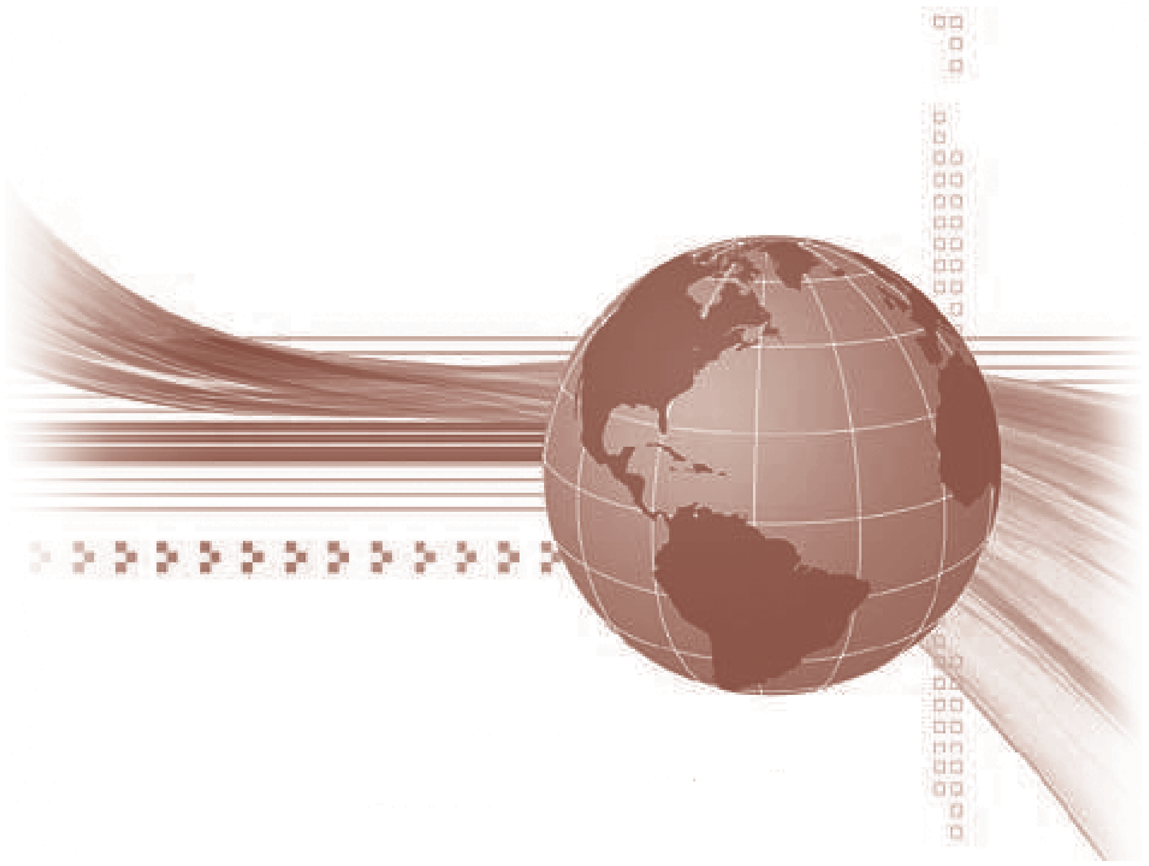
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

2/2017

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 2/2017

July - December

EDITORIAL BOARD

YEAR Volume 62 (LXII) 2017
MONTH DECEMBER
ISSUE 2

S T U D I A
UNIVERSITATIS BABEȘ-BOLYAI
INFORMATICA

2

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.01

A HYBRID APPROACH FOR SCHOLARLY INFORMATION EXTRACTION

ZALÁN BODÓ AND LEHEL CSATÓ

Abstract. Metadata extraction from documents forms an essential part of web or desktop search systems. Similarly, digital libraries that index scholarly literature require to find and extract the title, the list of authors and other publication-related information from an article. We present a hybrid approach for metadata extraction, combining classification and clustering to extract the desired information without the need of a conventional labeled dataset for training. An important asset of the proposed method is that the resulting clustering parameters can be used in other problems, e.g. document layout analysis.

1. Introduction

Since its inception in the 1970s, information retrieval is heavily used in several domains of science, not to mention its indispensable everyday use as back-ends for search engines. Search engines collect the available documents and process the various formats to (a) extract information like title, abstract, and authors of a publication and (b) to rank these documents according to the query posed by the user. In this article we focus on document processing and authorship, respectively title extraction; we call this information metadata. The metadata for a scientific publication is of great importance for digital libraries, the ranking algorithms perform best when complete and unambiguous information is provided.

We present a generic hybrid method that considers the specifics of the metadata to be extracted, and optimizes a clustering procedure that concatenates text chunks from within a document. A beneficial by-product of this method is that the resulting clustering is amenable for other domains, such as document layout analysis [3]. We also mention that our system does not need labeled

Received by the editors: June 19, 2017.

2010 *Mathematics Subject Classification.* 62H30, 68P20.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE – *Learning*; I.5.3 [**Computing Methodologies**]: PATTERN RECOGNITION – *Clustering*.

Key words and phrases. information extraction, metadata, machine learning.

data as input, but only a set of PDF documents along with a metadata database, the labeling procedure is embedded into the training phase.

The paper is structured as follows: Section 2 introduces the problem of metadata extraction and enumerates popular PDF extraction tools, and Section 3 describes the proposed algorithm for metadata extraction. In Section 4 the features used for representing the text segments are described, and Section 5 concludes the paper by presenting the experiments and discussing the results.

2. Metadata extraction from scholarly articles

Metadata extraction is generally viewed as a classification problem, where specific text segments are needed to be identified and extracted from a document. That is, if we consider the text chunks extracted from a PDF document, these have to be labeled as being part of the title, of the author list, of the *other* category, etc. Hence, the problem can be viewed as a supervised learning task: given a set of labeled examples, learn the mapping from the data to labels. This can be done either by a classifier (e.g. support vector machine [7]) or by a structured learning method (e.g. conditional random field [14]). In either case labeled examples or sequences are needed in order to train the learning model chosen. However, the problem can also be approached by unsupervised learning: without using any labeled data, find the cohesive text segments of the page, then label these using for example some kind of rule set [8]. The works [6, 10] give an excellent overview of the available methods and tools.

In this paper we consider the problem as a supervised learning task, but we also make use of clustering methods to find the cohesive text segments, that will be used for learning.

Although scholarly articles can be found in a wide variety of file formats on the Internet, the *Portable Document Format* is without doubt the most popular one. Therefore, it is sufficient to consider this format for metadata extraction.

The most popular tools used for obtaining the text fragments from PDF documents—for Java, C# and Python—are PDFBox[1], iText[2] and PDFMiner[3] [1, 4, 16]. All of these libraries can extract text along with font and positional information from a document, however, the implementations and functionalities—obviously—differ. Thus, some of them can recognize and separate ligatures, others cannot, they return different reading orders for the text

[1]`http://pdfbox.apache.org/`

[2]`http://itextpdf.com/`

[3]`http://www.unixuser.org/~euske/python/pdfminer/`

chunks, etc.[4] Figure 1(a) shows the text chunks returned by PDFMiner for a test document.

3. A hybrid approach for metadata extraction

In order to use the described method, a large training set of scholarly articles is required with metadata information attached to them, e.g the title of the paper, author names, journal or conference proceedings the article appeared in, etc., depending on what kind of metadata is going to be extracted, in textual format. This can be obtained by using a specialized crawler, which seeks for scientific articles on specific websites, and finds the associated metadata. This is by itself a complex task to perform, therefore the setup of such a system is not detailed here. Another possibility is to use a digital library, from where the documents and metadata can be obtained in a more straightforward manner. One such digital library is CiteSeerX [5, 17], which offers an OAI collection for metadata harvesting.[5] Other approaches may include the utilization of large research article databases such as the ACL Anthology[6], PubMed Central Open Access Subset[7] or the arXiv e-Print Archive[8]. Online and open bibliography websites such as DBLP[9] or the Collection of Computer Science Bibliographies[10] also offer a huge amount of bibliographic data. Combined with a web search engine with high coverage, one can obtain a large collection of articles and associated metadata from various journals and conference proceedings. In order to generalize well, it is of central importance to train the metadata extraction system using a large variety of article formats.

PDF extraction tools return the extracted text as separate text chunks or segments along with positional and font information. We consider the problem of information extraction as a two-step procedure: (a) cluster the segments to find the cohesive parts, e.g. the title of the article, (b) use the output of the clustering as input for a supervised learning method. We do not require

[4]Using 100 random articles as a test set, it resulted that the fastest is iText, PDFBox is about twice as slow, while PDFMiner is the slowest of the three libraries, slower than iText by a factor of 7. The tested versions were 7.0.1, 2.0.3 and 20140328, respectively.

[5]CiteSeerX also employs a metadata extraction system to extract these information from the crawled files, hence it might seem odd to use the output of a metadata extraction system as input for training another metadata extraction system. However, a similar wrapper method, where the output is fed back as input, called *self-training*, is a common approach in semi-supervised learning to strengthen the confidence of the underlying classifier [18].

[6]`http://acl-arc.comp.nus.edu.sg/`

[7]`ftp://ftp.ncbi.nlm.nih.gov/pub/pmc/`

[8]`https://arxiv.org/`

[9]`http://dblp.uni-trier.de/`

[10]`http://liinwww.ira.uka.de/bibliography/`

Algorithm 1 Finding the clustering parameters

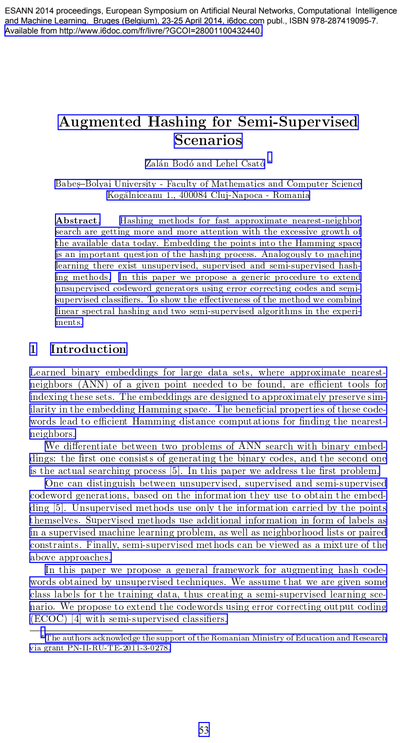
```
1: Choose similarity threshold $t$
2: **while** best clustering parameters $P$ are found **do**
3:     $P \leftarrow$ next parameter set
4:     Perform clustering using $P$
5:     Count matches using threshold $t$
6:     Evaluate $P$: #matches/#all cases
7: **end while**
```

a conventional labeled dataset for training, but only a set of PDF documents with a metadata database.

Our method searches for the best clustering of the extracted text chunks, such that to maximize the number of matches between the metadata and the obtained text segments. A match is found if the similarity between the metadata and the text segment does exceed a given threshold, and this happens exactly once. No match or multiple matches are equivalently considered as no matches. The proposed method has a number of parameters to set, including the clustering algorithm and therefore its parameters, the similarity measure and threshold. However, since we are working with textual data, we recommend using the bag-of-n-grams representation with raw frequency weighting scheme, considering the relative shortness of the text segments, and the cosine similarity measure [15, 13]. Algorithm 1 summarizes the described process. Searching for the optimal clustering parameters can be done using either randomized or grid search.

As possible benefits of the proposed procedure we enumerate the following:

- Constructing a labeled dataset for metadata extraction is a costly process. The approach presented in this paper does not need a conventional labeled dataset for training, but a set of PDF files along with a metadata database containing the metadata to be found/extracted, which can be much more less expensive to produce.
- The clustering procedure (i.e. the clustering parameters) can be used in another system, that requires to determine cohesive text segments in a document, for example in document layout analysis [3].
- The output text segments can be used in either a classification or a sequence tagging algorithm, thus, the learning method can use font and position-related features. Fewer text segments can increase the performance of the learning method.
- Using a measure of central tendency, the font name, font size, font style are determined for the entire segment, thus reducing the probability of misguiding the classifier—when different font styles or sizes



ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7. Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Augmented Hashing for Semi-Supervised Scenarios

Zalán Bodó and Lehel Csató

Babeş–Bolyai University - Faculty of Mathematics and Computer Science
Kogălniceanu 1., 400084 Cluj-Napoca - Romania

Abstract. Hashing methods for fast approximate nearest-neighbor search are getting more and more attention with the excessive growth of the available data today. Embedding the points into the Hamming space is an important question of the hashing process. Analogously to machine learning there exist unsupervised, supervised and semi-supervised hashing methods. In this paper we propose a generic procedure to extend unsupervised codeword generators using error correcting codes and semi-supervised classifiers. To show the effectiveness of the method we combine linear spectral hashing and two semi-supervised algorithms in the experiments.

1 Introduction

Learned binary embeddings for large data sets, where approximate nearest-neighbors (ANN) of a given point needed to be found, are efficient tools for indexing these sets. The embeddings are designed to approximately preserve similarity in the embedding Hamming space. The beneficial properties of these codewords lead to efficient Hamming distance computations for finding the nearest-neighbors.

We differentiate between two problems of ANN search with binary embeddings: the first one consists of generating the binary codes, and the second one is the actual searching process [5]. In this paper we address the first problem.

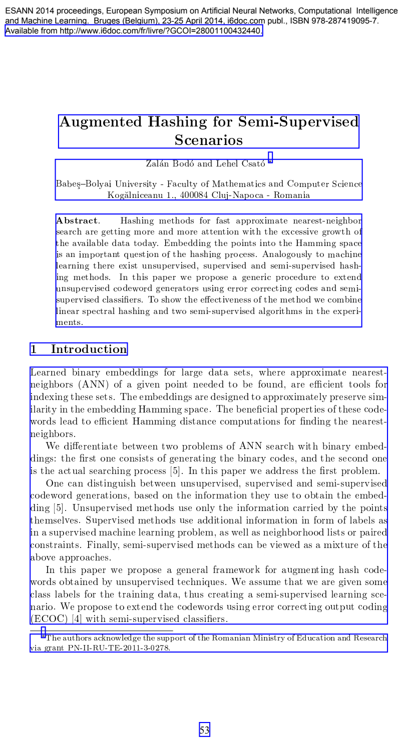
One can distinguish between unsupervised, supervised and semi-supervised codeword generations, based on the information they use to obtain the embedding [5]. Unsupervised methods use only the information carried by the points themselves. Supervised methods use additional information in form of labels as in a supervised machine learning problem, as well as neighborhood lists or paired constraints. Finally, semi-supervised methods can be viewed as a mixture of the above approaches.

In this paper we propose a general framework for augmenting hash codewords obtained by unsupervised techniques. We assume that we are given some class labels for the training data, thus creating a semi-supervised learning scenario. We propose to extend the codewords using error correcting output coding (ECOC) [4] with semi-supervised classifiers.

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

53

(a)



ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7. Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Augmented Hashing for Semi-Supervised Scenarios

Zalán Bodó and Lehel Csató

Babeş–Bolyai University - Faculty of Mathematics and Computer Science
Kogălniceanu 1., 400084 Cluj-Napoca - Romania

Abstract. Hashing methods for fast approximate nearest-neighbor search are getting more and more attention with the excessive growth of the available data today. Embedding the points into the Hamming space is an important question of the hashing process. Analogously to machine learning there exist unsupervised, supervised and semi-supervised hashing methods. In this paper we propose a generic procedure to extend unsupervised codeword generators using error correcting codes and semi-supervised classifiers. To show the effectiveness of the method we combine linear spectral hashing and two semi-supervised algorithms in the experiments.

1 Introduction

Learned binary embeddings for large data sets, where approximate nearest-neighbors (ANN) of a given point needed to be found, are efficient tools for indexing these sets. The embeddings are designed to approximately preserve similarity in the embedding Hamming space. The beneficial properties of these codewords lead to efficient Hamming distance computations for finding the nearest-neighbors.

We differentiate between two problems of ANN search with binary embeddings: the first one consists of generating the binary codes, and the second one is the actual searching process [5]. In this paper we address the first problem.

One can distinguish between unsupervised, supervised and semi-supervised codeword generations, based on the information they use to obtain the embedding [5]. Unsupervised methods use only the information carried by the points themselves. Supervised methods use additional information in form of labels as in a supervised machine learning problem, as well as neighborhood lists or paired constraints. Finally, semi-supervised methods can be viewed as a mixture of the above approaches.

In this paper we propose a general framework for augmenting hash codewords obtained by unsupervised techniques. We assume that we are given some class labels for the training data, thus creating a semi-supervised learning scenario. We propose to extend the codewords using error correcting output coding (ECOC) [4] with semi-supervised classifiers.

