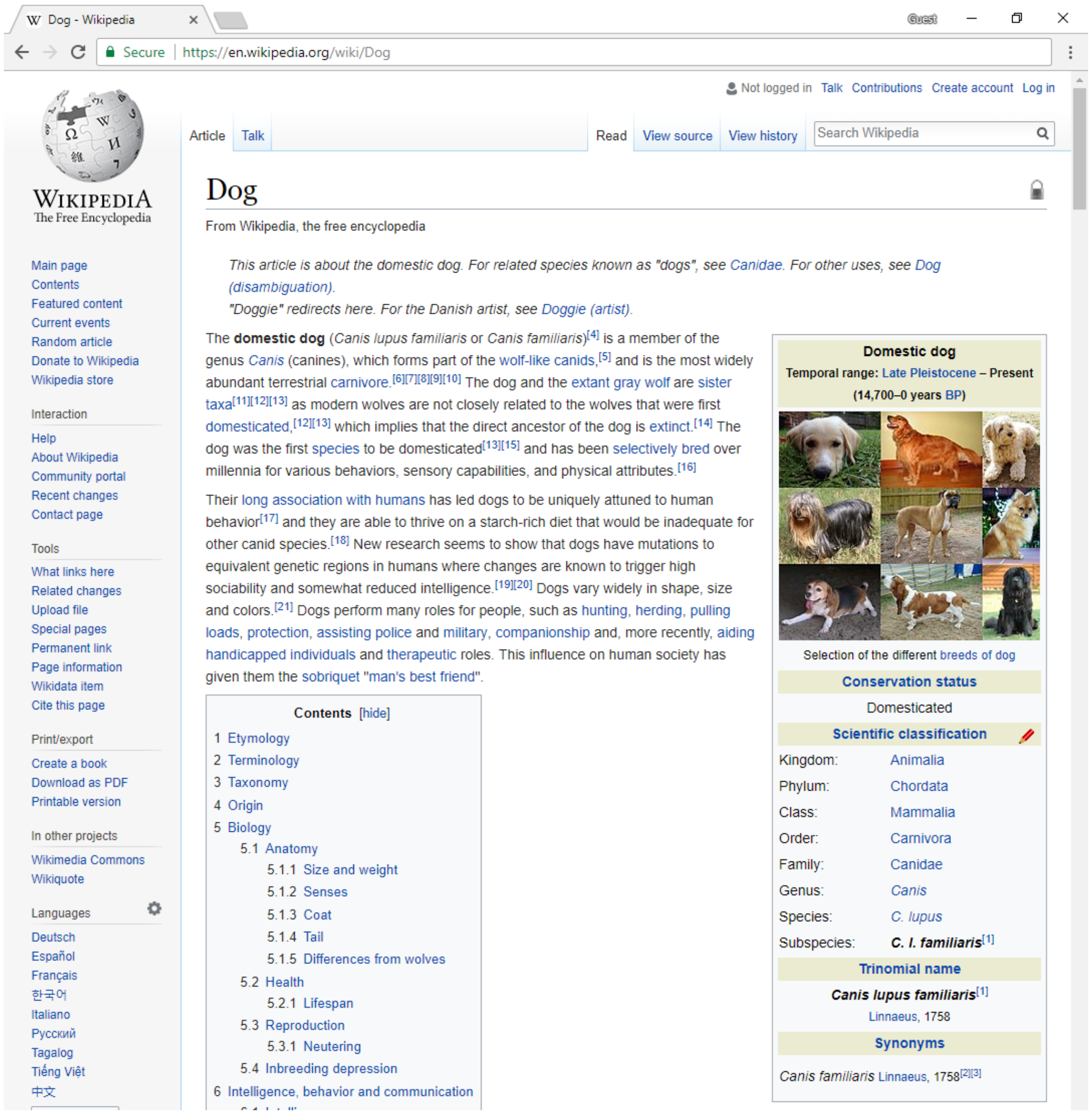
```
                                        # derive natural language phrase of adjective CYC
concept
                                        try:
                                            if '(' in obj_adj_term:
                                                obj_adj_nl =
access.converse().converseString("(generate-phrase '{0})".format(obj_adj_term. replace(' (', '(').replace(' ', '
#$').replace('(', ' (#$')))
                                            else:
                                                obj_adj_nl =
access.converse().converseString('(generate-phrase #${0})'.format(obj_adj_term))
                                        except:
                                            print "CYC Api error when retrieving NL
phrase for adjective: {0}".format(obj_adj_term)
                                            obj_adj_nl = ''
                                    else:
                                        obj_adj_weight = 0
                                        obj_adj_term = None
                                        obj_adj_nl = ''
                # SUMMARY
                # record each Subject - Subject-Adjective - Predicate - Predicate-Adverb - Object - Object-
Adjective
                # into an output file as a newly created sentence
                with open(path, 'a') as f:
                  f.write("{0} / {1} | {2} / {3} | {4} / {5} | {6} / {7} | {8} / {9} | {10} / {11}\n{12} | {13} |
{14} | {15} | {16} | {17}\n\n".format(subj_adj_term, subj_adj_weight, k, v, pred_adv_term,
pred_adv_weight, pred_keys, pred_values, obj_adj_term, obj_adj_weight, obj_keys, obj_values,
subj_adj_nl, subj_nl, pred_adv_nl, predicate_nl, obj_adj_nl, object_nl))
            else:
              obj_values = 0
              obj_keys = None
              object_nl = ''
              obj_adj_term = None
              obj_adj_weight = 0
              obj_adj_nl = ''
              with open(path, 'a') as f:
                                    f.write("{0} / {1} | {2} / {3} | {4} / {5} | {6} / {7} | {8} / {9} |
{10} / {11}\n{12} | {13} | {14} | {15} | {16} | {17}\n\n".format(subj_adj_term, subj_adj_weight, k, v,
pred_adv_term, pred_adv_weight, pred_keys, pred_values, obj_adj_term, obj_adj_weight, obj_keys,
obj_values, subj_adj_nl, subj_nl, pred_adv_nl, predicate_nl, obj_adj_nl, object_nl))
  return
```

APPENDIX B

DOCUMENTS USED FOR TESTING

B.1 "Dog" Wikipedia article.

The article was accessed in March 2018.



Dog - Wikipedia

Guest

Secure | https://en.wikipedia.org/wiki/Dog

Not logged in Talk Contributions Create account Log in

WIKIPEDIA
The Free Encyclopedia

Article Talk Read View source View history Search Wikipedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikiquote

Languages
Deutsch
Español
Français
한국어
Italiano
Русский
Tagalog
Tiếng Việt
中文

Dog

From Wikipedia, the free encyclopedia

This article is about the domestic dog. For related species known as "dogs", see Canidae. For other uses, see Dog (disambiguation).

"Doggie" redirects here. For the Danish artist, see Doggie (artist).

The **domestic dog** (*Canis lupus familiaris* or *Canis familiaris*)[4] is a member of the genus *Canis* (canines), which forms part of the wolf-like canids,[5] and is the most widely abundant terrestrial carnivore.[6][7][8][9][10] The dog and the extant gray wolf are sister taxa[11][12][13] as modern wolves are not closely related to the wolves that were first domesticated,[12][13] which implies that the direct ancestor of the dog is extinct.[14] The dog was the first species to be domesticated[13][15] and has been selectively bred over millennia for various behaviors, sensory capabilities, and physical attributes.[16]

Their long association with humans has led dogs to be uniquely attuned to human behavior[17] and they are able to thrive on a starch-rich diet that would be inadequate for other canid species.[18] New research seems to show that dogs have mutations to equivalent genetic regions in humans where changes are known to trigger high sociability and somewhat reduced intelligence.[19][20] Dogs vary widely in shape, size and colors.[21] Dogs perform many roles for people, such as hunting, herding, pulling loads, protection, assisting police and military, companionship and, more recently, aiding handicapped individuals and therapeutic roles. This influence on human society has given them the sobriquet "man's best friend".

Contents [hide]

Domestic dog

Temporal range: Late Pleistocene – Present (14,700–0 years BP)

Selection of the different breeds of dog

Conservation status

Domesticated

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Family:	Canidae
Genus:	*Canis*
Species:	*C. lupus*
Subspecies:	***C. l. familiaris***[1]

