2016

A Stochastic Petri Net Based NLU Scheme for Technical Documents Understanding

Adamantia Psarologou

Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/1493

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
A Stochastic Petri Net based NLU Scheme for Technical Documents Understanding

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

By

ADAMANTIA PSAROLOGOU
B.S., University of Piraeus, 2012

2016
Wright State University

Committee on Final Examination

Nikolaos Bourbakis, Ph.D.
Soon M. Chung, Ph.D.
Yong Pei, Ph.D.
Sukarno Mertoguno, Ph.D.
Maria Virvou, Ph.D.

Dissertation Director

Michael Raymer, Ph.D.
Director, Computer Science & Engineering Ph.D. Program

Robert E. W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT


Natural Language Understanding (NLU) is a very old research field, which deals with machine reading comprehension. Despite the many years of work and the numerous accomplishments by several researchers in the field, there is still place for significant improvements.

Here, our goal is to develop a novel NLU methodology for detecting and extracting event/action associations in technical documents. In order to achieve this goal we present a synergy of methods (Kernel extraction, Formal Language Modeling, Stochastic Petri-nets (SPN) mapping and Event Representation via SPN graph synthesis). In particular, the basic meaning of a natural language sentence is given by its kernel (Agent $\rightarrow$ Action $\rightarrow$ Patient), which is “who”, is doing “what”, to “whom”. Thus, we have developed a methodology that automatically extracts the kernels of NL sentences based on their parse trees. Then, we represent the kernel’s structure in a form of a formal language, called Glossa, for efficient processing. Next, we map the formal representation of kernels to an SPN state machine in order to embed timing in the representation of NL sentences. Finally,
we synthesize the SPN representation of kernels for expressing the association of events/actions of different sentences.

Results of our methodology are presented to prove the concept and validate the overall approach. Moreover, we provide two different application that our proposed NLU methodology can be used for. The first application is a quick and easy way for modifying technical documents, by multiple users. The second one is document summarization, where two different types of summarization are described.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................. iii

TABLE OF CONTENTS ................................................................................................... v

LIST OF FIGURES ......................................................................................................... viii

LIST OF TABLES ........................................................................................................... xii

ACKNOWLEDGMENTS .................................................................................................... xiii

Chapter 1 : Introduction .................................................................................................. 1

Chapter 2 : Filtering/Categorization of Technical Documents ....................................... 6

  2.1. User’s weights for AI and security methods ......................................................... 9

  2.2. Methodology ....................................................................................................... 10

  2.3. Illustrative examples .......................................................................................... 19

  2.4. Conclusions ........................................................................................................ 23

Chapter 3 : Kernel extraction of NL sentences .............................................................. 24

  3.1. Parse trees ........................................................................................................... 25

  3.2. Our current algorithm for kernel extraction ....................................................... 27

Chapter 4 : Glossa: A formal representation of kernels .................................................. 31

  4.1. Notations and definitions .................................................................................... 32

  4.2. Glossa: A formal language .................................................................................. 33

    4.2.1. The alphabet ................................................................................................ 34

    4.2.2. Grammar ..................................................................................................... 35

    4.2.3. Operators .................................................................................................... 35
4.2.4. Examples words of *Glossa* language .......................................................... 37

4.3. Examples .............................................................................................................. 37

4.4. Conclusions ......................................................................................................... 40

Chapter 5: NLU: Kernel to SPN graph mapping ......................................................... 41

5.1. Convert natural language sentences to graphs .................................................... 43

5.1.1. Constructing graphs ......................................................................................... 43

5.2. Convert graphs to SPNs ..................................................................................... 44

5.2.1. SPNs are state machines ............................................................................... 45

5.2.2. Constructing SPN graphs ............................................................................... 47

5.3. Conclusions ......................................................................................................... 49

Chapter 6: NLU: Sentences into SPN graphs representation ........................................ 50

6.1. Mapping *Glossa* expressions into an SPN ...................................................... 55

6.1.1. Stochastic Petri Nets prerequisites .................................................................. 55

6.1.2. Mapping of *Glossa* expression into SPN graphs ......................................... 57

6.1.3. Examples ......................................................................................................... 65

6.2. Synthesis of SPN graphs .................................................................................... 67

6.2.1. Anaphora Resolution ..................................................................................... 67

6.2.2. Combined SPN graphs .................................................................................. 69

6.2.3. Examples ......................................................................................................... 75

6.2.4. Conclusions .................................................................................................... 78

6.3. Examples ............................................................................................................ 79

6.4. Conclusions ......................................................................................................... 87

Chapter 7: Modifications of technical documents ...................................................... 89
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: The way that the documents pass the first two steps of the classification procedure.</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2: Weight calculation of a specific document (first two stages).</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3: The five classes and their interdependency.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4: Parse trees of the text: “Tom has a dog. The dog is brown.”</td>
<td>27</td>
</tr>
<tr>
<td>Figure 5: Graph of the kernel of a sentence.</td>
<td>44</td>
</tr>
<tr>
<td>Figure 6: (a) Basic state machine (b) SPNs to represent the kernel: V changes state of P to P’ and A to A.</td>
<td>45</td>
</tr>
<tr>
<td>Figure 7: SPN graphs: symbolic representation of kernel.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 8: Syntactic tree of the sentence: “John teases Kate”.</td>
<td>48</td>
</tr>
<tr>
<td>Figure 9: (a) Graph of sentence: “John teases Kate.” (b) SPN graph of the kernel of: “John teases Kate” without using the symbolic representation (c) Detailed SPN graph of the kernel of: “John teases Kate” using the symbolic representation (d) Simplified SPN graph of the kernel of: “John teases Kate” using the symbolic representation.</td>
<td>48</td>
</tr>
<tr>
<td>Figure 10: A simple Petri Net graph.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 11: Example of conversion of SPN graph to Glossa expression.</td>
<td>65</td>
</tr>
<tr>
<td>Figure 12: (a) SPN graph of sentence: “John and Mike play chess.” (b) SPN graph of sentence: “John teases and loves Kate.” (c) SPN graph of sentence: “John teases Kate or Anna.”.</td>
<td>67</td>
</tr>
<tr>
<td>Figure 13: (a) Uncombined SPN graph: A, A’, A” are different places (nodes) of the same agent. (b) Combined coloured SPN graph: there is only one place, A, for the agent. (c) Uncombined SPN graph of the text: “John likes music. He is a musician.” (d) Combined coloured SPN graph of the same text.</td>
<td>72</td>
</tr>
<tr>
<td>Figure 14: (a) Uncombined SPN graph: P, P’, P” are different places (nodes) of the same patient. (b) Combined coloured SPN graph: there is only one place, P, for the</td>
<td></td>
</tr>
</tbody>
</table>
patient. (c) Uncombined SPN graph of the text: “John has a guitar. Peter likes it.” (d) Combined coloured SPN graph of the same text. 

Figure 15: (a) Uncombined SPN graph, where A, A’ are different places (nodes) of the same agent and P and P’ are different places of the same patient. (b) Combined SPN graph: there is only one place, A, for agent and only one place, P, for the patient. (c) Uncombined SPN graph of the text: “Mike bought a candy. He ate it.” (d) Combined coloured SPN graph of the same text.

Figure 16: (a) Uncombined SPN graph of text: “Mary has a violin. She likes it. It is nice.” (b) Combined coloured SPN graph of the same text.

Figure 17: (a) Uncombined SPN graph of text: “The strongest rain ever recorded in India shut down the financial hub of Mumbai, snapped communication lines, closed airports and forced thousands of people to sleep in their offices or walk home during the night, officials said today.” (b) Combined coloured SPN graph of the same text.

Figure 18: (a) Output uncombined SPN graph of text: “I knew John would finish the work by 5:00 PM. At 8:00 PM tonight, Sally is going to make a beautiful dinner. The pies would be made by my mother, as always. John is going to be washing the dishes after the dinner.” produced by our program. (b) Output combined Coloured SPN graph of the same text.

Figure 19: (a) Output uncombined SPN graph of the text: “Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don't remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom's mother was serving food and his father was playing guitar. They all had fun.”. (b) Output combined Coloured SPN graph of the same text.

Figure 20: (a) Output uncombined SPN graph of text: “The strongest rain ever recorded in India shut down the financial hub of Mumbai, snapped communication lines, closed airports and forced thousands of people to sleep in their offices or walk home during the night, officials said today.” produced by our program. (b) Combined Coloured SPN graph of the same text produce by our program. (c) SPN graph of the same text as shown in [15] by Bourbakis and Mills.

Figure 21: (a) Output uncombined SPN graph of text: “I saw Tom with the telescope. The telescope sat on the table. The table and telescope was seen by all of us. We had a great time at the party. It was Tom's birthday. Tom received several presents from all of us. The telescope was a Tom's favorite present. The telescope appeared expensive. I looked through the telescope. A great time was had by all.” produced by our program (b) Combined Coloured SPN graph of the same text produce by our program. (c) SPN graph of the same text as shown in [15] by Bourbakis and Mills.
all had fun.” with target word “Tom” from the sentence “Tom had his birthday yesterday.”

Figure 32: (a) Uncombined SPN graph produced by our program (b) Combined Coloured
SPN graph of the same text produce by our program.

Figure 33: Combined SPN graph of the summary produced by our Free Summarization
technique with threshold value equal to 5%.

Figure 34: (a) Uncombined SPN graph produced by our program (b) Combined Coloured
SPN graph of the same text produce by our program.

Figure 35: Combined SPN graph of the summary produced by our Free Summarization
technique with threshold value equal to 10%.

Figure 36: Combined SPN graph of the summary produced by our Free Summarization
technique with threshold value equal to 20%.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1: Contains words and phrases which correspond with AI methodologies (first column) and an assigned weight for each one (second column).</td>
<td>11</td>
</tr>
<tr>
<td>Table 2: Contains words and phrases which correspond with security areas (first column), an assigned weight for each (second column), and the class to which the words/phrases of the first column belong to, if they are not considered a separate class by themselves (third column).</td>
<td>13</td>
</tr>
<tr>
<td>Table 3: Experimental results of papers’ filtering.</td>
<td>20</td>
</tr>
<tr>
<td>Table 4: POS (Part of Speech) tags</td>
<td>25</td>
</tr>
<tr>
<td>Table 5: Chunk tags</td>
<td>26</td>
</tr>
<tr>
<td>Table 6: Representation of Glossa’s terminal symbols with SPN components</td>
<td>57</td>
</tr>
<tr>
<td>Table 7: Representation of Glossa’s operators with SPN components</td>
<td>58</td>
</tr>
<tr>
<td>Table 8: Examples of the visual representation of expressions of Glossa language with SPN graphs</td>
<td>62</td>
</tr>
<tr>
<td>Table 9: A more complex example of the visual representation of expression of Glossa language with SPN graph</td>
<td>64</td>
</tr>
<tr>
<td>Table 10: Examples of output text after Anaphora Resolution</td>
<td>68</td>
</tr>
<tr>
<td>Table 11: Evaluation of different versions of our kernel extraction algorithm</td>
<td>142</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to thank all the individuals who helped me complete this enriching and exciting doctoral journey. Professor Nikolaos Bourbakis provided constant support and invaluable guidance through the past several years. I appreciate all his contributions of ideas, time, and funding to make my Ph.D. experience productive and successful.

I also wish to thank the Ph.D. committee members, Soon M. Chung, Yong Pei, and Sukarno Mertoguno. Moreover, I would like to thank Professor Maria Virvou for believing in me and supporting me. They all had major positive influence on this Ph.D. work and it was a real pleasure working with them. Moreover, I would like to acknowledge all my colleagues Anargyros Angeleas, Zacharias Hasparis, Stavros Mallios, Konstantinos Michalopoulos, Iosif Papadakis-Ktistakis, Giorgia Rematska, Anna Trikalinou, Michael Tsakalakis, and Athanasios Tsitsoulis for their support during the past 3 years.

I would like to express my gratitude to my family who have always been there for me, bringing out the best in me. My parents Konstantinos Psarologos and Anna Protopsalti, and my brother Spiros Psarologos. Also, a special thank you to the group of my friends from Piraeus University, for always cheering me up and standing by me even from thousands of miles away. Last but not least, I would like to thank Corey D. Sears for his patient, support and help during the last year of my studies. Without all of them, this dissertation would have been an even tougher venture.
This dissertation is dedicated to my parents, Konstantinos Psarologos and Anna Protopsalti, for their endless love, support and encouragement.
Chapter 1 : Introduction

Natural Language Processing (NLP) and Natural Language Understanding (NLU) have been studied by various researchers for decades now. The ambiguity of Natural Language, as well as the diversity and possible complexity of NL sentences, have made the discovery of a generalized automated NLP, and especially NLU, methodology very difficult and almost impossible. Thus, the areas of NLP and NLU are still open and possess need for improvement.

For our research we focus on the NLU field, and specifically we develop a new NLU methodology to represent the meaning of NL documents using the state machine, Stochastic Petri Nets (SPNs). State machines and especially Hidden Markov Models (HMMs) have been used in NLP for the production of parse trees of NL sentences [1], [2]. Also, HMMs have been used in Speech Processing and Recognition for the determination of the concept/meaning of group of words, and the prediction of meanings and/or words within spoken texts [3].

Our proposed NLU methodology uses the state machine, Stochastic Petri Nets (SPNs), in order to efficiently associate events and actions for technical document understanding. Specifically, we represent NL documents into SPN graphs. Graphs offer structural information, while SPNs offer functional information. Thus, SPN graphs hold the basic meaning of NL sentences and preserve the timing and flow of events/actions. Particularly, the SPN graph of a NL sentence provide a first quick view of its basic meaning of NL
sentences since we use it to represent the kernel of NL sentences. The kernel of an NL sentence carries its basic meaning since it is the Agent→Action→Patient carrier. Another way to say this is, who→is doing what→to whom. Additionally, through the synthesis of SPN graphs we associate the different actions of an agent or patient preserving the flow and timing.

Through all the years, several techniques have been used for NLP and NLU. For example, first order logic, several machine learning algorithms, neural networks, and clustering techniques have all been applied to achieve NLP and NLU. Furthermore, different forms of representation of the extracted knowledge have been used [4]. A general survey of different methodologies that have been use for different tasks of NLP or NLU processing is [5], which covers the area of NLP and NLU until 2012. So, for our research, we look at published papers 2012 and later. We especially focus on NLU of written documents. Until today, researchers on this topic are still using same or closely similar techniques for NLU and knowledge representation. Some of them are, ontologies [6], Markov Logic Networks [7], neural networks [8], deep belief networks [9], hidden Markov support vector machines (HM-SVMs) [10], tf x idf representation, lexical chain representation, and various other [4], [11]. Furthermore, as it is presented in [12] the same techniques, such as formal grammars and machine learning, have been used in Automated Speech Recognition, which is one type of NLU. Moreover, graph representation has been widely used because of the familiarity and preference of humans to a visual representation in cases of large amounts of information. A recent survey paper on graph representation for purposes of NLP and NLU is [13], which covers the area until 2014.
The representation of an NL sentence’s meaning using a state machine, and especially SPNs, provides the user of our methodology with its essential elements. These are the meaning, and timing and flow of events/actions throughout an NL document. In known literature, until today, only Bourbakis and Mills [14], [15], [16], have used SPNs for NLU purposes. Bourbakis and Mills’ proposed methodology is an early theoretical step for representation of NL documents into SPN graphs. They focus on the description of images using NL text and sets the general idea of SPN graph representation of said texts. However, our work goes deeper into the subject. So, in contrast to Bourbakis and Mills, who give only the general idea of representing NL sentences into SPN graphs, we present a formal model for the extraction of the meaning of NL sentences and their representation into SPN graphs. Additionally, our proposed methodology can process and represent more complicated forms of NL sentences in greater detail. Our model is able to hold more information and provide a more advanced way of representation through SPN graphs than Bourbakis and Mills’.

Our NLU methodology can be used in a wide range of applications that include or target NLP and NLU schemes by introducing timing and event/action associations. The proposed methodology is designed to automatically process English natural language documents and produce their SPN graphs. SPN graphs created after the processing of NL documents can be used for the production of summaries, document searching, information retrieval, text-based search engines, monitoring of social media for security purposes, question-answering systems, and more. For example, the produced SPN can be processed and used for information retrieval purposes in the following way: A query on a processed document is given, then using appropriate colored tokens (triggers) of the corresponding SPN graphs,
a specific part of the graph which holds the answer to the query will be activated. The produced combined SPN graphs in combination to the use of graph search algorithms can be very useful in order to provide faster and better results for any of the mentioned applications. In our work we use the proposed NLU methodology for processing of technical documents.

In this dissertation, Chapter 2 introduces a methodology for a first level quick filtering of technical documents based on the user’s preferences. We use this technique as a preprocessing step in order to filter out documents which we don’t want to further process.

Chapter 3 presents our algorithm for the extraction of kernels from an NL text. This algorithm uses the Stanford CoreNLP toolkit, and specifically the Stanford parser, in order to produce the parse tree of each NL sentence. Then, based on the structure of the produced parse tree, it extracts the kernel(s).

Chapter 4 defines the formal representation of the different forms that a kernel can take. Specifically, we define a formal language which we named *Glossa*. *Glossa* language is composed of the basic components of a kernel, A, V and P (agent, action/verb, and patient). Thus, the kernel of any NL sentence can be formally articulated by a *Glossa* expression.

In Chapter 5 we describe the representation of a simple kernel to a general graph. Then, we focus on the conversion of kernels into SPN graphs. Especially, we explain how SPNs, which are state machines, can be used for the representation of kernels, what the different states mean, and how we visually represent them.
In Chapter 6 we move from the representation of just a simple kernel to the representation of a sentence into SPN graphs. A sentence may contain complicated forms of kernels, thus, we illustrate a set of rules that maps the formal representation of kernels produced by Glossa language into SPN graphs. We also present a set of rules for the synthesis of SPN graphs produced after the processing of each paragraph within an NL document.

In Chapter 7 we present one of the application of our proposed NLU methodology. Particularly, we use our methodology to produce modified versions of the initial document. The modifications are made by the user on the SPN graph. Then, our system produces the new version of the document as well as the corresponding SPN graph. This could be useful in cases where a group of people need to exchange the same document and make some quick changes.

In Chapter 8 we describe two summarization techniques, as a second application of our proposed NLU methodology. For the first summarization technique, the produce summary contains information about a target agent or patient given by the user. For the second one, the produced summary contains information about a percentage (according to user’s preference) of the “main character” in the initial text. The “main characters” are defined based on their weight (frequency in the initial text and degree in the corresponding SPN graph).

Finally, Chapter 9 we conclude this work and summarize the major findings and results. It also includes possible extensions our NLU methodology, where cases of incomplete kernels and complicated cases of NL sentences will be taken under consideration.
Chapter 2: Filtering/Categorization of Technical Documents

The vast amount of available scientific information in every research topic makes search and retrieval efforts difficult. Additionally, the domain information is distributed in many different sources, with the main source being the Internet. The Internet houses a plethora of documents in virtually every scientific branch (medicine, psychology, biology, computer science, etc.). The amount of these documents exponentially increasing over time. As a result, it has been noticed in an increasing amount of survey papers on categorization of available documents of a specific domain. Each categorization technique takes into consideration different criteria. This emerging need to organize relevant information in text collections led to the creation of automatic text categorization machines. Text categorization aims to automatically place predefined labels on previously unseen documents. Since we want to apply NLU techniques to scientific documents, we have to use some kind of document classifier in order to limit the range of the natural language domain which we are targeting. Thus, in this chapter we propose a document filtering/categorization technique, which we use in order to filter domain specific documents before we process them further.