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

53

(b)

FIGURE 1. Results of text chunk extraction using PDFMiner (a) before and (b) after clustering.

are assigned for consecutive text chunks actually belonging to the same segment.[11]

Figure 1(b) shows the result obtained by clustering the text chunks extracted from a PDF document using PDFMiner. The parameters of the clustering method were determined using the proposed algorithm.

4. Clustering and building the feature vectors

Since PDFMiner returns concatenated text chunks from the same line (see Figure 1), the clustering is performed in vertical direction only. For clustering the text chunks/segments, a distance measure between the objects is needed,

[11]However, we mention that the clustering method is faced with the same challenge during the clustering process.

for which the following metric was used:

$$d(x,z) = \min(\min(d_1(x,z), d_2(x,z)), d_3(x,z) + d_4(x,z)) \\ /(lineStretch \cdot minSize) + sizeFactor \cdot d_5(x,z) \tag{1}$$

where d_1, d_2, d_3 and d_4 return the distances between the bottom left and top right, top right and bottom left, bottom left and bottom left, and top right and top right y-coordinates of the bounding boxes, respectively. The parameter $minSize$ gives the minimum of the most frequent font sizes of the two text segments, while d_5 is the absolute value of the font size difference. In case the majority font styles and font names differ between the two segments, the distance is multiplied by a $styleFactor$ and $fontFactor$ parameter, respectively. These parameters of the distance function, along with $lineStretch$ and $sizeFactor$, were determined using cross-validation.

For representing text chunks for the supervised learning phase we use the following 3 feature categories:

(a) distances, sizes;
(b) regular expression-based features;
(c) dictionary-based features.

Distance and size features include the distances between the previous and trailing text chunks (i.e. their bounding boxes), vertical positions of the bounding box, index of the text chunk, text length, number of words, font sizes (current, before and after, most frequent in the segment), font style (regular, bold, italic). Regular expressions-based features include the ratios of uppercase letters and terms (current, before and after), presence of email and URL addresses, ratio of numbers and special characters (non-word characters). Additionally, a database of first and last names[12] was used for calculating the ratio of names found in a text chunk.

5. Experiments and discussion

5.1. **The dataset.** The training and test data were obtained from CiteSeerX by retrieving 5000 scholarly articles from it, using the CiteSeerX OAI collection via the harvest URL.[13]. The dataset was compiled between September 5 and 7, 2016, using selective harvesting with datestamps of 01.01.2005 and 01.01.2016. We stored the metadata of an article only if the *source* field was valid, and stopped when the number of 5000 articles was reached. The downloaded files were processed using PDFMiner omitting the erroneous (e.g. non-PDF) documents, resulting in a total of 4217 articles. This collection was split randomly into 3 sets: C_1 with 1000 documents, C_2 also with 1000 documents,

[12]`https://github.com/SeerLabs/CiteSeerX`

[13]`http://citeseerx.ist.psu.edu/oai2`

```
{...
"http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.5389": {
  "source": "http://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf",
  "author": [
    "Ting-fan Wu",
    " Chih-Jen Lin",
    " Ruby C. Weng"
  ],
  "title": "Probability Estimates for Multi-class Classification by
          Pairwise Coupling"
  }
...}
```

FIGURE 2. Data stored for an article in our CiteSeerX-based dataset.

Matching errors	Cluster distance threshold								
Similarity threshold	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
0.6	124	111	104	99	92	**88**	89	**88**	89
0.65	91	84	82	80	77	**75**	77	78	81
0.7	79	75	74	71	71	**69**	72	73	75
0.75	91	88	**86**	**86**	87	87	89	90	92
0.8	134	122	118	115	**113**	**113**	116	116	118

TABLE 1. Matching errors obtained on set C_1 for titles by varying the similarity threshold from 0.6 to 0.8 by 0.05 (rows) and the cluster distance threshold from 0.4 to 2.0 by 0.2 (columns).

and C_3 with the remaining 2217 articles. C_1 was used for determining the best clustering parameters, C_2 was the training set, while the system was tested on C_3.

The dataset, which by its nature contains errors and inaccuracies, can be freely downloaded and used for further research.[14] The data is availabe in JSON format, where the key is the CiteSeerX identifier/URL of the article, and each item is described by three fields, *title*, *author* and *source*, the latter containing the URL to the downloadable file. Figure 2 shows the description of an article in this dataset.

5.2. **Experimental results.** Because our entire system was implemented in Python, despite its disadvantages, such as relative slowness and omission of certain chunks at extraction (see Figure 1(a)), the PDFMiner package was used.

For clustering the text chunks single linkage hierarchical clustering was used with a parametrized distance function, taking into account the distances between the chunks, the font names, sizes and styles, as described in Section 4.

[14]`http://www.cs.ubbcluj.ro/~zbodo/citeseerx4217.html`

Matching errors	Cluster distance threshold								
Similarity threshold	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
0.6	**274**	290	299	317	332	347	362	371	385
0.65	**297**	319	332	353	379	392	409	420	434
0.7	**337**	358	372	391	418	433	451	467	479
0.75	**389**	412	425	446	475	491	506	520	530
0.8	**438**	461	468	487	515	530	544	555	567

TABLE 2. Matching errors obtained on set C_1 for authors, by varying the similarity threshold from 0.6 to 0.8 by 0.05 (rows) and the cluster distance threshold from 0.4 to 2.0 by 0.2 (columns).

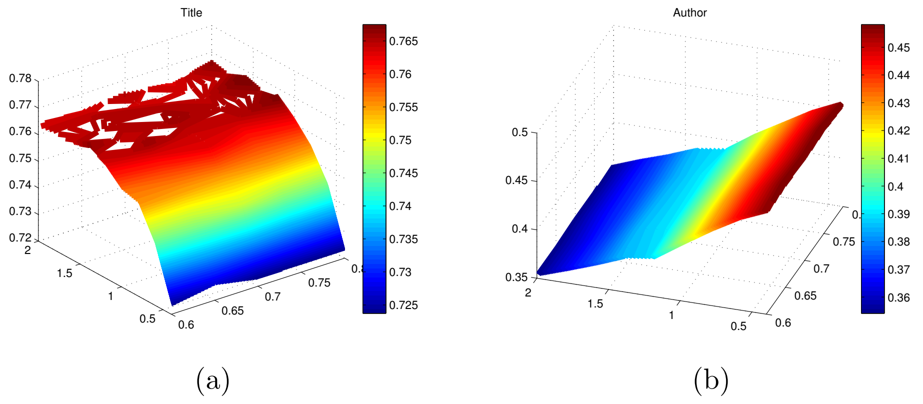


FIGURE 3. Errors obtained on set C_3 for (a) titles and (b) authors, using the same search grid as in Tables 1–2.

From the design of the distance metric it follows that the optimal threshold should be somewhere around 1, therefore we performed a grid search around this value.

The applied method for metadata extraction is similar to the one presented in [9], but for determining the text segments, we used the method described in Section 3. Our system currently extracts only the title and the authors of the article, without segmenting the author names. The feature vectors—including dictionary-based features, regular expressions, positional and font information—are constructed for each segment, and random forest classifiers

Accuracy	Title	Authors
SVMHeaderParse with PDFBox	0.6982	0.4713
SVMHeaderParse with PDFMiner	0.6414	0.4848
GROBID	0.7780	0.6337
Our method	0.7631	0.4091

TABLE 3. Accuracy results using C_3 for testing.

[2] are trained on these data. Two sets of 100 random decision trees were trained using the *scikit-learn* library[15], one for each class.

The distance threshold of the hierarchical clustering and the similarity threshold for finding the corresponding metadata influence each other, therefore we performed a joint search for the optimal values. In Tables 1–2 the number of matching errors obtained on set C_1 are shown, varying the clustering distance and similarity threshold. For these parameter combinations the metadata extraction system was trained on set C_2 and evaluated on C_3: Figure 3 shows the errors obtained. At evaluation we used a testing similarity threshold of 0.9 for checking whether the corresponding metada was found. Similarity was measured using cosine similarity, representing the texts as bag-of-words weighted by the frequencies. The author names were considered as one object.

We selected the best parameters by standardizing the rows of the matching error matrices, summing the two and finding the minimum of each row. Other methods for finding the optimal joint parameters can be applied as well, for example using an importance weighted linear combination of the matching errors for the different categories. Thus, for example using a similarity threshold 0.7 we obtained the best cluster threshold of 1.0. For these values we get accuracies of 0.7631 and 0.4091 for titles and respectively authors, using the above-mentioned similarity threshold of 0.9 in testing.

We compared our approach to other existing methods, as shown in Table 3. However, we were faced with the following difficulties when testing other systems. First, unlike our approach, existing systems [10] use supervised learning for training—either a classifier or a structured learning method—and use distinct sets of classes. Our solution, however, does not need labeled data, but only a set of PDF files along with a metadata database (see Figure 2), the labeling procedure is embedded into the training phase. Second, existing systems were primarily designed to use them just as they were trained, therefore re-training is not well-documented and difficult to perform. Hence, we were

[15]`http://scikit-learn.org/`

not able train these using set C_2, but only tested them on C_3. SVMHeaderParse is the metadata extraction module used by CiteSeerX implementing the method described in [7]. For text extraction it uses PDFBox, but since this component can be easily replaced, we also tested it using PDFMiner. GROBID[16] (GeneRation Of BIbliographical Data, version 0.4.1) is a complex metadata extraction tool for header metadata and bibliographical extractions. GROBID uses pdftoxml[17] for content and layout extraction and conditional random fields for learning [11, 12]. As the results in Table 3 show, our method performs well on title extraction, getting almost the same accuracies as GROBID, which obtained the best overall results in the experiments of [10]. For author extraction our system achieved the lowest accuracy results, thus a careful examination of the obtained results is required to be able to improve on the performance.

5.3. **Discussion.** In this paper we described a hybrid metadata extraction system, that uses clustering to identify cohesive text segments, after which, based on the features representing a segment, supervised learning is used to automatically find parts containing metadata information. The method described does not need a conventional labeled dataset for training, but only a set of PDF documents along with a metadata database. We also compiled a small dataset from CiteSeerX's database and used it to train and evaluate our method.

From the experiments it can be seen that finding titles is easier—this can be argued by the fact that the most important information about an article is its title, which always constitutes the most accentuated part of the title page. The relatively low accuracy obtained for authors can be explained by the noise in the training data, as well as by the fact that the author list is often broken up by additional information, e.g. affiliation, which makes the recognition process more difficult. Using less noisy training and test data our system achieves accuracies up to 90% and 70% for titles and authors, respectively.

The obtained results show that matching errors vary differently for distinct metadata categories, therefore a sound methodology is needed how to select the overall optimal parameters of the clustering method and possibly the similarity threshold too. Future investigations also include the application and testing of structured prediction models in metadata extraction using the proposed method.

[16]`https://github.com/kermitt2/grobid`

[17]`https://github.com/eliask/pdf2xml`

Acknowledgements. This work was supported by a grant of the Romanian National Authority for Scientific Research, CNDI–UEFISCDI, project number PN-II-PT-PCCA-2011-3.2-0895.

References

[1] J. Beel, S. Langer, M. Genzmehr, and C. Müller. Docear's PDF inspector: title extraction from PDF files. In *JCDL*, pages 443–444. ACM, 2013.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5, 2001.

[3] T. M. Breuel. High performance document layout analysis. In *Proceedings of the Symposium on Document Image Understanding Technology*, pages 209–218, 2003.

[4] B. H. Butt, M. Rafi, A. Jamal, R. S. U. Rehman, S. M. Z. Alam, and M. B. Alam. Classification of research citations (CRC). In *CLBib@ISSI*, volume 1384 of *CEUR Workshop Proceedings*, pages 18–27. CEUR-WS.org, 2015.

[5] C. Caragea, J. Wu, A. M. Ciobanu, K. Williams, J. P. F. Ramírez, H.-H. Chen, Z. Wu, and C. L. Giles. CiteseerX: A scholarly big dataset. In *ECIR*, volume 8416 of *Lecture Notes in Computer Science*, pages 311–322. Springer, 2014.

[6] M. Granitzer, M. Hristakeva, K. Jack, and R. Knight. A comparison of metadata extraction techniques for crowdsourced bibliographic metadata management. In *SAC*, pages 962–964. ACM, 2012.

[7] H. Han, C. L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. A. Fox. Automatic document metadata extraction using support vector machines. In *JCDL*, pages 37–48. IEEE Computer Society, 2003.

[8] A. Ivanyukovich and M. Marchese. Unsupervised metadata extraction in scientific digital libraries using a-priori domain-specific knowledge. In *SWAP*, volume 201 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[9] R. Kern, K. Jack, M. Hristakeva, and M. Granitzer. Teambeam - meta-data extraction from scientific literature. *D-Lib Magazine*, 18(7/8), 2012.

[10] M. Lipinski, K. Yao, C. Breitinger, J. Beel, and B. Gipp. Evaluation of header metadata extraction approaches and tools for scientific PDF documents. In *JCDL*, pages 385–386. ACM, 2013.

[11] P. Lopez. Grobid: Combining automatic bibliographic data recognition and term extraction for scholarship publications. In *International Conference on Theory and Practice of Digital Libraries*, pages 473–474. Springer, 2009.

[12] P. Lopez and L. Romary. Humb: Automatic key term extraction from scientific articles in grobid. In *Proceedings of the 5th international workshop on semantic evaluation*, pages 248–251. Association for Computational Linguistics, 2010.

[13] C. D. Manning, H. Schütze, and P. Raghavan. *Introduction to information retrieval.* Cambridge University Press, 2008.

[14] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *HLT-NAACL*, pages 329–336, 2004.

[15] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[16] J. Wu, J. Killian, H. Yang, K. Williams, S. R. Choudhury, S. Tuarob, C. Caragea, and C. L. Giles. PDFMEF: A multi-entity knowledge extraction framework for scholarly documents and semantic search. In *K-CAP*, pages 13:1–13:8. ACM, 2015.

[17] J. Wu, K. M. Williams, H.-H. Chen, M. Khabsa, C. Caragea, S. Tuarob, A. Ororbia, D. Jordan, P. Mitra, and C. L. Giles. CiteseerX: AI in a digital library search engine. *AI Magazine*, 36(3):35–48, 2015.

[18] X. Zhu. Semi-supervised learning literature survey. Technical Report TR 1530, University of Wisconsin, 2005.

Faculty of Mathematics and Computer Science, Babeş–Bolyai University, Cluj-Napoca, Romania

E-mail address: {`zbodo,lehel.csato`}`@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.02

IMPROVING SIFT FOR IMAGE FEATURE EXTRACTION

RENATA DEAK, ADRIAN STERCA, AND IOAN BĂDĂRÎNZĂ

Abstract. This paper reviews a classical image feature extraction algorithm, namely SIFT (i.e. Scale Invariant Feature Transform) and modifies it in order to increase its repeatability score. We are using an approach that is inspired from another computer vision algorithm, namely FAST. The tests presented in the evaluation section show that our approach (i.e. SIFT-FAST) obtains better repeatability scores over classical SIFT.

1. Introduction

Image matching techniques can be separated into two major categories - *feature-based* or *direct*. Direct image matching techniques involve directly matching one image's pixels values to the values of the pixels of another image. Therefore, this method looks at how much the pixels of two, or more, images agree [17]. This approach can be split into two steps: finding an appropriate error metric like Mean Squared Error and deciding on an efficient search technique. Although, exhaustive search can be applied, it is not an efficient approach, in particular when dealing with high resolution images.

On the other hand, feature-based matching techniques extract interest points from the input images and aim to minimize the distance between these interest points. This method is preferred to the direct-based method as it is more robust, due to the fact that feature extraction algorithms output features that are invariant to scale, rotation and translation. One of the earliest and most notable feature extraction algorithms is SIFT - Scale Invariant Feature Transform - developed by David Lowe, first published in 1999 [2]. This enabled the development of more feature extraction methods such as SURF [8], ASIFT [11], ORB [12], BRISK [15] and FAST [18], to name a few.

Feature extraction methods have two phases. The first one is detecting the interest points in an image. There is no formal definition of what constitutes

Received by the editors: August 22, 2017.

2010 *Mathematics Subject Classification.* 68U10, 94A08.

1998 *CR Categories and Descriptors.* I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement – *Feature Representation*; I.5.2 [**Pattern Recognition**]: Design Methodology – *Feature evaluation and selection.*

Key words and phrases. image feature extraction, SIFT, FAST.

an interest point, most papers defining it as an - interesting - part of an image, parts that are easily recognizable in two or more different images. This property is called repeatability, and it is used to measure the effectiveness of feature detection algorithms. It is desirable that extracted features are invariant to scale, rotation or translation. This is the reason why feature detection algorithms aim to detect corners, rather than detecting edges, as edges are invariant to translation only along their principal axis [5]. Invariance to scale is obtained by building a scale space of the input image that simulates different levels of zoom and blur applied on the initial image. The second one is building descriptors for these points, which will be used to identify a point within an image. These descriptors are later on used for matching features between images or for object detection purposes. Matching between features is done by minimizing the distance between their descriptors.