Trinomial name

Canis lupus familiaris[1]
Linnaeus, 1758

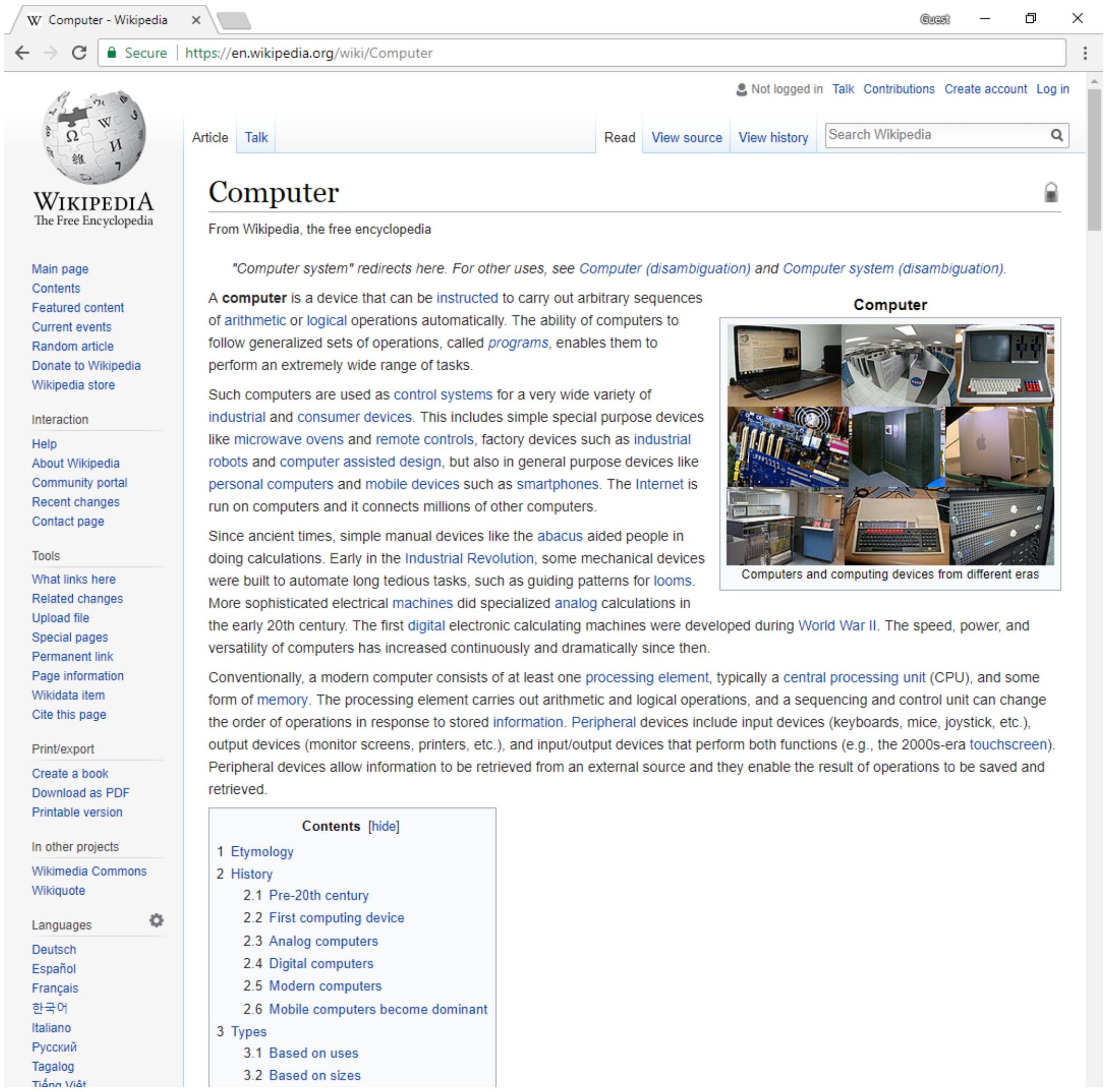
Synonyms

Canis familiaris Linnaeus, 1758[2][3]

Figure B-1: Screenshot of the first page of "Dog" Wikipedia article.

B.2 "Computer" Wikipedia article.

The article was accessed in March 2018.



Computer

From Wikipedia, the free encyclopedia

"Computer system" redirects here. For other uses, see Computer (disambiguation) and Computer system (disambiguation).

A **computer** is a device that can be instructed to carry out arbitrary sequences of arithmetic or logical operations automatically. The ability of computers to follow generalized sets of operations, called *programs*, enables them to perform an extremely wide range of tasks.

Such computers are used as control systems for a very wide variety of industrial and consumer devices. This includes simple special purpose devices like microwave ovens and remote controls, factory devices such as industrial robots and computer assisted design, but also in general purpose devices like personal computers and mobile devices such as smartphones. The Internet is run on computers and it connects millions of other computers.

Since ancient times, simple manual devices like the abacus aided people in doing calculations. Early in the Industrial Revolution, some mechanical devices were built to automate long tedious tasks, such as guiding patterns for looms. More sophisticated electrical machines did specialized analog calculations in the early 20th century. The first digital electronic calculating machines were developed during World War II. The speed, power, and versatility of computers has increased continuously and dramatically since then.

Conventionally, a modern computer consists of at least one processing element, typically a central processing unit (CPU), and some form of memory. The processing element carries out arithmetic and logical operations, and a sequencing and control unit can change the order of operations in response to stored information. Peripheral devices include input devices (keyboards, mice, joystick, etc.), output devices (monitor screens, printers, etc.), and input/output devices that perform both functions (e.g., the 2000s-era touchscreen). Peripheral devices allow information to be retrieved from an external source and they enable the result of operations to be saved and retrieved.

Computer

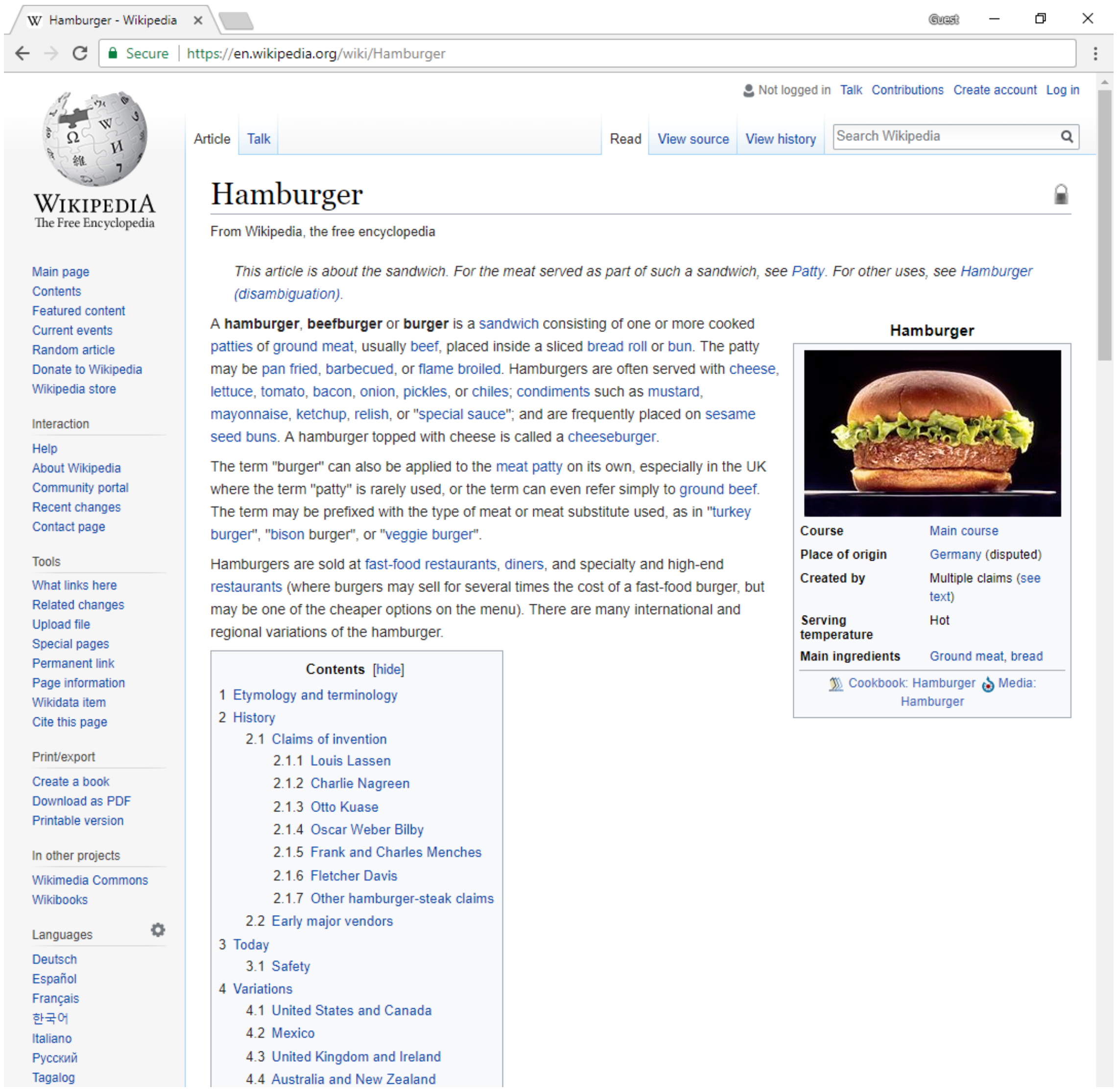
Computers and computing devices from different eras

Contents [hide]

Figure B-2: Screenshot of the first page of "Computer" Wikipedia article.

B.3 “Hamburger” Wikipedia article.

The article was accessed in March 2018.



Hamburger - Wikipedia

Secure | https://en.wikipedia.org/wiki/Hamburger

Not logged in Talk Contributions Create account Log in

Article Talk Read View source View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikibooks

Languages
Deutsch
Español
Français
한국어
Italiano
Русский
Tagalog

Hamburger

From Wikipedia, the free encyclopedia

This article is about the sandwich. For the meat served as part of such a sandwich, see Patty. For other uses, see Hamburger (disambiguation).

A **hamburger**, **beefburger** or **burger** is a sandwich consisting of one or more cooked patties of ground meat, usually beef, placed inside a sliced bread roll or bun. The patty may be pan fried, barbecued, or flame broiled. Hamburgers are often served with cheese, lettuce, tomato, bacon, onion, pickles, or chiles; condiments such as mustard, mayonnaise, ketchup, relish, or "special sauce"; and are frequently placed on sesame seed buns. A hamburger topped with cheese is called a cheeseburger.

The term "burger" can also be applied to the meat patty on its own, especially in the UK where the term "patty" is rarely used, or the term can even refer simply to ground beef. The term may be prefixed with the type of meat or meat substitute used, as in "turkey burger", "bison burger", or "veggie burger".

Hamburgers are sold at fast-food restaurants, diners, and specialty and high-end restaurants (where burgers may sell for several times the cost of a fast-food burger, but may be one of the cheaper options on the menu). There are many international and regional variations of the hamburger.