Many attempts have been made to create an efficient and effective document classifier. Most of them are used as a first step in Information Retrieval (e.g. in search engines). First, a document representation technique is needed in these systems. There are many different types of document representation with the most common type being a representation of
documents as vectors of words. Then, two sets of labeled documents (the training set and the testing set) are used. Each time, a different classification algorithm extracts words or phrases (sets of words) which represent each class from the training set. The testing set is used for evaluation and adjustments of the algorithm. Every new document is compared with each class’ documents and a similarity measure is computed. One of the several existing similarity measure is been used each time. Finally, the new document is categorized in the class with the highest similarity to it. Support Vector Machines, Naïve Bayes, and K-nearest neighbor are some classification algorithms commonly used. In [17] more classification algorithms are mentioned with full explanations.

It is out of the scope of our work to illustrate every way of document representation, filtering methods, categorization method, or similarity measure: we will briefly mention some of them, however. For example, in [18] the authors use a three layer feed-forward neural network as a classifier. Also, a vector which contains the tf x idf values of the words of the document is used for document representation. In [19], the documents are represented as a collection of transactions, while the items in each transaction are terms selected from both the document and the categories that the document is assigned to. In this case, the authors use association-rules to classify a new document, where a category is assigned to a document if a sufficient set of terms occur in the document. In [20], the authors generate a dictionary of rules (concurrent words) for each document. Hence, each class is a covering set of rules. This is the best set of rules that covers cases of only one class, out of the rules assigned in documents of this class. A similar outline of association rule generation is being used in [21], [22]. In [23], triples of words based on syntactical structure are extracted from the documents and represent them.
In this chapter, we propose a new method for document filtering/categorization that as a first step, filters the documents based on specific criteria, which are the use of AI methods for solving security problems. To achieve this we use two tables: a table of AI methods and a table of security areas. We have assigned a weight value to each entry of each table. Then, we scan the document in order to find word/phrases that match those in the tables. For each matching word/phrase the document acquires the corresponding weight. At the end of this phase we classify the documents to different security areas according to their acquired weights. In the second phase we apply Natural Language Understanding (NLU) techniques to the documents and classify them based on their meaning, in one or more of the following classes: Detection, Prevention, Interruption, Extermination, and Healing. To extract the meaning of the documents we can use our proposed NLU methodology.

Although, the first phase of our methodology is similar to a search engine, the second phase or NLU phase, makes the fundamental difference between our technique and a simple search engine or information retrieval technique. Our advantage over the above mentioned techniques is the filtering/categorization of documents based on their extracted meaning or content (expressed into SPN graphs using the proposed NLU methodology), in contrast to the use of keywords or phrases alone. One more difference is that the above mentioned techniques mainly classify web documents, where our goal is to categorize published papers (technical documents).

We need to point out here that although in this work we focused on scientific documents that use AI methodologies for solving security problems, the criteria and area of interest can change any time. In other words, we can use the proposed filtering/categorization
technique to filter and keep documents on any area of interest just by changing the content of the tables.

This chapter is organized as follows. In Section 1 we explain the use of the tables which consist of AI methods, security areas and their assigned weights. Detailed description of our proposed methodology for document filtering/categorization is presented in Section 2. Some experimental examples are given in Section 3. Finally, we conclude in Section 4.

2.1. User's weights for AI and security methods

As we mentioned before, we are interested in documents that are associated with security problems and use AI methodologies. First, we scan the document to find word or phrases (more than one word sequentially) within its title and keywords relevant to AI methods. Then, we scan it to find words or phrases relevant to security areas. Specifically, we look within each document’s content (titles and key words) for words or phrases contained in the two tables we have created. We evaluate every document based on the weights of the common pairs that will arise from this comparison. Next, we calculate its total weight. If the total weight, in each stage, exceeds a threshold the document passes to the next stage, otherwise the document is rejected as irrelevant to our interests.

As we have mentioned already, we created two tables. The first table, Table 1, contains words and phrases which correspond to AI methodologies (first column), and a weight for each (second column). The weights are derived from our personal opinion in accordance to how germane these words/phrase are to AI. We regard these words/phrase as germane to AI, only if the methods incorporate or indicate learning. The existence of learning in each AI methodology is symbolized by the assigned weight in this table. The second table,
Table 2, contains words and phrases that are connected with security areas (first column), and a weight (second column). The weights here express how much each entry is connected entirely to security problems, according to our opinion. The third column of this table indicates the class in which the words/phrases of the first column belong to, if they are not considered a separate class by themselves. This column has also been created based on our opinion.

Due to the fact that both tables are created based on our opinions, weights and words/phrases within them can be changed, or deleted at any time. Additionally, new words/phrases can be added. Moreover, anytime the field of interest changes, we can create new tables that contain important words/phrases of the focused area.

2.2. Methodology

In this section we will describe our proposed method for document filtering/categorization into five classes (detailed explanation of each class is given below). These classes are representative of five different ways to deal with security problems (intrusions, attacks, etc). It is important to notice that all of the documents that we will categorize have the same specific structure; they are scientific published papers and as a result they are subjected to the same set of rules. They have a main title, abstract, key words, and subtitles and body text for every section. Taking advantage of this structure we want to classify them according to the following described procedure.

At first, we search every document for words/phrases among its titles and key words that match with words/phrases of Table 1 (this table includes AI methodologies). Every match assigns a weight, \( a_i \), to the document. All of the weights of each document collected at the
end of this stage are multiplied together and give way to a new total weight, \( V_A \). Consequently, the calculation of \( V_A \) weight can be written as:

\[
V_A = \prod_{i=1}^{N} a_i
\]  

[1]

where \( a_i, i = 1,2,\ldots,N \) denotes each weight of the document for every match with Table 1, and \( N \) being the number of entries in Table 1.

Table 1: Contains words and phrases which correspond with AI methodologies (first column) and an assigned weight for each one (second column).

<table>
<thead>
<tr>
<th>Words/phrases indicates AI techniques</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support vector machines (SVMS)</td>
<td>( a_{14} = 5 )</td>
</tr>
<tr>
<td>Multiple Instance Logistic Regression</td>
<td>( a_{2} = 5 )</td>
</tr>
<tr>
<td>Neural Network</td>
<td>( a_{3} = 5 )</td>
</tr>
<tr>
<td>Data mining</td>
<td>( a_{4} = 1 )</td>
</tr>
<tr>
<td>Artificial Neural Network</td>
<td>( a_{5} = 5 )</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>( a_{6} = 5 )</td>
</tr>
<tr>
<td>k-Means clustering</td>
<td>( a_{7} = 5 )</td>
</tr>
<tr>
<td>k-Nearest Neighbor classifier</td>
<td>( a_{8} = 5 )</td>
</tr>
<tr>
<td>Decision-Tree</td>
<td>( a_{9} = 1 )</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>( a_{10} = 5 )</td>
</tr>
<tr>
<td>Multilayer Perceptrons</td>
<td>( a_{11} = 5 )</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>( a_{12} = 5 )</td>
</tr>
<tr>
<td>Perceptron</td>
<td>( a_{13} = 5 )</td>
</tr>
<tr>
<td>Recircular Neural Networks (RNNs)</td>
<td>( a_{14} = 5 )</td>
</tr>
<tr>
<td>Adaboost</td>
<td>( a_{15} = 5 )</td>
</tr>
<tr>
<td>Artificial Immune System (AIS)</td>
<td>( a_{16} = 5 )</td>
</tr>
<tr>
<td>Self-Organizing Feature Map</td>
<td>( a_{17} = 5 )</td>
</tr>
<tr>
<td>Boosted Modified Probabilistic Neural Network (BMPNN)</td>
<td>( a_{18} = 5 )</td>
</tr>
<tr>
<td>Rule Based Induction</td>
<td>( a_{19} = 5 )</td>
</tr>
<tr>
<td>Adaptive Resonance Theory</td>
<td>( a_{20} = 5 )</td>
</tr>
<tr>
<td>Back Propagation (BP) network</td>
<td>( a_{21} = 5 )</td>
</tr>
<tr>
<td>Case-Based Reasoning</td>
<td>( a_{22} = 5 )</td>
</tr>
<tr>
<td>Adaptive Intelligent Agent</td>
<td>( a_{23} = 5 )</td>
</tr>
<tr>
<td>Multiple Kernel Learning</td>
<td>( a_{24} = 5 )</td>
</tr>
<tr>
<td>Instance-Based Reasoning</td>
<td>( a_{25} = 5 )</td>
</tr>
<tr>
<td>Multiple Instance Learning</td>
<td>( a_{26} = 5 )</td>
</tr>
<tr>
<td>Synthetic Data Learning (SDL) algorithm</td>
<td>( a_{27} = 5 )</td>
</tr>
<tr>
<td>Algorithm TL (Transfer Learning)</td>
<td>( a_{28} = 5 )</td>
</tr>
<tr>
<td>Grammar Based Learning</td>
<td>( a_{29} = 5 )</td>
</tr>
</tbody>
</table>
Semi-supervised learning | a30 = 5
Intelligent assistant system | a31 = 1
Classification Algorithms | a32 = 1
Confidence-Weighted (CW) Algorithm | a33 = 1
Fuzzy rules generation | a34 = 1
Fuzzy classifiers | a35 = 1
Forward-Backward Algorithm | a36 = 1
Clustering | a37 = 1
Agent-based | a38 = 1
Artificial Intelligence Techniques | a39 = 1
Association Rules | a40 = 1
Rule-Based | a41 = 1
Multiple Instance Bag Creation | a42 = 1
Radical Basis Function | a43 = 1
Adaptive Resonance Theory | a44 = 1
Multi-Classifier | a45 = 1
Expert system | a46 = 1
Computer Immunology | a47 = 1
Pattern classification | a48 = 1
Directed Acyclic Graph Support Vector Machine | a49 = 1
Intelligent agents | a50 = 1
Hidden Markov Model (HMM) | a51 = 1
Learning Bias | a52 = 1
Generalization Variance | a53 = 1
Nonparametric Techniques | a54 = 1
Natural-Language Processing | a55 = 1
Self-including agent | a56 = 1
Fuzzy logic | a57 = 1
Artificial Intelligence | a58 = 1

The document passes to the second stage if $V_A \geq T_1$, where $T_1$ is a predefined threshold.

Otherwise the document is rejected and it does not take part in the second stage. The $T_1$ threshold is being used to assure the document uses AI methods.

At the second stage, we follow the same procedure as the first, with the only difference being that now we compare each document to Table 2 (this table includes security areas).

In this stage, every match assigns to the document a weight, $s_j$. All of the collected weights
of the document are multiplied together and give way to a new total weight, \( U_s \). Consequently, the calculation of \( U_s \) weight can be written as:

\[
U_s = \prod_{j=1}^{M} s_j
\]  

where \( s_j, j = 1,2,\ldots,M \), denotes each weight of the document for every match with Table 2, and \( M \) is the number of entries in Table 2.

Table 2: Contains words and phrases which correspond with security areas (first column), an assigned weight for each (second column), and the class to which the words/phrases of the first column belong to, if they are not considered a separate class by themselves (third column).

<table>
<thead>
<tr>
<th>Words/phrases indicates security areas</th>
<th>weights</th>
<th>classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intelligent Intrusion Detection</td>
<td>( s_1 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Misuse detection</td>
<td>( s_2 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Intrusion Detection and Recognition</td>
<td>( s_3 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Adaptive Intrusion Detection</td>
<td>( s_4 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Recognize Intrusions</td>
<td>( s_5 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>( s_6 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Malicious Email Detection</td>
<td>( s_7 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Distributed Intrusion Detection</td>
<td>( s_8 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Intrusion Detection</td>
<td>( s_9 = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Network Intrusion Detection</td>
<td>( s_{10} = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Detecting Web-Based Attacks</td>
<td>( s_{11} = 5 )</td>
<td>Malicious Web Pages Detection</td>
</tr>
<tr>
<td>Network security state assessment</td>
<td>( s_{12} = 5 )</td>
<td>Malicious Web Pages Detection</td>
</tr>
<tr>
<td>Content-Based Detection of Terrorists Browsing the Web</td>
<td>( s_{13} = 5 )</td>
<td>Malicious Web Pages Detection</td>
</tr>
<tr>
<td>E-mail-Based Intrusion Detection System (EBIDS)</td>
<td>( s_{14} = 5 )</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>Spam Assassin</td>
<td>( s_{15} = 5 )</td>
<td>Spam filtering/detection</td>
</tr>
<tr>
<td>Threat Prevention</td>
<td>( s_{16} = 5 )</td>
<td></td>
</tr>
<tr>
<td>Intelligent Threat Sensing Engine (ITSE), Intelligent Prevention Engine (IPE)</td>
<td>( s_{17} = 5 )</td>
<td></td>
</tr>
<tr>
<td>Detection of Polymorphic Networks Attacks</td>
<td>( s_{18} = 5 )</td>
<td></td>
</tr>
<tr>
<td>Attack Recognition</td>
<td>( s_{19} = 5 )</td>
<td></td>
</tr>
<tr>
<td>Antivirus Software</td>
<td>( s_{20} = 5 )</td>
<td>Spam filtering/detection</td>
</tr>
<tr>
<td>Detection of Unknown Viruses</td>
<td>$s_{21} = 5$</td>
<td></td>
</tr>
<tr>
<td>Detecting SQL Injection Attacks</td>
<td>$s_{22} = 5$</td>
<td></td>
</tr>
<tr>
<td>Detect Malicious Web Pages</td>
<td>$s_{23} = 5$</td>
<td></td>
</tr>
<tr>
<td>Malware Executable</td>
<td>$s_{24} = 5$</td>
<td></td>
</tr>
<tr>
<td>Network Anomaly Detection</td>
<td>$s_{25} = 5$</td>
<td></td>
</tr>
<tr>
<td>Immunity-Based Anomaly Detection</td>
<td>$s_{26} = 5$</td>
<td></td>
</tr>
<tr>
<td>SpamHunting</td>
<td>$s_{27} = 5$</td>
<td></td>
</tr>
<tr>
<td>Anti-spam</td>
<td>$s_{28} = 5$</td>
<td></td>
</tr>
<tr>
<td>Spam Filtering</td>
<td>$s_{29} = 5$</td>
<td></td>
</tr>
<tr>
<td>Content-based Spam Detection</td>
<td>$s_{30} = 5$</td>
<td></td>
</tr>
<tr>
<td>Link-based Spam Detection</td>
<td>$s_{31} = 5$</td>
<td></td>
</tr>
<tr>
<td>Cloud Based Malware Detection</td>
<td>$s_{32} = 5$</td>
<td></td>
</tr>
<tr>
<td>Good Word Attacks</td>
<td>$s_{33} = 5$</td>
<td></td>
</tr>
<tr>
<td>Privacy Recommendation</td>
<td>$s_{34} = 1$</td>
<td></td>
</tr>
<tr>
<td>Information Security</td>
<td>$s_{36} = 1$</td>
<td></td>
</tr>
<tr>
<td>Computer security</td>
<td>$s_{37} = 1$</td>
<td></td>
</tr>
<tr>
<td>Cyber Defense</td>
<td>$s_{38} = 1$</td>
<td></td>
</tr>
<tr>
<td>Cyber Threats</td>
<td>$s_{39} = 1$</td>
<td></td>
</tr>
<tr>
<td>Cyber Security</td>
<td>$s_{40} = 1$</td>
<td></td>
</tr>
<tr>
<td>Proactive Defense</td>
<td>$s_{41} = 1$</td>
<td></td>
</tr>
<tr>
<td>Computer Attacks</td>
<td>$s_{42} = 1$</td>
<td></td>
</tr>
<tr>
<td>Threats to the Privacy and Security</td>
<td>$s_{43} = 1$</td>
<td></td>
</tr>
<tr>
<td>Cloud Security</td>
<td>$s_{44} = 1$</td>
<td></td>
</tr>
<tr>
<td>Malicious Attacks</td>
<td>$s_{45} = 1$</td>
<td></td>
</tr>
<tr>
<td>Detection and Prevention of Attacks</td>
<td>$s_{46} = 1$</td>
<td></td>
</tr>
<tr>
<td>Network Security</td>
<td>$s_{47} = 1$</td>
<td></td>
</tr>
<tr>
<td>Social-Engineering Attacks</td>
<td>$s_{48} = 1$</td>
<td></td>
</tr>
<tr>
<td>E-Mail Authentication</td>
<td>$s_{49} = 1$</td>
<td></td>
</tr>
<tr>
<td>Spams</td>
<td>$s_{50} = 1$</td>
<td></td>
</tr>
<tr>
<td>Inferring Private Information</td>
<td>$s_{51} = 1$</td>
<td></td>
</tr>
<tr>
<td>Threats to Information Security</td>
<td>$s_{52} = 1$</td>
<td></td>
</tr>
<tr>
<td>Spam Detection</td>
<td>$s_{53} = 5$</td>
<td></td>
</tr>
</tbody>
</table>

Malicious Web Pages Detection

Spam filtering/detection
The document passes to the next phase if \( U_S \geq T_2 \), where \( T_2 \) is also a predefined threshold. Otherwise the document is rejected. In this case, \( T_2 \) threshold is being used to assure that the document is about security issues.

We decided to multiply all the collected weights in each stage, Formulas (1) and (2), because we want to put emphasis on documents that use more than one AI technique or solve more than one security problem.

When the first two stages are done, we calculate an indicator (G) of the document’s described methodology, which shows how strong AI is on security. We calculate G using the following formula:

\[
G = N_A - N_S \tag{3}
\]

where \( N_A \) (\( N_S \)) is the number of different weights \( a_i \) (\( s_j \)) that the document acquired in each stage. Furthermore, the document is temporarily categorized in the class or classes that correspond to the greatest \( s_j \) weight or weights derived from the document.

Figure 1 shows the way in which documents pass from one step to another as we explained above. Figure 2 is an example of weight calculation for a specific document.
Figure 1: The way that the documents pass the first two steps of the classification procedure.
Figure 2: Weight calculation of a specific document (first two stages).

The indicator $G$ of each document is being used to give us a clue, as to whether the document uses many different methods to solve a problem, or if it uses few (one or two maybe) methods to solve many different problems. Thus, $G$ shows how strong the methodology that each document describes is. Generally, the use of several methodologies for solving one problem is stronger than use of one methodologies for solving many problems.
The next phase of the described methodology is to apply Natural Language Understanding (NLU) techniques to the remaining of the documents. For this phase, we can use the proposed NLU methodology. In this phase we get deeper in the body of the document and extract its meaning. Then, we classify the document according to its meaning in one of the following categories:

- Detection: if the document’s technique detects and/or recognizes an undesirable attack.
- Prevention: if the document’s technique prevents or stops an undesirable attack. Firewalls work in this way.
- Interruption: if the document’s technique disconnects and isolates an undesirable attack.
- Extermination: if the document’s technique cleans or exterminates an undesirable attack.
- Healing: if the document’s technique restores and heals a software program from the effect of an undesirable attack.