In this paper we start wih the classical SIFT algorithm and update it by adding a mechanism inspired from FAST that is meant to increase the repeatability of SIFT.

2. Related work

One of the first detectors introduced in the literature is the Harris corner detector, which detects points based on eigenvalues of the second-moment matrix [5]. However, this detector was not scale-invariant. It wasn't until Lindeberg introduced the concept of automatic scale selection [6] that the scale invariance of features was a property that feature detector algorithms pursued. In order to attach to each point its characteristic scale, Lindeberg used both the Hessian matrix and the Laplacian. What followed was a refinement of this method by Mikloajczyk and Schmidt called Harris-Laplace and Hessian-Laplace which issued robust, scale-invariant features. In order to avoid computation of the Laplacian of Gaussians, the idea to approximate this by the Difference of Gaussians was introduced by David Lowe, idea that was later used in his implementation of SIFT [2].

In terms of feature descriptors, there has been a wide variety of methods introduced, such as Gaussian derivatives [7], complex features [9] [10] and descriptors such as SIFT [2] that capture information about the spatial intensity patterns in the neighbourhood of the interest point. To date, this descriptor has proved to be the most robust one. Although there have been alternatives, to improve performance - such as PCA-SIFT, which encodes information in a 36-dimensional vector [19] - they have proven to be less distinctive [14]. GLOH was proposed as an alternative [14], which has proved to be more distinctive, but it is computationally expensive.

Another alternative to the SIFT descriptor was proposed by Se et al. [13], which is both fast and distinctive enough, however it has a drawback in the fact that the vectors extracted are of high dimensions, making the matching phase more difficult.

In [8] we can find another feature extraction method, named SURF (Speeded Up Robust Features), which defines three main steps for searching for discrete image correspondences: detection step, description step and matching step. In the first step, they are selecting distinctive locations in the image, like blobs, corners and T-junctions. These distinctive locations are called 'interest points'. In the second step, a feature vector is defined for each neighbourhood of every interest point. These vectors are used further in step three where they are matched between different images, by computing the Mahalanobis or Euclidean distance. Even though SURF is know for its robustness and speed. there are other algorithms, like BRISK (Binary Robust Invariant Scalable Keypoints) [15] that can achieve comparable quality for matching but with much less computation time. BRISK is a method for generating keypoints from an image in two phases: detecting scale-space keypoints using a saliency criterion and keypoint description.

ORB (Oriented FAST and Rotated BRIEF) is another very fast binary descriptor that is build on top of FAST keypoint detector and BRIEF descriptor. Combining these two methods, you can achive very good performance and very low cost. Our own approach combines the robustness of the SIFT descriptors with the detection of the FAST algorithm in order to achieve finding better candidate points from the first extraction of points, but also increasing the repeatability of the features extracted.

3. SIFT - Scale Invariant Feature Transform

The SIFT algorithm outputs features that are invariant to scale, rotation and translation. The first step in SIFT feature extraction is to compute a Gaussian scale-space from the initial image. 5 octaves of images are constructed from the initial image where each octave contains variants of the initial picture with a decreased samplerate. In each octave there are 5 images with the same samplerate, but with increasing blur levels (Gaussian blur is used). A part of this Gaussian scale-space is depicted in Fig. 1 for an image that is later used in the evaluation section. After this, the DOG (i.e. Difference of Gaussian) space is computed by applying the differential operator to the Gaussian scale-space and the 3D extremum points are extracted from the DOG space. Similarly, a part of the DOG representation is displayed in Fig. 2 for the same image. The 3D extremum points are coarsely detected from the

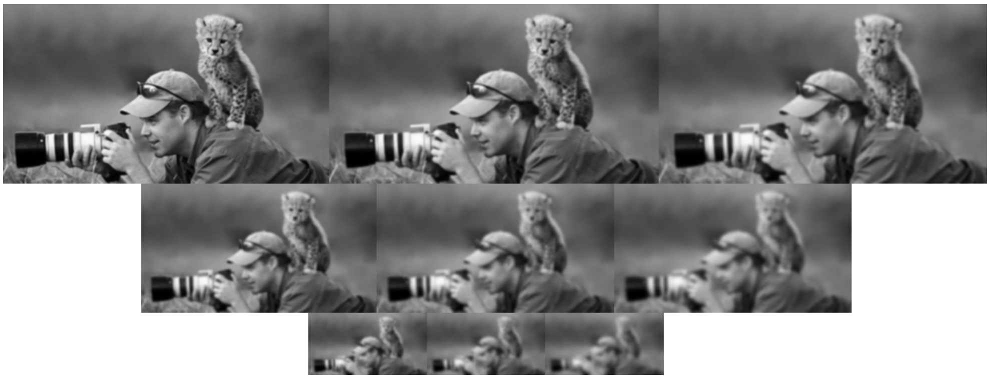


Figure 1. Gaussian scale space. The first row represents the first 3 images in the first octave, the second row contains the first 3 images from the second octave, and the third from the third octave.

DoG scale-space by taking the minimum or the maximum from a neighbourhood of 26 pixels (see Fig. 3). This is done in order to have extremum points that are scale invariant. Unfortunately, these extremum candidate points are very are sensitive to noise. Several filters are applied then in order to discard low importance extreme points like low contrast keypoints or candidate keypoints that are on the edges. Furthermore, maximum or minimum points are not situated directly on the pixel, they usually lie in between pixels. So in order to be able to address such a pixel, a position refinement is applied, as well as a scale refinement, since points detected previously are constrained to the sampling grid. After low-contrasted pixels are discarded, local interpolation (using an approximation of the second order Taylor polynomial) is applied on the remaining candidate points to refine their location and scale.

The final step in SIFT is constructing a 128 byte descriptor for each keypoint found. SIFT feature descriptors are computed by extracting one or more dominant orientations for each keypoints over a normalized patch. This ensures the rotation-invariant property of the extracted keypoints. Due to the fact that there may be more than one dominant orientation for a single keypoint, the number of feature descriptors extracted may be higher than the number of features detected in the previous steps. The first step in extracting the dominant orientation is to build a patch around the keypoint which contains pixels that lie at a distance smaller than a threshold value from the keypoint's position. Then, within the normalized patch, a magnitude and orientation are computed from the gradient of the image with respect to the x-coordinate and

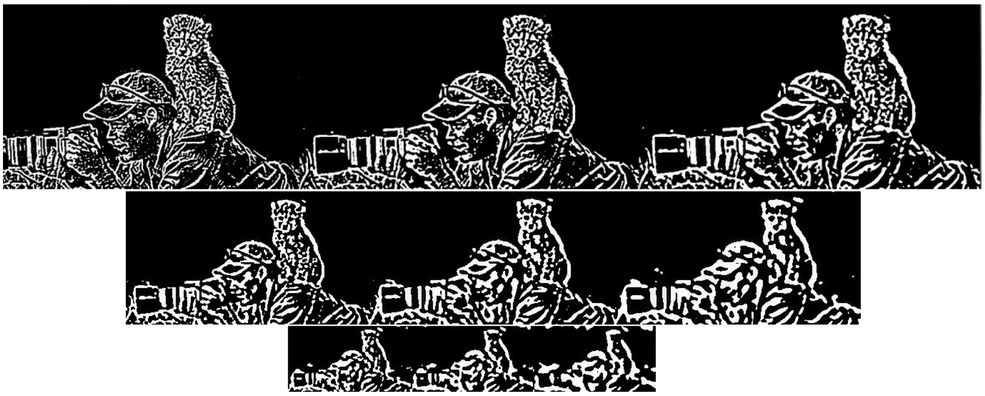


FIGURE 2. The DoG representation. The first row represents the first 3 images in the first octave, the second row contains the first 3 images from the second octave, and the third from the third octave.

the y-coordinate. A histogram of orientations will be built from the orientations extracted. The histogram is built by dividing the interval $[0, 2\pi]$ into 36 bins of 10 degrees each. Then the orientation is assigned to the closest bin. For example, if the orientation is $\pi/4$, the corresponding bin will be the one for degrees 40-49. In order to smooth out noise and make distant pixels have a smaller influence on the gradient assignation, the entries are weighted by a Gaussian window function which is centered at the interest point. The next step in extracting dominant orientations is to smooth the histogram by applying a circular convolution six times with a three-tap box filter. From the smoothed histogram, the orientations will be extracted from local maxima positions that are larger 0.8 times than the global maximum. Consequently, we may extract more than one orientation for a single interest point. Once the orientations have been computed for each keypoint, this information is quantized into 128 dimension vectors. In order to compute the descriptor, the information of the local spatial distribution of the gradient orientation on a particular neighborhood must be encoded. As a neighborhood, there have been papers where the entire image was used [1]. However, the original SIFT method takes a square normalized patch aligned with the orientation of the point, to induce invariance to scale, rotation and translation [2].

[1]Image taken from http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoints/

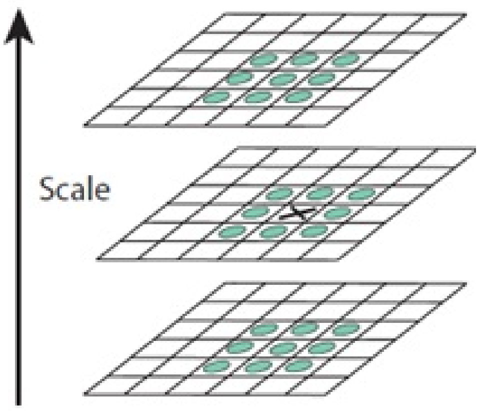


Figure 3. DoG 3D extremum candidate, if pixel 'x' is either smaller or larger than all its 26 neighbours. The point marked with X in the middle image represents the candidate point, the image below is the image with a lower blur level and the image above is the image with higher blur level [1]

4. Improving the repeatability of SIFT

In the classical SIFT implementation candidate points are extracted by finding 3D extrema points, in neighborhoods of 27 pixels. This means that each pixel of the image will be compared to its 26 neighbors - 9 from the image with a smaller level of blur, 8 from the current image and 9 from the image with a greater level of blur. A pixel is considered a candidate point if its value is either smaller or larger than all of its 26 neighbors. Our new approach is to use FAST [3] for detecting candidate keypoints of the DoG scale-space. FAST is a corner detection algorithm and in FAST, a circle of sixteen pixels is also known as the Bresenham circle of radius 3 [20] - around a candidate pixel p is considered. The pixel is a corner if there is a consecutive sequence of n pixels in the circle which are all either brighter than the candidate pixel by a certain threshold or darker than the candidate point by the same threshold. In our approach for increasing the repeatability of SIFT, instead of searching through a square of $3x3$ neighbors in 3 dimensions, the search for extrema is done by searching through a circle of radius 3 of 16 pixels a sequence of n pixels that are all either brighter than the candidate point by a certain threshold, or are darker than the candidate point by the same threshold (see Fig. 4).

The motivation behind choosing this combination of methods is the fact that this detection has the potential of extracting keypoints with high repeatability score. The repeatability is a desired property of the extracted features, as it evaluates whether or not the feature will be detected in other images containing the same scene. The FAST detection shows great potential towards this goal, since it does not restrict the pixel to be brighter or darker than all its nearest 26 neighbors. Instead, a circle is considered, from which a sequence of n pixels must be either brighter or darker than the candidate point. So the chances that the feature might be detected in a blurrier image, for example, are higher than with the classical SIFT detection. For the threshold t, tests have been run in order to evaluate how it affects the repeatability score of the extracted features, and the best value obtained was 0.018. Based on the tests from [16], the value for n was left to 12, as it provides the best results in terms of number of features extracted and the redundancy of the extracted features. This means that from each of the three images a sequence of 12 consecutive circle pixels is found, leading to 36 pixels that should be either darker or brighter than the candidate pixel by a threshold t.

5. Evaluation

In order to evaluate our SIFT-FAST approach, we have implemented classical SIFT and SIFT-FAST and compared the two algorithms on photos from different domains (thus having different entropy levels): a photo with an animal in nature (containing a reasonable amount of color changes and large blurred areas in the background), a human face (containing distinct areas of color changes) and a landscape photo (containing many small areas with small color fluctuations - i.e. the grass). The dimensions of these 3 photos are also different: 915x497, 500x366 and respectively, 425x378 pixels. We have compared the two implementations using two metrics: *the computation time* and *the repeatability score*. The computation time metric is evaluated for both approaches in order to see how much computational overhead does SIFT-FAST introduce and the second metric, repeatability, is used to evaluate if SIFT-FAST is more suited than classic SIFT for object recognition in images. For the first metric, considering that the only difference between these implementations is the extraction of the initial set of keypoints, the time required to extract these points was measured for both approaches and the results obtained are in figures 5, 6 and 7.

As it can be observed in these figures, classical SIFT detection is more efficient from the point of view of the execution time, compared to SIFT-FAST. This is due to the fact that for SIFT-FAST, a consecutive sequence of 12 pixels needs to be found in three different images, whereas with SIFT the coordinates of the pixels that are used in the comparison are known beforehand. However, this is not a big drawback for the SIFT-FAST approach, as it is still executed

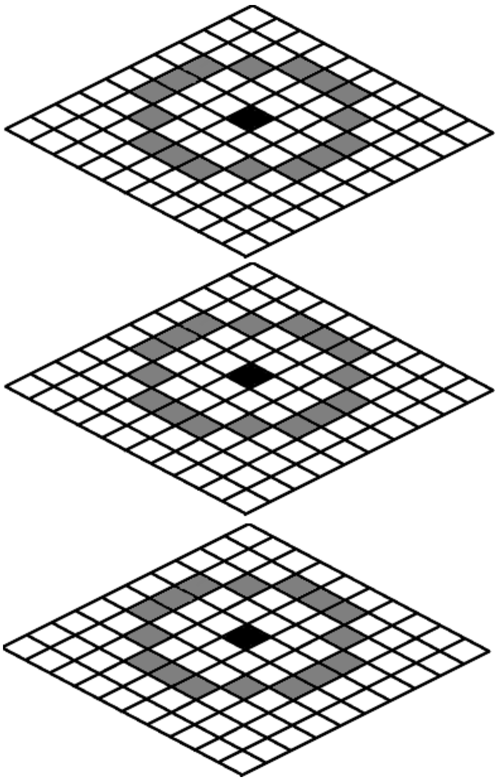


Figure 4. Candidate point detection for FAST-SIFT keypoint. The three grids represent 3 consecutive scales of an image within an octave. The black pixel is the candidate point and the grey pixels represent the circle from which a sequence of n consecutive pixels needs to be darker or brighter than the black pixel by a certain threshold.

under 0.6 seconds. It just means with SIFT the check is completed faster. Although SIFT-FAST takes more time because it extracts more reliable points in this first step than classical SIFT.

Another important property of feature detecting algorithms is the *repeatability of the features*. This means that having two different images of the same scene, the features detected in the first image are detected in the second image as well. The repeatability score for an image was computed by taking the image and a blurrier version of the same image (obtained by applying a Gaussian blur filter) and computing the percentage of keypoints that are found in both images (i.e. percentage of common keypoints).

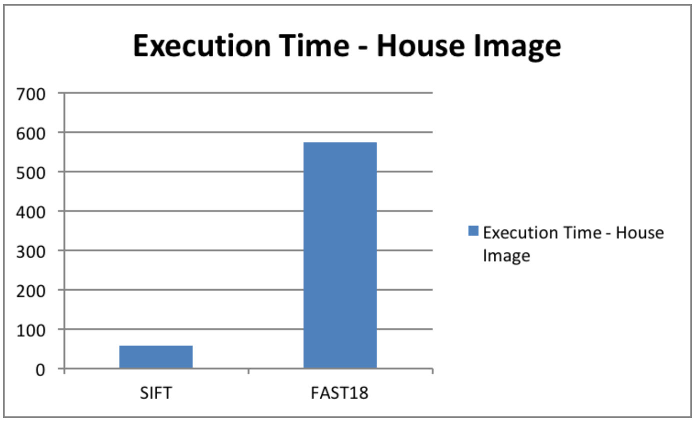


FIGURE 5. Execution time for extracting first set of keypoints for an image with a house. FAST18 represents running FAST detection with threshold=0.018. (time is measured in milliseconds)

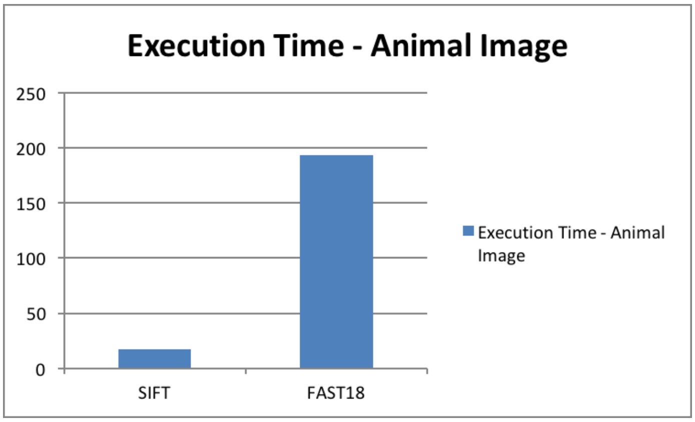


FIGURE 6. Execution time for extracting first set of keypoints for the image with an animal. (time is measured in milliseconds)

The results obtained for the same set of input images are illustrated in figures 8, 9, and 10, respectively. The notations FAST16, FAST17, FAST18 and FAST19 represent the SIFT-FAST algorithm with the value 0.016, 0.017, 0.018 and 0.019 for the threshold used for comparisons. From these tests, the conclusion is that overall the most suitable value for the threshold is 0.018.