Hamburger

Course	Main course
Place of origin	Germany (disputed)
Created by	Multiple claims (see text)
Serving temperature	Hot
Main ingredients	Ground meat, bread

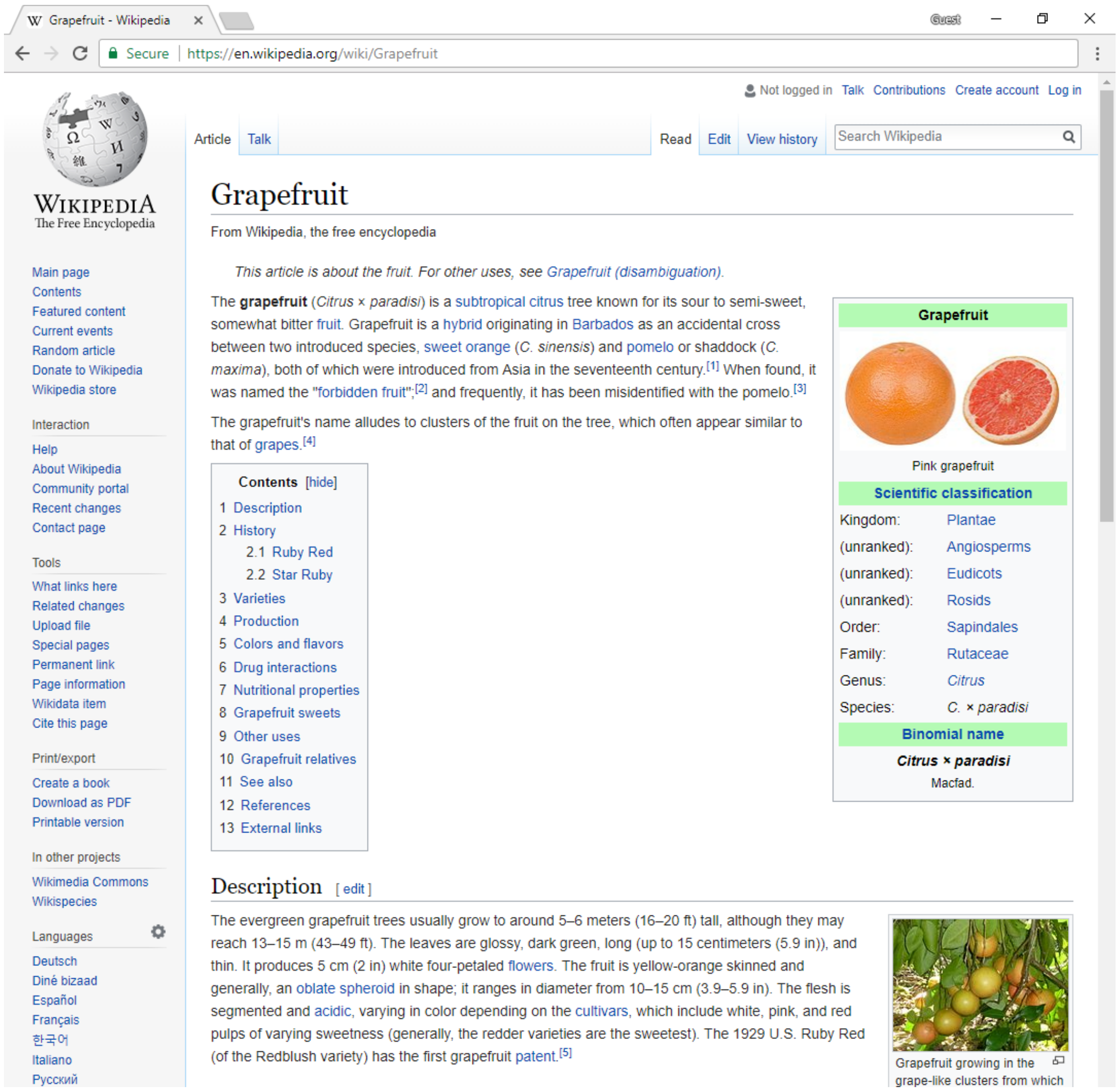
Cookbook: Hamburger Media: Hamburger

Contents [hide]

Figure B-3: Screenshot of the first page of “Hamburger” Wikipedia article.

B.4 "Grapefruit" Wikipedia article.

The article was accessed in March 2018.



W Grapefruit - Wikipedia ×

Guest

Secure | https://en.wikipedia.org/wiki/Grapefruit

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikispecies

Languages
Deutsch
Diné bizaad
Español
Français
한국어
Italiano
Русский

Grapefruit

From Wikipedia, the free encyclopedia

This article is about the fruit. For other uses, see Grapefruit (disambiguation).

The **grapefruit** (*Citrus × paradisi*) is a subtropical citrus tree known for its sour to semi-sweet, somewhat bitter fruit. Grapefruit is a hybrid originating in Barbados as an accidental cross between two introduced species, sweet orange (*C. sinensis*) and pomelo or shaddock (*C. maxima*), both of which were introduced from Asia in the seventeenth century.[1] When found, it was named the "forbidden fruit";[2] and frequently, it has been misidentified with the pomelo.[3]

The grapefruit's name alludes to clusters of the fruit on the tree, which often appear similar to that of grapes.[4]

Contents [hide]

Grapefruit

Pink grapefruit

Scientific classification

Kingdom:	Plantae
(unranked):	Angiosperms
(unranked):	Eudicots
(unranked):	Rosids
Order:	Sapindales
Family:	Rutaceae
Genus:	*Citrus*
Species:	*C. × paradisi*

Binomial name

Citrus × paradisi
Macfad.

Description [edit]

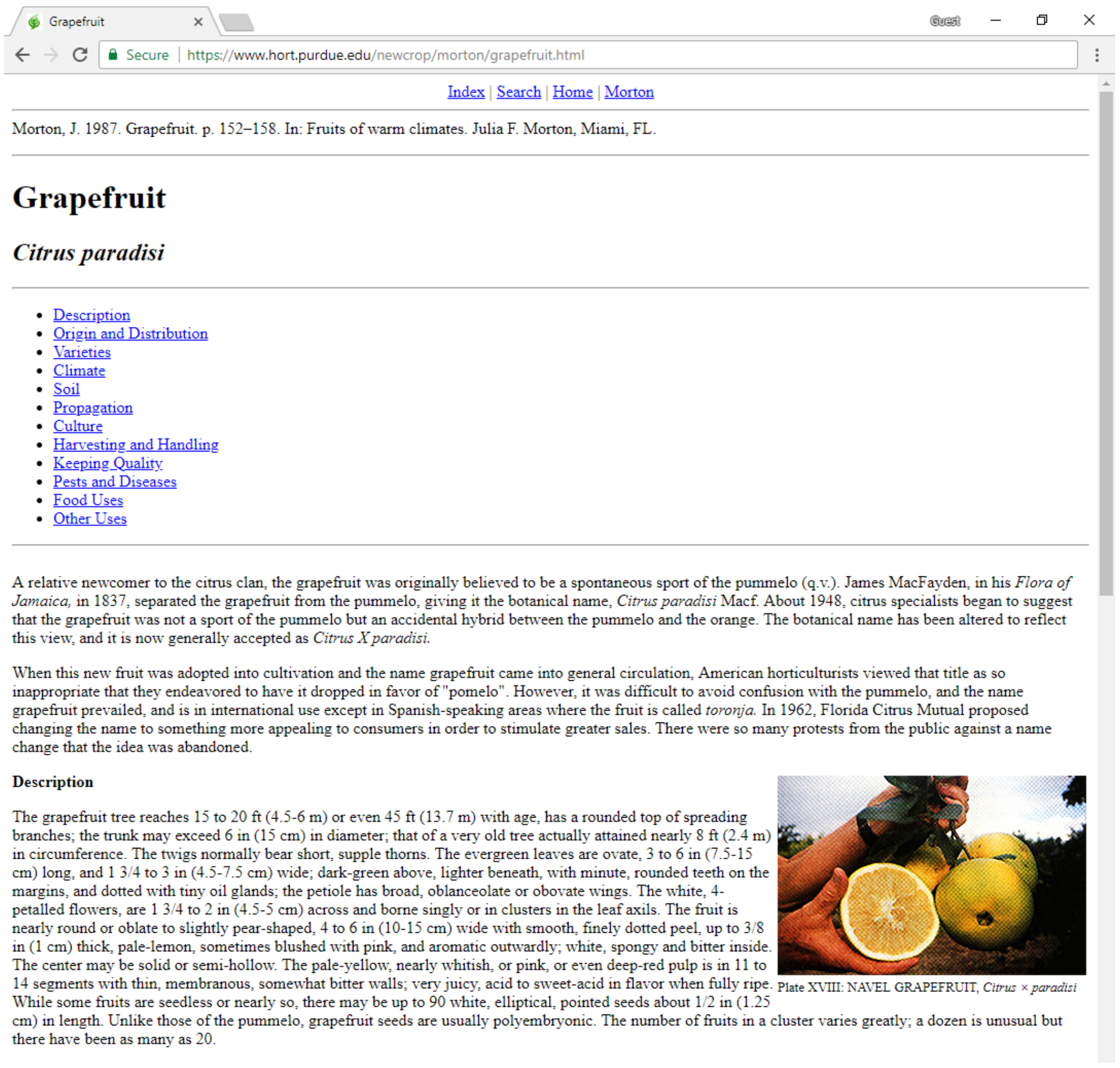
The evergreen grapefruit trees usually grow to around 5–6 meters (16–20 ft) tall, although they may reach 13–15 m (43–49 ft). The leaves are glossy, dark green, long (up to 15 centimeters (5.9 in)), and thin. It produces 5 cm (2 in) white four-petaled flowers. The fruit is yellow-orange skinned and generally, an oblate spheroid in shape; it ranges in diameter from 10–15 cm (3.9–5.9 in). The flesh is segmented and acidic, varying in color depending on the cultivars, which include white, pink, and red pulps of varying sweetness (generally, the redder varieties are the sweetest). The 1929 U.S. Ruby Red (of the Redblush variety) has the first grapefruit patent.[5]

Grapefruit growing in the grape-like clusters from which

Figure B-4: Screenshot of the first page of "Grapefruit" Wikipedia article.

B.5 “Grapefruit” Morton encyclopedia article.

The article was accessed in March 2018.



Index | Search | Home | Morton

Morton, J. 1987. Grapefruit. p. 152–158. In: Fruits of warm climates. Julia F. Morton, Miami, FL.