The above categories are defined in respect to our opinion about the security problems and how these are treated. Figure 3 shows the dependencies between the five classes mentioned above.
If there are documents that belong to more than one category simultaneously, there is a possibility to create new intermediate categories, depending on the number of these documents. An additional factor that is taken into consideration in order to move documents from one category to the other, is the indicator $G$ (Formula 3).

### 2.3. Illustrative examples

In the following section we present some examples of document filtering/categorization using our proposed methodology. Particularly, we categorized several scientific documents. A small sample of the results of the filtering stage is in Table 3 (we didn’t include the whole table because of its size). It is necessary to mention that we set both the values of the thresholds $T_1$ and $T_2$ equal to 5. The first column of Table 3 is the number that references the document. The second column of Table 3 is the weights ($a_i$) that each document acquires from the comparison with Table 1. The total weight of each document
from this step is the value \( V \) (depicted in bold and underlined), which appears at the same column below all of the document’s \( a_i \) weights. If the document is rejected, which means that \( V < T_1 \), the value of \( V \) is depicted with a red background. The third column is the weights \((s_j)\) that each document acquires from the comparison with Table 2. The total weight of each document from this step is the value \( U \) (depicted in bold and underlined), which appears at the same column below all of the document’s \( s_j \) weights. If the document is rejected, which means that \( U < T_2 \), the value of \( U \) is depicted with a red background. The fourth column of Table 3 is the provisional classes of each document which correspond to the security areas in which the document has the greatest weights (equal to 5). The fifth column is the classes of each document after the NLU process. The document is classified in one or more classes according to its meaning, and to the definition of the five classes (Detection, Prevention, Interruption, Extermination, Healing).

Table 3: Experimental results of papers’ filtering.

<table>
<thead>
<tr>
<th>Title</th>
<th>Weights from AI</th>
<th>Weight s from security</th>
<th>Classes of highest weights</th>
<th>Pre-class</th>
<th>Final classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[24]</td>
<td>( a_{34} = 1 )</td>
<td>( s_1 = 5 )</td>
<td>Intelligent ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( a_{35} = 1 )</td>
<td>( s_9 = 5 )</td>
<td>ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( s_2 = 5 )</td>
<td>Misuse ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( s_2 = 5 )</td>
<td>Misuse ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( s_{42} = 1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V = 1 )</td>
<td>( U = 125 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td>( a_3 = 5 )</td>
<td>( s_5 = 5 )</td>
<td>Recognize Intrusion</td>
<td>Intrusion Detection</td>
<td>Detection</td>
</tr>
<tr>
<td></td>
<td>( a_{51} = 1 )</td>
<td>( s_6 = 5 )</td>
<td>Anomaly ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( a_{36} = 1 )</td>
<td>( s_9 = 5 )</td>
<td>ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V = 5 )</td>
<td>( U = 125 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[26]</td>
<td>( a_1 = 5 )</td>
<td>( s_7 = 5 )</td>
<td>Malicious Email Detection</td>
<td>Malicious Email Detection</td>
<td>Detection</td>
</tr>
<tr>
<td></td>
<td>( V = 5 )</td>
<td>( U = 5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Formula</td>
<td>Description</td>
<td>Detection Count</td>
<td>Prevention Count</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-------------</td>
<td>-----------------</td>
<td>------------------</td>
<td></td>
</tr>
</tbody>
</table>
| [27]  | \( a_6 = 5 \)  
  \( a_{15} = 5 \)  
  \( s_{11} = 5 \)  
  \( s_9 = 5 \) | Detecting Web-Based Attacks  
  Detecting Web-Based Attacks | \( V = 25 \)  
  \( U = 25 \) | |
| [28]  | \( a_{58} = 1 \)  
  \( s_{16} = 5 \) | Threat Prevention  
  Intelligent Threat Sensing Engine (ITSE), Intelligent Prevention Engine (IPE) | Threat Prevention,  
  Intelligent Threat Sensing Engine (ITSE),  
  Intelligent Prevention Engine (IPE) | Detection, Preventio  
  n and Isolation |
|       | \( a_{39} = 1 \)  
  \( s_{17} = 5 \) | | | |
|       | \( a_{46} = 1 \) | | | |
|       | \( a_{19} = 5 \) | | | |
|       | \( a_3 = 5 \) | | | |
|       | \( a_{32} = 1 \) | | | |
|       | \( V = 25 \)  
  \( U = 25 \) | | | |
| [29]  | \( a_3 = 5 \)  
  \( s_9 = 5 \) | ID  
  Intrusion Detection | Detection, Preventio  
  n and Isolation | |
|       | \( a_{20} = 5 \) | | | |
|       | \( a_{10} = 5 \) | | | |
|       | \( a_{11} = 5 \) | | | |
|       | \( a_{42} = 1 \) | | | |
|       | \( a_{17} = 5 \) | | | |
|       | \( a_{21} = 5 \) | | | |
|       | \( V = 15625 \)  
  \( U = 5 \) | | | |
| [30]  | \( a_{29} = 5 \)  
  \( s_{18} = 5 \) | Detection of Polymorphic Networks Attacks | Detection of Polymorphic Networks Attacks | Detection |
|       | \( a_1 = 5 \) | | | |
|       | \( V = 25 \)  
  \( U = 5 \) | | | |
| [31]  | \( a_{22} = 5 \)  
  \( s_4 = 5 \) | Adaptive ID  
  Detecting SQL Injection Attacks | Intrusion Detection,  
  Detecting SQL Injection Attacks | Detection and Healing |
|       | \( a_1 = 5 \)  
  \( s_{22} = 5 \) | | | |
|       | \( a_5 = 5 \)  
  \( s_9 = 5 \) | ID | | |
|       | \( a_{23} = 5 \) | | | |
|       | \( V = 625 \)  
  \( U = 125 \) | | | |
| [32]  | \( a_6 = 5 \)  
  \( s_{23} = 5 \) | Detect Malicious Web Pages  
  Detect Malicious Web Pages, Malware Executable | Detect Malicious Web Pages,  
  Malware Executable | Detection |
|       | \( a_{32} = 1 \)  
  \( s_{24} = 5 \) | Malware Executable | | |
|       | \( V = 5 \)  
  \( U = 25 \) | | | |
It needs to be mentioned that all of the steps in all phases were carried out manually by humans due to the primary nature of the proposed methodology.

All of the documents that we used in our experiments are about security issues and use AI methodologies. Although, we must notice that there are some documents in Table 3, which have been rejected during the procedure of filtering. This happened because the authors of these documents have not used the right terms in the documents’ main title, subtitles, and key words. We also observed, and needs to be said, that most of the documents have no key words. The lack of key words or the wrong choice of them affect the weights, $V_A$ and $U_S$, of the documents and may lead to their rejection.

Another inference that occurs from Table 3 is that some survey papers pass the steps and have heavy weights, $V_A$ and $U_S$. This is something we expected, and is not a problem. The
survey papers can easily be recognized from their big value of the \( V_A \) and the small value of \( U_S \) weight and vice versa. So, these papers may be assigned to a new separate class.

Finally, it is interesting to notice that most of the papers belong only to the Detection category, with fewer in the Prevention category, and so on. At last, the Healing category has the fewest papers. This allocation of the papers is expected since it is much more difficult to create a methodology that can heal a system after an attack.

2.4. Conclusions

In this chapter we suggest a new methodology for document filtering/categorization. We categorized published scientific papers that solve security problems using AI techniques. First, we set some criteria that the documents must meet. For every document that meets all the criteria, we extracted its meaning and classified it in predefined categories. We also demonstrated some examples of this procedure.
Chapter 3: Kernel extraction of NL sentences

Since our goal is to create an NLU methodology, the extraction of the kernel, or kernels of NL sentences is of the utmost importance. This is because the kernel of each sentence gives us its general meaning. Thus, kernel extraction is a necessary step. In this chapter we present our current algorithm written in Java, for kernel extraction based on the parse tree of NL sentences.

Kernel extraction or triplet extraction (these terms can be used interchangeable) has been study for many years. In [40], and [41] the authors present their methodology for triple extraction. This work is one of the most recent works similar to ours. Similarly to us the authors of these paper use the structure of parse tree of a sentence to extract the subjects, predicates and objects (or in other words the agents, actions and patients). They also map the extracted triplets to a semantic graph. However, our proposed methodology it doesn’t only focus on the extraction of the kernel. Since our goal is the representation of kernels through SPN graphs for event/action association, the type of connection between the different elements of the graph, as well as timing and flow of the different events play significant role. Thus, our kernel extraction methodology extracts the relations/connections between different agents, actions, and/or patients, in addition to the agents, actions, patients. Also, it keeps information about the verb’s tense, which is later been used for the representation of kernels into SPN graph. As a result, our methodology follows the same main steps as most of the existing kernel extraction methods. Then again, it has all the
unique features that serve our purpose to map the event/action associations (existing in a NL documents) to SPN graphs.

3.1. Parse trees

The first step in extracting the kernel of an NL sentence is to produce the parse tree. To do this, we use the Stanford parser [37], which is one of the Stanford CoreNLP toolkid components [38]. Then, we follow a set of rules that extracts the kernel based on the structure of the parse tree.

The parse tree of a NL sentence contains all of its words tagged according to their grammatical and syntactical role. Table 4 and Table 5 present the most common used Part of Speech (POS) and chunk tags, respectively, of syntactic trees as it is given in [39]. More detailed explanation of all of the tags can be found in [39].

Table 4: POS (Part of Speech) tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>conjunction, coordinating</td>
<td>and, or, but</td>
</tr>
<tr>
<td>CD</td>
<td>cardinal number</td>
<td>five, three, 13%</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
<td>the, a, these</td>
</tr>
<tr>
<td>EX</td>
<td>existential there</td>
<td>there were six boys</td>
</tr>
<tr>
<td>FW</td>
<td>foreign word</td>
<td>mais</td>
</tr>
<tr>
<td>IN</td>
<td>conjunction, subordinating or preposition</td>
<td>of, on, before, unless</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
<td>nice, easy</td>
</tr>
<tr>
<td>JJR</td>
<td>adjective, comparative</td>
<td>nicer, easier</td>
</tr>
<tr>
<td>JJS</td>
<td>adjective, superlative</td>
<td>nicest, easiest</td>
</tr>
<tr>
<td>LS</td>
<td>list item marker</td>
<td></td>
</tr>
<tr>
<td>MD</td>
<td>verb, modal auxiliary</td>
<td>may, should</td>
</tr>
<tr>
<td>NN</td>
<td>noun, singular or mass</td>
<td>tiger, chair, laughter</td>
</tr>
<tr>
<td>NNS</td>
<td>noun, plural</td>
<td>tigers, chairs, insects</td>
</tr>
<tr>
<td>NNP</td>
<td>noun, proper singular</td>
<td>Germany, God, Alice</td>
</tr>
<tr>
<td>NNPS</td>
<td>noun, proper plural</td>
<td>we met two Christmases ago</td>
</tr>
<tr>
<td>PDT</td>
<td>predeterminer</td>
<td>both his children</td>
</tr>
<tr>
<td>POS</td>
<td>possessive ending</td>
<td>'s</td>
</tr>
<tr>
<td>PRP</td>
<td>pronoun, personal</td>
<td>me, you, it</td>
</tr>
</tbody>
</table>

25
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Words</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>noun phrase</td>
<td>DT+RB+JJ+NN + PR</td>
<td>the strange bird</td>
</tr>
<tr>
<td>PP</td>
<td>prepositional phrase</td>
<td>TO+IN</td>
<td>in between</td>
</tr>
<tr>
<td>VP</td>
<td>verb phrase</td>
<td>RB+MD+VB</td>
<td>was looking</td>
</tr>
<tr>
<td>ADVP</td>
<td>adverb phrase</td>
<td>RB</td>
<td>Also</td>
</tr>
<tr>
<td>ADJP</td>
<td>adjective phrase</td>
<td>CC+RB+JJ</td>
<td>warm and cozy</td>
</tr>
<tr>
<td>SBAR</td>
<td>subordinating conjunction</td>
<td>IN</td>
<td>whether or not</td>
</tr>
<tr>
<td>PRT</td>
<td>Particle</td>
<td>RP</td>
<td>up the stairs</td>
</tr>
<tr>
<td>INTJ</td>
<td>Interjection</td>
<td>UH</td>
<td>Hello</td>
</tr>
</tbody>
</table>

Table 5: Chunk tags

The grammatical role of each word is defined by its POS tag, while chunk tags are assigned to groups of words and define their syntactical role. Specifically, “S” tag indicates the beginning of a sentence. “NP”, “VP” or “PP” chunk tags indicate a group of word that belongs to a noun, verb, or prepositional phrase, respectively. POS tags that have a first
letter N, V, or P indicate nouns, verbs, or prepositions, respectively. Other letters of a noun
POS tag may indicate plural, singular, proper or non-proper noun. Additional letters of a
verb POS tag may indicate plural, singular, the verb’s tense and grammatical person. Figure
4 shows an example parse trees produced by Stanford parser for the following text: “Tom
has a dog. The dog is brown.”.

```
(ROOT (S (NP (NNP Tom)) (VP (VBZ has) (NP (DT a) (NN dog))))).
```

```
(ROOT (S (NP (DT The) (NN dog)) (VP (VBZ is) (ADJP (JJ brown))))).
```

Figure 4: Parse trees of the text: “Tom has a dog. The dog is brown.”.

### 3.2. Our current algorithm for kernel extraction

Based on the study of different cases (structures) of parse trees of NL sentences, we define
our set of rules for the extraction of kernels. Thus, it is more than possible that our proposed
algorithm, written in Java, does not work correctly in cases that haven’t been seen yet.
Although, we experimented by applying our algorithm on different NL texts, which helped
us to make improvements and incorporate different kernel structures. This way we cover a
large percentage of different cases, while focusing on generally common forms of kernels
without considering extreme cases. However, because of the great variety of the English
language, there will still be cases in which our algorithm doesn’t extract parts or even the
whole kernel correctly. Appendix A illustrates the evaluation of different versions of our algorithm.

Our current algorithm (latest version) uses the following rules to extract the kernel(s) based on the parse tree of an NL sentence:

1. Collects all the nouns and adjectives that are before a verb phrase (VP) and the connections between them.
2. Associates each adjective with its corresponding noun(s) and keeps it in the verb’s record.
3. Collects all the verbs in a verb phrase (VP tag) and the connections between them.
4. Do step 1 and 2 for nouns and adjectives inside the verb phrase.
5. When a verb phrase is closed (at the end of a VP tag), checks the tense of the verb:
   a. If the verb is in past tense then it forms kernels when the very first verb phrase is closed.
   b. Otherwise, it forms kernels every time that a verb phrase is closed.
6. In order to form the kernel it checks the voice of the verb:
   a. If the verb is in active voice then all the nouns before the verb phrase are considered agents and all the nouns in the verb phrase are considered patients.
   b. Otherwise, if the verb is in passive voice (has a VBN tag) and there is a preposition after it then swap agents with patients.
7. Saves kernel(s).

On top of the above described algorithm, we have set some extra rules that lead to the correct extraction of kernels. These rules, which can be considered assumptions, are as follows:

1. Phrases (in a sentence) that contain verbs appearing in one of the forms: a) “by performing”, “by generating”, etc or b) “for updating”, “for performing”, etc, are considered explanatory phrases. These phrases give more information for the preceding action (previous kernel). Thus, they are missing the agent, because they
reference to the preceding agent. Explanatory verbs, as well as, their corresponding nouns, are associated with their “main” verb (preceding verb) and kept in its record. For example, in the sentence, “The main processor controls the entire recognition process by generating proper signals to the two co-processors (PE1, PE2).” the explanatory phrase is ‘by generating proper signals to the two co-processors (PE1, PE2)’ and the “main” verb is ‘controls’.

2. Phrases (sub-sentences) starting with the word ‘which’ or ‘that’ are also considered explanatory phrases.

3. Adjectives are considered agents and patients only if there are no other agents or patients.

4. Nouns that follow a 'before' or 'after' preposition are not considered agents or patients (e.g. The project will have been completed before the deadline).

5. Some words, although tagged as nouns (NN tag), are not considered agents or patients. For example, these are: ‘pm’, ‘tonight’, ‘week’, ‘year’, ‘years’, ‘month’, ‘months’, ‘yesterday’

6. The word ‘time’ is considered agent or patient if, and only if, it does not follow a preposition. Otherwise, it is excluded from agents or patients.

7. If there is a personal pronoun (PRP tag), the program checks if it is inside a prepositional phrase (PP tag). If so, it checks if the verb is in active voice. If both conditions are true, then the personal pronoun is excluded from agents and patients. For example, in the sentence, “Tom received several presents from all of us.” the personal preposition ‘us’ is excluded from agents or patients. Otherwise, the preposition is considered an agent or patient.

8. If a modal auxiliary verb precedes a verb, then it is considered part of the verb (e.g. My mother would always make the pies).

9. If a preposition immediately follows a verb, it checks if the combination of the verb and the preposition belong to the table with prepositional verbs. If so, the preposition is considered part of the verb.

Further, we have to mention that elements such as adjectives (when they aren’t considered agents or patients) and cardinal numbers are collected and associated with their corresponding nouns. These elements are kept inside the verb’s record. The remaining elements of a sentence (which aren’t nouns, verbs, adjectives, or cardinal numbers) such as articles, and prepositions, can also be kept in the verb’s record. Each of these elements
has to be associated to the word that it is referred to. In this case there is no information loss. Thus, the recovery of the initial sentence is easy and lossless. We have to point out that at this point we are interested in keeping adjectives and cardinal numbers only. However, our algorithm can be easily extended to hold every single element of a sentence, if it is needed in the future.

We chose to keep different elements inside verb records, because it fits our purposes the best. Our goal is to associate events/actions of the same agent or patient. To do this, we combine nodes that are reference to the same agent or patient at the SPN graph. Although, we do not combine action nodes in order to avoid confusion of different kernels. Thus, keeping adjectives and/or cardinal numbers inside the verb’s record also helps the avoidance of possible confusions. Each adjective and/or cardinal number has first been associated with corresponding noun. More detailed explanation is given in the following sections.

As soon as the extraction of kernels is complete, our program automatically constructs the corresponding SPN graphs. The procedure of constructing an SPN graph, once a kernel is given, is explained in detail in the following chapters.
Chapter 4 : *Glossa: A formal representation of kernels*

The previous chapter presented the algorithm for the extraction of kernels of NL sentences based on their parse trees. Every natural language (NL) sentence is composed of a "kernel". This kernel represents "who is doing what to whom" (Agent → Action → Patient). However, a principal problem that one deals with during the NLU procedure, is the several different forms a kernel can take on. This depends on how many agents, actions and patients the kernel contains, and how these agents, actions and patients are connected.

In this chapter we present a formal language (FL), called *Glossa* language (*Glossa* is the Greek word for language), which we developed based on the kernel (A→V→P) of a NL sentence for NLU purposes. Specifically, we define an FL that assists in the modeling of all different forms of a kernel using its basic components, which are: A, V, P (agents, verbs - actions, and patients). The proposed FL will also provide an easy method for the combination of different kernels and representation of these combinations.