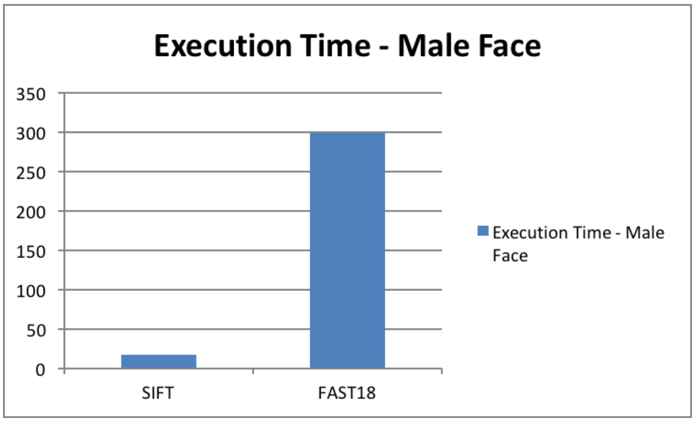


FIGURE 7. Execution time for extracting first set of keypoints for an image with a male face. (time is measured in milliseconds)

It is clearly visible in figures 8 - 10 that the SIFT-FAST approach yields better repeatability results than the classical SIFT algorithm irrespective of the threshold used, although some threshold values give better results than others. This result is ensured by the initial extraction of the keypoints. If a point is chosen as candidate keypoint in one image, in means that it has found the sequence of 12 pixels that are either brighter or darker in the current image, in an image with a smaller level of blur and an image with a higher level of blur. Consequently, there is a high probability that if a pixel was chosen as candidate in one image, it will be found as candidate in another image of the same scene (with different blur level, luminosity or small translate transformations applied).

The advantages of the SIFT-FAST approach over classical SIFT can also be depicted visually. In Fig. 11 we show the initial set of candidate points extracted from the three images using classical SIFT. As it can be seen, in this first phase of feature detection, the features are scattered all over the image, and they are not reliable in this phase. Then, following the SIFT workflow, a number of filters are applied to this initial set of candidate points: threshold filter, low-contrast filter, quadratic interpolation and removal of keypoints located on edges. The final set of keypoints is depicted in Fig. 12. Fig. 13 illustrates this first initial set of keypoints obtained by our SIFT-FAST approach in the same three images. It can be observed that unlike the initial detection with classical SIFT, the keypoints are much better positioned. For example in the first image, there is no point selected in the background, where

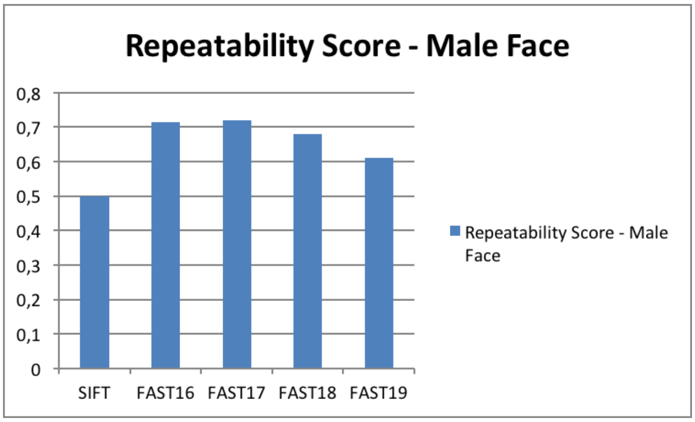


FIGURE 8. Repeatability score for image with male face.

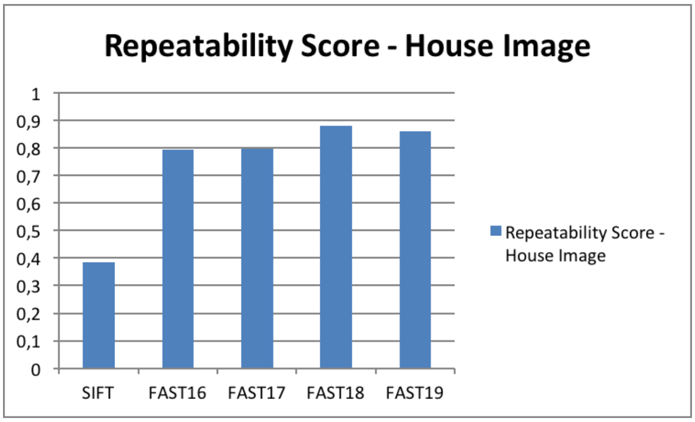


FIGURE 9. Repeatability score for image with house.

the image is out of focus, and even with the human eye, no object can be uniquely distinguished, there are no keypoints detected. With classical SIFT initial detection, the points where scattered all over the image, as it can be observed in figure 11. Then, the workflow continues the same as for classical SIFT algorithm and the final set of keypoints extracted with this approach is represented in Fig. 14. Comparing the final keypoints extracted by this approach and the classical SIFT approach it can be observed that this approach extracts fewer candidate keypoints than SIFT. However, defining what makes an extracted keypoint important is highly dependent on the application domain. For example, by comparing the points extracted for the first

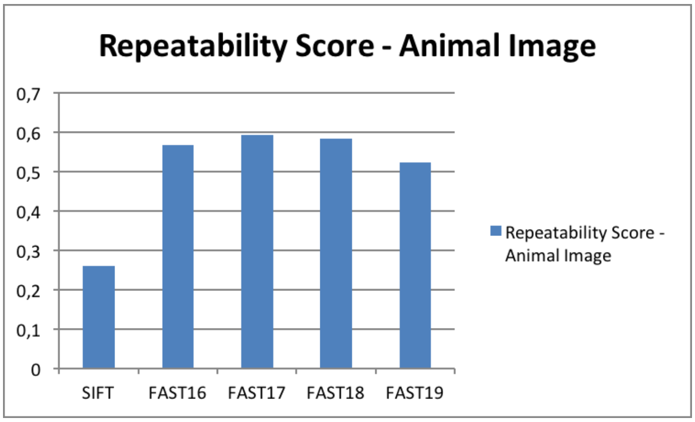


Figure 10. Repeatability score for image with animal.

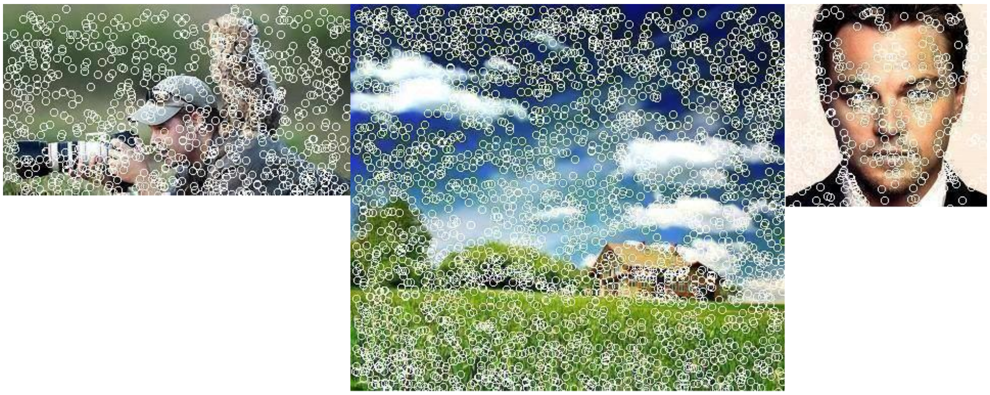


Figure 11. The first set of keypoints detected in our test images using classical SIFT.

images in the three test images by SIFT and the SIFT-FAST approach, it can be observed that the latter extracts little to no points on the man's shirt, which seems correct as the variations of contrast in that area of the image are generated by shadow only.

6. Conclusions and future work

We considerred in this paper the SIFT feature extraction algorithm introduced by David Lowe in 1999 [2]. A new approach was proposed by combining the classical SIFT algorithm with a FAST-like detection of initial keypoints.

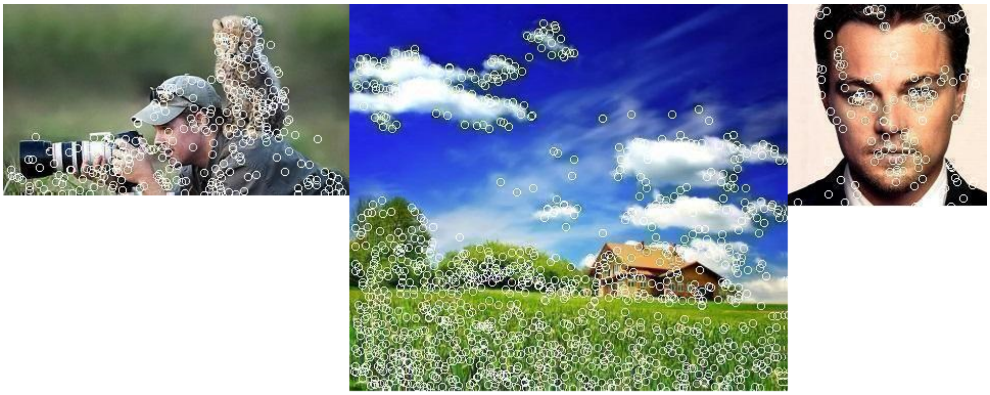


FIGURE 12. The final set of keypoints detected in our test images using classical SIFT (after low-contrast filtering, interpolation)

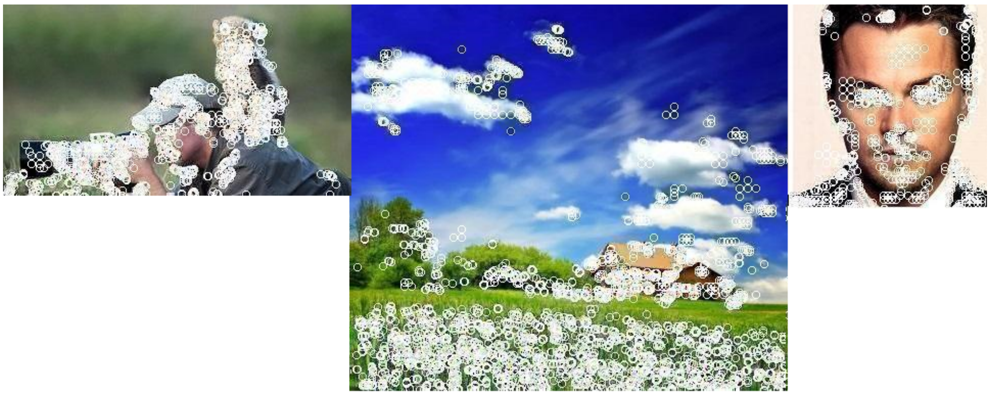


FIGURE 13. The first set of keypoints detected in our test images using SIFT-FAST approach.

Instead of scanning for 3D local extrema points using the 26 neighbors as the SIFT algorithm does, the points are scanned in a FAST-like way. That is, the candidate point is selected if on the circle of radius 3 having the point in its center, there are 12 pixels either brighter or darker than the point by a threshold. This check is done on the current level of blur, on the previous sample image with a smaller level of blur and on the next sample image with a higher level of blur.

In our tests, we compared these two methods in terms of execution time and repeatability of features. We showed that although using the FAST detector

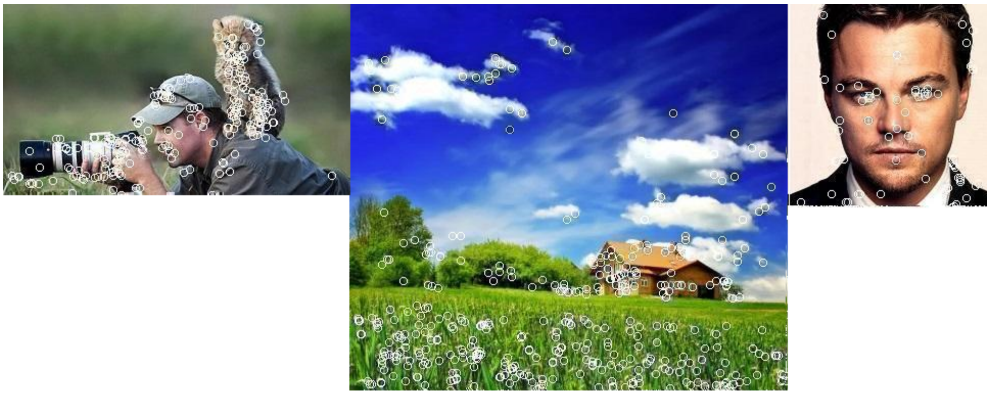


FIGURE 14. The final set of keypoints detected in our test images using SIFT-FAST approach (after low-contrast filtering, interpolation)

for extracting the initial set of keypoints was more time-consuming, it yielded better results in terms of repeatability of features which is important for image object recognition tasks. The reason why execution time is higher using SIFT-FAST algorithm is that a consecutive sequence of 12 pixels needs to be found, on three levels of blur, whereas for the classical SIFT, there are 26 neighbors that are checked only. However, the fact that the repeatability test had better results of this approach, than the classical SIFT, makes this drawback have lesser importance.

References

[1] Hassner, T., Mayzels, V., Zelnik-Manor, L., On SIFTs and their scales, IEEE Conference on Computer Vision and Pattern Recognition, Washington DC, USA, June 16-21, 2012, pp. 1522-1528.

[2] Lowe, D., Object recognition from local scale-invariant features, In Proceedings of the 7th International Conference on Computer Vision, Washington DC, USA, September 20-25, 1999, pp. 1150-1157.

[3] Rosten, E., Drummond, T.: Fusing points and lines for high performance tracking, 10th IEEE International Conference on Computer Vision, Washington DC, USA, October 17-20, 2005, pp. 1508-1515.

[4] Otero, I.R., Delbracio, M., The anatomy of the SIFT method, Image Processing On Line, vol. 4, 2014, pp. 370-396.

[5] Harris, C., Stephens, M.: A combined corner and edge detector, Proceedings of the 4th Alvey Vision Conference, Manchester, 31 August - 2 September, 1988, pp. 147-151.

[6] Lindeberg, T., Scale-space theory in computer vision, Kluwer Academic Publishers Norwell, MA, USA,1994.

[7] Florack, L.M.J., Haar Romeny, B.M.T., Koenderink, J.J., Viergever, M.A.: General intensity transformations and differential invariants, Journal of Mathematical Imaging and Vision, May 1994, Volume 4, Issue 2, pp 171-187.
[8] Bay, H., Tuytelaars, T., Van Gool, L., Surf: Speeded up robust features., Proceedings of the European Conference on Computer Vision, Graz, Austria, May 2006, pp 404-417.
[9] Baumberg, A., Reliable feature matching across widely separated views, Conference on Computer Vision and Pattern Recognition, Hilton Head Island, South Carolina, 15 June 2000, pp. 774-781.
[10] Schaffalitzky, F., Zisserman, A., Multi-view matching for unordered image sets, or "How do I organize my holiday snaps', European Conference on Computer Vision, Copenhagen, Denmark, May 28-31, 2002, pp. 414-431.
[11] G. Yu and J-M. Morel, ASIFT: An Algorithm for Fully Affine Invariant Comparison, Image Processing On Line, vol. 1, 2011, pp. 438-469.
[12] Rublee, E., Rabaud, V., Konolige, K., Bradski, G., ORB: An efficient alternative to SIFT or SURF, Proceedings of IEEE International Conference on Computer Vision, Washington DC, USA, November 06-13, 2011, pp. 2564-2571.
[13] Se, S., Ng, H., Jasiobedzki, P., Moyung, T., Vision based modeling and localization for planetary exploration rovers, Proceedings of the 55th International Astronautical Congress, Vancouver, Canada, 4-8 October pp. 364-375.
[14] Mikolajczyk, K., Schmid, C., A performance evaluation of local descriptors. Pattern Analysis and Machine Intelligence, vol. 27, issue 10, 2005, pp. 1615-1630.
[15] Leutenegger, S., Chli, M., Siegwart, R.Y., BRISK: Binary Robust Invariant Scalable Keypoints, Proceedings of IEEE International Conference on Computer Vision, Barcelona, Spain, 6-13 November, 2011, pp. 2548-2555.
[16] Rosten, E., Porter, R., Drummond, T., Faster and better: A machine learning approach to corner detection, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 32, issue 1, pp. 105-119, 2010.
[17] Szeliski, R., Image alignment and stitching: a tutorial, Foundations and Trends in Computer Graphics and Computer Vision, Now Publishers, pp. 1-104, 2006.
[18] Rosten E., Drummond, T., Machine learning for high-speed corner detection, European Conference on Computer Vision, Graz, Austria, May 07-13, 2006, pp. 430-443.
[19] Ke, Y., Sukthankar, R., PCA-SIFT: A more distinctive representation for local image descriptors, IEEE Conference on Computer Vision and Pattern Recognition, Washington DC, USA, 27 June-2 July 2004.
[20] Donald Hearn, M. Pauline Baker, Computer graphics, Prentice-Hall, USA, 1994.