Grapefruit

Citrus paradisi

A relative newcomer to the citrus clan, the grapefruit was originally believed to be a spontaneous sport of the pummelo (q.v.). James MacFayden, in his *Flora of Jamaica,* in 1837, separated the grapefruit from the pummelo, giving it the botanical name, *Citrus paradisi* Macf. About 1948, citrus specialists began to suggest that the grapefruit was not a sport of the pummelo but an accidental hybrid between the pummelo and the orange. The botanical name has been altered to reflect this view, and it is now generally accepted as *Citrus X paradisi.*

When this new fruit was adopted into cultivation and the name grapefruit came into general circulation, American horticulturists viewed that title as so inappropriate that they endeavored to have it dropped in favor of "pomelo". However, it was difficult to avoid confusion with the pummelo, and the name grapefruit prevailed, and is in international use except in Spanish-speaking areas where the fruit is called *toronja.* In 1962, Florida Citrus Mutual proposed changing the name to something more appealing to consumers in order to stimulate greater sales. There were so many protests from the public against a name change that the idea was abandoned.

Description

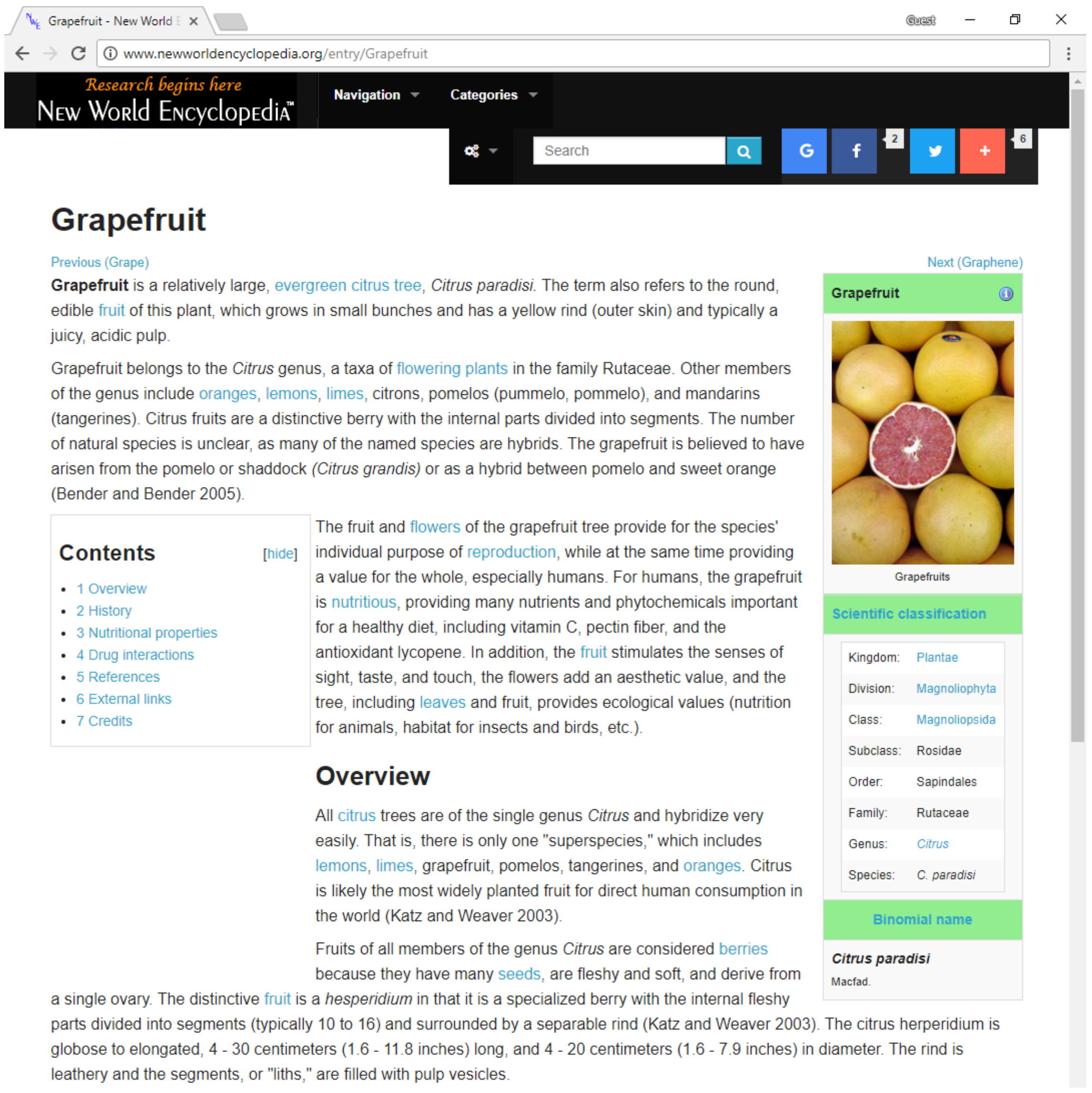
Plate XVIII: NAVEL GRAPEFRUIT, *Citrus × paradisi*

The grapefruit tree reaches 15 to 20 ft (4.5-6 m) or even 45 ft (13.7 m) with age, has a rounded top of spreading branches; the trunk may exceed 6 in (15 cm) in diameter; that of a very old tree actually attained nearly 8 ft (2.4 m) in circumference. The twigs normally bear short, supple thorns. The evergreen leaves are ovate, 3 to 6 in (7.5-15 cm) long, and 1 3/4 to 3 in (4.5-7.5 cm) wide; dark-green above, lighter beneath, with minute, rounded teeth on the margins, and dotted with tiny oil glands; the petiole has broad, oblanceolate or obovate wings. The white, 4-petalled flowers, are 1 3/4 to 2 in (4.5-5 cm) across and borne singly or in clusters in the leaf axils. The fruit is nearly round or oblate to slightly pear-shaped, 4 to 6 in (10-15 cm) wide with smooth, finely dotted peel, up to 3/8 in (1 cm) thick, pale-lemon, sometimes blushed with pink, and aromatic outwardly; white, spongy and bitter inside. The center may be solid or semi-hollow. The pale-yellow, nearly whitish, or pink, or even deep-red pulp is in 11 to 14 segments with thin, membranous, somewhat bitter walls; very juicy, acid to sweet-acid in flavor when fully ripe. While some fruits are seedless or nearly so, there may be up to 90 white, elliptical, pointed seeds about 1/2 in (1.25 cm) in length. Unlike those of the pummelo, grapefruit seeds are usually polyembryonic. The number of fruits in a cluster varies greatly; a dozen is unusual but there have been as many as 20.

Figure B-5: Screenshot of the first page of “Grapefruit” Morton article.

B.6 “Grapefruit” New World Encyclopedia article.

The article was accessed in March 2018.



Grapefruit

Previous (Grape) Next (Graphene)

Grapefruit is a relatively large, evergreen citrus tree, *Citrus paradisi*. The term also refers to the round, edible fruit of this plant, which grows in small bunches and has a yellow rind (outer skin) and typically a juicy, acidic pulp.

Grapefruit belongs to the *Citrus* genus, a taxa of flowering plants in the family Rutaceae. Other members of the genus include oranges, lemons, limes, citrons, pomelos (pummelo, pommelo), and mandarins (tangerines). Citrus fruits are a distinctive berry with the internal parts divided into segments. The number of natural species is unclear, as many of the named species are hybrids. The grapefruit is believed to have arisen from the pomelo or shaddock *(Citrus grandis)* or as a hybrid between pomelo and sweet orange (Bender and Bender 2005).

The fruit and flowers of the grapefruit tree provide for the species' individual purpose of reproduction, while at the same time providing a value for the whole, especially humans. For humans, the grapefruit is nutritious, providing many nutrients and phytochemicals important for a healthy diet, including vitamin C, pectin fiber, and the antioxidant lycopene. In addition, the fruit stimulates the senses of sight, taste, and touch, the flowers add an aesthetic value, and the tree, including leaves and fruit, provides ecological values (nutrition for animals, habitat for insects and birds, etc.).

Contents [hide]

Grapefruit

Grapefruits

Scientific classification

Kingdom:	Plantae
Division:	Magnoliophyta
Class:	Magnoliopsida
Subclass:	Rosidae
Order:	Sapindales
Family:	Rutaceae
Genus:	*Citrus*
Species:	*C. paradisi*

Binomial name

Citrus paradisi
Macfad.

Overview

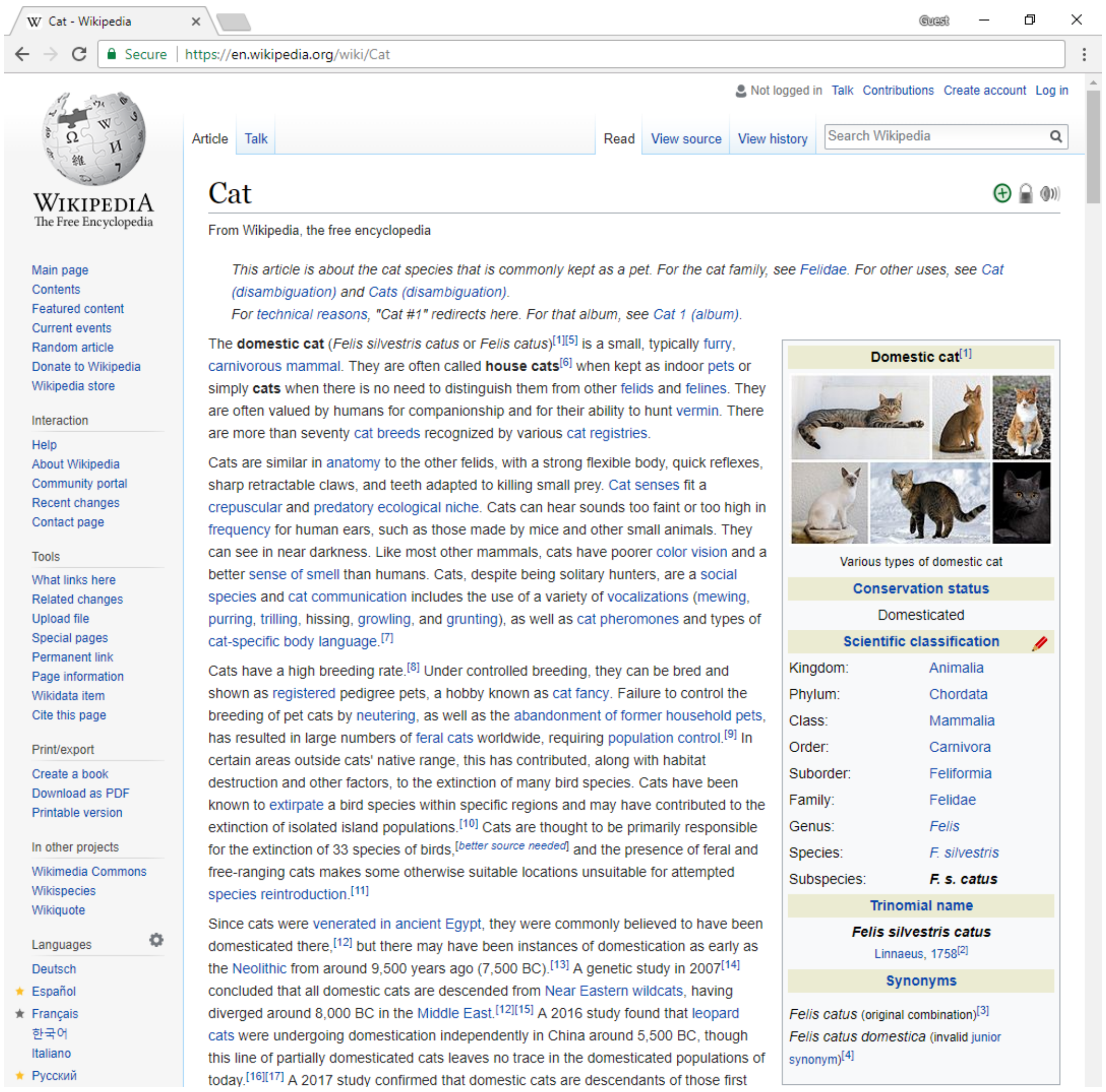
All citrus trees are of the single genus *Citrus* and hybridize very easily. That is, there is only one "superspecies," which includes lemons, limes, grapefruit, pomelos, tangerines, and oranges. Citrus is likely the most widely planted fruit for direct human consumption in the world (Katz and Weaver 2003).