Our formal language will help us to easier represent NL sentences into a state machine in order to assist the NLU process. This is because it provides a formal representation of the extracted kernels. In other words, the formal language that we define in this chapter will lead to an easy and automatic transformation of NL sentences into SPN graphs. We chose to use Stochastic Petri-nets (SPNs) as the state machine for the representation of kernels, due to its capabilities to represent structural and functional information (timing, flow) as well.
The selection of a formal language for modeling the structural behavior of kernels, was based on the idea that others have used similar formal models in the past for expressing kernels [2]. The formal representation of kernels reveals different aspects of their structure and behavior and helps us to avoid inconsistencies.

This chapter is organized into four main sections. Section 1 is a brief overview of the basic definition of a formal language. This section helps the reader to follow the remainder of the chapter. In Section 2, the formal definition, the properties, and the alphabet of our proposed formal language, called Glossa language, are presented. Section 3 contains examples of the representation of NL sentences with Glossa language expressions. Finally, we conclude in Section 4.

4.1. Notations and definitions

In this section we give the theoretical background on which we based the construction of the proposed formal language, Glossa. This brief overview will be useful for the reader in order to better understand our work. As we mentioned, our goal in this chapter is to introduce a formal language for the representation of kernels of NL sentences. For this reason, we provide the reader with the basic definition of a formal language, as well as its basic components and how we can define a formal language.

At first we give the basic definition of a formal language. According to [42], a formal language is a set of strings of symbols (alphabet) that may be constrained by rules that are specific to that particular language. The alphabet (Σ) of a formal language is the set of symbols, letters, or tokens from which the strings of the language may be formed. The strings formed from this alphabet are called words, and the words that belong to a particular
formal language are sometimes called well-formed words. The set of all words over an alphabet $\Sigma$, is usually denoted by $\Sigma^*$. A formal language, $L$, over an alphabet $\Sigma$, is a subset of $\Sigma^*$, which is a set of words over that alphabet.

A formal language is often defined (or generated) by means of a formal grammar. A formal (generative) grammar $G$, is an ordered four-tuple $(V_N, V_T, S, F)$ where $V_N$ and $V_T$ are finite alphabets of terminal and non-terminal symbols, respectively. Where $V_N \cap V_T = \emptyset$, $S$ is a distinguished symbol of $V_N$, and $F$ is a finite set of ordered pairs $(B, Q)$ such that $P$ and $Q$ are in $(V_N \cup V_T)^*$, and $B$ contains at least one symbol from $V_N$ [43]. Thus, the formal language that corresponds to the previous grammar $G$ is: $L(G) = \{ W | S \xrightarrow{G} W \text{ and } W \in V_T^* \}$ [43].

**4.2. Glossa: A formal language**

This section presents the definition of *Glossa* language, but first it provides the reader with the motivation behind its construction. For a moment, consider all the possible different forms that a kernel of an NL sentence can take. For example, the simplest case is when the kernel is composed of one agent, one action and one patient. Although, there are also kernels that have more than one agent, action or patient connected with an “and” or an “or” relationship in each case. Moreover, there are kernels that include combinations of multiple agents, actions and patients. As one may realize, these combinations are many and sometimes complicated. Thus, the problem that arises is how someone can formally model all of the different forms of a kernel (different combinations of agents, actions, and patients). The problem becomes more complex when someone wants to model a sentence that contains more than one kernel connected with an “and” or an “or” relation. In order to
overcome these problems we need a model that is able to represent, in a quick and automated way, any form of a kernel, no matter how many agents, actions or patients it contains, and how these are related (e.g. “and”, “or” connection). For this reason, we propose a formal language whose key components are symbols which represent agents, actions and patients of an NL sentence. We call this formal language *Glossa*. We need to stress that we focus only on the kernel of NL sentences and not on the remaining words, because the kernel keeps the basic meaning of a sentence.

### 4.2.1. The alphabet

Before we define *Glossa* language we need to set its alphabet. Thus, suppose that we associate to each agent of a kernel of an NL sentence a symbol $A_i$. Then, each $A_i$ represents a word of the English vocabulary that can be used as an agent in an NL sentence. So, $\forall i \in \mathbb{Z}, A_i = \{x \mid x$ is a noun, proper noun, personal noun, wh-pronoun, possessive wh-pronoun, or an adjective of the English vocabulary\}. We also associate to each verb of a kernel of an NL sentence a symbol $V_i$. Then, each $V_i$ represents a word of the English vocabulary that can be used as a verb in an NL sentence. So, $\forall i \in \mathbb{Z}, V_i = \{x \mid x$ is a verb or a modal auxiliary verb of the English vocabulary\}. Finally, we associate to each patient of a kernel of an NL sentence a symbol $P_i$. Then, each $P_i$ represents a word of the English vocabulary that can be used as a patient in an NL sentence. So, $\forall i \in \mathbb{Z}, P_i = \{x \mid x$ is a noun, proper noun, personal noun, wh-pronoun, possessive wh-pronoun, or an adjective of the English vocabulary\}. We need to point out here that $A_i$s (agents) and $P_i$s (patients) take values in the same set, because in natural language words that can be in the place of an agent can also be in the place of a patient of an NL sentence. Based on all the above, we have the following set $\Sigma = \{A_i, V_i, P_i \mid i \in \mathbb{Z}\}$ that makes up the alphabet of *Glossa* language.
4.2.2. Grammar

Now that we have set the alphabet of *Glossa* language, we can describe the grammar $G$ ($(V_N, V_T, S, F)$) that defines it. So, we have:

- $V_N$ is the set of non-terminal symbols with $V_N = \{K, A, V, P, S\}$
- $V_T$ is the set of terminal symbols with $V_T = \Sigma \cup \{i | i \in \mathbb{Z}\} \cup \{#, !, *, \%, ^, ~, @, $, (, ), [ , ]\}$
- $S$ is the start symbol of the grammar $G$

$F$ is the set of production rules which are defined as follows:

- $S \rightarrow [K] \mid [K] \# S \mid [K] ! S$
- $K \rightarrow (A) (V) (P)$
- $A \rightarrow A_1 * A_2 * A_3 * \ldots * A_n$ (represents “and” connection between agents)
- $A \rightarrow A_1 \% A_2 \% A_3 \% \ldots \% A_n$ (represents “or” connection between agents)
- $P \rightarrow P_1 @ P_2 @ P_3 @ \ldots @ P_n$ (represents “and” connection between patients)
- $P \rightarrow P_1 S P_2 S P_3 S \ldots S P_n$ (represents “or” connection between patients)
- $V \rightarrow V_1 ^ V_2 ^ V_3 ^ \ldots ^ V_n$ (represents “and” connection between verbs)
- $V \rightarrow V_1 \sim V_2 \sim V_3 \sim \ldots \sim V_n$ (represents “or” connection between verbs)

In all cases above $n \in \mathbb{Z}$.

4.2.3. Operators

In the production rules that we have defined above we used some special symbols, which are $\#$, $!$, $\ast$, $\%$, $^$, $\sim$, $@$, $\$$. These are the operators of *Glossa* language. The explanation of each one of these operators are as follows:

- “$\#$” is an operator that represents “and” connection between kernels
- “$!$” is an operator that represents “or” connection between kernels
• “∗”, “^”, and “@” are operators that represent “and” connection between agents, verbs and patients, respectively

• “%”, “~”, and “$” are operators that represent “or” connection between agents, verbs and patients, respectively

Also, we have to mention that “(”, “)”, “[”, “]” are used in order to determine the scope of different operators.

Since we explained the meaning of each operator, now we can demonstrate their algebraic properties in order to have a complete definition. Thus, the operators ∗, %, ^, ~, @, $ are associative. This means that if e1, e2 and e3 are terminal or non-terminal symbols of Glossa language then the combinations (e1 ? e2) ? e3 and e1 ? (e2 ? e3), where “?” can be any of these operators, generate the same result. Also, the operators ∗, %, ^, ~, @, $ are commutative. This means that if e1 and e2 are two terminal or non-terminal symbols of Glossa language then the combinations e1 ? e2 and e2 ? e1, where “?” can be any of these operators, generate the same result. Additionally, the operators #, ! are associative. This means that if e1, e2 and e3 are kernels (whichever form of a kernel, e.g. [(A) (V) (P)]) then the combinations (e1 # e2) # e3 (or using ! operator) and e1 # (e2 # e3) (or using ! operator) generate the same result. In addition, the operators #, ! are commutative. This means that if e1 and e2 are kernels (whichever form of a kernel, e.g. [(A) (V) (P)]) then the combinations e1 # e2 (or using ! operator) and e2 # e1 (or using ! operator) generate the same result.

Moreover, we have to point out that using Glossa language, an expression with a combination of operators # and !, may occur. As an example, we may have (e1 # e2) ! e3, where e1, e2 and e3 are kernels. A case like that where there are three kernels connected
with an “and” relation first and then an “or” relation isn’t an unusual phenomenon in English natural language. In such a case, it is not clear which operator (relation) has higher priority, leading to ambiguity of the meaning of the NL sentence. This phenomenon is rare in natural language. Thus, the expressions: \((e_1 \# e_2) \! e_3\) and \((e_1 \! e_2) \# e_3\), don’t generate the same result, which means that we cannot tell if they have the same meaning.

### 4.2.4. Examples words of Glossa language

The grammar \(G\) that we defined in Section 4.2.2 can be classified as context-free. The generated language, denoted by \(L(G)\), is therefore a context-free language. The grammar \(G\) is called Glossa grammar and the generated language is called Glossa language. Some examples of words that are produced by Glossa grammar are:

\[
\begin{align*}
w_1 &= [(A_1) (V_1) (P_1)] \\
w_2 &= [(A_1) (V_1) (P_1)] \# [(A_1 \ast A_2) (V_1) (P_1)] \\
w_3 &= [(A_1) (V_1 \sim V_2) (P_1)] \! [(A_1) (V_1) (P_1)] \\
w_4 &= [(A_1 \% A_2) (V_1) (P_1 \& P_2)] \! [(A_1 \% A_2) (V_1) (P_1)] \# [(A_1) (V_1) (P_1)]
\end{align*}
\]

The definition of the previous \(G\) grammar can be slightly modified so that allowing the definition of the empty word \(\varepsilon\) by adding a new rule: \(S \rightarrow \varepsilon\). Then, the new grammar \(G'\) remains context-free, and \(\varepsilon \in L(G')\) so that: \(L(G') = L(G) \cup \varepsilon\).

### 4.3. Examples

In this section we present examples of Glossa expressions of different forms of kernels of NL sentences that contain different operators. Particularly, we take an NL text with few
lines and we convert it into expressions of *Glossa* language, using the previously defined rules, alphabet and grammar.

The text that we process is the following: “*Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don’t remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom’s mother was serving food and his father was playing guitar. They all had fun.*”. First, we split this text into its sentences and for each sentence we extract its kernel (one or many). We have to stress that there are already many different tools implemented that someone can use to separate a text into sentences. In our case we use the Stanford CoreNLP toolkid for the production of parse trees. For the extraction of kernels we used our proposed algorithm presented in Chapter 3. Then, for each sentence we represent its kernel (one or many) using *Glossa* language.

For example the first sentence, “*Tom had his birthday yesterday.*” of the processed text has only one kernel, which is “*Tom→had→birthday*” (Agent→Verb→Patient). This is a simple form of kernel which is composed of one agent, one action and one patient. The steps that we follow in order to represent this kernel using *Glossa* language are as follows:

\[
S \rightarrow [K] \\
\rightarrow [(A) (V) (P)] \\
\rightarrow [(A1) (V1) (P1)] \\
\rightarrow [(Tom) (had) (birthday)]
\]

In the case of the sentence “*Tom was teasing and talking to Kate and Sue.*” the kernel is composed of two verbs connected with an “and” relation (which are “teasing” and “talking
to”) and also two patients connected again with an “and” relation (which are “Kate” and “Sue”). As a result the steps that we follow, using the production rules of *Glossa* language, in order to represent this kernel are:

\[
S \rightarrow [K] \\
\rightarrow [(A) (V) (P)] \\
\rightarrow [(A1) (V1 \land V2) (P1 @ P2)] \\
\rightarrow [(Tom) (teasing \land talking_to) (Kate @ Sue)]
\]

In the case of the sentence “Tom’s mother was serving food and his father was playing guitar.” there are two kernels that are connected with an “and” relation. The first kernel is “mother→serving→food” and the second kernel is “father→playing→guitar”. So, the steps that we follow are:

\[
S \rightarrow [K] \ # S \\
\rightarrow [K] \ # [K] \\
\rightarrow [(A) (V) (P)] \ # [(A) (V) (P)] \\
\rightarrow [(A1) (V1) (P1)] \ # [(A1) (V1) (P1)] \\
\rightarrow [(mother) (serving) (food)] \ # [(father) (playing) (guitar)]
\]

Following similar procedure with the above examples we convert each one of the sentences of the previous NL text into expressions of *Glossa* language. Therefore, we have the following expressions:

- [(Tom) (has) (birthday)]
- [(friends) (went) (house)]
- [(Mike * Kate) (brought) (cake)]
- [(Sue % Penny) (brought) (juices)]
We have to point out that each expression above corresponds to a sentence of the previous NL text.

### 4.4. Conclusions

In this chapter we introduced the *Glossa* formal language which is used in order to model and represent NL sentences, based on the basic components of their kernels (which are agents, actions and patients). We also demonstrate some clarification examples of how someone can represent and model different forms of kernels of NL sentences using *Glossa* language.

The reason that we constructed the *Glossa* language is because our future goal is to represent NL sentences into state machines and more specifically into SPN graphs. Hence, we need a model that a machine can interpret and formally represent NL sentences. It should also support the transformation of NL sentences into SPNs. Since we now have this model, *Glossa* language, we will further define a visual representation of NL sentences using SPN graphs in Chapter 6.
Chapter 5 : NLU: Kernel to SPN graph mapping

In the previous chapters we described the extraction of kernels and their formal representation through *Glossa* language for NLU purposes. *Glossa* language constitutes our basis for the mapping of kernels into SPN graphs. Glossa expressions hold the structural information of kernels. So, now we introduce the function components (timing and flow of events) through their SPN graph representation.

Some of the research activities on the field of NLP and NLU are about the following topics: event resolution (ER), grammar annotation (GrA), information mining (IM), knowledgebase (K), labeling (Lab), novelty detection (ND), question/answer (QA), redundancy reduction (Red), semantic relatedness (SR), similarity measure (SM), summarization (Sum), textual entailment (TE), word sense disambiguation (WSD), and word sense induction (WSI) [5]. Over the past ten years, research in these areas has moved toward graph-based methods. The reduced complexity of graph methods over vector methods offers a more compressed and efficient concept representation of text.

Most researchers in this field use simple structural graphs. Nodes of graphs may represent words, sentences, events or senses (or meanings). They usually apply different clustering and searching (walk) algorithms on graphs and use different similarity measures for many purposes. As well, some researchers use machine-learning algorithms like Naïve Bayes and Support Vector Machines. Others use techniques, such as singular value decomposition (SVD), vector space modes and ranking algorithms [5]. In particular, their purposes are as
follows: find similar words or topics in nearby sentences or similar documents [44], [45], [46], [47]; extract word synonyms [48]; determine the intended meanings of words [44], [49], [50], [51], [52], [53], [54], [55]; measure semantic relatedness of text [56], [57]; other.

In NL document understanding the graph-based methods and machine learning techniques don’t provide an efficient events association scheme. Thus, the purpose of this chapter is to introduce a new methodology that captures the natural language understanding of events and models those using Stochastic Petri Nets (SPNs). In particular, we obtain the syntactic and semantic analysis of events in terms of agents, actions, and patients from subject nouns, verbs, and object nouns within each phrase and sentence of a text. Then, we produce a graph consisting of nodes representing nouns and verbs, and edges representing their relations. More specifically the graph provides a baseline implementation, which we could relate to other graph methodologies, and offers a structured approach to NLP and NLU from text. Next, we embed into our model a new NL text to Stochastic Petri Net (SPN) graph conversion methodology in order to represent events associated with NL text. Moreover, SPN graphs provide not only the structured representation that graphs do, but also other capabilities, such as representing and adjusting timing using its transition components, constraining flow with its inhibiting places, stochastic behavior of its markings, and color markings [58], [14]. We use these added capabilities of SPN modeling to capture new NLU capabilities of events from NL text.

The emphasis of this chapter is on putting the basic foundation for the representation of simple kernels (which contain one agent, one action and one patient) of natural language
sentences into SPN graphs. In order to make the transition from kernels to SPN graphs easier to the reader we use simple graphs as an intermediate step. In Chapter 6 we describe the conversion of NL sentences (which include any form of a kernel) into SPN graphs, using their form representation through Glossa expressions. Also, our work in [16] uses SPN graph model to remove ambiguity of events. To achieve this disambiguation we combine all the different meanings (SPN graphs) of each ambiguous sentence into one colored SPN graph. Although, we won’t further discuss disambiguation of events in this study.

The main organizational parts of this chapter are as follows. Section 1 presents the conversion of a simple kernel of an NL sentence into a graph. Section 2 provides an explanation of the mapping of graphs into SPN graphs. Finally, we conclude in Section 3.

5.1. Convert natural language sentences to graphs

5.1.1. Constructing graphs

The methodology here transforms subject nouns, verbs, and object nouns in each phrase and sentence of a text document into agents, actions, patients and then into graphs. The triplet of Agent → Action → Patient is the kernel of the sentence, based on which we construct the graph of the sentence. To find the kernel we first need to develop the syntactic tree of the sentence. Thus, for this purpose we use the Stanford parser that produces the Chomsky trees [59] of sentences.

In particular, to produce the graph we extract the kernel using the algorithm described in Chapter 3. Every component of the extracted kernel becomes a node of the graph. We find the connections (edges) between nodes extracting each word and detecting its function
(agent, action, patient, or other) based on its position (or level) in the parse tree. Also, we use the tags to identify nouns and verbs. For example, we use verb tag VBG to recognize verbs that are gerund verbs and the verb tag VBN to detect passive verbs. Comparing the relative values of the levels of two or three symbols (i.e., “S”, “NP”, and “VP”) can help determine a noun’s relationship to a near-by verb. Levels and order of words help determine whether the noun is an agent or a patient. In more detail, during the construction of the syntactic tree, each word of each sentence is analyzed to be associated to its properties and type (nouns, what type of noun, prepositions, etc). Next, we extract the kernel, using the algorithm presented in Chapter 3. After, we develop the graph connecting the nodes that correspond to words of the sentence’s kernel. In particular we connect the agent node to the action node and the action node to the patient node (Figure 5).

![Graph of the kernel of a sentence](image)

**Figure 5:** Graph of the kernel of a sentence.

### 5.2. Convert graphs to SPNs

The next step is the conversion of graphs, discussed in the previous section, into SPN graphs. Our goals are: (1) the representation and understanding of natural language sentences and (2) the association of events/action of different natural language sentences. To implement these goals we use SPNs which are state machines. We introduce a symbolic representation for them to achieve better visualization (the detailed explanation of this representation is in Section 5.2.1).
5.2.1. SPNs are state machines

SPNs are state machines, so SPN graphs represent the change of states. Representing NL sentences into SPNs, we capture the change of states of agents and patients after the action of verbs on them. In a SPN graph a transition represents an action. Places that are connected with a transition may change their states after the firing of it (Figure 6a). In our case (NL sentences) the state of an agent, A, and a patient, P, (at time $T_0$) may change to states $A'$ and $P'$ (at time $T_1$), respectively, after the impact of an action that a verb causes. Figure 6b shows the change of states of the agent and the patient after the impact of the verb. This is the SPN graph for the kernel on which we will based the representation of NL sentences into SPNs.