Faculty of Mathematics and Computer Science, Babeş–Bolyai University, Cluj-Napoca, Romania
E-mail address: `drhp0888@scs.ubbcluj.ro`
E-mail address: `forest@cs.ubbcluj.ro`
E-mail address: `ionutb@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.03

LOGICAL TIME AND SPACE OF THE NETWORK INTRUSION

DANIEL MIHÁLYI, JÁN PERHÁČ, AND PATRIK BÁLINT

Abstract. Nowadays, one of the biggest threats for modern computer networks are the cyber attacks. One of the possible ways how to increase the level of computer networks security is a deployment of a network intrusion detection system. This paper deals with the behavior of the network intrusion detection system during specific network intrusion. We formally describe this network intrusion by the modal linear logic formula. Based on this formula, logical space and logical time is expressed from the attacker, and the network environment point of view in the usage of the Ludics theory.

1. Introduction

Just as people communicate with each other, so do the computers. This communication takes place within computer networks. As computer users, people encounter the term 'computer network' on a daily basis. Computer security threats are relentlessly inventive. These threats constantly evolve the possibilities how to find new ways to annoy, steal, or harm the user's data. Intrusion Detection System (IDS) [6] is one of the ways how to protect computer network from security threats. IDS is a device or software application that monitors a network or computer systems for malicious activities or policy violations. Each detected activity or violation is typically reported either to an administrator or collected centrally. A sequence of such causal activities can be described by a resource-oriented logical system properly.

In our approach, we use Linear Logic, which is a suitable logical system for usage in the field of computer science. This Logic [1] [2], is a substructural logic proposed as a refinement of classical and intuitionistic logic. Linear Logic brings new possibilities how to reason about formulæ in the resource oriented

Received by the editors: October 15, 2017.

2010 *Mathematics Subject Classification.* 18C10, 68M10.

1998 *CR Categories and Descriptors.* I.2.7 [**Mathematical Logic and Formal Languages**]: Formal Languages – *Language models.*

Key words and phrases. Linear Logic, DDoS Attack, SYN Flood, Ludics, Logical Space, Logical Time.

form. It means that a formula can be considered as a action or a resource that is performed in control manner. For example, the linear implication is causal which means that after performing it, the assumption is consumed.

In our previous work, we have created the network laboratory and described IDS's behavior during a ARP spoofing attack by linear logic formula [10]. From the coalgebraic point of view, we have described behavior of a IDS step by step through the coalgebra for a polynomial endofunctor in [8]. Then we have translated real network intrusion signatures based to coalgebraic one [9]. In this paper, we present usage of the Modal Linear Logic, which is a suitable logic to describe the behavior of state-oriented dynamic of an executed program system. The whole process of performing network attack and catching a network intrusion by a IDS is specified by behavioral the resource oriented logical formula. Then, we apply a logical time and a logical space from the Ludics theory [3], which was proposed by the J. Y. Girard [1]. In terms of this theory, we can consider a behavioral formula as a polarized game between an attacker and a network environment.

2. Modal linear logic

For the exact description of intrusion detection system's behavior, we have introduced our new logical system. We have proposed the Modal Linear Logic for IDS, which results from generalization of the linear logic's multiplicative fragment and the coalgebraic logic [2] [7]. Compared to the other logical systems, the significant feature of linear logic is resource-oriented approach of dealing with formulæ [1], which creates a strong expressive power for describing the real processes [2], e.g. causality, pleonasm or parallelism and many more [5]. These, together with a modal operators of the coalgebraic modal logic, create an appropriate formalism for describing a behavior of state-oriented program systems such as IDS.

2.1. **Syntax of Modal linear logic.** We formulate the syntax of Modal linear logic in [5], by the following production rule in the Backus-Naur form:

$$\varphi ::= a_n \mid 1 \mid \perp \mid \varphi \otimes \psi \mid \varphi \parr \psi \mid \varphi \multimap \psi \mid \varphi^{\perp} \mid \Box\varphi \mid \Diamond\varphi,$$

where:

- a_n represents the elementary formulæ, where $n = \{1,2\ ...\}$,
- $\varphi \otimes \psi$ with its neutral element 1 is the multiplicative conjunction, which describes the process of performing of both actions simultaneously,
- $\varphi \parr \psi$ with its neutral element $\perp$ is the multiplicative disjunction, which expresses the commutativity of duality between available and

consumed resources by performing either the action φ or the action ψ,

- $\varphi \multimap \psi$ is the linear implication, which expresses that the (re)action ψ is the causal consequence of the action φ and after performing this implication, the resource φ became consumed ($\varphi^{\perp}$),
- $\varphi^{\perp}$ is the linear negation, which expresses duality between action (φ) and reaction ($\varphi^{\perp}$), in the other words, an available and a consumed resource, respectively,
- $\Box\varphi$ is the modal operator expressing necessity of the action φ,
- $\Diamond\varphi$ is the modal operator expressing possibility of the action φ.

2.2. **The proof system.** The proof system of Modal linear logic is defined in the Gentzen's double side sequent calculus. The building block of this calculus is a sequent, which has the following form:

$$\Gamma \vdash \Delta, \tag{1}$$

where Γ, Δ represent (finite) sets of formula(e).

The inference rules have following

$$\frac{assumption(s)}{conclusion}(rule\ name), \tag{2}$$

where the *assumption*(*s*) and the *conclusion* are sequents. There is a special type of rules without assumption, called *axioms*.

They are defined as follows:

- The identity rule:
$$\frac{}{\varphi \vdash \varphi}(id)$$
- The structural rules are a cut rule and exchange rules:
$$\frac{\Gamma \vdash \varphi \quad \Delta, \varphi \vdash \psi}{\Gamma, \Delta \vdash \psi}(cut)$$
$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \psi, \varphi \vdash \Delta}(exl) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \psi, \varphi, \Delta}(exr)$$
- The logical rules deal with logical connectives:
$$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}(1_l) \qquad \frac{}{\vdash 1}(1_r) \qquad \frac{}{\perp \vdash}(\perp_l) \qquad \frac{}{\Gamma \vdash \perp, \Delta}(\perp_r)$$
$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \otimes \psi \vdash \Delta}(\otimes_l) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Phi \vdash \psi, \Sigma}{\Gamma, \Phi \vdash \varphi \otimes \psi, \Delta, \Sigma}(\otimes_r)$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Phi, \psi \vdash \Sigma}{\Gamma, \Phi, \varphi \multimap \psi \vdash \Delta, \Sigma}(\multimap_l) \qquad \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \multimap \psi, \Delta}(\multimap_r)$$

$$\frac{\Gamma, \varphi \vdash \Delta \quad \Phi, \psi \vdash \Sigma}{\Gamma, \Phi, \varphi \parr \psi \vdash \Delta, \Sigma}(\parr l) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \parr \psi, \Delta}(\parr r)$$

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \varphi^\perp \vdash \Delta}(()^\perp_l) \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \varphi^\perp, \Delta}(()^\perp_r)$$

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \Box\varphi, \Delta}(\Box r) \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \Box\varphi \vdash \Delta}(\Box l)$$

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \Diamond\varphi, \Delta}(\Diamond r) \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \Diamond\varphi \vdash \Delta}(\Diamond l)$$

The proof of a formula is proof tree, constructed from the root (the bottom of the tree) up to the leaves. The proof tree leaves have to be axioms, which implies that Gentzen's style proof tree is constructed correctly. When all leaves are axioms, the formula is proven.

3. Motivation Example

In this section, we briefly introduce basic notions of the used methods related to the detection of a network intrusion by the intrusion detection system, and an informal description of the particular attack, that we demonstrate in the motivation example bellow.

IDS is a security system that monitors the computer system's activities and its network traffic, and analyzes that traffic for possible hostile attacks originating from outside of an organization, and also for a system misuse or attacks originating from inside of an organization. It provides the three significant functions: monitoring, detecting, and responding [4] to unauthorized activities by company insiders and outsider intrusions. IDS uses policies to define certain events that if detected, will issue an alert.

Our motivation example is based on the execution of a Distributed Denial of Service (DDOS) type of attack, which is "extended" Denial of Service (DOS) attack type. The point of the DOS is flooding a target (e.g. server) by requesting attempts to overload it. In case of a DDOS, the attack is performed from more hosts at the same target(s) at the same time. Nowadays there are plethora of DDOS attacks. We have chosen the Syn Flood attack. This attack exploits the Transmission Control Protocol's (TCP) "three way handshake", during a client attempt to connect with a server. The server first passively listens at a port for possible connections. To establish a connection, the client sends a **SYN** to the server. The **SYN** contains various information, but what is important, it contains IP address of the client. The server allocate resources for possible connection for the IP address for a some time *(half-open connection)*. Then the server responses by sending the SYN-ACK, to which the client response with **ACK**. After that a connection is established. The SYN Flood attack exploits the first step of this process. It sends multiple requests (**SYN**) for connections to the server, but with spoofed IP addresses. This can result into the server's overload, which cause its malfunction.

To demonstrate the SYN Flood attack, we have created the laboratory environment (as shown in Figure 1), where we can see the Attacker's machine, its Terminals, the Victim's machine (with a localhost running) and the Router. Attacker uses five terminals to flood the Victim's web server services.

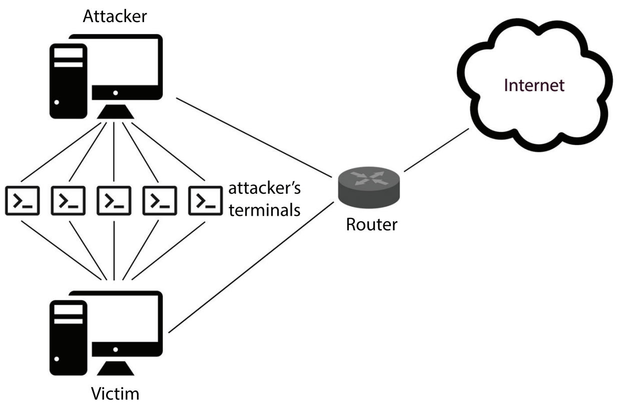


FIGURE 1. Laboratory network environment.

In our case, it is necessary to do the following steps to perform chosen attack:

(1) examination of the Local Area Network (LAN) topology (address space, network mask, default gateway etc.), e.g. by the tool *nmap*,
(2) perform a check for open ports on clients (potential victims) connected to the LAN by port scan,
(3) execution of the Syn Flood attack at chosen client, from the 5 terminals simultaneously.

3.1. **Formula in Modal Linear Logic.** Now, we can describe the formula (see Figure 3) of the network intrusion by Modal Linear Logic for IDS (section 2):

(3) $(((N \multimap S_1) \multimap \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2)) \multimap \Box U_N$,

where:

- N represents a vertical port scan of the victim's host port,
- S_1 is a reaction of IDS to vertical scanning of the victim's host ports by creating a log about a potential attack,
- $\Diamond U_M$ represents possible network intrusion,
- elements H_1 ... H_5 represent executing the SYN Flood attack from Attacker's five terminals to the Victim's machine,
- S_2 is a reaction to the SYN Flood attack from the Attacker's terminals,
- $\Box U_N$ represents the necessity of successful network attack.

The formula (3) can be interpreted as follows.

- *"Vertical port scan executed by the attacker (N)*
- *implies ($\multimap$)*
- *an action of the IDS by creating a log (S_1),*
- *and that implies $\multimap$*
- *a possible network intrusion ($\Diamond$ U_M),*
- *and ($\otimes$)*
- *performing the SYN Flood attack from the attacker's five terminals ($H_1\otimes, \dots, \otimes H_5$),*
- *implies ($\multimap$)*
- *an action of IDS by creating a log (S_2),*
- *and that all implies ($\multimap$)*
- *the necessity of the network intrusion. ($\Box U_N$)"*

Next step is to create a proof tree in Linear Logic proof system, which is constructed from the root to leaves, as shown in Figure 2. All leaves have to be identities. The whole proof tree represents a process of the SYN Flood attack from the Attacker's point of view. The contexts in the proof tree are defined in Figure (3). Every deduction step in the proof tree above (Figure 2) is realized by using an appropriate rule (defined in the Section 2.2) of the linear Gentzen's calculus.

3.2. **De Morgan's form.** The original formula (3) demonstrate a process of the attack from the attacker's point of view. To show the same process from the network environment, it is necessary to transform it to the orthogonal one. It can be done by application of the De Morgan's laws Table 1. By applying

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash N}\,(\mathrm{id}) \qquad \dfrac{\dfrac{}{S_1,\Gamma_2 \vdash U_M,\Gamma_3}\,(\mathrm{id})}{S_1,\Gamma_2 \vdash \Diamond U_M,\Gamma_3}\,(\Diamond\mathrm{r})}{N \multimap S_1,\Gamma_4 \vdash \Diamond U_M,\Gamma_5}\,(\multimap\mathrm{l})}{(N \multimap S_1),\Gamma_4 \vdash \Diamond U_M,\Gamma_5}\,(\mathrm{brac{-}l})}{\vdash (N \multimap S_1) \multimap \Diamond U_M,\Gamma_6}\,(\multimap\mathrm{r})}{\vdash ((N \multimap S_1) \multimap \Diamond U_M),\Gamma_6}\,(\mathrm{brac{-}r}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H_1,H_2,H_3,H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\mathrm{id})}{H_1 \otimes H_2,H_3,H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\otimes\mathrm{l})}{(H_1 \otimes H_2),H_3,H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\mathrm{brac{-}l})}{(H_1 \otimes H_2) \otimes H_3,H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\otimes\mathrm{l})}{((H_1 \otimes H_2) \otimes H_3),H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\mathrm{brac{-}l})}{((H_1 \otimes H_2) \otimes H_3) \otimes H_4,H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\otimes\mathrm{l})}{(((H_1 \otimes H_2) \otimes H_3) \otimes H_4),H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\mathrm{brac{-}l})}{(((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5,\Gamma_7 \vdash S_2,\Gamma_8}\,(\otimes\mathrm{l})}{((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5),\Gamma_7 \vdash S_2,\Gamma_8}\,(\mathrm{brac{-}l})}{\vdash ((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2,\Gamma_9}\,(\multimap\mathrm{r})}{\vdash (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2),\Gamma_9}\,(\mathrm{brac{-}r})}{\vdash ((N \multimap S_1) \multimap \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2),\Gamma_{10}}\,(\otimes\mathrm{r})}{\vdash (((N \multimap S_1) \multimap \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2)),\Gamma_{10}}\,(\mathrm{brac{-}r}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{U_N \vdash U_N}\,(\mathrm{id})}{\Box U_N \vdash U_N}\,(\Box\mathrm{l})}{\Box U_N, U_N^{\perp} \vdash}\,(()^{\perp}_1)}{\Box U_N, \Diamond U_N^{\perp} \vdash}\,(\Diamond\mathrm{l})}{\Box U_N, \Diamond(U_N^{\perp}) \vdash}\,(\mathrm{brac{-}l})}{(((N \multimap S_1) \multimap \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2)) \multimap \Box U_N \vdash \Gamma}\,(\multimap\mathrm{l})$$

FIGURE 2. Proof tree in Linear Logic.

$$\begin{aligned}
\Gamma &= \{N^{\perp}, S_1^{\perp}, \Diamond(U_M^{\perp}), H_1^{\perp}, H_2^{\perp}, H_3^{\perp}, H_4^{\perp}, H_5^{\perp}, S_2^{\perp}, \Box(U_N^{\perp})\}\\
\Gamma_1 &= \{N^{\perp}\}\\
\Gamma_2 &= \{S_1^{\perp}\}\\
\Gamma_3 &= \{U_M^{\perp}\}\\
\Gamma_4 &= \{N^{\perp}, S_1^{\perp}\}\\
\Gamma_5 &= \{\Diamond(U_M^{\perp})\}\\
\Gamma_6 &= \{N^{\perp}, S_1^{\perp}, \Diamond(U_M^{\perp}\}\\
\Gamma_7 &= \{H_1^{\perp}, H_2^{\perp}, H_3^{\perp}, H_4^{\perp}, H_5^{\perp}\}\\
\Gamma_8 &= \{S_2^{\perp}\}\\
\Gamma_9 &= \{H_1^{\perp}, H_2^{\perp}, H_3^{\perp}, H_4^{\perp}, H_5^{\perp}, S_2^{\perp}\}\\
\Gamma_{10} &= \{N^{\perp}, S_1^{\perp}, \Diamond(U_M^{\perp}), H_1^{\perp}, H_2^{\perp}, H_3^{\perp}, H_4^{\perp}, H_5^{\perp}, S_2^{\perp}\}
\end{aligned}$$

FIGURE 3. Contexts for the proof tree (2)

them, we can obtain dual view between the two participants: in our case, the attacker and the environment.