Fruits of all members of the genus *Citrus* are considered berries because they have many seeds, are fleshy and soft, and derive from a single ovary. The distinctive fruit is a *hesperidium* in that it is a specialized berry with the internal fleshy parts divided into segments (typically 10 to 16) and surrounded by a separable rind (Katz and Weaver 2003). The citrus herperidium is globose to elongated, 4 - 30 centimeters (1.6 - 11.8 inches) long, and 4 - 20 centimeters (1.6 - 7.9 inches) in diameter. The rind is leathery and the segments, or "liths," are filled with pulp vesicles.

Figure B-6: Screenshot of the first page of “Grapefruit” New World Encyclopedia article.

B.7 “Cat” Wikipedia article.

The article was accessed in March 2018.



Cat - Wikipedia

Guest

Secure | https://en.wikipedia.org/wiki/Cat

Not logged in Talk Contributions Create account Log in

Article Talk Read View source View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export

Create a book
Download as PDF
Printable version

In other projects

Wikimedia Commons
Wikispecies
Wikiquote

Languages

Deutsch
Español
Français
한국어
Italiano
Русский

Cat

From Wikipedia, the free encyclopedia

This article is about the cat species that is commonly kept as a pet. For the cat family, see Felidae. For other uses, see Cat (disambiguation) and Cats (disambiguation).

For technical reasons, "Cat #1" redirects here. For that album, see Cat 1 (album).

The **domestic cat** (*Felis silvestris catus* or *Felis catus*)[1][5] is a small, typically furry, carnivorous mammal. They are often called **house cats**[6] when kept as indoor pets or simply **cats** when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to hunt vermin. There are more than seventy cat breeds recognized by various cat registries.

Cats are similar in anatomy to the other felids, with a strong flexible body, quick reflexes, sharp retractable claws, and teeth adapted to killing small prey. Cat senses fit a crepuscular and predatory ecological niche. Cats can hear sounds too faint or too high in frequency for human ears, such as those made by mice and other small animals. They can see in near darkness. Like most other mammals, cats have poorer color vision and a better sense of smell than humans. Cats, despite being solitary hunters, are a social species and cat communication includes the use of a variety of vocalizations (mewing, purring, trilling, hissing, growling, and grunting), as well as cat pheromones and types of cat-specific body language.[7]

Cats have a high breeding rate.[8] Under controlled breeding, they can be bred and shown as registered pedigree pets, a hobby known as cat fancy. Failure to control the breeding of pet cats by neutering, as well as the abandonment of former household pets, has resulted in large numbers of feral cats worldwide, requiring population control.[9] In certain areas outside cats' native range, this has contributed, along with habitat destruction and other factors, to the extinction of many bird species. Cats have been known to extirpate a bird species within specific regions and may have contributed to the extinction of isolated island populations.[10] Cats are thought to be primarily responsible for the extinction of 33 species of birds,[*better source needed*] and the presence of feral and free-ranging cats makes some otherwise suitable locations unsuitable for attempted species reintroduction.[11]

Since cats were venerated in ancient Egypt, they were commonly believed to have been domesticated there,[12] but there may have been instances of domestication as early as the Neolithic from around 9,500 years ago (7,500 BC).[13] A genetic study in 2007[14] concluded that all domestic cats are descended from Near Eastern wildcats, having diverged around 8,000 BC in the Middle East.[12][15] A 2016 study found that leopard cats were undergoing domestication independently in China around 5,500 BC, though this line of partially domesticated cats leaves no trace in the domesticated populations of today.[16][17] A 2017 study confirmed that domestic cats are descendants of those first

Domestic cat[1]

Various types of domestic cat

Conservation status

Domesticated

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Suborder:	Feliformia
Family:	Felidae
Genus:	*Felis*
Species:	*F. silvestris*
Subspecies:	***F. s. catus***

Trinomial name

Felis silvestris catus
Linnaeus, 1758[2]

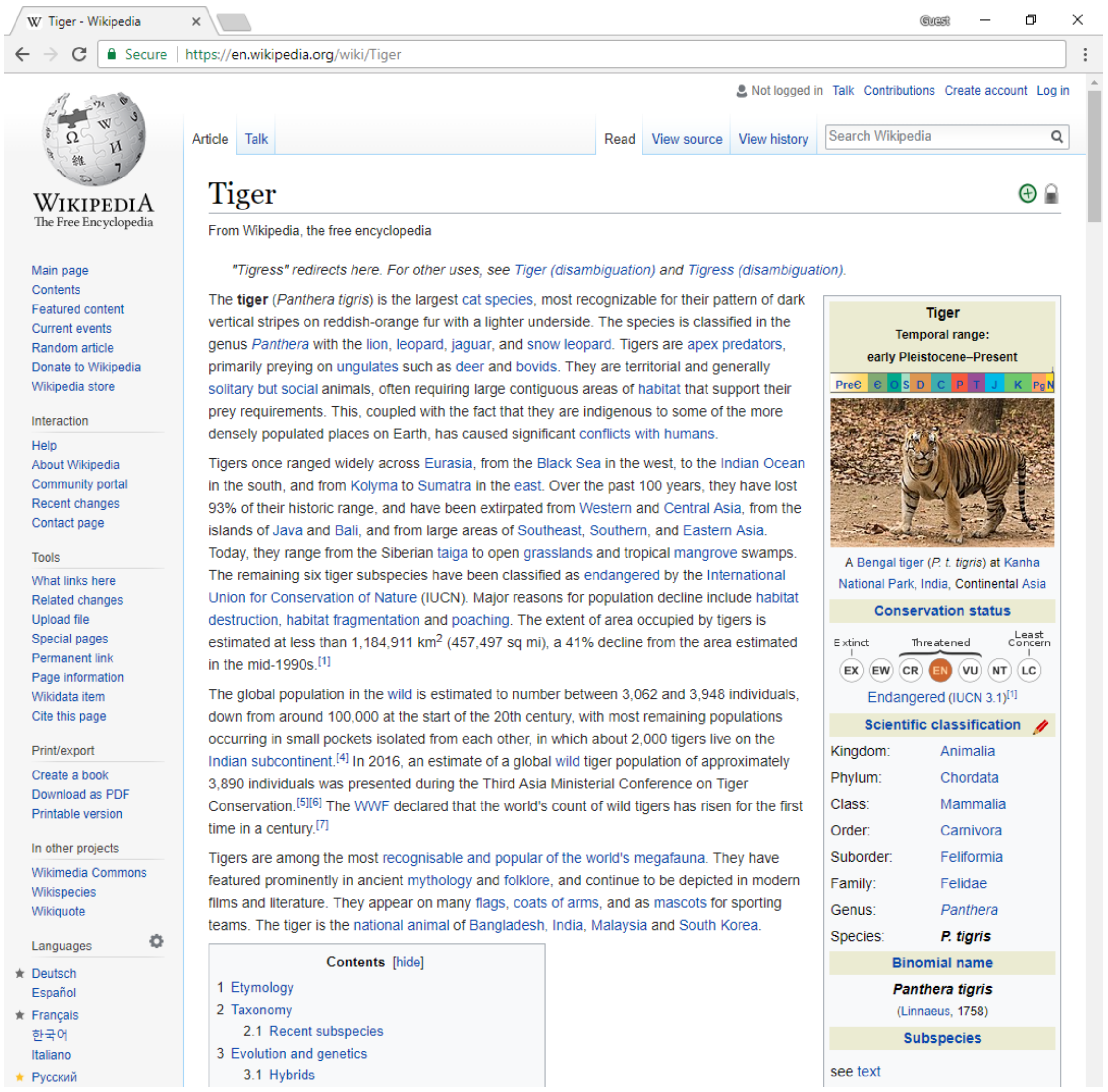
Synonyms

Felis catus (original combination)[3]
Felis catus domestica (invalid junior synonym)[4]

Figure B-7: Screenshot of the first page of “Cat” Wikipedia article.

B.8 "Tiger" Wikipedia article.

The article was accessed in March 2018.



Tiger

From Wikipedia, the free encyclopedia

"Tigress" redirects here. For other uses, see Tiger (disambiguation) and Tigress (disambiguation).