Since transitions represent actions, verbs are represented by timed transitions (thick rectangles) in SPNs. Moreover, agents and patients are represented by places in an SPN graph. Agents, patients and verbs are connected with arcs that represent their relations.

![Figure 6: (a) Basic state machine (b) SPNs to represent the kernel: V changes state of P to P' and A to A.](image)

In order to reduce the visual complexity of the produced SPN graphs, we consider a symbolic representation of the kernel. In this representation the two states of each agent
(patient) that correspond to the state before and after the action of the verb, are compressed into one place. This place of each agent (patient) incorporates both of its states (Figure 7). As a result, each agent (patient) is represented by only one “parent” place in an SPN graph. Consequently, all the transitions (verbs) that act on the same agent (patient), in any state of this agent (patient), are connected with the “parent” (representative) place of the agent (patient). Figure 7 also shows the feedback of the action to the agent with a dashed arc. In our future work we will not show the feedback arc anymore for simplicity reasons because it does not play a significant role in the meaning of a sentence.

![Figure 7: SPN graphs: symbolic representation of kernel.](image)

As written above this symbolic representation is only for clear and better visualization. It is very important to mention that there is no loss of information in this implementation. However, this representation helps us to visualize the combined SPN graphs better, in which the number of nodes and arcs is increasing. Most importantly the symbolic representation does not impact the extracted meaning of a sentence.
5.2.2. Constructing SPN graphs

The conversion of graphs, discussed previously, into SPNs is a straightforward procedure. Each agent and patient node is represented by a place and each action node is represented by a transition.

In SPN graphs, places (agents and patients) may change states according to transitions (verbs). Consequently, we should have two different states (places) of every agent or patient. Nevertheless, using the symbolic representation that we explained previously, we will have only one place of each agent and one place of each patient.

For example we consider the sentence, “John teases Kate.” and we want to construct its SPN graph. First, we need to construct its “simple” graph. The produced syntactic tree of the sentence is shown in Figure 8. The extracted kernel of this sentence is: agent is “John”, action is “teases”, and patient is “Kate”. Figure 9a is the simple graph of this sentence.

The produced SPN graph of this sentence without using the symbolic representation is in Figure 9b. This figure shows the two different states of the agent “John” (places John and John’) and the two different states of the patient “Kate” (places Kate and Kate’), which are the states of them before and after the action of the verb.

However, using the symbolic representation we have only one place for the agent and one place for the patient. Each of these places contains all the different states of the word that represents (Figure 9c). Figure 9d is another representation of the SPN graph of Figure 9c. In this representation only the “parent” place is showed and the dashed arc has been omitted, for simplicity reasons. We will use the symbolic representation of SPN graphs and its simplified version from this point forward.
Figure 8: Syntactic tree of the sentence: “John teases Kate”.

Figure 9: (a) Graph of sentence: “John teases Kate.” (b) SPN graph of the kernel of: “John teases Kate” without using the symbolic representation (c) Detailed SPN graph of the kernel of: “John teases Kate” using the symbolic representation (d) Simplified SPN graph of the kernel of: “John teases Kate” using the symbolic representation.
5.3. Conclusions

In this chapter we explained the conversion of simple kernels of NL sentences into SPN graphs. The use of SPN graphs for the representation of natural language events gives us the opportunity to represent, except the structural information, the time (sequence) and the flow of the events of natural language documents. For a better understanding for readers we first show the mapping of kernels into simple graphs and then into SPN graphs. In the next chapter we describe the conversion of whole NL sentences into SPN graphs, based on their Glossa expressions.
Chapter 6: NLU: Sentences into SPN graphs representation

In this chapter we move from the representation of simple kernels to the representation of NL sentences into SPN graphs. Although significant research contributions on Natural Language Processing (NLP) and Natural Language Understanding (NLU) have been made over the last thirty years, several researchers are still working on these two fields, proposing new techniques or improving existing ones. This is because of the richness of the NL field and the ambiguity of the different meanings that NL words can get in different content.

NL input statements are analyzed and information is extracted. The extracted information must be represented in a useful way. Several ways to represent information or knowledge exist, and each method has distinct advantages and disadvantages. Example methods for knowledge representation are: Formal Logic, Statistical Based, Natural Logic, and Ontology Based.

Through the years, some systems that model English language into formal logic, such as first order logic, have been proposed [60], [61]. Formal logic representation provides limited flexibility, because it can only model basic types of quantifiers and provides only true or false answers. As a result, in the case of NLU a more scalable system that can handle the wide range of variations and exceptions of natural language is needed.

Several statistical methods have been used in different task of NLP and NLU, such as speech of text, build parse trees, identify a word sense, and understand the meaning of a
word [62], [63], [64], [65], [66]. For example, understanding the meaning of a word via statistical methods, usually are based on bag of words representation of documents. This representation is used for counting of words’ appearance, examination of surrounding words and calculation of distance to different situations that have been seen previously, in order to cluster them into a particular concept [67]. However, bag of words representation does not always achieve a higher level of understanding. All these statistical techniques have been widely studied for information retrieval in the NLP domain of clustering concepts [68], [69], [70], [71]. In our work we use the statistical based parser, included in Stanford CoreNLP toolkit, as the first processing step.

Natural Logic is another model for inferring and interpreting language. It attempts to encompass reasoning capacities and understand everything that can be stated in a language. It is mainly used for Natural Language Inference (NLI). NatLog, presented in [72], [73], [74], is the first computational model of natural logic working towards textual inference. NatLog has been added to the Stanford RTE (Recognition of Textual Entailment) system, which is also based on natural logic, leading to a better accuracy. However, Natural Logic also has its limitations, such that it cannot handle all types of inferences, and also because there is no universal natural logic for which all languages agree.

A number of systems use ontologies for knowledge representation [75], [76], [77], [78], [79]. In ontology-based representation the world knowledge of a target domain is represented by ontologies that have been created by hand-crafted rules. Ontologies can incorporate semantics into a system, and provide link between the syntax of the input and the semantics of the knowledge. They have been widely used in NLU lately. The main
weak point in this type of representation is that it must be specially crafted for a particular
domain and cannot easily handle new information outside of the specified domain.

Thus, all these activities expressed in forms of algorithmic techniques, systems, ontologies,
and logic have driven the field mostly in the direction of NL interpretation, where there are
still open problems to be resolved. On the other hand the Speech Recognition research
community is pushing the NL component towards Signal processing and understanding
direction using Hidden Markov Models (HMM). But at the end of the day, both groups are
talking about the same message NLU either in the form of documents or in the form of
signals.

Our proposed work, in this chapter, is based on the intersection of these domains in an
effort to bridge possible gaps and offer a vehicle for solving problems, by using features
from these domains. Thus, our proposed methodology presents NLU features, like the
extraction of the kernel (Agent → Action → Patient) from NL sentences. While at the same
time we use a formal language mechanism (Glossa) to map the kernel(s) into a state
machine. This will provide the timing-synchronization features to the methodology for
efficient association of events.

In particular, the core component of a simple natural language sentence is its kernel that
carries its basic meaning. State machines can efficiently represent the kernel of every
English natural language sentence [80], [16]. Each agent and each patient changes its state
after the impact of an action (verb) on them. As a result, in every action, each agent and
each patient participates with a different state. A state machine can easily capture and
represent all these different states of agents and patients, keeping trace of the continuity of
events. In particular, we use SPN graphs that provide us with additional capabilities of modeling timing and flow of events of natural language sentences. Same SPN-based representation of natural language sentences has been used in [14], [15], [16]. Although, in contrast to [14], [15], [16] our approach creates a more detailed graph of the kernels of English natural language sentences. The different types of connections (conjunction) between agents, actions, patients or kernels are taken into consideration and are represented in SPN graphs.

More specifically, we focus on NLU and propose a methodology for the understanding of English technical documents. In Chapter 4 we developed and presented a formal language (FL), Glossa, that models the structure of the kernel (A→V→P) of a NL sentence for purposes of NLU [81]. The basic components of the proposed FL are A, V, P (agents, verbs - actions, and patients) of a kernel and its purpose is to assist the modeling of all the different forms of a kernel. Glossa language leads to an easy and automatic transformation of NL sentences into a state machine; SPN graphs. Particularly, in this chapter we focus on a mapping between the FL and the visual representation of Stochastic Petri Nets (SPNs) for better understanding.

We chose to use Stochastic Petri-nets (SPNs) as the state machine due to their capabilities to represent structural and functional information as well (timing, flow), as we have already mentioned. Different state machines, such as hidden Markov models, have been widely used for several natural language processing and information extraction tasks [66], [15], [82], [83]. FASTUS is one of the most known systems that uses state machine for information extraction [83]. Similarly, to our approach they use various states of a finite
state machine, in order to model and extract nouns and verbs. They also associate extracted incidents of different phrases. However, in our work we use a state machine for purposes of natural language understanding and not for natural language processing or information extraction (as FASTUS).

In addition to sentences representation into SPN graphs, we also present a set of rules to combine SPN graphs for events/actions association. The proposed methodology merges two or more SPN graphs, which reference the same agent or patient and belong to different kernels. Thus, in the combined form of SPN graphs, actions from different NL sentences that reference the same agent or patient are being associated. This association of actions/events provides efficiency and makes the information retrieval easier and faster. Moreover, it is visually simpler, which leads to a better and faster capturing/understanding by humans.

The whole proposed procedure for NLU consist of four main steps; we first apply Anaphora Resolution (AR) to the processing text, then we extract the kernel(s) of each sentence separately, represent the kernel into SPN graphs and finally we apply a set of rules that we have defined for the combination of graphs. The step of kernel extraction has already been studied in Chapter 3. Each one of the remaining steps, as well as the motivation behind them, is presented in detail in this chapter.

This chapter is organized into four main sections. In Section 1 the mapping between Glossa language and SPNs graphs is presented, as well as some examples of converting natural language sentences into SPN graphs. Section 2.1 explains the use of AR in our program. Section 2.2 provides the reader with the set of rules for the combination of SPN graphs.
Section 3 illustrates examples of the representation of Glossa expression using SPNs, along with their combined representation. Section 4 is the summary of the chapter.

6.1. **Mapping Glossa expressions into an SPN**

As we have already mentioned, our goal is to represent NL sentences into SPNs in order to assist the NLU process. To accommodate such transformation, we define a mapping between Glossa language and SPN graphs. Through this mapping Glossa language that is used to represent NL sentences using the “kernel”, can be visually expressed by SPN graphs Glossa ↔ SPN, where Glossa is the formal language and SPN is the stochastic Petri-net model.

In Section 6.1.1 we explain and illustrate the basic components of Petri net visual representation in order to help readers that aren’t familiar with Petri nets, to better understand the proposed mapping. In Section 6.1.2 we define the mapping of Glossa language to SPN graphs, and illustrate the visual representation of Glossa language expressions through SPN graphs.

6.1.1. **Stochastic Petri Nets prerequisites**

In this section we demonstrate the visual representation of the basic components of Petri Nets. SPN is a specific category of Petri Nets. As a result, SPNs and Petri Nets have the same visualization components and follow the same visualization rules.

According to [84], [85], a Petri net consists of places, transitions, and arcs. Arcs can connect a place to a transition or vice versa, but never two places or two transitions. Places in a Petri net may contain a discrete number of tokens. Any distribution of tokens over the
places will represent a configuration of the net called a marking. Arcs are characterized by their capacity, which is the number of tokens that they can transfer. The default capacity is 1 and we use this in our mapping. A transition of a Petri net is enabled when there are sufficient tokens in all of its input places, which means that the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition. A transition may fire if it is enabled. When the transition fires, it consumes the required input tokens, and creates tokens in its output places. This results in a new marking of the net, a state description of all places. An additional type of arc in Petri nets is an inhibitor arc, which imposes the precondition that the transition may only fire when the place is empty.

In a graphic representation of a Petri net (Figure 10), places are depicted with circles (where each circle contains or not one or more dots called tokens), transitions with long narrow rectangles, and arcs as one-way arrows that show connections of places to transitions or transitions to places. Labels above arcs indicate their capacity, which means the maximum number of tokens that an arc can carry simultaneously. An inhibitor arc is represented by an arc terminated with a small empty circle [86]. More information about Petri Nets and Stochastic Petri Nets can be found in the corresponding literature [84], [85], [86], [87].

Figure 10: A simple Petri Net graph.
The representation of simple kernels into SPNs has been explained in detail in Chapter 5. Although, we need to mention that we will use the compressed, symbolic representation of the kernel into SPN as it is proposed in Chapter 5 [16]. Here, we present again some basic points for the representation of agents, actions and patients into SPN graphs. So, verbs are represented by timed transitions (thick rectangles), since transitions represent actions. Immediate transitions (thin rectangles) do not represent a verb, but we use them as auxiliary transitions for the correct connection of agents, action and patients under the rules of visual representation of SPNs. Places (big circles) mostly represent agents and patients. However, we also use some places (small circles), named neutral nodes, that do not represent an agent or a patient. Instead, they are used as auxiliary places for the correct connection of agents, action and patients under the rules of visual representation of SPNs.

6.1.2. Mapping of Glossa expression into SPN graphs

Since we already talked about the basic components of Glossa language and SPN graphs, we will now define the mapping between the two. Through this mapping we get a visual representation of NL sentences through SPN graphs. Thus, our NLU methodology receives benefit from the special capabilities of SPNs. Table 6 shows the visual representation of all the terminal symbols of Glossa language, with the exception of the operators, using SPN graphs. The visual representation of the operators of Glossa language using SPN graphs are shown in Table 7.

Table 6: Representation of Glossa’s terminal symbols with SPN components

<table>
<thead>
<tr>
<th>Glossa terminal symbol</th>
<th>NL meaning</th>
<th>SPN meaning</th>
<th>Visual Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ai</td>
<td>agent</td>
<td>place - state</td>
<td><img src="Ai" alt="Ai" /></td>
</tr>
</tbody>
</table>
Table 7\(^1\): Representation of Glossa’s operators with SPN components

<table>
<thead>
<tr>
<th>Glossa operator</th>
<th>NL meaning</th>
<th>Visual Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>“and” between kernels</td>
<td><img src="image1" alt="Diagram 1" /></td>
</tr>
<tr>
<td>!</td>
<td>“or” between kernels</td>
<td><img src="image2" alt="Diagram 2" /></td>
</tr>
<tr>
<td>*</td>
<td>“and” between agents</td>
<td><img src="image3" alt="Diagram 3" /></td>
</tr>
<tr>
<td>e.g. [A1 * A2 * ... * An]</td>
<td></td>
<td><img src="image4" alt="Diagram 4" /></td>
</tr>
<tr>
<td>%</td>
<td>“or” between agents</td>
<td><img src="image5" alt="Diagram 5" /></td>
</tr>
<tr>
<td>e.g. [A1 % A2 % ... % An]</td>
<td></td>
<td><img src="image6" alt="Diagram 6" /></td>
</tr>
<tr>
<td>^ after *</td>
<td>“and” between verbs following “and” between agents</td>
<td><img src="image7" alt="Diagram 7" /></td>
</tr>
<tr>
<td>e.g. [A1 * A2 * ... * An] [V1 ^ V2 ^ ... ^ Vn]</td>
<td></td>
<td><img src="image8" alt="Diagram 8" /></td>
</tr>
</tbody>
</table>

\(^1\) In all the visual representations of this table: n, r, f, k, j, d, p, g, h, m ∈ Z and 1 ≤ i ≤ m

\(^2\) For the visual representation of this operator each pair of agents must be connected with the depicted structure.
<table>
<thead>
<tr>
<th>^ after %</th>
<th>“and” between verbs following “or” between agents</th>
</tr>
</thead>
</table>
| e.g. [A1 % A2 % … % An] [V1 ^ V2 ^ … ^ Vn] | ![Diagram](image1)

<table>
<thead>
<tr>
<th>~ after *</th>
<th>“or” between verbs following “and” between agents</th>
</tr>
</thead>
</table>
| e.g. [A1 * A2 * … * An] [V1 ~ V2 ~ … ~ Vn] | ![Diagram](image2)

<table>
<thead>
<tr>
<th>~ after %</th>
<th>“or” between verbs following “or” between agents</th>
</tr>
</thead>
</table>
| e.g. [A1 % A2 % … % An] [V1 ~ V2 ~ … ~ Vn] | ![Diagram](image3)

<table>
<thead>
<tr>
<th>@</th>
<th>“and” between patients</th>
</tr>
</thead>
</table>
| e.g. [P1 @ P2 @ … @ Pn] | ![Diagram](image4)

<table>
<thead>
<tr>
<th>$</th>
<th>“or” between patients</th>
</tr>
</thead>
</table>
| e.g. [P1 $ P2 $ … $ Pn] | ![Diagram](image5)

In the first row of Table 7, there is the visual representation of the “and” connection between kernels. In this case we use one neutral place, which enables an immediate transition that fires, and enables, all kernels simultaneously (gives a token to every kernel).

On the other hand, in the case of the “or” connection between kernels, which is represented in the second row of Table 7, the neutral place enables only one of the immediate
transitions. Each immediate transition is connected with one kernel. When it fires, it enables the corresponding kernel.

In the third row of Table 7, there is the visual representation of the “and” connection between agents. In this case each agent place outputs arrowed arcs in order to connect to the verbs. In the case of the “or” connection between agents, which is represented in the fourth row of Table 7, each agent is connected with an immediate transition that fires, and enables, all the verbs. At the same time it blocks (inhibits) all the immediate transitions of the rest of the agents.

For the visual representation of the connection between verbs, we have to take under consideration how the agents before them are connected. As a result, we have four different cases. The first one is when there is an “and” connection between verbs following an “and” connection between agents (fifth row of Table 7). In this case we use an immediate transition which fires and enables all the neutral places connected to the verbs simultaneously (one neutral place for each verb). Also, each verb outputs one or more arrowed arcs in order to enable (connect to) the patients. The second case is when there is an “and” connection between verbs after an “or” connection between agents (sixth row of Table 7). The difference here is that each verb also outputs one or more arrowed arcs (feedback arrowed arcs) in order to unblock the rest of the agents. Also, all the verbs are connected with an immediate transition, using an intermediate neutral place for each one of them, which is used to enable the patients. The third case is when there is an “or” connection between verbs after an “and” connection between agents (seventh row of Table 7). In this case each verb is connected directly with each agent or with the corresponding
structure which connects the agents. Also, each verb outputs one or more arrowed arcs in order to enable the patients. The forth case is when there is an “or” connection between verbs after an “or” connection between agents (eighth row of Table 7). In this case each verb is again connected directly with each agent or with the corresponding structure which connects the agents. The difference is that besides the outputted arrowed arcs of verbs in order to enable the patients, each verb also outputs one or more additional arrowed arcs (feedback arrowed arcs) in order to unblock the rest of the agents.