TABLE 1. De Morgan's rules

$1^{\perp}$	$\equiv_{dm1}$	$\perp$
$\perp^{\perp}$	$\equiv_{dm2}$	1
$(\varphi^{\perp})^{\perp}$	$\equiv_{dm3}$	φ
$(\varphi \otimes \psi)^{\perp}$	$\equiv_{dm4}$	$\varphi^{\perp} \parr \psi^{\perp}$
$(\varphi \parr \psi)^{\perp}$	$\equiv_{dm5}$	$\varphi^{\perp} \otimes \psi^{\perp}$
$(\varphi \otimes \psi)$	$\equiv_{dm6}$	$(\varphi^{\perp} \parr \psi^{\perp})^{\perp}$
$\varphi \multimap \psi$	$\equiv_{dm7}$	$\varphi^{\perp} \parr \psi$
$\varphi \parr \psi$	$\equiv_{dm8}$	$(\varphi^{\perp} \otimes \psi^{\perp})^{\perp}$
$(\Diamond\ \varphi)^{\perp}$	$\equiv_{dm9}$	$\Diamond\ (\varphi)^{\perp}$
$(\Box\ \varphi)^{\perp}$	$\equiv_{dm10}$	$\Box\ (\varphi)^{\perp}$

To achieve this, we must use the following De Morgan rules (see Table 1 above) to the original formula (3).

$$(((\underline{N \multimap S_1}) \multimap \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2)) \multimap \Box U_N \equiv_{dm7}$$

$$\equiv_{dm7} ((\underline{(\mathrm{N}^{\perp} \parr S_1) \multimap \Diamond U_M}) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2)) \multimap \Box U_N \equiv_{dm7}$$

$$\equiv_{dm7} (((\mathrm{N}^{\perp} \parr S_1)^{\perp} \parr \Diamond U_M) \otimes (\underline{((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5) \multimap S_2})) \multimap \Box U_N \equiv_{dm7}$$

$$\equiv_{dm7} (\underline{((\mathrm{N}^{\perp} \parr S_1)^{\perp} \parr \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2)) \multimap \Box U_N} \equiv_{dm7}$$

$$\equiv_{dm7} ((\underline{(\mathrm{N}^{\perp} \parr S_1)}^{\perp} \parr \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm5}$$

$$\equiv_{dm5} (((((\mathrm{N}^{\perp}\underline{)^{\perp}} \otimes S_1^{\perp})^{\perp})^{\perp} \parr \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm5} ((((\mathrm{N} \otimes S_1^{\perp})\underline{{}^{\perp})^{\perp}} \parr \Diamond U_M) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} ((\underline{(\mathrm{N} \otimes S_1^{\perp}) \parr \Diamond U_M}) \otimes (((((H_1 \otimes H_2) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm8}$$

$$\equiv_{dm8} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes ((((\underline{(H_1 \otimes H_2}) \otimes H_3) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm6}$$

$$\equiv_{dm6} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes ((((\underline{(H_1^{\perp} \parr H_2^{\perp})^{\perp} \otimes H_3}) \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm6}$$

$$\equiv_{dm6} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes ((((((H_1^{\perp} \parr H_2^{\perp})\underline{{}^{\perp})^{\perp}} \parr H_3^{\perp})^{\perp} \otimes H_4) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes (((\underline{((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp})^{\perp} \otimes H_4}) \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm6}$$

$$\equiv_{dm6} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes ((((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp})\underline{{}^{\perp})^{\perp}} \parr H_4^{\perp})^{\perp} \otimes H_5)^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes (\underline{((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp})^{\perp} \otimes H_5})^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm6}$$

$$\equiv_{dm6} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes (((((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp})\underline{{}^{\perp})^{\perp}} \parr H_5^{\perp})^{\perp})^{\perp} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} (((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes ((((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp})\underline{{}^{\perp})^{\perp}} \parr S_2))^{\perp} \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} (\underline{((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})^{\perp} \otimes (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)})^{\perp} \parr \Box U_N \equiv_{dm4}$$

$$\equiv_{dm4} ((((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp})\underline{{}^{\perp})^{\perp}} \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp}) \parr \Box U_N \equiv_{dm3}$$

$$\equiv_{dm3} (\underline{((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\Diamond U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp}) \parr \Box U_N} \equiv_{dm8}$$

$$\equiv_{dm8} ((((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes (\underline{\Diamond U_M)^{\perp}}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})^{\perp} \otimes (\Box U_N)^{\perp})^{\perp} \equiv_{dm9}$$

$$\equiv_{dm9} ((((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes \Box(U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})^{\perp} \otimes (\underline{\Box U_N)^{\perp}})^{\perp} \equiv_{dm10}$$

$$\equiv_{dm10} ((((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes \Box(U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})^{\perp} \otimes \Diamond(U_N)^{\perp})^{\perp}$$

FIGURE 4. De Morganized formula.

In the Figure (4), we have translated the formula (3) in Modal linear logic *(from the attacker point of view)* to the De Morganized one *(from the network environment point of view)*. In every step of the formula translation, we underline the appropriate part, where a particular De Morgan's law was applied. Later, we construct a polarized proof tree (see Figure 5), where the root of tree is De Morganized formula and every derivation step is realized by using an appropriate rule applied to obtain a new proof instance.

3.3. **Logical space and logical time.** To successfully express the logical space and time, it is necessary to identify changes in the polarity within the proof tree. Logical connectives are divided into the two groups depending on their polarity. The positive connectives are $\otimes$, 1, $\Box$ and the connectives $\parr$, $\perp$, $\Diamond$, are the negative ones.

The polarity of formula depends on its outermost connective [11]. Change of polarity within the proof tree instance of a linear formula characterizes an incrementation of time. The steps of a proof with the same polarity can be enclosed into a cluster [11]. The actions in a cluster can be performed simultaneously. Application of the negation rule is causing a leaping of an appropriate formula or its subformula between left and right side of the turnstile. It occurs when a new cluster of the same polarity passes. To achieve this, a polarized proof tree must be created with the root of De Morganized formula as shown in the Figure (4). At the end all the leaves of the proof tree have to be identities.

$$\begin{array}{cl}
\dfrac{\dfrac{}{\vdash N, \Gamma_1}\,(\text{id}) \qquad \dfrac{}{\vdash S_1, U_M, \Gamma_2}\,(\text{id})}{\vdash N \otimes S_1^\perp, U_M, \Gamma_3} & (\otimes\text{r}) \\
\hline \vdash N \otimes S_1^\perp, U_M, \Gamma_3 & (\otimes\text{r}) \\
\hline \vdash (N \otimes S_1^\perp), U_M, \Gamma_3 & (\text{brac}-\text{r}) \\
\hline (\mathrm{N} \otimes S_1^\perp)^\perp \vdash U_M, \Gamma_3 & (()_1^\perp) \\
\hline (\mathrm{N} \otimes S_1^\perp)^\perp \vdash (U_M), \Gamma_3 & (\text{brac}-\text{r}) \\
\hline (\mathrm{N} \otimes S_1^\perp)^\perp \vdash \Box(U_M), \Gamma_3 & (\Box\text{r}) \\
\hline (\mathrm{N} \otimes S_1^\perp)^\perp, \Box(U_M)^\perp \vdash \Gamma_3 & (()_1^\perp) \\
\hline (\mathrm{N} \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp \vdash \Gamma_3 & (\otimes\text{l}) \\
\hline ((\mathrm{N} \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \vdash \Gamma_3 & (\text{brac}-\text{l})
\end{array}$$

$$\begin{array}{cl}
\hline \vdash H_1^\perp, H_2^\perp, H_3^\perp, H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\text{id}) \\
\hline \vdash H_1^\perp \parr H_2^\perp, H_3^\perp, H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\parr\text{r}) \\
\hline \vdash (H_1^\perp \parr H_2^\perp), H_3^\perp, H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\text{brac}-\text{r}) \\
\hline \vdash (H_1^\perp \parr H_2^\perp) \parr H_3^\perp, H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\parr\text{r}) \\
\hline \vdash ((H_1^\perp \parr H_2^\perp) \parr H_3^\perp), H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\text{brac}-\text{r}) \\
\hline \vdash ((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp, H_5^\perp, S_2, \Gamma_4 & (\parr\text{r}) \\
\hline \vdash (((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp), H_5^\perp, S_2, \Gamma_4 & (\text{brac}-\text{r}) \\
\hline \vdash (((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp, S_2, \Gamma_4 & (\parr\text{r}) \\
\hline \vdash ((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp), S_2, \Gamma_4 & (\text{brac}-\text{r}) \\
\hline \vdash ((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2, \Gamma_4 & (\parr\text{r}) \\
\hline \vdash (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2), \Gamma_4 & (\text{brac}-\text{r}) \\
\hline (((((\mathrm{H}_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp \vdash \Gamma_4 & (()_1^\perp)
\end{array}$$

$$\begin{array}{cl}
\hline ((\mathrm{N} \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp \vdash \Gamma_5 & (\parr\text{l}) \\
\hline (((\mathrm{N} \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp) \vdash \Gamma_5 & (\text{brac}-\text{l}) \\
\hline \vdash (((N \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp)^\perp, \Gamma_5 & (()_r^\perp)
\end{array}$$

$$\begin{array}{cl}
\hline \mathrm{U}_N \vdash \Gamma_6 & (\text{id}) \\
\hline (\mathrm{U}_N) \vdash \Gamma_6 & (\text{brac}-\text{l}) \\
\hline \Diamond(U_N) \vdash \Gamma_6 & (\Diamond\text{l}) \\
\hline \vdash \Diamond(U_N)^\perp, \Gamma_6 & (()_r^\perp)
\end{array}$$

$$\begin{array}{cl}
\hline \vdash (((N \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp)^\perp \otimes \Diamond(U_N)^\perp, \Gamma & (\otimes\text{r}) \\
\hline \vdash ((((N \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp)^\perp \otimes \Diamond(U_N)^\perp), \Gamma & (\text{brac}-\text{r}) \\
\hline ((((\mathrm{N} \otimes S_1^\perp)^\perp \otimes \Box(U_M)^\perp) \parr (((((H_1^\perp \parr H_2^\perp) \parr H_3^\perp) \parr H_4^\perp) \parr H_5^\perp) \parr S_2)^\perp)^\perp \otimes \Diamond(U_N)^\perp)^\perp \vdash \Gamma & (()_1^\perp)
\end{array}$$

FIGURE 5. Polarized proof tree.

where the contexts are defined as follows:

$$\begin{aligned}
&\Gamma = \{N^\perp, S_1, U_M^\perp, H_1, H_2, H_3, H_4, H_5, S_2^\perp, U_N^\perp\} \\
&\Gamma_1 = \{N^\perp\} \\
&\Gamma_2 = \{S_1^\perp, U_M^\perp\} \\
&\Gamma_3 = \{N^\perp, S_1, U_M^\perp\} \\
&\Gamma_4 = \{H_1, H_2, H_3, H_4, H_5, S_2^\perp\} \\
&\Gamma_5 = \{N^\perp, S_1, U_M^\perp, H_1, H_2, H_3, H_4, H_5, S_2^\perp\} \\
&\Gamma_6 = \{U_N^\perp\}
\end{aligned}$$

FIGURE 6. Contexts for the polarized proof tree.

In the polarized proof tree Figure (5), we observe the change of polarity. We identify the clusters of polarities determined by the linear negation rule. Where it is applied, a proof step expresses a time incrementation. The actions enclosed in clusters can be performed simultaneously. The time incrementation reflects the fact, that the use of the negation rule causes tilting of the corresponding formula between the right and left sides of the turnstile. The following proof tree Figure (7) is constructed from its clusters. Proof trees with clusters are not only simpler but also indicate the time incrementation. The cluster proof tree (depicted in the Figure (7)), is derived from the polarized proof tree (depicted in the Figure (5)) in such a way, that it contains only those tree forms where the rule of linear negation was used.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash E,\, \Lambda_3}(\maltese) \quad \dfrac{}{\vdash F,\, \Lambda_4}(\maltese)}{\mathrm{C} \vdash \Lambda_1}(-,\mathrm{C}\vdash,\{\{\mathrm{E}\},\{\mathrm{F}\}\}) \qquad \dfrac{}{\mathrm{D} \vdash \Lambda_2}(\maltese)}{\vdash B,\, \Lambda}(+,\vdash\mathrm{B},\{\{\mathrm{C}\},\{\mathrm{D}\}\})}{\mathrm{A} \vdash \Lambda}(-,\mathrm{A}\vdash,\{\mathrm{B}\})$$

FIGURE 7. Cluster proof tree.

Appropriate contexts are depicted in the Figure (8).

$$A = ((((\mathrm{N} \otimes S_1^{\perp})^{\perp} \otimes \Box(U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})^{\perp} \otimes \Diamond(U_N)^{\perp})^{\perp}$$

$$\mathrm{B} = ((((N \otimes S_1^{\perp})^{\perp} \otimes \Box(U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})^{\perp} \otimes \Diamond(U_N)^{\perp})$$

$$\mathrm{C} = (((N \otimes S_1^{\perp})^{\perp} \otimes \Box(U_M)^{\perp}) \parr (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)^{\perp})$$

$$\mathrm{D} = \Diamond(U_N)$$

$$\mathrm{E} = (N \otimes S_1^{\perp}),\ U_M$$

$$\mathrm{F} = (((((H_1^{\perp} \parr H_2^{\perp}) \parr H_3^{\perp}) \parr H_4^{\perp}) \parr H_5^{\perp}) \parr S_2)$$

FIGURE 8. Cluster proof tree contexts.

In the linear logic, we consider a space in terms of locations. Every formula has a location, i.e. its address [11]. Based on that, we remove the content of subformulæ, and we replace it by its locations. Proof trees containing only locations are called designs, where any logical information about the original subformula is substituted by appropriate locative addresses, i.e. loci in the design (Figure 9).

The following rules are used in the process of constructing a proof tree in the time-spatial Ludics theory.

- *Positive rule* is used when the outermost formula has positive polarity,

$$\frac{\ldots, \xi * i, \ldots \ \vdash \Lambda_i, \ldots}{\vdash \Lambda, \xi}(+,\xi,I), \tag{4}$$

  where the ξ is the address of a formula, and for every $i \in I$, and the Λ_i is set of addresses of every immediate subformulas.
- *Negative rule* is used when the outermost formula has negative polarity:

$$\frac{\ldots \ \vdash \Lambda_I, \xi * I, \ldots}{\xi \vdash \Lambda}(-,\xi,N), \tag{5}$$

  where the N is set of ramifications, where for the every $I \in N$ holds, that $\xi * I$.
- The *daemon* rule is used otherwise, mostly in the leaves:

$$\frac{}{\vdash \Lambda, \xi}(\maltese). \tag{6}$$

A location of proved formula in the design is denoted by ξ, where ξ is the location address. If the formula has its immediate subformula ξ_1, their locations are called biases (the bias Λ_1 or the concentrated biases Λ_{11}, Λ_{12} etc.). The structure of space occupied by a formula is uniquely identified by a finite sequence of biases [11].

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \xi_{111}, \Lambda_{111}}(\maltese) \quad \dfrac{}{\vdash \xi_{112}, \Lambda_{112}}(\maltese)}{\xi_{11} \vdash \Lambda_{11}}(-,\xi_1\vdash,\{\{1\},\{2\}\}) \qquad \dfrac{}{\xi_{12} \vdash \Lambda_{12}}(\maltese)}{\vdash \xi_1, \Lambda_1}(+,\vdash\xi_1,\{\{1\},\{2\}\})}{\xi \vdash \Lambda}(-,\xi\vdash,\{1\})$$

FIGURE 9. Design.

Appropriate contexts are depicted below.

$$\begin{aligned}
\Delta &= \xi \\
\Delta_1 &= \xi_1 \\
\Delta_{11} &= \xi_{11} \\
\Delta_{12} &= \xi_{12} \\
\Delta_{111} &= \xi_{111} \\
\Delta_{112} &= \xi_{112}
\end{aligned}$$

Finally, we obtain the design on the network attack as shown in Figure (9) for expressing the locative structure of the network intrusion. Designs are the significant objects of the Ludics theory. The design in the Figure (9) consists of the three time lines of comparable loci with respect to ordering relation $\sqsubseteq$.