The **tiger** (*Panthera tigris*) is the largest cat species, most recognizable for their pattern of dark vertical stripes on reddish-orange fur with a lighter underside. The species is classified in the genus *Panthera* with the lion, leopard, jaguar, and snow leopard. Tigers are apex predators, primarily preying on ungulates such as deer and bovids. They are territorial and generally solitary but social animals, often requiring large contiguous areas of habitat that support their prey requirements. This, coupled with the fact that they are indigenous to some of the more densely populated places on Earth, has caused significant conflicts with humans.

Tigers once ranged widely across Eurasia, from the Black Sea in the west, to the Indian Ocean in the south, and from Kolyma to Sumatra in the east. Over the past 100 years, they have lost 93% of their historic range, and have been extirpated from Western and Central Asia, from the islands of Java and Bali, and from large areas of Southeast, Southern, and Eastern Asia. Today, they range from the Siberian taiga to open grasslands and tropical mangrove swamps. The remaining six tiger subspecies have been classified as endangered by the International Union for Conservation of Nature (IUCN). Major reasons for population decline include habitat destruction, habitat fragmentation and poaching. The extent of area occupied by tigers is estimated at less than 1,184,911 km^2 (457,497 sq mi), a 41% decline from the area estimated in the mid-1990s.[1]

The global population in the wild is estimated to number between 3,062 and 3,948 individuals, down from around 100,000 at the start of the 20th century, with most remaining populations occurring in small pockets isolated from each other, in which about 2,000 tigers live on the Indian subcontinent.[4] In 2016, an estimate of a global wild tiger population of approximately 3,890 individuals was presented during the Third Asia Ministerial Conference on Tiger Conservation.[5][6] The WWF declared that the world's count of wild tigers has risen for the first time in a century.[7]

Tigers are among the most recognisable and popular of the world's megafauna. They have featured prominently in ancient mythology and folklore, and continue to be depicted in modern films and literature. They appear on many flags, coats of arms, and as mascots for sporting teams. The tiger is the national animal of Bangladesh, India, Malaysia and South Korea.

Contents [hide]

Tiger
Temporal range: early Pleistocene–Present

A Bengal tiger (*P. t. tigris*) at Kanha National Park, India, Continental Asia

Conservation status

Endangered (IUCN 3.1)[1]

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Suborder:	Feliformia
Family:	Felidae
Genus:	*Panthera*
Species:	***P. tigris***

Binomial name

Panthera tigris
(Linnaeus, 1758)

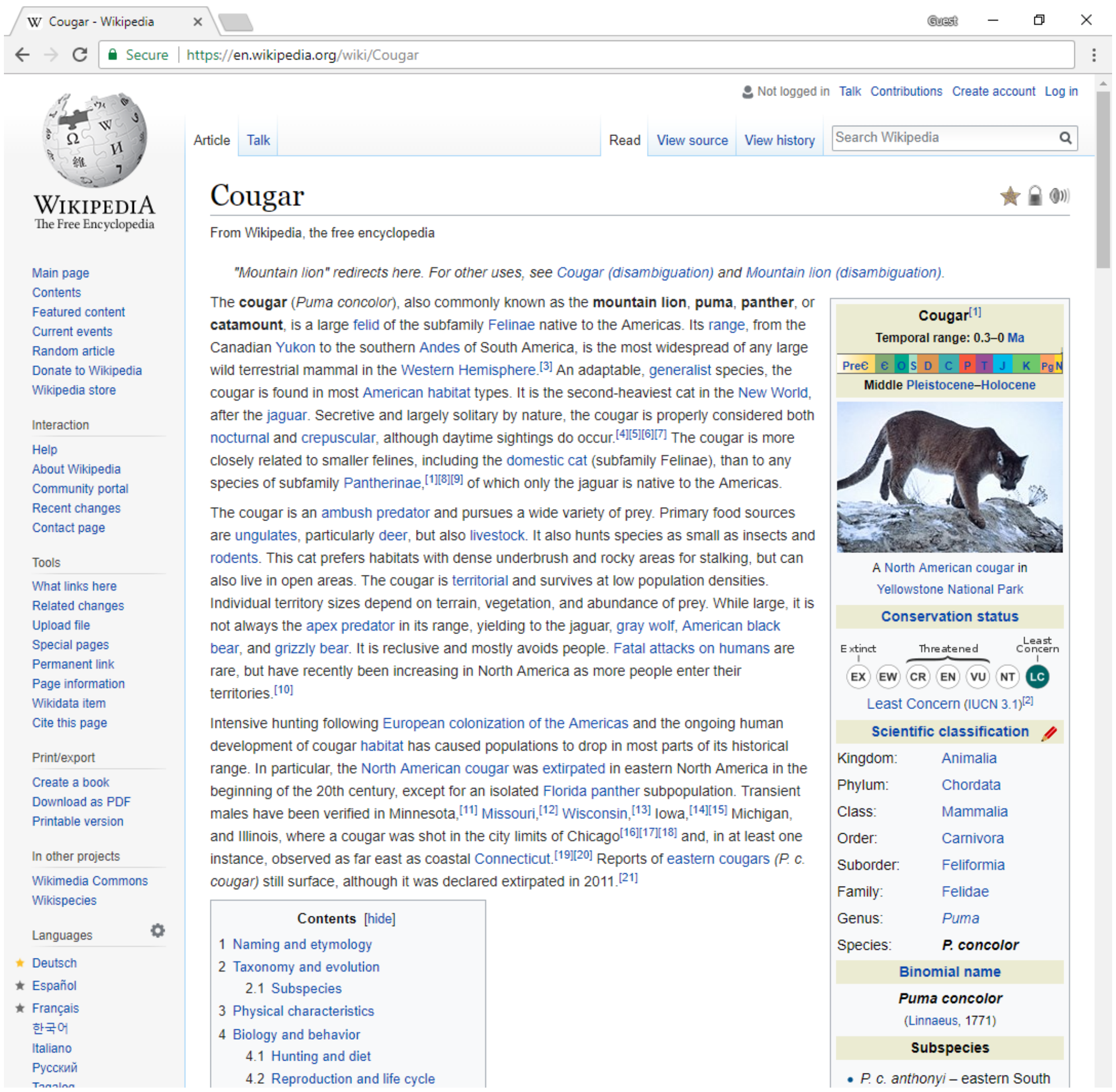
Subspecies

see text

Figure B-8: Screenshot of the first page of "Tiger" Wikipedia article.

B.9 “Cougar” Wikipedia article.

The article was accessed in March 2018.



Cougar - Wikipedia

Guest

Secure | https://en.wikipedia.org/wiki/Cougar

Not logged in Talk Contributions Create account Log in

WIKIPEDIA
The Free Encyclopedia

Article Talk Read View source View history Search Wikipedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikispecies

Languages
Deutsch
Español
Français
한국어
Italiano
Русский
Tagalog

Cougar

From Wikipedia, the free encyclopedia

"Mountain lion" redirects here. For other uses, see Cougar (disambiguation) and Mountain lion (disambiguation).

The **cougar** (*Puma concolor*), also commonly known as the **mountain lion**, **puma**, **panther**, or **catamount**, is a large felid of the subfamily Felinae native to the Americas. Its range, from the Canadian Yukon to the southern Andes of South America, is the most widespread of any large wild terrestrial mammal in the Western Hemisphere.[3] An adaptable, generalist species, the cougar is found in most American habitat types. It is the second-heaviest cat in the New World, after the jaguar. Secretive and largely solitary by nature, the cougar is properly considered both nocturnal and crepuscular, although daytime sightings do occur.[4][5][6][7] The cougar is more closely related to smaller felines, including the domestic cat (subfamily Felinae), than to any species of subfamily Pantherinae,[1][8][9] of which only the jaguar is native to the Americas.

The cougar is an ambush predator and pursues a wide variety of prey. Primary food sources are ungulates, particularly deer, but also livestock. It also hunts species as small as insects and rodents. This cat prefers habitats with dense underbrush and rocky areas for stalking, but can also live in open areas. The cougar is territorial and survives at low population densities. Individual territory sizes depend on terrain, vegetation, and abundance of prey. While large, it is not always the apex predator in its range, yielding to the jaguar, gray wolf, American black bear, and grizzly bear. It is reclusive and mostly avoids people. Fatal attacks on humans are rare, but have recently been increasing in North America as more people enter their territories.[10]

Intensive hunting following European colonization of the Americas and the ongoing human development of cougar habitat has caused populations to drop in most parts of its historical range. In particular, the North American cougar was extirpated in eastern North America in the beginning of the 20th century, except for an isolated Florida panther subpopulation. Transient males have been verified in Minnesota,[11] Missouri,[12] Wisconsin,[13] Iowa,[14][15] Michigan, and Illinois, where a cougar was shot in the city limits of Chicago[16][17][18] and, in at least one instance, observed as far east as coastal Connecticut.[19][20] Reports of eastern cougars (*P. c. cougar*) still surface, although it was declared extirpated in 2011.[21]

Contents [hide]

Cougar[1]

Temporal range: 0.3–0 Ma

PreЄ Є O S D C P T J K Pg N

Middle Pleistocene–Holocene

A North American cougar in Yellowstone National Park

Conservation status

Extinct Threatened Least Concern

EX EW CR EN VU NT LC

Least Concern (IUCN 3.1)[2]

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Suborder:	Feliformia
Family:	Felidae
Genus:	*Puma*
Species:	***P. concolor***

Binomial name

Puma concolor
(Linnaeus, 1771)