In the ninth row of Table 7, there is the visual representation of the “and” connection between patients. In this case each agent place inputs arrowed arcs in order to connect to the verbs. In the case of the “or” connection between patients, which is represented in the tenth row of Table 7, we use a neutral place that when it is enabled, fires one of the immediate transitions connected to a patient (one immediate transition for each patient).

We have to stress some important points about how we connect the outgoing and incoming arcs of the visual representation of different operators. So, the following conditions must be true:

1. \( r_1 + \ldots + r_m = k \), which corresponds to the connection between agents and verbs (see Table 7)
2. \( \forall i \in Z, j_i = h_1 + \ldots + h_m \), which corresponds to the connection between verbs and patients (see Table 7)
3. \( j_1 + \ldots + j_m = g \), which also corresponds to the connection between verbs and patients (see Table 7)

One additional rule that must be followed is: \( \forall i \in Z, f_i = j_1 + \ldots + j_m \) or \( f_1 + \ldots + f_m = d \). This rule corresponds to the connection between agents and verbs, in cases where the agents are connected with an “or” relation (see Table 7).
For more clarification of the representation of different operators, we give some examples of the visual representation of expressions of Glossa language through SPN graphs in Table 8. In each example (row) we put emphasis on a different operator. The first column of the table contains the NL sentence that we want to convert into a Glossa expression. In the second column there are the derivation steps using Glossa language for the representation of the corresponding NL sentence into a Glossa expression. The third column is the visual representation of the corresponding Glossa expression into SPN graphs using the defined mapping.

Table 8: Examples of the visual representation of expressions of Glossa language with SPN graphs

<table>
<thead>
<tr>
<th>NL sentence</th>
<th>Glossa expression</th>
<th>Visual representation (SPN graph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>John teases Anna and Mike plays tennis.</td>
<td>$S \rightarrow [K] # S \rightarrow [K] # [K] # [(A) (V) (P)] # [(A1) (V1) (P1)] # [(A1) (V1) (P1)] # [(John) (teases) (Anna)] # [(Mike) (plays) (tennis)]$</td>
<td><img src="image1.png" alt="Diagram" /></td>
</tr>
<tr>
<td>(“and” connection between kernels)</td>
<td><img src="image2.png" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td>John teases Anna or Mike plays tennis.</td>
<td>$S \rightarrow [K] # S \rightarrow [K] # [(A) (V) (P)] # [(A) (V) (P)] # [(A1) (V1) (P1)] # [(A1) (V1) (P1)] # [(John) (teases) (Anna)] # [(Mike) (plays) (tennis)]$</td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
<tr>
<td>(“or” connection between kernels)</td>
<td><img src="image4.png" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td>John and Mike play chess.</td>
<td>$S \rightarrow [K] # [(A) (V) (P)] # [(A1 * A2) (V) (P)] # [(A1 * A2) (V1) (P1)] # [(John * Mike) (play) (chess)]$</td>
<td><img src="image5.png" alt="Diagram" /></td>
</tr>
<tr>
<td>(“and” connection between agents)</td>
<td><img src="image6.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>
At this point, we have to remind the reader that every agent or patient changes its state after the impact of an action, because of the state machine representation. For example, in the NL sentence “John sat down” the state of the agent, John, before the action, sat down, was “standing John”. The state of agent, John, after the action, is “sitting John”. Because SPNs are state machines, in an SPN graph of an NL sentence, all the different states of agents and patients must be represented. As we already mentioned we follow the representation
introduced in Chapter 5 [16], so we consider that each place of an agent or a patient, in
SPN graphs, contains all the different states of the corresponding agent or patient, before
and after the action. We have to point out that the different states of an agent or a patient
can’t be represented with Glossa language. However, they are implied in both cases of
natural language and Glossa language.

Table 9 contains a more complex example compared to those in Table 8. This example
contains two operators instead of one. Again the first column of the table contains the NL
sentence that we want to convert into a Glossa expression. In the second column there are
the derivation steps using Glossa language for the representation of the corresponding NL
sentence into a Glossa expression. The third column is the visual representation of the
corresponding Glossa expression using SPN graphs.

Table 9: A more complex example of the visual representation of expression of Glossa
language with SPN graph

<table>
<thead>
<tr>
<th>NL sentence</th>
<th>Glossa expression</th>
<th>Visual representation (SPN graph)</th>
</tr>
</thead>
</table>
| John and Mike tease and love Kate. | S → [K] → [(A) (V) (P)]
  → [(A1 * A2) (V) (P)]
  → [(A1 * A2) (V1 ^ V2) (P)]
  → [(A1 * A2) (V1 ^ V2) (P1)]
  → [(John * A2) (V1 ^ V2) (P1)]
  → [(John * Mike) (V1 ^ V2) (P1)]
  → [(John * Mike) (tease ^ V2) (P1)]
  → [(John * Mike) (tease ^ love) (P1)]
  → [(John * Mike) (tease ^ love) (Kate)] | ![Visual representation (SPN graph)](image) |

In the example that follows, we do the reverse procedure compared to the previous ones.
Given the SPN graph, we produce the Glossa expression. Then, the NL sentence is easily
acquired by the Glossa expression. This example is used in order to emphasize that there
is no ambiguity in the conversion of a Glossa expression to an SPN graph and vice versa,
due to the defined mapping. On the top of Figure 11 is the given SPN graph and on the
bottom is the corresponding *Glossa* expression. The part of the graph in the blue rectangle represents an “or” connection between two agents. The part of the graph in the red rectangle represents one verb (no connection with other verbs). Then finally, the part of the graph in the green rectangle represents an “and” connection between two patients. The NL sentence that is derived from the corresponding *Glossa* expression is the following: “*John or Mike brings cake and candles*”.

![Figure 11: Example of conversion of SPN graph to *Glossa* expression.](image)

6.1.3. **Examples**

In this section we illustrate some example SPN graphs of NL sentences produced by our proposed program, which is written in Java. As a visualization tool for the representation of SPN graphs, we use the java library called JUNG.

For ease of visual representation we choose to represent agents with filled green circles, patients with cyan filled circles, explanatory patients (patients that are part of an explanatory phrase) with orange, neutral (or pseudo) places with smaller gray filled circles,
verbs with red filled rectangles, explanatory verbs (verbs that are part of an explanatory phrase) with pink, and immediate transitions with thin black filled rectangles. Also, we have to point out that due to the limitation of JUNG library we aren’t able to represent inhibitor arcs with an arc that ends in a small empty circle. As a consequence we represent them with dashed arcs that end in an arrow.

Figure 12a shows the SPN graph of the sentence: “John and Mike play chess.”. Figure 12b shows the SPN graph of the sentence: “John teases and loves Kate.”. Figure 12c shows the SPN graph of the sentence: “John teases Kate or Anna.”. All the SPN graphs of Figure 12 have been produced automatically using our proposed NLU methodology.
Figure 12: (a) SPN graph of sentence: “John and Mike play chess.” (b) SPN graph of sentence: “John teases and loves Kate.” (c) SPN graph of sentence: “John teases Kate or Anna.”

6.2. Synthesis of SPN graphs

Since we presented the production of SPN graphs from kernels, we want to synthesize SPN graphs for event/action association purposes. In order to do this, we take a step back and incorporate anaphora resolution features to our program. This eliminates the appearance of pronouns in the place of agents or patients. Then, we define a set of rules for the correct combination of SPN graphs.

The proposed program, which is written in Java, takes the plain text as input and outputs its SPN graph. To produce the SPN graph, it applies anaphora resolution, parses the text using the Stanford parser, and extracts the kernels from each parsed tree. Subsequently, following the rules of Section 6.1 for the conversion of a kernel into an SPN graph, the program produces the graph of each kernel. Finally, it is able to produce the combined SPN graph of the whole text, combining different appearances (nodes/places) of the same agent or patient node. For the visual representation of SPN graphs we use the java library called JUNG [88].

6.2.1. Anaphora Resolution

As we mentioned, the first step of our program is to apply anaphora resolution. Anaphora resolution (AR), which most commonly appears as pronoun resolution, is the problem of resolving references to earlier or later items in the discourse. These items are usually noun phrases representing objects in the real world called referents, but can also be verb phrases,
whole sentences or paragraphs [89]. In other words, AR is the problem of resolving what a pronoun or a noun phrase refers to.

Since we represent kernels as graphs, the appearance of pronouns in the place of agents or patients makes it difficult or impossible to infer the referred to noun. As a result, before we extract the kernel(s) of an NL sentence we apply anaphora resolution and replace the pronouns with the corresponding noun. In order to do this, we use the Name Entity Recognition (NER) component of Stanford CoreNLP toolkit. We apply NER to each paragraph of the initial text separately, then replace only the pronouns with appropriate nouns as they are given by the output. NER in addition to associating pronouns with the appropriate nouns, it also associates nouns with other nouns (i.e. a person with his/her profession). Our program doesn’t take into consideration cases like this, but rather, it only uses the associations of pronouns with nouns. Thus, it replaces a word if, and only if, it is a pronoun. The examples in Table 10 shows the input and output text produced by our program after anaphora resolution.

Table 10: Examples of output text after Anaphora Resolution

<table>
<thead>
<tr>
<th></th>
<th><strong>Example 1</strong></th>
<th><strong>Example 2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input text</strong></td>
<td>John likes music. He is a musician.</td>
<td>Mary has a violin. She likes it. It is nice.</td>
</tr>
<tr>
<td><strong>Output text</strong></td>
<td>John likes music. John is a musician.</td>
<td>Mary has a violin. She likes the violin. The violin is nice.</td>
</tr>
</tbody>
</table>

In Example 1 of Table 10, anaphora resolution associates “a musician” with “John”. However, as we mentioned above our program doesn’t replace “a musician” with “John” because “a musician” isn’t a pronoun.
6.2.2. Combined SPN graphs

In the section we present one more feature of our system, which is the generation of what we call combined coloured SPN graphs (CSPN). These graphs have only one node/place for each different agent or patient. To contrast with uncombined SPN graphs, the same agent or patient may appear in more than one node of different kernels. In other words, kernels that refer to the same agent (i.e. John) are all connected to the same node/place of the graph. The same applies for kernels that refer to the same patient nodes/places. We call these graphs combined graphs because we combine all the different appearances of a specific agent/patient into one node/place. We have to point out that we don’t combine transitions that represent the same verb. This is because the created graph would make it impossible to define which agent does what (acts) to which patient.

The advantage of combined SPN graphs is the association of events/actions. Also, all of the information (actions) referenced to an agent or a patient is gathered together. This provides easier and faster information retrieval, along with a better visual representation, which leads to better and faster human understanding. However, the creation of combined SPN graphs requires a set of rules in order to prevent the information flow, the meaning of each kernel and also, provide consistency to the general rules of SPN graphs. As a consequence, we combine SPNs to coloured SPN graphs, because we need to take advantage of the features of CSPNs.

Coloured SPNs are high-level Petri nets, introduced by Kurt Jensen in [90]. CSPNs support the extensions with time, color and hierarchy. Particularly, each token has attached a color value, indicating the identity of the token. Moreover, each place and each transition has
attached a set of colors. All the rules, properties, and functions of original Petri Nets apply to CSPNs, while always taking into consideration the color feature. Specifically, a transition can fire with respect to each of its colors. By firing a transition, tokens are removed from the input places and added to the output places in the same way as original Petri Nets, except that a functional dependency is specified between the color of the transition firing and the colors of the involved tokens. Additionally, the color attached to a token may be changed by a transition firing [86], [91], [92].

In our case we use the different colors of tokens and places in order to differentiate between verb tenses (past, present, future). Specifically, we attached green, red, or blue to the appropriate tokens and places in order to fire (activate) transitions that represent verbs/actions of past, present, or future tense, respectively. We also use tokens and places of black color in order to fire any transition independent of the verb’s tense. For better understanding, more details and examples are given below.

The combination of SPNs are done as follows: while combining two or more nodes/places of the same agent we keep only one of these nodes, add an immediate transition, and connect that node with the immediate transition. Then, we add a neutral node for each transition (action/verb) that was connected with the agent’s nodes, connect the neutral node with the corresponding transition, and also connect all the neutral nodes with the immediate transition that we added earlier. After that, we assign black color to each one of the neutral nodes. Additionally, we also assign green, red or blue to each one of the neutral nodes according to the verb’s (transition) tense, which they are connected to. To represent the assigned colors we label the neutral node appropriately. “Gr”, “Rd” and “Bl” labels indicate
green, red and blue color, for past, present and future tense verbs respectively. Since all of neutral nodes have been assigned with black color, we consider it as a default value and we don’t use any specific label for this. Figure 13 shows the combination of agent’s nodes/places as it is described above. It also shows the uncombined and combined graphs of the simple text, “John likes music. He is a musician.”, as are produced by our program. Particularly, in Figure 13a, there are three different nodes/places, A, A’ and A”, of the same agent, while in Figure 13b all of these nodes/places have been combined in one node/place, A. The same applies for the different places of “John” in Figure 13c, which are combined in one place in Figure 13d. We have to point out here that the neutral nodes of Figure 13b would be assigned with different colors, depending on the tense of the verbs $V_1$, $V_2$ and $V_3$. In Figure 13d the neutral nodes are assigned with red color because both verbs, “like” and “is” are in present tense.
Figure 13: (a) Uncombined SPN graph: A, A’, A’’ are different places (nodes) of the same agent. (b) Combined coloured SPN graph: there is only one place, A, for the agent. (c) Uncombined SPN graph of the text: “John likes music. He is a musician.” (d) Combined coloured SPN graph of the same text.

For the combination of two or more nodes/places of the same patient we only keep one of these nodes. Then, the node that we kept becomes connected with all of the transitions that were connected to the removed nodes. Figure 14 shows the combination of patient nodes/places as it is described above. It also shows the uncombined and combined graphs of the simple text, “John has a guitar. Peter likes it.”, as are produced by our program. In Figure 14a there are three different nodes/places, P, P’ and P”, of the same agent, while in Figure 14b all of these nodes/places have been combined in one node/place, P. The same applies for the different places of “guitar” in Figure 14c, which are combined in one place in Figure 14d.
Figure 14: (a) Uncombined SPN graph: P, P’, P”’ are different places (nodes) of the same patient. (b) Combined coloured SPN graph: there is only one place, P, for the patient. (c) Uncombined SPN graph of the text: “John has a guitar. Peter likes it.” (d) Combined coloured SPN graph of the same text.

However, a different connection is used when we have to combine two (or more) graphs that are composed of different nodes/places of the same agent, while at the same time different nodes/places of the same patient. In this case we have to decide which action happened earlier in time. Then, the action that is preceded in time, after its completion,
activates the action that follows in time. If the different verbs (actions) are in different
tenses, then it is automatically known which one happened first. If not, a list of verbs with
assigned priorities is being used. Specifically, we have assigned to different verbs priorities
so that lower priority means the action must precede another with higher priority. In case
there is a conflict (two verbs with the same priority), or there is not a defined priority then
a message is shown to the user asking him/her to assign priority. When the sequence of
actions (priorities) has been decided, the combined graph is being produced so: again, we
keep only one of the agent nodes/places and only one of the patient nodes/places following
the same rules as above for the connection with the appropriate transitions. However,
because we want the earlier action to activate the latest one, we add a neutral node, then
connect the “first” transition (action) to the neutral node and finally connect the neutral
node to the “second” transition (action). This type of connection is shown in Figure 15a,
where A and A’ are different nodes/places of the same agent and P and P’ are different
nodes/places of the same patient. Similar to the previous examples, the neutral nodes of
Figure 15b, preceding each verb would be assigned with different colors, depending on the
tense of the verbs V₁, and V₂. Figure 15c shows the uncombined and combined graphs of
the simple text, “Mike bought a candy. He ate it.”, as are produced by our program after
using lower priority for the verb “bought” than the verb “ate”. All the appearances of the
agent node “Mike” in Figure 15c have been combined in one node in Figure 15d. The same
applies for all the appearances of the patient nodes of “candy”. Additionally, in Figure 15d
the neutral nodes are assigned with green because both verbs, “bought” and “ate” are in
past tense.
6.2.3. **Examples**

In this section we present further examples of combined and their corresponding uncombined graph. Figure 16a shows the SPN graph of the following text: “*Mary has a violin. She likes it. It is nice.*”, while Figure 16b shows the combined coloured SPN graph of the same text. We have to notice here that in Figure 16b the node, violin, is purple. The
word “violin” appears in the place of agent and also in the place of patient in the given text.
So, its color changes to purple only for visualization reasons.

![Diagram](image)

(a)

![Diagram](image)

(b)

Figure 16: (a) Uncombined SPN graph of text: “Mary has a violin. She likes it. It is nice.” (b) Combined coloured SPN graph of the same text.

Figure 17a shows the SPN graph of the following text: “The strongest rain ever recorded in India shut down the financial hub of Mumbai, snapped communication lines, closed airports and forced thousands of people to sleep in their offices or walk home during the
night, officials said today.”, while Figure 17b shows the combined coloured SPN graph of the same text.
6.2.4. Conclusions

In this section we illustrated the combination of SPN graphs that are refer to the same agent and/or patient node. Since our general goal is to propose a Natural Language Understanding methodology, the combination of SPN graphs is a small part of the methodology. In order to achieve effective SPN graph combinations we first applied Anaphora Resolution to the given input text and defined a set of rule for the extraction of kernels based on the parse trees of sentences. Then, we described our set of rules for
combining graphs. We also provided some illustrative examples of the combined and uncombined graphs of different sentences.

6.3. Examples

In this section we illustrate examples of output SPN graph and the corresponding combined coloured SPN graphs produced by our current program given as input NL text. These examples are representative of different cases of connections between agents, actions and patients. The purpose of these is to demonstrate the representation of NL text into uncombined and combined SPN graphs as it is proposed in previous chapters. Finally, in order to compare our work with [14], [15], [16], we give the output SPN graphs of our program for two example texts that are used in [14], [15], [16].

The first example shows the produced uncombined and combined SPN graph of a text that consists of sentences of both active and passive voice. Specifically, Figure 18a shows the output uncombined SPN graph of the text, “I knew John would finish the work by 5:00 PM. At 8:00 PM tonight, Sally is going to make a beautiful dinner. The pies would be made by my mother, as always. John is going to be washing the dishes after the dinner.”. Figure 18b shows the corresponding combined SPN graph of the same text.
Figure 18: (a) Output uncombined SPN graph of text: “I knew John would finish the work by 5:00 PM. At 8:00 PM tonight, Sally is going to make a beautiful dinner. The pies would be made by my mother, as always. John is going to be washing the dishes after the dinner.” produced by our program. (b) Output combined Coloured SPN graph of the same text.

In Figure 19a we can see the output uncombined SPN graph of the text, “Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don’t remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom’s mother was serving food and his father was playing guitar. They all had fun.”. Figure 19b shows the corresponding combined SPN graph of the same text.
Figure 19: (a) Output uncombined SPN graph of the text: “Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don’t remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom’s mother was serving food and his father was playing guitar. They all had fun.”. (b) Output combined Coloured SPN graph of the same text.