(1) $\xi \sqsubseteq \xi_1 \sqsubseteq \xi_{11} \sqsubseteq \xi_{111}$, represents the linear time line of the possibility of the vertical portscan intrusion,
(2) $\xi \sqsubseteq \xi_1 \sqsubseteq \xi_{11} \sqsubseteq \xi_{112}$, represents the linear time line of necessity of the SYN Flood network attack,
(3) $\xi \sqsubseteq \xi_1 \sqsubseteq \xi_{12}$, represents the linear time line of neccessity of network intrusion.

A design can have one or more branches and it expresses two relationships [11]: time and space. The addresses in the same branch of design are comparable addresses and they have time relationship. The addresses in different branches are incomparable, i.e. they have space relationship in order to relation $\sqsubseteq$.

We can also interpret this design as the polarized game, where the linear negation is conductive to move alternation between the proponent (attacker) and the opponent (network environment). From the computer science point of view, we were able to express logical space that represents the computer memory and also logical time, which represents the calculation of computer processor.

4. Conclusion

In this contribution, we show how the resource-oriented logical system can be used to describe real processes in network environment, such as network intrusion. We have expressed IDS's behavior during network intrusion by a formula of Modal Linear Logic. Our method is helpful that proof of such a formula ensures the correctness of component software system design.

The main goal of this paper is to apply the time-spatial calculus from Girard's Ludics theory. Finally, we were able to express logical space that represents the computer memory and also logical time, which represents the calculation time of operation.

In the future, we would like to extend our approach by joining the host-based intrusion detection systems with the network-based one i.e. create a complex security of program systems. Such a combination of the both types of IDSs will secure computer systems even more. The next step in our work, will be extending IDS by applying the resource oriented Belief-Desire-Intention logical system. We plan to create a BDI architecture, that will perform automated IDSs reactions, instead of a system administrator intervention as it is now.

Acknowledgment

This paper was supported by KEGA project ViLMA: Virtual Laboratory for Malware Analysis (079TUKE-04/2017).

This work is a result of international cooperation under the CEEPUS network No.CIII-HU-0019-12-1617.

References

[1] J.-Y. Girard. *Linear Logic*, Theoretical Computer Science 50, Elsevier Science Publishers Ltd. Essex, UK, 1987

[2] J.-Y. Girard. *Linear Logic: its syntax and semantics*, Laboratoire de Mathematiques Descretes, UPR 9016 - CRNS, 1995

[3] J.-Y. Girard. *Locus Solum: From the rules of logic to the logic of rules*, Mathematical Structures in Computer Science, Vol. 11, N. 3, 2001

[4] P. Innella, O. McMillan, T. Digital Integrity, LLC. *An Introduction to IDS*, Symantec Connect, 2001

[5] J. Perháč, D. Mihályi. *Intrusion Detection System Behavior as Resource-Oriented Formula*, Acta Electrotechnica et Informatica, Vol. 15, No. 3, 2015

[6] SANS Institute. *Intrusion Detection Systems: Definition, Need and Challenges", SANS Institute Reading Room*, 2011

[7] J. Perháč, D. Mihályi, V. Novitzká, *Between Syntax and Semantics of Resource Oriented Logic for IDS Behavior Description*, The Publishing Office of Czestochowa University of Technology, Journal of Applied Mathematics and Computational Mechanics, Vol. 15, No. 2, ISSN:2353-0588, 2016

[8] J. Perháč, D. Mihályi, *Coalgebraic modeling of IDS behavior*, 2015 IEEE 13th International Scientific Conference on Informatics, November 18-20, 2015, Poprad, Slovakia, 2015

[9] J. Perháč, D. Mihályi, *Coalgebraic specification of network intrusion signatures*, Studia Universitatis Babes-Bolyai, Informatica, Vol. 61, N. 2 pp. 83-94, 2016

[10] J. Perháč, D. Mihályi, *Intrusion Detection System Behavior as Resource-Oriented Formula*, Acta Electrotechnica et Informatica. Vol. 15, N. 3, 2015, pp. 9-13. ISSN 1335-8243

[11] W. Steingartner, A. Poláková, P. Prazňák, V. Novitzká, *Linear Logic in Computer Science*, Journal of Applied Mathematics and Computational Mechanics, Vol. 14(1), 91-100, 2015

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic,

E-mail address: `Daniel.Mihalyi@tuke.sk`

E-mail address: `Jan.Perhac@tuke.sk`

E-mail address: `Patrik.Balint@globallogic.com`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.04

ROEMOLEX - A ROMANIAN EMOTION LEXICON

ANAMARIA BRICIU AND MIHAIELA LUPEA

Abstract. In Natural Language Processing tasks, semantical and lexical resources are of paramount importance for efficient implementations of solutions. However, availability of tools for any other language than English is fairly limited, therefore leaving the field open to improvements and new developments. This paper presents the second version of RoEmoLex, a lexicon containing approximately eleven thousand Romanian words tagged with a series of emotions and two valences. We describe the steps followed in the improvement process of the first version of the resource: the addition of new terms, and the extension of emotional concepts to finer-grained tags.

1. Introduction

Semantical and lexical resources are widely used in natural language processing tasks, either as supports of knowledge-based methods, or as aids for hybrid techniques involving machine learning. Hand-crafted lexicons, exhaustive thesauri or semantic tools (e.g. FrameNet) add an element of language understanding to traditional statistical models, generally improving performance in tasks like sentiment analysis, word sense disambiguation or document summarization.

However, there are relatively few resources designed for languages other than English, which makes it difficult to implement viable solutions for the analysis of non-English content. As far as the Romanian language is concerned, the existence of RoWordNet [15], a large semantic network that encompasses information in the form of synsets (i.e. groups of full synonyms) linked by lexical-semantic relations, marks the central point of Romanian language processing research due to its size, completeness, and information richness. Some

Received by the editors: November 2, 2017.

2010 *Mathematics Subject Classification.* 68T50, 91F20.

1998 *CR Categories and Descriptors.* I.2.7. [**Computing Methodologies**]: Artificial Intelligence - *Natural Language Processing*; H.3.1. [**Information Systems**]: Information Storage and Retrieval - *Content Analysis and Indexing*.

Key words and phrases. natural language processing, linguistic processing, emotion analysis, lexicon.

aligned corpora for machine translation, as well as a number of small-scale emotion lexicons have also been developed, which is encouraging, but leaves the field open to exploration and expansion. This was our motivation for taking a translated version of the lexicon proposed by Mohammad and Turney in [7], processing the existing data and bringing a series of modifications and additions to it in order to create a dependable Romanian emotion lexicon.

This work discusses the continuation of the work presented in [6], where a series of processing and structuring steps applied to the original, automatically translated lexicon were described. Enhancements to this processed first version in the form of new term additions brought the lexicon to its current form, consisting of approximately 11000 words tagged with affect information, specifically Plutchik's [9] eight basic emotions (*Anger, Anticipation, Surprise, Joy, Trust, Fear, Sadness, Disgust*) and polarity tags (*Positivity, Negativity*), and a series of derived emotions formed through the combination of two basic emotions.

The paper is structured as follows: in Section 2, related work is presented, while in Section 3 we describe RoEmoLex, following the development process from the initial revision of the English translation to this new round of enhancements. Section 4 is reserved for conclusions and proposals for future work.

2. RELATED WORK

As relevant to the current article, there are two directions of research that need to be discussed: the available lexical and semantic resources for Romanian text, and the particularities of existent affect lexicons.

First, to address the former issue, RoWordNet [3], [15] must be introduced. This is the Romanian version of the Princeton WordNet, and it is a lexical database which allows the modeling of linguistic knowledge with regards to aspects of the real world. There is sentiment information in form of three polarity scores: *positive, objective, negative*, domain data (every term is associated with a conceptual type or domain - see SUMO, Section 3.4.), and a series of relationships that structure the data in an intuitive manner (e.g. *is a*, or *part of* relations between lexical units). Beyond RoWordNet, there are a few other resources that offer the possibility of modular integration in a more complex system (e.g dexonline[1]), but none with the applicability and richness in information that RoWordNet offers.

Secondly, in terms of resources for affect detection and recognition in text, for English, a widely used tool is LIWC [8], [13], a text analysis application with a comprehensive dictionary in which terms are marked as representative

[1]http://pydex.lemonsoftware.eu/

of a series of categories and subcategories, including Affective Processes. Other resources are DepecheMood [10] and ANEW [2]. The first, DepecheMood, is a lexicon created in an entirely automated manner by using affective annotation provided implicitly by readers of articles on a news site. The second resource, ANEW, provides a set of normative emotional ratings for a series of words. It is important to note that, in contrast to our lexicon, in the case of ANEW, emotion is viewed as a space in three dimensions [5], pleasure, arousal and dominance, rather than as represented by fixed atomic units in the form of basic emotions from which other emotions are formed.

More relevant to the current work are EmoLex [7], which will be described in detail in Section 3.1, as a starting point of our work, and the multilingual WordNet-Affect project described in [1]. The latter represents an aligned English-Russian-Romanian affect lexicon, considering six emotions (*Anger, Disgust, Fear, Joy, Sadness, Surprise*) and the corresponding synsets present in WordNet-Affect. It is a carefully crafted resource, with valuable emotional information that we integrated in RoEmoLex. Details about this process can be found in Section 3.3.

3. ROEMOLEX – ROMANIAN EMOTION LEXICON

The starting point of this research is a lexicon proposed by Mohammad and Turney in 2010, the NRC Word-Emotion Association Lexicon or EmoLex [7], consisting of a series of English words and their associations with Plutchik's eight emotions and two polarity tags. We used the automatically translated Romanian version of this lexicon, with the proposed improvements consisting in removing non-emotive words (i.e. words with an overall emotion and polarity score equal to 0), eliminating duplicate lines, and adding RoWordNet information for each word. Finally, we added new terms to the lexicon, aiming to build a more comprehensive resource.

3.1. **Original lexicon.** The NRC Word-Emotion Association Lexicon [7], or EmoLex, was created by crowdsourcing the task through Amazon's Mechanical Turk. Hundreds of online annotators were given a series of words and asked which of eight emotions (*Anger, Anticipation, Surprise, Joy, Trust, Fear, Sadness, Disgust*) and two valences (*Positivity, Negativity*) the term expresses, and to what extent it does (*none, weak, moderate, strong*). Then, based on the annotators' feedback, the authors considered the number of assignments into each intensity category for a word-emotion pair. If there were more appearances of *no*, or *weak* intensity, it was considered that the word did not evoke the emotion in question. Alternatively, if the majority of responses said *moderate* or *strong* intensity, the word was tagged with the specified emotion. The resulting lexicon contained 2000 words tagged with eight emotions and sentiment valences, but was later expanded to 14182 words and 25000 senses for English.

Through automatic translation, EmoLex has been made available for over twenty languages, including Romanian. English words are converted using Google Translate, which maintains the emotional correspondence to a relatively high extent [14], but, as is to be expected, also introduces noisy data, for which a series of post-processing steps are required.

3.2. **Post-processing of translated data.** Initially, the Romanian version of the lexicon consisted of 8581 lines (i.e. words). However, many of these were null lines, i.e. terms expressing no sentiment valence or emotion. These were *non-evocative* terms, which we considered to be of little use in further emotion analysis applications. Therefore, we discarded them along with any duplicate entries. Note that the notion of duplicate entry is defined as the same term having two entries in the lexicon with all polarity and emotion tags equal. A series of identical terms have remained in the lexicon, as they have different scores, and account for polysemantic words.

After these steps, we were left with 3989 unique lines, which we aligned with RoWordNet by means of assigning each term to the corresponding synset (where possible). Using this synset information, we introduced part of speech, positive, negative and objective SentiWordNet scores, and SUMO categories for each term. More details regarding these processing steps, and about the first version of RoEmoLex, respectively, can be found in [6].

In the current paper, we introduce a new structural addition in the form of affective annotation for primary, secondary and tertiary emotions, as theorized by Plutchik in the form of dyads.

In his general psychoevolutionary theory of emotion, Plutchik identified eight basic, biologically primitive emotions: *Anger*, *Anticipation*, *Disgust*, *Fear*, *Joy*, *Sadness*, *Surprise*, *Trust* and formulated ten postulates to characterize his theory. Of these, of relevance to the current work is the sixth, which states that beyond these eight, all other emotions are mixed or derivative states; that is, they occur as combinations, mixtures, or compounds of the primary emotions [9]. Considering this postulate, we included tags for a series of secondary emotions defined by the author as results of meaningful combinations. Utilizing the Wheel Of Emotions shown in Figure 1 as an expressive visual representation of Plutchik's theory, the following combinations of two basic emotions can be identified:

- **Primary dyads** (emotions combined are one petal apart): *Optimism* (Anticipation and Joy), *Love* (Joy and Trust), *Submission* (Trust and Fear), *Awe* (Fear and Surprise), *Disapproval* (Surprise

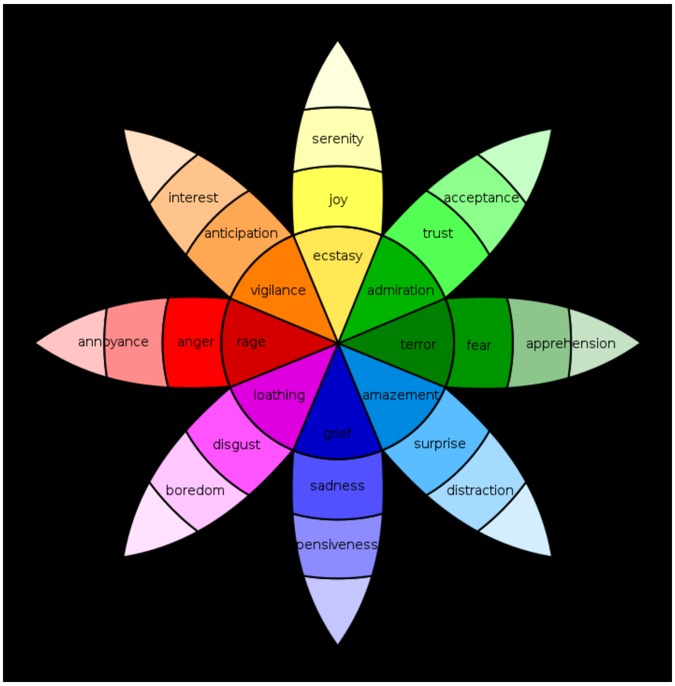


FIGURE 1. Plutchik's Wheel Of Emotions

and Sadness), *Remorse* (Sadness and Disgust), *Contempt* (Disgust and Anger), *Aggressiveness* (Anger and Anticipation)

- **Secondary dyads** (emotions combined are two petals apart): *Hope* (Anticipation and Trust), *Guilt* (Joy and Fear), *Curiosity* (Trust and Surprise), *Despair* (Fear and Sadness), *Disbelief* (Surprise and Disgust), *Envy* (Sadness and Anger), *Cynicism* (Disgust and Anticipation), *Pride* (Anger and Joy)
- **Tertiary diads** (emotions combined are three petals apart): *Anxiety* (Anticipation and Fear), *Delight* (Joy and Surprise), *Sentimentality* (Trust and Sadness), *Shame* (Fear and Disgust), *Outrage* (Surprise and Anger), *Pessimism* (Sadness and Anticipation), *Morbidness* (Disgust and Joy), *Dominance* (Anger and Trust)

Therefore, for every word having both affective tags from a pair, we considered the term to express the emotion resulted from their combination, and tagged them in our lexicon accordingly. Table 1 shows a series of representative words for a few categories.

Emotion	**Terms**
Optimism	credință/**faith** căsătorie/**marriage**
Aggressiveness	ghilotină/**guillotine** duel/**duel**
Hope	a aspira (la)/**to aspire** determinat/**determined** obiectiv/**objective**
Despair	demoralizat/**demoralized** chin/**anguish** cimitir/**cemetery**
Anxiety	anxietate/**anxiety** diagnostic/**diagnostic** vigilență/**vigilence**
Pessimism	condamnare/**condemnation** îngrozitor/**awful** moarte/**death**

Table 1. Examples of words associated with derived emotions

3.3. **Lexicon enrichment.** A final step in the process of improving RoEmoLex was the addition of new words. The expansion of the lexicon was done in two steps: the addition of synonyms of existent entries from RoWordNet, and integration of the data present in the similar resource *Russian-Romanian WordNet-Affect.*

3.3.1. *Lexicon enrichment through addition of RoWordNet synonyms.* The first proposed enrichment method was the addition of synonyms of existent entries. We considered these new terms as having the same emotional and polarity content as the original terms, and we thus assigned them the same emotion and valence tags as the original words. This was done with the help of RoWordNet, taking as *synonyms* all words in the same synset with the entry present in RoEmoLex. After this step, we obtained an additional 6455 entries.