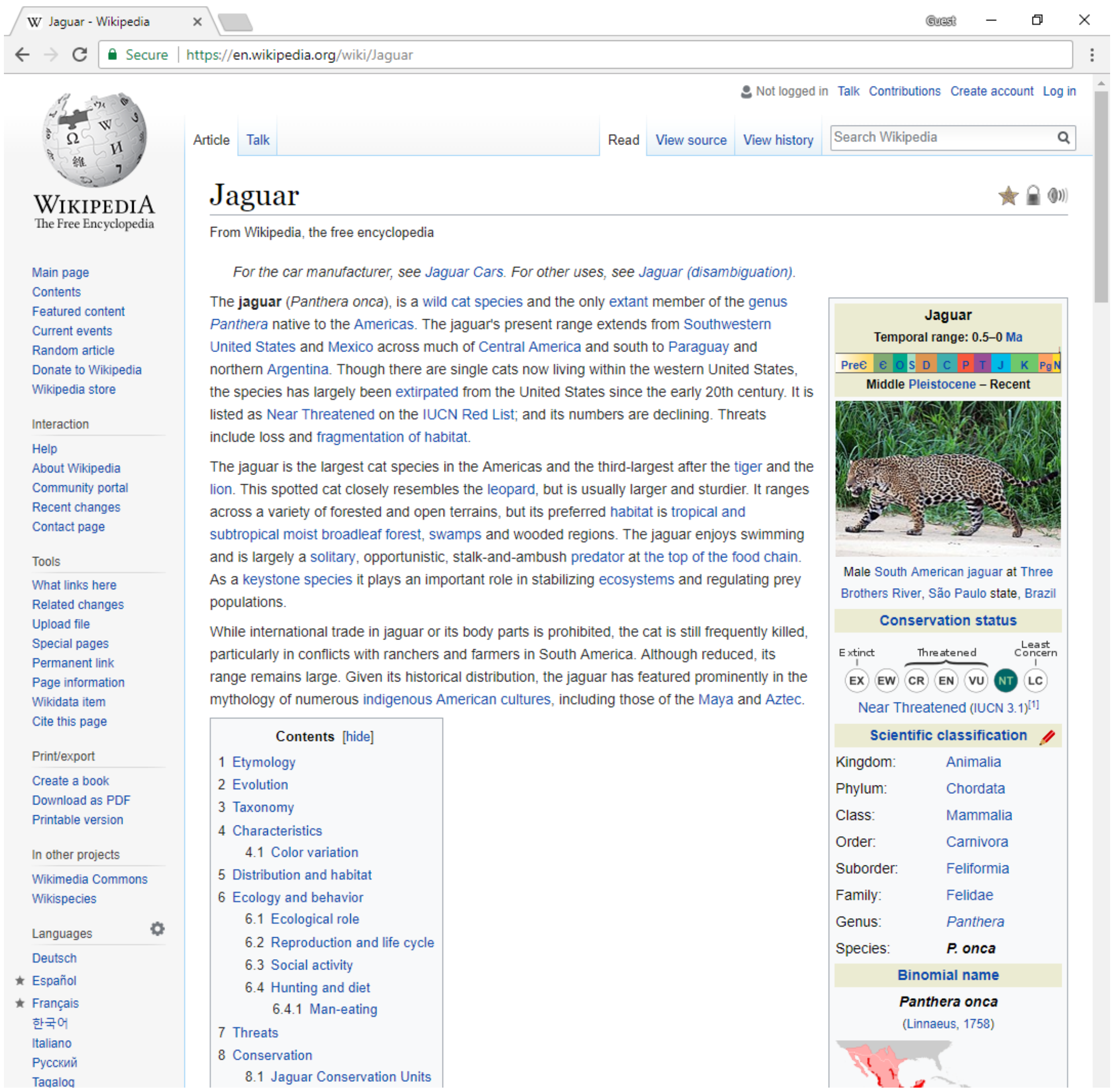
Subspecies

- *P. c. anthonyi* – eastern South

Figure B-9: Screenshot of the first page of “Cougar” Wikipedia article.

B.10 “Jaguar” Wikipedia article.

The article was accessed in March 2018.



Jaguar

From Wikipedia, the free encyclopedia

For the car manufacturer, see Jaguar Cars. For other uses, see Jaguar (disambiguation).

The **jaguar** (*Panthera onca*), is a wild cat species and the only extant member of the genus *Panthera* native to the Americas. The jaguar's present range extends from Southwestern United States and Mexico across much of Central America and south to Paraguay and northern Argentina. Though there are single cats now living within the western United States, the species has largely been extirpated from the United States since the early 20th century. It is listed as Near Threatened on the IUCN Red List; and its numbers are declining. Threats include loss and fragmentation of habitat.

The jaguar is the largest cat species in the Americas and the third-largest after the tiger and the lion. This spotted cat closely resembles the leopard, but is usually larger and sturdier. It ranges across a variety of forested and open terrains, but its preferred habitat is tropical and subtropical moist broadleaf forest, swamps and wooded regions. The jaguar enjoys swimming and is largely a solitary, opportunistic, stalk-and-ambush predator at the top of the food chain. As a keystone species it plays an important role in stabilizing ecosystems and regulating prey populations.

While international trade in jaguar or its body parts is prohibited, the cat is still frequently killed, particularly in conflicts with ranchers and farmers in South America. Although reduced, its range remains large. Given its historical distribution, the jaguar has featured prominently in the mythology of numerous indigenous American cultures, including those of the Maya and Aztec.

Contents [hide]

Jaguar

Temporal range: 0.5–0 Ma

PreЄ Є O S D C P T J K Pg N

Middle Pleistocene – Recent

Male South American jaguar at Three Brothers River, São Paulo state, Brazil

Conservation status

Extinct — Threatened — Least Concern

EX EW CR EN VU NT LC

Near Threatened (IUCN 3.1)[1]

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Suborder:	Feliformia
Family:	Felidae
Genus:	*Panthera*
Species:	***P. onca***

Binomial name

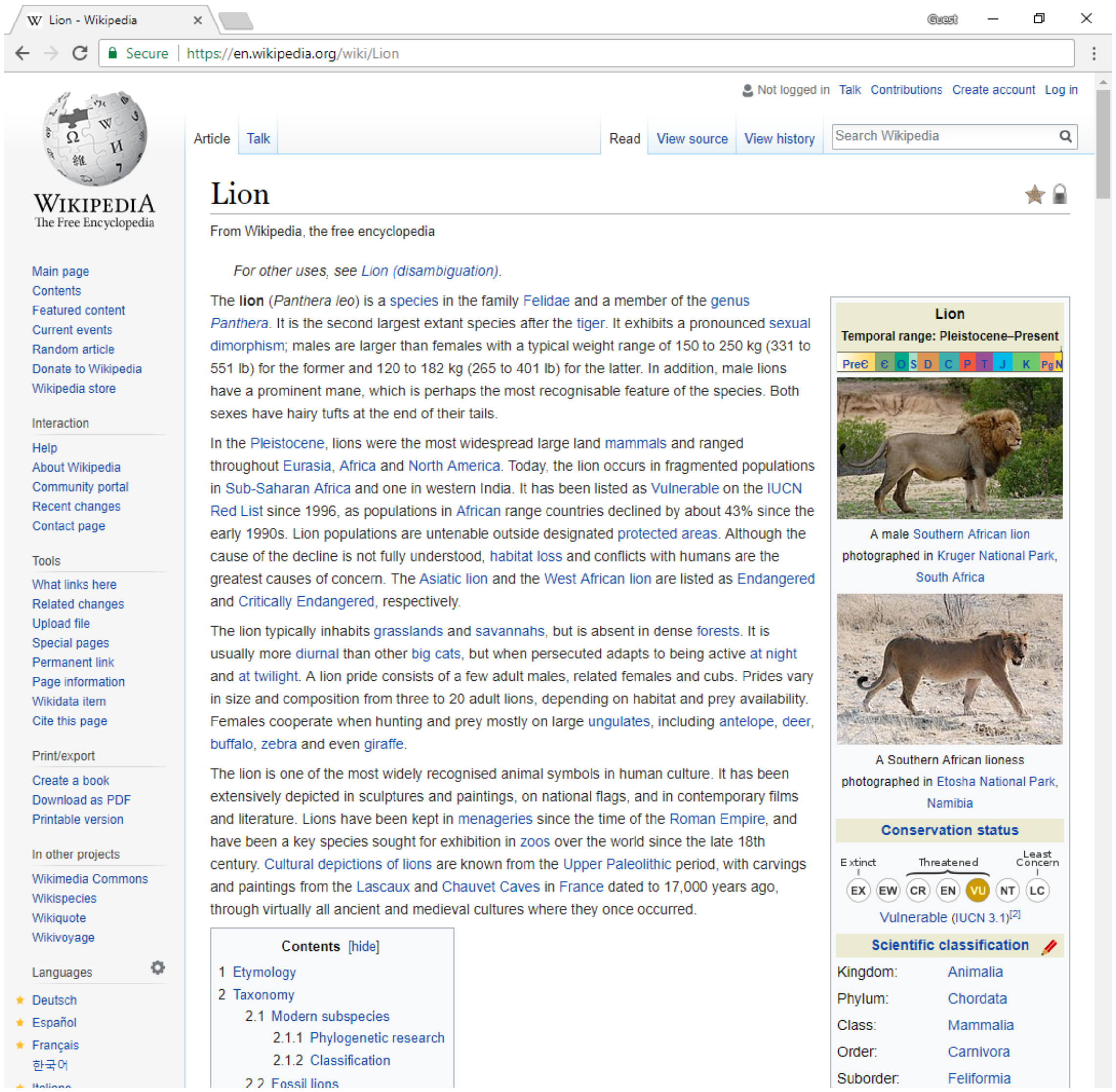
Panthera onca

(Linnaeus, 1758)

Figure B-10: Screenshot of the first page of “Jaguar” Wikipedia article.

B.11 “Lion” Wikipedia article.

The article was accessed in March 2018.



W Lion - Wikipedia
Guest
Secure | https://en.wikipedia.org/wiki/Lion
Not logged in Talk Contributions Create account Log in
WIKIPEDIA
The Free Encyclopedia
Article Talk
Read View source View history Search Wikipedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikispecies
Wikiquote
Wikivoyage

Languages
Deutsch
Español
Français
한국어

Lion

From Wikipedia, the free encyclopedia

For other uses, see Lion (disambiguation).

The **lion** (*Panthera leo*) is a species in the family Felidae and a member of the genus *Panthera*. It is the second largest extant species after the tiger. It exhibits a pronounced sexual dimorphism; males are larger than females with a typical weight range of 150 to 250 kg (331 to 551 lb) for the former and 120 to 182 kg (265 to 401 lb) for the latter. In addition, male lions have a prominent mane, which is perhaps the most recognisable feature of the species. Both sexes have hairy tufts at the end of their tails.