Figure 20 shows the output SPN graph of the following text, “The strongest rain ever recorded in India shut down the financial hub of Mumbai, snapped communication lines, closed airports and forced thousands of people to sleep in their offices or walk home during the night, officials said today.” that is included in [15]. Particularly, Figure 20a is the uncombined graph that has been produced automatically by our program. Figure 20b is the corresponding combined Coloured SPN graph of the same text, and Figure 20c is the graph
that Bourbakis and Mills have produced in [15] for the same text. As we can notice the extracted kernels are almost the same in both cases (ours and [15]). However, there are small differences. In our case we have combined together agent or patient words when one is used as an adjective, or a determiner, to the other. In addition, in our case we don’t consider as agents or patients, words that specify time or place (e.g. the word “time” if it follows a preposition).
Figure 20: (a) Output uncombined SPN graph of text: “The strongest rain ever recorded in India shut down the financial hub of Mumbai, snapped communication lines, closed airports and forced thousands of people to sleep in their offices or walk home during the night, officials said today.” produced by our program. (b) Combined Coloured SPN graph of the same text produce by our program. (c) SPN graph of the same text as shown in [15] by Bourbakis and Mills.

Figure 21 shows the output SPN graph of the following text, “I saw Tom with the telescope. The telescope sat on the table. The table and telescope was seen by all of us. We had a great time at the party. It was Tom's birthday. Tom received several presents from all of us. The telescope was a Tom's favorite present. The telescope appeared expensive. I looked through the telescope. A great time was had by all.”, that is included in [15]. Particularly, Figure 21a is the uncombined graph that has been produced automatically by our program. Figure 21b is the corresponding combined coloured SPN graph of the same text, and Figure 21c is the graph that Bourbakis and Mills have produced in [15], for the same text. As we can notice the extracted kernels are almost the same in both cases (ours and [15]). We have to notice that there are again the same differences as the ones that we mentioned in the previous example.
Figure 21: (a) Output uncombined SPN graph of text: “I saw Tom with the telescope. The telescope sat on the table. The table and telescope was seen by all of us. We had a great time at the party. It was Tom's birthday. Tom received several presents from all of us. The telescope was a Tom's favorite present. The telescope appeared expensive. I looked through the telescope. A great time was had by all.” produced by our program (b) Combined Coloured SPN graph of the same text produce by our program. (c) SPN graph of the same text as shown in [15] by Bourbakis and Mills.

6.4. Conclusions

In this chapter we presented the mapping of Glossa expressions (kernels of NL sentences) into state machines and more specifically into SPN graphs. Glossa language formally represents NL sentences and also supports the transformation of NL sentences into SPNs. So, we defined a mapping of Glossa language expression to SPN graphs. Through this mapping we are able to produce a visual representation of NL sentences using SPN graphs,
along with preserving the timing and flow of events/actions. We also illustrated some examples of conversion of *Glossa* expressions to SPN graphs and vice versa.

Then, we explained the use of Anaphora Resolution which is use for elimination of preposition and as a result, the limitation of ambiguity in the graph. Moreover, we provided a set of rules to combine SPN graphs with common agents or patients, which enriches our NLU methodology with the capability of event/action association.

Finally, we showed the output uncombined and combined SPN graphs of various NL texts that are produced automatically by our program following the given algorithm and mapping rules. Also, we have included some comparative examples of our work with the work of Bourbakis and Mills [15].
Chapter 7 : Modifications of technical documents

In the previous chapters we presented our NLU methodology for extracting events/actions associations of technical documents. Our methodology represents that extracted associations through SPN graphs. In this chapter, we describe one application that our proposed methodology can be used. Specifically, we use our methodology to produce modified versions of the initial NL document. Particularly, we will create a program that generates a new/modified version of a given NL document, based on the modifications made on its SPN graph.

At this stage the modifications on SPN graphs will be small, such as insertion or change of agents, or patients, and will be made manually. This type of application can be useful for quick and easy exchange of information between people regarding modifications made on the same document. For example, between teacher and students, or two people that work on the same scientific paper, etcetera.

This chapter is organized into three main sections. In Section 1 we describe the process of making modifications and producing a new version of the initial text. Section 2 illustrates examples of modified versions of technical documents produced using our methodology. Section 3 is the summary of the chapter.
7.1. Modifications

For this application, the user manually makes modifications on the produced SPN graph of the initial text (paragraph) of interest. Thus, we first use our NLU methodology to produce the SPN graph of the given text. The main steps for the production of the SPN graph are:

1. Produce the parse tree of each sentence, using the Stanford parser.
2. Extract the kernel(s) (Agent→Action→Patient) of each sentence, using our proposed algorithm presented in Chapter 3 [93].
3. Convert the kernels into SPN graphs, using our proposed methodology presented in Chapter 4 [81].
4. Finally, we synthesize the different SPN graphs into one, for event/action association purposes, as it is presented in Chapter 6 [93].

After the SPN graph have been produced, the user has the ability to make modifications on it using the given graphical environment. Particularly, the user can choose and change the name of an agent or patient node/vertex of the graph. The changes are made on the uncombined SPN graph, where each kernel is appeared separately. Every change is automatically appeared to the graph. When the user is done editing the graph, the modified version of the text and the combined SPN graph are automatically produced.

7.2. Examples

In this section we demonstrate examples of small modifications made on three different technical documents. For each document we choose some paragraphs that we process and produced their uncombined and combined SPN graphs. Then, we make small modifications on the uncombined SPN graph and observe their impact on the produced modified text and combined SPN graph.
We have to remind the reader, that our algorithm for the extraction of kernels (at each point) is created based on studies of different structures of parse trees. As a result, errors may occur when sentences with parse tree structures different than the already studied ones are being processed. However, we continually experiment on new structures of parse trees and we keep improving our algorithm. Additionally, errors that may occur during the creation of the parse tree (using Stanford parser), such as wrongly tagged words. For example words such as, ‘design’, could be misidentified as noun when it has the role of a verb and viva versa. Errors like these are inherited by our kernel extraction algorithm and lead to incorrectly extracted kernels.

For this first example we chose the paper [94] and specifically focused on section 2.1. The initial text and the corresponding uncombined and combined SPN graph are shown below.

Initial text: “The general structural design of the Anagnostis system is presented in Fig. 1. The system consists of a main processor (ME), two co-processors (PE1 and PE2) and ten units: a focusing and zooming unit, a segmentation and text binarization unit, a text sentences detection and paragraph synthesis unit, a raster scanner unit, a horizontal and vertical projection unit, a character pre-processing circuit, a chain code generation unit, a line generator/recognizer unit, a graph generator unit, and a matching processor. The main processor controls the entire recognition process by generating proper signals to the two co-processors (PE1, PE2). The PE1 controls the first six units and therefore is responsible in controlling the scanning-in process while PE2 controls the chain code generator, the line generator/recognizer, the graph generator and the matching processor. The PE2 is responsible for the entire recognition process. Since the scanner parts are
mechanical in nature, any delays caused in scanning the characters are negated by performing processes in parallel into the two co-processors, achieving some degree of parallelism.”
We now modify the initial text given above by replacing the phrase “chain code generation unit” with “character propagation unit”. The changes on the text are shown below in bold and underlined. The produced modified version of the text, the combined SPN graph, and the uncombined SPN graph are also shown below. The modified vertices of each graph are shown inside the red circle.

**Modified Text:** “The general structural design of the Anagnostis system is presented in Fig. 1. The system consists of a main processor (ME), two co-processors (PE1 and PE2) and ten units: a focusing and zooming unit, a segmentation and text binarization unit, a
text sentences detection and paragraph synthesis unit, a raster scanner unit, a horizontal and vertical projection unit, a character pre-processing circuit, a character propagation unit, a line generator/recognizer unit, a graph generator unit, and a matching processor.

The main processor controls the entire recognition process by generating proper signals to the two co-processors (PE1, PE2). The PE1 controls the first six units and therefore is responsible in controlling the scanning-in process while PE2 controls the chain code generator, the line generator/recognizer, the graph generator and the matching processor. The PE2 is responsible for the entire recognition process. Since the scanner parts are mechanical in nature, any delays caused in scanning the characters are negated by performing processes in parallel into the two co-processors, achieving some degree of parallelism.”
For the second example we chose the paper [95] and specifically focused on paragraphs 2, 3, and 4 of section VI.A. The initial text and the corresponding uncombined and combined SPN graph are shown below.

**Text:** “Every operation specifies its destination register at the time of initiation. A modifier ("dest-bank") specifies which register file contains the target register: the local general register bank, the general register bank in the paired F unit, the store file in the paired F unit, a general register bank in another Z unit, or a branch bank on an Z or F board. Branch banks are small 1-bit register files used to control branching; see Section VI-E-2.”
Substantial support was provided for injecting immediate constants into the computation. Each ALU can get a 6-bit, 17-bit, or 32-bit immediate provided on one operand leg, under the control of the instruction word. A 32-bit immediate field is flexibly shared between ALUO, ALUI, and a 32-bit PC adder which generates branch target addresses.

Included in the Z board instruction set are pipelined load and store instructions for referencing memory. Memory addresses are 32-bit byte pointers. The memory system hardware operates only on 32-bit or 64-bit quantities; access to fields of other sizes is provided via extract/merge/shift operations which are arranged to accept the same 32-bit pointer, using the low bits to specify the field position.”
Figure 24: (a) Uncombined SPN graph. (b) Combined Coloured SPN graph of the same text produce by our program.

We now modify the initial text given above by replacing the word “ALU0” with “Comparator0”. The changes on the text are shown below in bold and underlined. The produced modified version of the text, the combined SPN graph, and the uncombined SPN graph are also shown below. The modified vertices of each graph are shown inside the red circle.

Modified text: “Every operation specifies its destination register at the time of initiation. A modifier ("dest-bank") specifies which register file contains the target register: the local
general register bank, the general register bank in the paired F unit, the store file in the paired F unit, a general register bank in another Z unit, or a branch bank on an Z or F board. Branch banks are small 1-bit register files used to control branching; see Section VI-E-2.

Substantial support was provided for injecting immediate constants into the computation. Each ALU can get a 6-bit, 17-bit, or 32-bit immediate provided on one operand leg, under the control of the instruction word. A 32-bit immediate field is flexibly shared between Comparator0, ALUI, and a 32-bit PC adder which generates branch target addresses.

Included in the Z board instruction set are pipelined load and store instructions for referencing memory. Memory addresses are 32-bit byte pointers. The memory system hardware operates only on 32-bit or 64-bit quantities; access to fields of other sizes is provided via extract/merge/shift operations which are arranged to accept the same 32-bit pointer, using the low bits to specify the field position.”
For the third example we chose the paper [96] and specifically focused on the paragraphs of section VI. The initial text and the corresponding uncombined and combined SPN graph are shown below.

Text: “Lastly, in this section, we will discuss future work and more specifically the hardware implementation of EDIFT, which will greatly minimize the method's real-time performance overhead. As mentioned previously, our technique can accurately detect both Control-Flow Hijacking and Data-Only attacks; however, in our current, software-only implementation it introduces an average of 230% real-time slowdown in recompiled applications.”
In order to tackle this problem, we can incorporate our security operations within the architecture's instruction pipeline as part of a Secure Coprocessor, similar to (J. R. Crandall & F. T. Chong, 2004); (M. Dalton, H. Kannan, & C. Kozyrakis, 2007); (M. Dalton, H. Kannan, & C. Kozyrakis, 2008). Fig. 9 illustrates how EDIFT can be applied into a standard 5-stage architecture, where the upper part represents the system's CPU and the lower part represents the EDIFT parallel coprocessor. After an instruction has been fetched, signals go through both the CPU’s Decode stage and through EDIFT coprocessor, where the tags of the used registers are retrieved. Register File Tags1 refers to tags of the CPU registers (i.e. edx, ecx, eax in x86 Assembly) that follow the upward-growing stack, while Register File Tags2 follow the downwards-growing stack. The Stack1 & Stack2 registers component holds additional, EDIFT internal registers, which are used for the tag stacks' manipulation. Parallel to the CPU's execution stage is the EDIFT's tag Propagation Logic, which then stores the new tags in the Stack1 Tags and Stack2 Tags memory. These refer to the tags that track untrusted information inside the program's stack, for the upward-growing and downward-growing stacks respectively. Finally, the last stage, includes the Tags Writeback Logic for updating the tag information in the Register Files or the Check Logic, which checks for inconsistent tag information and sends the Detection Signal when an attack is being performed.

Hence, EDIFT could be completely masked within the CPU’s instruction pipeline and introduces no delay since the comparison and the execution of the security operations are done in parallel. Thus, the time overhead is 0% vs to RSE (10-15%). In addition, this approach does not require any modification in the existing CPU architecture and would be ideal for security- and performance-critical applications.”
We now modify the initial text given above by replacing the phrase “Stack1 & Stack2” with “Adder1 & Adder2”. The changes on the text are shown below in bold and underlined. The produced modified version of the text, the combined SPN graph, and the uncombined SPN graph are also shown below. The modified vertices of each graph are shown inside the red circle.

**Modified text:** “Lastly, in this section, we will discuss future work and more specifically the hardware implementation of EDIFT, which will greatly minimize the method’s real-time performance overhead. As mentioned previously, our technique can accurately detect both Control-Flow Hijacking and Data-Only attacks; however, in our current, software-
only implementation it introduces an average of 230% real-time slowdown in recompiled applications.

In order to tackle this problem, we can incorporate our security operations within the architecture’s instruction pipeline as part of a Secure Coprocessor, similar to (J. R. Crandall & F. T. Chong, 2004); (M. Dalton, H. Kannan, & C. Kozyrakis, 2007); (M. Dalton, H. Kannan, & C. Kozyrakis, 2008). Fig. 9 illustrates how EDIFT can be applied into a standard 5-stage architecture, where the upper part represents the system's CPU and the lower part represents the EDIFT parallel coprocessor. After an instruction has been fetched, signals go through both the CPU’s Decode stage and through EDIFT coprocessor, where the tags of the used registers are retrieved. Register File Tags1 refers to tags of the CPU registers (i.e. edx, ecx, eax in x86 Assembly) that follow the upward-growing stack, while Register File Tags2 follow the downwards-growing stack. The Adder1 & Adder2 registers component holds additional, EDIFT internal registers, which are used for the tag stacks' manipulation. Parallel to the CPU’s execution stage is the EDIFT’s tag Propagation Logic, which then stores the new tags in the Stack1 Tags and Stack2 Tags memory. These refer to the tags that track untrusted information inside the program's stack, for the upward-growing and downward-growing stacks respectively. Finally, the last stage, includes the Tags Writeback Logic for updating the tag information in the Register Files or the Check Logic, which checks for inconsistent tag information and sends the Detection Signal when an attack is being performed.

Hence, EDIFT could be completely masked within the CPU’s instruction pipeline and introduces no delay since the comparison and the execution of the security operations are
done in parallel. Thus, the time overhead is 0% vs to RSE (10-15%). In addition, this approach does not require any modification in the existing CPU architecture and would be ideal for security- and performance-critical applications.”
We have to point out here that changes were made directly to the text can be pasted through our methodology, where the corresponding combined and uncombined SPN graphs will be produced automatically. So, our methodology works both ways; for modifications directly on the text and for modifications on the graph. Thus, the user can decide the form of representation that prefers to work on.
7.3. Conclusions

In this section we used our proposed NLU methodology for generation of modified version of technical documents. This is one of the applications that our methodology can be used for. Our methodology, proposed in this chapter, for modifying documents could be really useful in cases where two or more people want to quickly modify and share a technical document. It gives the user the ability to find and change different elements easily and fast, through the SPN graph, without having to read the text line by line. Additionally, it provides a fast and easy way of tracking changes made by others in a document.
Chapter 8 : Summarization

In this chapter, we use our methodology for summarization purposes. Particularly, we will describe two different types of summarization; for the first one our methodology produces a summary of the text based on a given agent or patient, and for the second one the summary is produced based on weights (importance) of agents and patients in the text. We named these two types Driven Summarization and Free Summarization, respectively.

This chapter is organized into three main sections. In Section 1 and Section 2 we describe in details the Driven Summarization and Free Summarization, respectively. Each one of the section includes examples of summaries generated using our proposed techniques. Section 3 is the summary of the chapter.

8.1. Driven Summarization

For this first form of summarization, the generated summary is consist of sentences of the original text that contain a preselected agent or patient. In other words, the produced summary contains all the information about the selected agent or patient (all the actions/events that the target agent or patient did as well as all the actions/event that occurred on the target agent or patient).

Our Driven Summarization technique includes 5 main steps. First we use our NLU methodology in order to extract the kernels and associations in the initial text and produce the corresponding SPN graph. Then, the user has to choose one of the nodes/places of the
produced uncombined SPN graph, that he/she is interested in. When the selection is made, our techniques goes through all the extracted kernels and retrieve those that contain the target agent or patient. Next, it recovers all sentences of the initial text correspond to the previously retrieved kernels. We have to point out here that the whole sentence is recovered and not only the part of the sentence that contains a retrieved kernel. Finally, it produces the summary (text contains all recovered sentences) and the corresponding combined SPN graph.

For example, we are given the following very simple text, “Once a week, Tom cleans the house. Right now, Sarah is writing the letter. Sam repaired the car. The salesman was helping the customer when the thief came into the store. Many tourists have visited that castle. Recently, John has been doing the work. George had repaired many cars before he received his mechanic's license. Tom loved and hated Anna. Anna is a beautiful girl.”. The uncombined and combined SPN graphs of this text produced by our NLU methodology are shown in Figure 28.
We choose the word “Tom” from the sentence “Tom loved and hated Anna.” as our target for the Driven Summarization. The outputted summary is “Tom loved and hated Anna. Once a week, Tom cleans the house.” and the corresponding combined SPN graph produced by our methodology is given in Figure 29.
Figure 29: Combined SPN graph of driven summary produced for text “Once a week, Tom cleans the house. Right now, Sarah is writing the letter. Sam repaired the car. The salesman was helping the customer when the thief came into the store. Many tourists have visited that castle. Recently, John has been doing the work. George had repaired many cars before he received his mechanic's license. Tom loved and hated Anna. Anna is a beautiful girl.” with target word “Tom” from the sentence “Tom loved and hated Anna.”.

For the next example, we are given the following very simple text, “Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don't remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom's mother was serving food and his father was playing guitar. They all had fun.”. The uncombined and combined SPN graphs of this text produced by our NLU methodology are shown in Figure 32.
We choose the word "Tom" from the sentence "Tom had his birthday yesterday." as our target for the Driven Summarization. The outputted summary is "Tom had his birthday yesterday. Tom was teasing and talking to Kate and Sue." and the corresponding combined SPN graph produced by our methodology is given in Figure 33.
Figure 31: Combined SPN graph of driven summary produced for text “Tom had his birthday yesterday. All of his friends went to his house. Mike and Kate brought a cake. Sue or Penny brought juices. I don't remember exactly her name. John was eating candies or cake all the time. Tom was teasing and talking to Kate and Sue. Tom's mother was serving food and his father was playing guitar. They all had fun.” with target word “Tom” from the sentence “Tom had his birthday yesterday.”.

8.2. Free Summarization

The second type of summarization, called Free Summarization, the summary is generated based on weight of the different agents and patients in the given text. The weight of each agent or patient is calculated based on its frequency (number of hits in different kernels), and the number of its connections (ingoing and outgoing edges) in the combined SPN graph. The produced summary consists of sentences that include agents or patients with weight above a threshold. The threshold is defined by the user based on his/her preference of the amount of information is returned.
Our Free Summarization technique includes 8 main steps. First we use our NLU methodology in order to extract the kernels of the initial text and produce the corresponding SPN graph. Then, for each agent and patient we calculate how many times it appears in the extracted kernels (of the whole text). Additionally, we calculate the number of connections that the specific agent or patient place/node has in the produced combined SPN graph. The total weight of an agent or patient is \( \text{frequency} \times \text{number of connections} \). We decided to multiply the two values, since the number of connections of an agent or patient is the leading value due to its importance. Afterwards, we sort the agents and patients based on their total weight. We consider that agents and patients, with weight above the threshold (user’s preference), are the “main/important characters” of the given text. Thus, the produced summary will contain all sentences (actions, information) that contain at least one of the “important” agents or patients. To do this, our techniques goes through all the extracted kernels and retrieve those that contain at least one of the “important” agents or patients. Next, it recovers all sentences of the initial text correspond to the previously retrieved kernels. Similarly to the Driven Summarization that we described previously, the whole sentence is recovered and not only the part of the sentence that contains a retrieved kernel. Finally, our technique produces the summary (text contains all recovered sentences) and the corresponding combined SPN graph.

Below we provide the user with some illustrative examples. For this first example we choose the following sort and simple text, “Snakes are elongated, legless, carnivorous reptiles of the suborder Serpentes[2] that can be distinguished from legless lizards by their lack of eyelids and external ears. Like all squamates, snakes are ectothermic, amniote vertebrates covered in overlapping scales. Many species of snakes have skulls with several
more joints than their lizard ancestors, enabling them to swallow prey much larger than their heads with their highly mobile jaws. To accommodate their narrow bodies, snakes' paired organs (such as kidneys) appear one in front of the other instead of side by side, and most have only one functional lung. Some species retain a pelvic girdle with a pair of vestigial claws on either side of the cloaca.

Living snakes are found on every continent except Antarctica, and on most smaller land masses; exceptions include some large islands, such as Ireland, Iceland, Greenland and the islands of New Zealand, and many small islands of the Atlantic and central Pacific.[3] Additionally, sea snakes are widespread throughout the Indian and Pacific Oceans. More than 20 families are currently recognized, comprising about 500 genera and about 3,400 species.[4][5] They range in size from the tiny, 10.4 cm-long thread snake[6] to the reticulated python of up to 6.95 meters (22.8 ft) in length.[7] The fossil species Titanoboa cerrejonensis was 12.8 meters (42 ft) long.[8] Snakes are thought to have evolved from either burrowing or aquatic lizards, perhaps during the Jurassic period, with the earliest known fossils dating to between 143 and 167 Ma ago.[9] The diversity of modern snakes appeared during the Paleocene period (c 66 to 56 Ma ago). The oldest preserved descriptions of snakes can be found in the Brooklyn Papyrus.

Most species are nonvenomous and those that have venom use it primarily to kill and subdue prey rather than for self-defense. Some possess venom potent enough to cause painful injury or death to humans. Nonvenomous snakes either swallow prey alive or kill by constriction.” found in Wikipedia. The uncombined and combined SPN graphs of this text produced by our NLU methodology are shown in Figure 34.
Using our proposed Free Summarization technique with threshold equal to 5% (percentage of “main characters” that our summary will include), the produced summary is, “Like all squamates, snakes are ectothermic, amniote vertebrates covered in overlapping scales. Living snakes are found on every continent except Antarctica, and on most smaller land masses; exceptions include some large islands, such as Ireland, Iceland, Greenland and the islands of New Zealand, and many small islands of the Atlantic and central Pacific. Additionally, sea snakes are widespread throughout the Indian and Pacific Oceans. Snakes are elongated, legless, carnivorous reptiles of the suborder Serpentes [2] that can
be distinguished from legless lizards by their lack of eyelids and external ears. Snakes are thought to have evolved from either burrowing or aquatic lizards, perhaps during the Jurassic period, with the earliest known fossils dating to between 143 and 167 Ma ago.”

The corresponding combined SPN graph is given in Figure 33.

Figure 33: Combined SPN graph of the summary produced by our Free Summarization technique with threshold value equal to 5%.

For the next example we process the section 1.2 of [97]. The text of this section is, “Even today, besides the numerous methodologies proposed and the collaborative effort of the researchers to produce robust and accurate systems and frameworks for activity recognition, several challenges still exist that have not been yet fully addressed. These challenges can hinder the performance of the methodologies significantly and render their applicability impossible to real-world scenarios. Even from the sensor level, camera effects
and distortions can introduce significant amounts of noise. Even in high quality video streams, quick motions can cause motion blurring and camera motion can alter the perception of the localized motions. Moreover, dynamic illumination and shadows, especially in appearance-based methods, are still a major challenge. It is commonly assumed that illumination conditions are controlled, however this is not true in real-world scenarios. Another problem that has to be dealt with are occlusions, either by the environment or moving objects, of self-occlusions by body parts, which limit the observations of potentially informative cues. Also, camera’s position plays a significant role in recognizing activities, because activities look different from different views ad the feature that are to be extracted have to be view-invariant so as to dissolve the imposed ambiguities. Finally, activity itself is inherently ambiguous to define. Different people perform the same action differently and each activity is so complex that can be decomposed to many primitive parts. As a result, the domain of application has to be limited, since human experts cannot possibly conceive and design every possible combination and machine-based methods are not expressive or powerful enough to capture this spectrum either, even with a huge amount of training data. Although many sophisticated methodologies have been developed and significant technological advancements have been made towards the direction of the general problem of surveillance and activity recognition, truly successful results cannot be claimed in real world conditions.”. The uncombined and combined SPN graphs of this text produced by our NLU methodology are shown in Figure 34.
Figure 34: (a) Uncombined SPN graph produced by our program (b) Combined Coloured SPN graph of the same text produced by our program.

We use our Free Summarization technique with threshold equal to 10% (percentage of “main characters” that our summary will include). The produced summary is, “Even today, besides the numerous methodologies proposed and the collaborative effort of the researchers to produce robust and accurate systems and frameworks for activity recognition, several challenges still exist that have not been yet fully addressed. Different people perform the same action differently and each activity is so complex that can be
decomposed to many primitive parts. Although many sophisticated methodologies have been developed and significant technological advancements have been made towards the direction of the general problem of surveillance and activity recognition, truly successful results can not be claimed in real world conditions. As a result, the domain of application has to be limited, since human experts can not possibly conceive and design every possible combination and machine-based methods are not expressive or powerful enough to capture this spectrum either, even with a huge amount of training data. Moreover, dynamic illumination and shadows, especially in appearance-based methods, are still a major challenge. Even from the sensor level, camera effects and distortions can introduce significant amounts of noise. Even in high quality video streams, quick motions can cause motion blurring and camera motion can alter the perception of the localized motions. Another problem that has to be dealt with are occlusions, either by the environment or moving objects, of self-occlusions by body parts, which limit the observations of potentially informative cues. ". The corresponding combined SPN graph is given in Figure 35.
Figure 35: Combined SPN graph of the summary produced by our Free Summarization technique with threshold value equal to 10%.

If we use a threshold of 20% for the same initial text, the produced summary is, “Even today, besides the numerous methodologies proposed and the collaborative effort of the researchers to produce robust and accurate systems and frameworks for activity recognition, several challenges still exist that have not been yet fully addressed. Different people perform the same action differently and each activity is so complex that can be decomposed to many primitive parts. Although many sophisticated methodologies have been developed and significant technological advancements have been made towards the direction of the general problem of surveillance and activity recognition, truly successful
results can not be claimed in real world conditions. As a result, the domain of application has to be limited, since human experts can not possibly conceive and design every possible combination and machine-based methods are not expressive or powerful enough to capture this spectrum either, even with a huge amount of training data. Moreover, dynamic illumination and shadows, especially in appearance-based methods, are still a major challenge. Even from the sensor level, camera effects and distortions can introduce significant amounts of noise. Even in high quality video streams, quick motions can cause motion blurring and camera motion can alter the perception of the localized motions. Another problem that has to be dealt with are occlusions, either by the environment or moving objects, of self-occlusions by body parts, which limit the observations of potentially informative cues. These challenges can hinder the performance of the methodologies significantly and render their applicability impossible to real-world scenarios. Also, camera’s position plays a significant role in recognizing activities, because activities look different from different views ad the feature that are to be extracted have to be view-invariant so as to dissolve the imposed ambiguities.”. The corresponding combined SPN graph is given in Figure 36.
Figure 36: Combined SPN graph of the summary produced by our Free Summarization technique with threshold value equal to 20%.

From the previous two example we can see the difference on the amount of information returned using different thresholds. Thus, higher threshold value (given by the use) produces more extended summaries, which contain more information. The threshold is chosen by the user according to the application and his/her preferences.
8.3. Conclusions

In this chapter we presented two different summarization techniques as an application of our proposed NLU methodology. Specifically, both of these techniques use, as a first step, our proposed NLU methodology to extracted the kernels of all sentences in the given text. Then, they use this information in order to produce the summary.

The first summarization technique, named Driven Summarization, produces the summary based on a target agent or patient, picked by the user. The result summary contains information about the target value. For the second summarization technique, named Free Summarization, the user chooses the preferred amount of information returned. The weight (given by the frequency and the number of connections in the graph) of all agents and patients nodes is calculated. The result summary contains information about the “main characters” of the given text (agents and patients with high weight value). The percentage of “main characters” that will appear on the summary is defined by the user’s input.

An extended version of the Free Summarization presented here, is given in Appendix B. This version looks for “main verbs” and their connection to the “main characters” of the text. As a result it is expected to have better results.
Chapter 9: Conclusions

Natural Language Understanding is an old field with several achievements throughout the years. Even though it is also an established field, it could afford improvement. This is because of the great variety and complexity of natural language. Thus, we created and proposed a novel NLU methodology for event/action association. Specifically, our methodology takes as input, English technical documents then extracts its kernels and represents them into SPN graphs.

In this work we presented a first level document classifier, which is used to filter technical documents. Documents that pass through the classifier are then processed using our proposed NLU methodology. Then, we describe in detail, each of the steps of our NLU methodology, which are: extraction of kernels, formal representation of kernels through Glossa language, conversion of kernels to SPN graphs for association representation, and synthesis of SPN graphs.

Particularly, we presented the algorithm for the extraction of kernels from NL sentences, based on their parse tree. Then, we defined a formal language, named Glossa, for the formal representation of different structures of kernels. Consequently, we used Glossa expressions of kernels in order to map them to SPN graphs. Finally, we defined a set of rules for the synthesis of SPN graphs. Through, combined SPN graphs we associate events/actions of different sentences while preserving their timing and flow.
We also presented two different applications of our NLU methodology. The first is a document modification methodology, where changes are made by the user on the produced SPN graph of the initial text. This could be useful in case of shared documents, since it provides the group of users the ability to create updated versions of the initial document quick and easy. For the second application we presented two document summarization types. The first one, named Driven Summarization, produces the summary, which is centered on a specific agent or patient chosen by the user. For the second summarization type all the agents and patients in the initial text are assigned with a weight, according to their frequency in the text and their degree in the graph. The produce summary is centered on a percentage (chosen by the user) of those agents and patients with higher weight.

Concluding, we would like to point out here the novel parts of our proposed NLU methodology:

- **We developed a new algorithm for kernel extraction, where the connections between agents, actions and patients and the tense of the different verbs were retained. Additionally, our algorithm associates adjectives with their corresponding nouns and “explanatory” kernels with their “main” verbs.**

- **We proposed a novel formal language, named Glossa, which helps mapping kernels to SPN graphs, while preserving the timing and the flow of events/actions.**

- **We defined a set of rules for the synthesis of different SPN graphs for event/action association. Timing and flow of events/actions is also preserved.**

For future work, we believe that our proposed representation of NL sentences to SPN graphs and the synthesis of SPN graphs can be useful for completion of incomplete kernel. We consider incomplete kernels where the agent and/or patent is either missing or implied.
Cases of incomplete kernels are really common in free writing, such as social networks, blogs, simply notes, etc.

Moreover, further extensions of our kernel extraction algorithm can be made, where more and more complicated structures of NL sentences will be included. For example, sentences that have a whole other sentence as agent or patient, conditionals and uncommon or extreme forms of kernels can be studied and incorporated into the initial algorithm. Finally, addition applications of our NLU methodology can be explored.
REFERENCES


133


Appendix A

As we already mentioned, our proposed algorithm for the extraction of kernels is incremental. This means that we built it in different stages by experimenting in more and more complex structures of parse trees. Thus, in this following appendix we evaluate the different stages of our algorithm, by presenting their success rate.

We processed different paragraphs of different technical documents, then tested different versions of our kernel extraction algorithm against each paragraph. For each experiment we count the number of correctly extracted kernels.

For the first cycle of experiments we process section 2.1 of [94]. The text of this paragraph is, “The general structural design of the Anagnostis system is presented in Fig. 1. The system consists of a main processor (ME), two co-processors (PE1 and PE2) and ten units: a focusing and zooming unit, a segmentation and text binarization unit, a text sentences detection and paragraph synthesis unit, a raster scanner unit, a horizontal and vertical projection unit, a character pre-processing circuit, a chain code generation unit, a line generator/recognizer unit, a graph generator unit, and a matching processor. The main processor controls the entire recognition process by generating proper signals to the two co-processors (PE1, PE2). The PE1 controls the first six units and therefore is responsible in controlling the scanning-in process while PE2 controls the chain code generator, the line generator/recognizer, the graph generator and the matching processor. The PE2 is responsible for the entire recognition process. Since the scanner parts are mechanical in
nature, any delays caused in scanning the characters are negated by performing processes in parallel into the two co-processors, achieving some degree of parallelism.”

For the second cycle of experiments we process paragraphs 5, 6, and 7 of section VI of [95]. The text of this paragraph is, “We partitioned the core processor into an integer and a floating unit (the "Z" and "F" boards), and provided separate physical register files for the floating point functional units and the integer ALU's. This makes intuitive sense, since there is little need for performing integer operations on floating point operands (and vice versa), while it is often the case that a chain of floating point operations can proceed while the integer units are performing the address computations in parallel. Fig. 2 shows the block diagrams of the "I" and "F" boards.

The unit of processor expansion is this integer-floating board pair. One, two, or four Z-F pairs can be configured, corresponding to a 256-bit, 512-bit, or 1024-bit instruction word.

Two 32-bit buses carry data traffic between the boards of a pair (through a dedicated front-edge path, rather than the backplane). Each board carries its own register file/crossbar, touching 12 32-bit data paths, handling four writes, four reads, and four bus-to-bus forwards in each minor cycle, plus bypassing from every write port to every read port. Sixty-four 32-bit registers are provided. This register file/crossbar is implemented in nine gate arrays; each is a 4-bit slice (byte parity is carried throughout the machine).”

For the third cycle of experiments we process section VI of [96]. The text of this paragraph is, “Lastly, in this section, we will discuss future work and more specifically the hardware implementation of EDIFT, which will greatly minimize the method's real-time performance
overhead. As mentioned previously, our technique can accurately detect both Control-Flow Hijacking and Data-Only attacks; however, in our current, software-only implementation it introduces an average of 230% real-time slowdown in recompiled applications.

In order to tackle this problem, we can incorporate our security operations within the architecture's instruction pipeline as part of a Secure Coprocessor, similar to (J. R. Crandall & F. T. Chong, 2004); (M. Dalton, H. Kannan, & C. Kozyrakis, 2007); (M. Dalton, H. Kannan, & C. Kozyrakis, 2008). Fig. 9 illustrates how EDIFT can be applied into a standard 5-stage architecture, where the upper part represents the system's CPU and the lower part represents the EDIFT parallel coprocessor. After an instruction has been fetched, signals go through both the CPU’s Decode stage and through EDIFT coprocessor, where the tags of the used registers are retrieved. Register File Tags1 refers to tags of the CPU registers (i.e. edx, ecx, eax in x86 Assembly) that follow the upward-growing stack, while Register File Tags2 follow the downwards-growing stack. The Stack1 & Stack2 registers component holds additional, EDIFT internal registers, which are used for the tag stacks’ manipulation. Parallel to the CPU’s execution stage is the EDIFT’s tag Propagation Logic, which then stores the new tags in the Stack1 Tags and Stack2 Tags memory. These refer to the tags that track untrusted information inside the program's stack, for the upward-growing and downward-growing stacks respectively. Finally, the last stage, includes the Tags Writeback Logic for updating the tag information in the Register Files or the Check Logic, which checks for inconsistent tag information and sends the Detection Signal when an attack is being performed.
Hence, EDIFT could be completely masked within the CPU’s instruction pipeline and introduces no delay since the comparison and the execution of the security operations are done in parallel. Thus, the time overhead is 0% vs to RSE (10-15%). In addition, this approach does not require any modification in the existing CPU architecture and would be ideal for security- and performance-critical applications.”

The percentage of correctly extracted kernels found by different version of our algorithm for each technical document is shown in Table 11.

Table 11: Evaluation of different versions of our kernel extraction algorithm

<table>
<thead>
<tr>
<th>Versions of our kernel extraction algorithm</th>
<th>Technical documents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[94]</td>
</tr>
<tr>
<td>Version 1 (24)</td>
<td>3 out of 12 = 25%</td>
</tr>
<tr>
<td>Version 2 (29)</td>
<td>6 out of 12 = 50%</td>
</tr>
<tr>
<td>Version 3 (35)</td>
<td>11 out of 12 = 91.66%</td>
</tr>
</tbody>
</table>

We have to mention here that version 1 of our kernel extraction algorithm of Table 11 is older than version 2, which is older than version 3.

The reader has to keep in mind that incorrectly extracted kernels could be a result of wrongly tagged words by the Stanford parser. For example, words could be tagged as verbs when they have the role of a noun in the sentence. Cases like affect the output of our algorithm.
Appendix B

In this appendix we propose an extension of the Free Summarization technique of Section 8.2. This extended version is an improvement of the initial summarization technique. We want to stress here that only the general idea of this extension is given here. This is because our goal is not to develop a summarization methodology, but to show our proposed NLU methodology can be useful for different applications.

We name this extended version, Level 2 Summarization. Level 2 Summarization includes the same main steps as the Free Summarization that we already described plus more. The extra steps are as follows:

1. Find all nouns in each subtitle of the given technical document.
2. Associate each one of the nouns, of step 1, with other agents or patients extracted from the corresponding subsection of the given technical documents.
3. Find all the verbs associated with the extracted agents or patients, of step 2.
4. Group together verbs, of step 3, that are synonyms or the same. For each group of verbs we also combine their associated agents and patients.
5. Define that “main verbs” based on the number of different agents and patients in the group that belong.
6. Define the “main characters” based on their total weight (frequency and number of connection found in previous steps) and the connection with the “main verbs”.
7. Retrieve all sentences of the subsection that include at least one of the “main characters”.
The logic behind this summarization technique is that the very important (top level) elements (methodologies, techniques, or products) of the presented work in each technical document should appear in its title and subtitles. Thus, extracting these nouns and their associated agents and patients in the specific subsection, we will be able to define the different components of the proposed system. Finally, based on the association, weights and connections we will be able to identify the “main characters” and compose the summary based on their actions.

We believe that Level 2 Summarization, as is described in this section, will produce a better summary than the Free Summarization. This is because this technique evaluates each one of the agents and patients based on their importance in the given document.