In the Pleistocene, lions were the most widespread large land mammals and ranged throughout Eurasia, Africa and North America. Today, the lion occurs in fragmented populations in Sub-Saharan Africa and one in western India. It has been listed as Vulnerable on the IUCN Red List since 1996, as populations in African range countries declined by about 43% since the early 1990s. Lion populations are untenable outside designated protected areas. Although the cause of the decline is not fully understood, habitat loss and conflicts with humans are the greatest causes of concern. The Asiatic lion and the West African lion are listed as Endangered and Critically Endangered, respectively.

The lion typically inhabits grasslands and savannahs, but is absent in dense forests. It is usually more diurnal than other big cats, but when persecuted adapts to being active at night and at twilight. A lion pride consists of a few adult males, related females and cubs. Prides vary in size and composition from three to 20 adult lions, depending on habitat and prey availability. Females cooperate when hunting and prey mostly on large ungulates, including antelope, deer, buffalo, zebra and even giraffe.

The lion is one of the most widely recognised animal symbols in human culture. It has been extensively depicted in sculptures and paintings, on national flags, and in contemporary films and literature. Lions have been kept in menageries since the time of the Roman Empire, and have been a key species sought for exhibition in zoos over the world since the late 18th century. Cultural depictions of lions are known from the Upper Paleolithic period, with carvings and paintings from the Lascaux and Chauvet Caves in France dated to 17,000 years ago, through virtually all ancient and medieval cultures where they once occurred.

Contents [hide]

Lion

Temporal range: Pleistocene–Present

A male Southern African lion photographed in Kruger National Park, South Africa

A Southern African lioness photographed in Etosha National Park, Namibia

Conservation status

Vulnerable (IUCN 3.1)[2]

Scientific classification

Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Order:	Carnivora
Suborder:	Feliformia

Figure B-11: Screenshot of the first page of “Lion” Wikipedia article.

REFERENCES

[1] IBM Marketing Cloud, "10 Key Marketing Trends for 2017," 2017.

[2] A. Lincoln, "FYI: TMI: Toward a holistic social theory of information overload," *First Monday*, vol. 16, no. 3, 2011.

[3] G. Chakraborty and M. K. Pagolu, "Analysis of unstructured data: Applications of text analytics and sentiment mining," *SAS global forum,* pp. 1288-2014, 2014.

[4] J. Hirschberg and C. D. Manning, "Advances in natural language processing," vol. *349, no*. 6245, pp. 261-266, 2015.

[5] J.-g. Yao, X. Wan and J. Xiao, "Recent advances in document summarization," *Knowledge and Information Systems,* vol. 53, no. 2, pp. 297-336, 2017.

[6] J. C. K. Cheung and G. Penn, "Towards Robust Abstractive Multi-Document Summarization: A Caseframe Analysis of Centrality and Domain," in ACL, 2013.

[7] Cycorp, "Home," July 2017. [Online]. Available: http://www.cyc.com/. [Accessed July 2017].

[8] A. Nenkova and K. McKeown, "A survey of text summarization techniques," in Mining *Text data,* Boston, Springer, 2012, pp. 43-76.

[9] H. P. Luhn, "The automatic creation of literature abstracts," IBM *Journal of Research and Development, pp*. 159-165, 1958.

[10] A. Nenkova and L. Vanderwende, "The impact of frequency on summarization," Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-2005, vol. 101, 2005.

[11] W.-t. Yih, J. Goodman, L. Vanderwende and H. Suzuki, "Multi-Document Summarization by Maximizing Informative Content-Words," in Proceedings of *the Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007.

[12] E. Hovy and C.-Y. Lin, "Automated text summarization and the SUMMARIST system," in Proceedings of *a workshop on held at Baltimore, Maryland: October* 13-15, Baltimore, Maryland, 1998.

[13] D. R. Radev, H. Jing, M. Styś and D. Tam, "Centroid-based summarization of multiple documents," Information Processing *& Management, vol.* 40, no. 6, pp. 919-938, 2004.

[14] E. Filatova and V. Hatzivassiloglou, "A formal model for information selection in multi-sentence text extraction," in Proceedings of *the 20th international conference on Computational Linguistics, 2004.*

[15] C.-Y. Lin and E. Hovy, "The automated acquisition of topic signatures for text summarization," in Proceedings of *the 18th conference on Computational linguistics - Volume* 1, Saarbrücken, Germany, 2000.

[16] J. M. Conroy, J. D. Schlesinger, J. Goldstein and D. P. O'leary, "Left-brain/right-brain multi-document summarization," in In Proceedings *of the Document Understanding Conference* (DUC 2004), 2004.

[17] S. Gupta, A. Nenkova and D. Jurafsky, "Measuring importance and query relevance in topic-focused multi-document summarization," in Proceedings of *the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, 2007.*

[18] R. Mihalcea and P. Tarau, "TextRank: Bringing Order into Text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 2004.

[19] G. Erkan and D. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, no. 22, pp. 457-479, 2004.

[20] J. Leskovec, N. Milic-Frayling and M. Grobelnik, "Impact of Linguistic Analysis on the Semantic Graph Coverage and Learning of Document Extracts," in *Proceedings of the AAAI*, 2005.

[21] X. Wan and J. Yang, "Improved affinity graph based multi-document summarization," in *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, 2006.

[22] J. Kupiec, J. Pedersen and F. Chen, "A trainable document summarizer," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, 1995.

[23] J. M. Conroy and D. P. O'Leary, "Text Summarization via Hidden Markov Models and Pivoted QR," *in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in* information retrieval, 2001.

[24] D. Shen, J.-T. Sun, H. Li, Q. Yang and Z. Chen, "Document Summarization Using Conditional Random Fields," in *Proceedings of International Joint Conference on Artificial Intelligence,* Hyderabad, India, 2007.

[25] M. Fuentes, E. Alfonseca and H. Rod, "Support vector machines for query-focused summarization trained and evaluated on pyramid data," in *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions,* 2007.

[26] K.-F. Wong, M. Wu and W. Li, "Extractive summarization using supervised and semi-supervised learning," in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1,* Manchester, United Kingdom, 2008.

[27] R. Barzilay and M. Elhadad, "Using Lexical Chains for Text Summarization," *Advances in automatic text summarization,* pp. 111-121, 1999.

[28] H. G. Silber and K. F. McCoy, "Efficiently computed lexical chains as an intermediate representation for automatic text summarization," *Computational Linguistics,* pp. 487-496, 2002.

[29] S. Ye, T.-S. Chua, M.-Y. Kan and L. Qui, "Document concept lattice for text understanding and summarization," *Information Processing & Management,* pp. 1643-1662, 2007.

[30] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval,* 2001.

[31] M. G. Ozsoy, I. Cicekli and F. Nur Alpaslan, "Text summarization of turkish texts using latent semantic analysis," in *Proceedings of the 23rd international conference on computational linguistics*, 2010.

[32] K. Filippova, "Multi-sentence compression: Finding shortest paths in word graphs," in *Proceedings of the 23rd International Conference on Computational Linguistics,* 2010.

[33] E. Lloret and M. Palomar, "Analyzing the Use of Word Graphs for Abstractive Text Summarization," in *Proceedings of the First International Conference*

on Advances in Information Mining and Management, Barcelona, Spain, 2011.

[34] I. F. Moawad and M. Aref, "Semantic Graph Reduction Approach for Abstractive Text Summarization," in *Computer Engineering & Systems (ICCES), 2012 Seventh International Conference on*, 2012.

[35] L. Bing, P. Li, Y. Liao, W. Lam, W. Gu and R. J. Passonneau, "Abstractive multi-document summarization," in *Proceedings of the ACL-IJCNLP*, 2015.

[36] S. Gerani, Y. Mehdad, G. Carenini, R. T. Ng and B. Nejat, "Abstractive Summarization of Product Reviews Using Discourse Structure," in *EMNLP*, 2014.

[37] F. Liu, J. Flanigan, S. Thomson, N. Sadeh and N. A. Smith, "Toward Abstractive Summarization Using Semantic Representations," in *Proceedings of the North American Association*, 2015.

[38] J. C. K. Cheung and G. Penn, "Unsupervised Sentence Enhancement for Automatic Summarization," in *EMNLP*, 2015.

[39] P. S. Sajja and R. Akerkar, "Knowledge-based systems for development," in *Advanced Knowledge Based Systems: Model, Applications & Research*, 2010, pp. 1-11.

[40] R. G. Smith, *Knowledge-based systems: Concepts, techniques, examples*, 1985.

[41] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross and K. J. Miller, "Introduction to WordNet: An on-line lexical database," *International journal of lexicography*, pp. 235-244, 1990.

[42] R. Navigli and S. P. Ponzetto, "BabelNet: Building a very large multilingual semantic network," in *Proceedings of the 48th annual meeting of the association for computational linguistics* , 2010.

[43] H. Liu and P. Singh, "ConceptNet—a practical commonsense reasoning tool-kit," *BT technology journal*, pp. 211-226, 2004.

[44] Cycorp, "Cyc: Knowledge Base," 8 March 2018. [Online]. Available: http://www.cyc.com/kb/. [Accessed 8 March 2018].

[45] E. Cambria and B. White, "Jumping NLP curves: A review of natural language processing research," IEEE *Computational intelligence magazine*, *vol*. 2, no. 9, pp. 48-57, 2014.

[46] D. Jurafsky and J. H. Martin, Speech and language processing, vol. 3, London: Pearson, 2014.

[47] M. Honnibal and M. Johnson, "An Improved Non-monotonic Transition System for Dependency Parsing," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015.

[48] JPype, "JPype - Java to Python integration," 2017. [Online]. Available: http://jpype.sourceforge.net/. [Accessed July 2017].

[49] B. Choi and X. Huang, "Creating New Sentences to Summarize Documents," in *The 10th IASTED International Conference on Artificial Intelligence and Application (AIA 2010),* Innsbruck, Austria, 2010.

[50] P. Baxendale, "Machine-made index for technical literature—an experiment," *IBM Journal of Research and Development,* pp. 354-361, 1958.

[51] H. Edmundson, "New methods in automatic extracting," *Journal of the ACM,* pp. 264-285, 1969.