Despite the fact that the lexicon almost tripled in size, the part of speech hierarchy was preserved, as can be seen in Table 2, with nouns and adjectives still accounting for the majority, but with the percent of verbs and idioms each rising about 4 percent. This is due to the fact that verbs represented the terms with most synonyms per entry (e.g. a vorbi - **to talk**: a discuta - **to discuss**, a grăi - **to speak**, a conversa - **to converse**, a dialoga; a sublinia - **to emphasize**: a accentua - **to accentuate**, a reliefa, a puncta), and quite a

Part of Speech	**% of lexicon in initial lexicon**	**% of lexicon after 1st round of enrichment**	**% of lexicon after 2nd round of enrichment**
	3989 words	10444 words	11051
Noun	57.48% — 2293 words	53.27% — 5563 words	51.841%— 5729 words
Adjective	23.41% — 933 words	20.22% — 2112 words	20.76% — 2295 words
Verb	12.86% — 513 words	16.42% — 1715 words	16.40% — 1813 words
Idiom	3.50% — 140 words	7.02% — 734 words	7.94% — 878 words
Adverb	2.65% — 106 words	3.03% — 316 words	3.00%— 332 words
Interjection	0.1% — 4 words	0.04% — 4 words	0.03%— 4 words

Table 2. Part of Speech Distribution

few of the synonyms added were, in fact, phrases and idioms (e.g.start - **start**: semnal de start - **start signal**) as opposed to terms with the same part of speech as the original word.

3.3.2. *Lexicon enrichment through integration of data from Romanian-Russian WordNet-Affect.* Another way to enrich the lexicon was the addition of terms from the Russian and Romanian WordNet-Affect [1]. This resource consists of six lists of representative synsets organized according to Ekman's [4] six emotions: *Anger, Disgust, Fear, Joy, Sadness, Surprise.* The motivation in integrating the Russian-Romanian WordNet-Affect data in our lexicon was the improvement of RoEmoLex through the addition of carefully annotated data with the starting point in WordNet-Affect [12], a dependable resource for affective computing.

	# of synsets	**# of Romanian words**
Anger	117	330
Disgust	17	60
Fear	80	248
Joy	209	641
Sadness	98	364
Surprise	27	87

Table 3. Russian-Romanian WordNet-Affect Data

The development of the Russian-Romanian WordNet-Affect started from a set of English synsets annotated for each of the six emotions and made available for the SemEval-2007 task "Affective Text" [11]. The authors then proceeded with automatic translation of the synsets from English, manually correcting any inconsistencies and checking the data, and furthermore adding any relevant synonyms of the generated Romanian terms. Statistics concerning the final form of the Russian-Romanian WordNet-Affect can be viewed in Table 3.

Emotion	**% of common words**
Anger	46.86%
Disgust	39.53%
Fear	42.85%
Joy	53.125%
Sadness	38.01%
Surprise	60.93%

Table 4. Percent of common data between RoEmoLex and Russian-Romanian WordNet-Affect

We compared this data with RoEmoLex content, hoping to discover sets of new words to add to our lexicon. As it can be seen in Table 4, the percent of common data varies little among emotions, which points to a balanced understanding of emotion in RoEmoLex. *Surprise* is the emotion with the most common words, while *Sadness* is the one with the fewest. It is interesting to note that a significant portion of the terms that were not in RoEmoLex, irrespective of emotion, were idioms and expressions (e.g. lipsit de veselie - **devoid of joy**, din nefericire - **unfortunately**, cu părere de rău - **with regret** for *Sadness*, lua pe neașteptate - **to catch someone off guard**, pune în încurcătură - **to discomfit** for Surprise). Another reason for the relatively high discrepancy is that the Russian-Romanian WordNet-Affect accounts for multiple spellings of a word. For example, in the multilingual resource, *a mâhni* - ***to dishearten*** can be found in both this form and *a mîhni*, a more infrequent spelling, typically encountered in older texts, while in RoEmoLex only the first form is present. We chose to include such terms in the lexicon, but, where existent, the emotional and polarity content of the term with the more common spelling are duplicated for the word with the less frequent form.

As for the degree of annotation agreement between the two resources, Table 5 shows that *Disgust*, *Fear* and *Sadness* are the emotions most consistently tagged, with more than 70% of the common words having the same emotion tag in both resources. For terms with conflicting tags, we did a manual verification and validation of the annotation in our lexicon.

Going back to Table 2, it can be seen that this second round of additions brought an increase in the number of adjectives and idioms in the RoEmoLex, with noun and verb percentages decreasing slightly. This is owed to the data structure of the Russian-Romanian WordNet-Affect, with a large amount of

Emotion	**% of words with matching tags**
Anger	69.29%
Disgust	82.35%
Fear	76%
Joy	56.37%
Sadness	77.17%
Surprise	48.71%

Table 5. Matching tags in Russian-Romanian WordNet-Affect and RoEmoLex

adjective synsets present, and the manual inclusion of translations (inimă grea - **heavy heart**, îndoială de sine - **self-doubt**).

In total, we acquired 607 new lexicon entries, with *Anger, Disgust, Fear, Joy, Surprise* and *Sadness* tags from the Russian-Romanian WordNet-Affect lexicon, and *Positivity, Negativity, Anticipation* and *Trust* manually added. This put the final number of terms in RoEmoLex at just over 11000.

3.4. **Lexicon samples.** In this section, we include a series of representative RoEmoLex entries for a better understanding of the structure and data format within the lexicon. We note that the presented table limits itself to introducing the tags for the eight basic emotions for reasons of space, and does not introduce the derived emotions presented in Section 3.2. For examples of terms corresponding to these derived tags, please refer to Table 1.

In Table 6, the **P** and **N** table headers refer to the *Positivity* and *Negativity* tag, respectively. **POS** stands for *Part of Speech* information. **SUMO** is an abbreviation for Suggested Upper Merged Ontology, and is a field mapped from RoWordNet designed to offer a semantic context to the term in question.

4. CONCLUSIONS AND FURTHER WORK

In this paper, we presented the development process of a new version of RoEmoLex, namely the enhancements we proposed in order to create a comprehensive emotion analysis resource for Romanian texts. We described the original lexicon, EmoLex, from which RoEmoLex was translated, briefly presenting the initial round of data processing. Finally, we outlined the mechanics of two rounds of lexicon enrichment, presenting a series of statistics and some sample entries to illustrate aspects of the current form of the lexicon.

We state that RoEmoLex can constitute a viable starting point for emotion analysis in Romanian texts, but note that there is still work that can be done in terms of improving the resource. For example, the inclusion of fuzzy logics

Word	**POS**	**P**	**N**	***Anger***	***Anticipation***	***Disgust***	***Fear***	***Joy***	***Sadness***	***Surprise***	***Trust***	**SUMO**
ascuns **concealed**	Adjective	0	1	0	1	0	1	0	0	1	0	Covering
ascuns **hidden**	Adjective	0	1	0	0	0	0	0	0	0	0	Subjective Assessment Attribute
demonic **demonic**	Adjective	0	1	1	0	1	1	0	1	0	0	Subjective Assessment Attribute
vacanță **holiday**	Noun	1	0	0	1	0	0	1	0	0	0	Vacationing
autenticitate **authenticity**	Noun	1	0	0	0	0	0	0	0	0	1	True
nedreptate **injustice**	Noun	0	1	1	0	1	0	0	1	0	0	Normative Attribute
bătaie de joc **mockery**	Idiom	0	1	1	0	1	0	0	0	0	0	Expressing
liniște sufletească **hearts-ease**	Idiom	1	0	0	0	0	0	1	0	0	0	EmotionalState
abia **barely**	Adverb	0	1	0	0	0	0	0	1	0	0	Subjective Assessment Attribute
vai **oh!**	Interjection	0	1	0	0	1	1	0	1	0	0	-
a îndrăzni **to dare**	Verb	1	0	0	0	0	0	0	0	0	1	Subjective Assessment Attribute
a sugruma **to strangle**	Verb	0	1	1	0	1	1	0	1	0	0	ViolentContest

TABLE 6. RoEmoLex - Sample entries

for evaluating the degree of membership of a word to all the emotion classes would be an approach that modeled human understanding of emotion and its expression better, and it is a track for further work that we will investigate.

The importance of continuous refinement and improvement of such a resource lies in the many interesting applications that can be developed provided a dependable emotion analysis module, from the simple study of emotional content and its evolution in a given text to integrating the module into more complex systems (e.g. customized interaction in tutoring systems, mental health monitoring applications).

References

[1] Victoria Bobicev, Victoria Maxim, Tatiana Prodan, Natalia Burciu, and Victoria Angheluş. Emotions in words: Developing a multilingual wordnet-affect. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 375–384. Springer, 2010.

[2] Margaret M. Bradley, Peter J. Lang, Margaret M. Bradley, and Peter J. Lang. Affective norms for english words (anew): Instruction manual and affective ratings, 1999.

[3] Ștefan Daniel Dumitrescu. Rowordnetlib - the first api for the romanian wordnet. *Proceedings of the Romanian Academy, Series A*, 16(1):87–94, 2015.

[4] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*, chapter 18. Lexicons for Sentiment and Affect Extraction.

[6] Mihaiela Lupea and Anamaria Briciu. Formal concept analysis of a romanian emotion lexicon. In *Proceedings of the 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 111–118, 2017.

[7] Saif M Mohammad and Peter D Turney. Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 26–34. Association for Computational Linguistics, 2010.

[8] James W Pennebaker, Roger J Booth, and Martha E Francis. Linguistic inquiry and word count: Liwc [computer software]. *Austin, TX: liwc. net*, 2007.

[9] Robert Plutchik. A general psychoevolutionary theory of emotion. *Emotion: Theory, Research, and Experience*, 1:3–33, 1980.

[10] Jacopo Staiano and Marco Guerini. Depeche mood: a lexicon for emotion analysis from crowd annotated news. pages 427–433, 2014.

[11] Carlo Strapparava and Rada Mihalcea. Learning to identify emotions in text. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1556–1560, New York, NY, USA, 2008.

[12] Carlo Strapparava and Alessandro Valitutti. Wordnet-affect: an affective extension of wordnet. In *In Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 1083–1086, 2004.

[13] Yla R. Tausczik and James W. Pennebaker. The psychological meaning of words: Liwc and computerized text analysis methods. *Journal of Language and Social Psychology*, 29(1):24–54, 2010.

[14] Vaibhav Tripathi, Aditya Joshi, and Pushpak Bhattacharyya. Emotion analysis from text: A survey. *Center for Indian Language Technology Surveys*, 2016.
[15] Dan Tufiș, Eduard Barbu, Verginica Barbu Mititelu, Radu Ion, and Luigi Bozianu. The romanian wordnet. *Romanian Journal of Information Science and Technology*, 7(1-2):107–124, 01 2004.

Faculty of Mathematics and Computer Science, Babeş–Bolyai University, Cluj-Napoca, Romania
E-mail address: `baic1326@scs.ubbcluj.ro`
E-mail address: `lupea@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.05

ANALYZING THE USEFULNESS OF THE USER'S BROWSER HISTORY FOR GENERATING QUERY SUGGESTIONS

IOAN BĂDĂRÎNZĂ

Abstract. A very useful feature of search engines that helps users while they browse the internet, where, they very often, try to satisfy an information need, is *query suggestion.* This mechanism shows the user a list of possible queries from where he can choose and be able to perform a search easier and faster. In this paper we tried to assess the usefulness of a user's recent web browsing history for generating new query suggestions. We performed a one month experiment in which we collected browsing history logs of several users and searched query terms submitted by those users to Google (using a Chrome plugin) and found that approximately 32% of the queries submitted can be predicted from the user's browsing history.

1. Introduction

Searching for information on the web can be very difficult sometimes. There are a lot of users that do not know what terms to enter in a search input of a search system to better describe their information need. In [8, 15] we can see that most of the search queries are very short, one or two words on average and in [9, 16] we can see that these words are ambiguous. In order to help the user when performing a search, most search engines like Google, Yahoo!, Bing and others, provide query auto-completion and query suggestions. In order to better explain how search suggestions are generated, we will first try to describe how query auto-completion works. In almost all modern browsers, search engines and text editors we can see how, after we start typing words, it automatically tries to predict what we actually want to type. These are called 'predictive auto-completion systems' where the candidates are matched against the prefix using information retrieval techniques and also Natural Language Processing techniques. This auto-completion is actually

Received by the editors: November 2, 2017.
2010 *Mathematics Subject Classification.* 68U10, 94A08.
1998 *CR Categories and Descriptors.* H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval.
Key words and phrases. query suggestion, browsing history, personalized suggestions.

the highest ranked suggestion from a suggestions list. The query suggestions list is a list that contains from eight to ten words (or group of words), which are usually prefixed with the subquery that the user is typing, items that are extracted from a huge log of queries submitted by all users. A very well known technique of extracting suggestions from a common query log is called Most Popular Completion, which we'll describe more in the next section of this paper.

The main focus of this paper is to analyse how user's personal browsing history and submitted query history are impacting the query suggestion list. In order to do this, we first created a Chrome extension which collects information about what web pages is the user visiting, what queries he submitted to Google and what suggestions Google returned for his subquery. By subquery we refer to the prefix of the query he started to type in the search input. Using this history, we perform an analysis on how important is this history on ranking future query suggestions. Moreover, we can later create user profiles that would improve the query suggestions offered by a search engine, like Google.

2. Related work

Query auto-completion. Auto-completion is used almost in all information retrieval engines. We have all seen how, in the search boxes of search engines, after we start typing the first character of our query, we immediately receive a possible auto-completion which will save us keystrokes when trying to fulfil an information need. What stands at the base of all these auto-completions is mostly the query logs of those particular search engines individually. We can see this kind of research in [2], [7], [5], [11], [10]. These approaches, do return pretty good suggestion lists but they lack a very particular thing, which is 'context'. This 'context' is composed by the immediately preceding queries that a user submitted. In [3], Bar-Yossef and Kraus demonstrated how recent user queries can significantly improve query auto-completion. They compare their results with the *Most Popular Completion (MPC)* which is one of the popular techniques for query suggestion. In [3], they say that the basic principle of MPC is users wisdom. This means that, if a particular query was used by a lot of users in the past, it is more likely that, that particular query will be the first candidate as an auto-completion. We can take, for example, a very popular and well known at the moment this article was written, social media website, "facebook". If we are trying to start typing letter "f" on www.google.com, the first auto-completion that we can see is "facebook" and that's because a lot of people are performing this particular query on google.com (see Fig. 1). In short terms, MPC is actually ranking suggestions based on their popularity. Let's say that we have a search log with

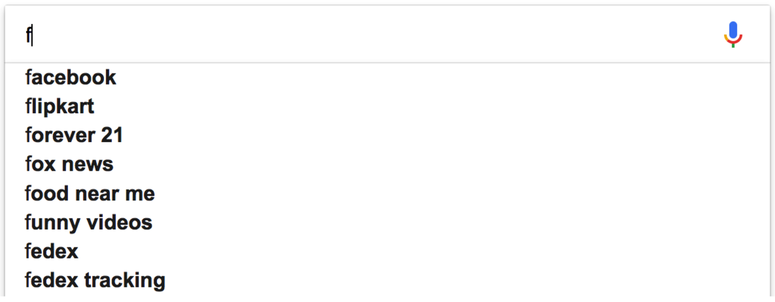

f
facebook
flipkart
forever 21
fox news
food near me
funny videos
fedex
fedex tracking

FIGURE 1. Typing letter "f" in Google's search input with all its suggestions.

previous queries $QLog$, a subquery (or the prefix of the intended query) sq and a list of query-completions $QC(sq)$, where all the items are starting with the desired subquery. Using MPC formula [23], we can calculate a rank for all items in $QC(sq)$ and order these items by their rank:

$$MPC(sq) = argmax_{q \in QC(sq)} w(q),$$

$$w(q) = \frac{freq(q)}{\sum\limits_{i \in QLog} freq(i)}$$

where $freq(q)$ is actually the number of ocurences of query q in $QLog$.

This formula is a *Maximum Likelihood Estimator*, which in [3], Bar-Yossef and Kraus, improve this popularity based algorithm and also take into consideration the previous queries of the user which are considered as 'query context'. They named this approach *NearCompletion*, where they compute similarity for this context and improve MPC and demonstrate using Mean-Reciprocal-Rank method, that the context of a query is very important when trying to generate suggestions. However, in their papers they only consider the query history of the user and not the personal browsing history of the user which is what we analyse in this paper.

Query Suggestion. Query suggestion and query auto-completion are very similar. The main scope of both of them is to save user keystrokes when performing a search. Query suggestion is an enhanced, proposed query that the user might be looking for, whereas an auto-completion is the possible query term that the user might want to type immediately after he started typing the first letter. Usually, auto-complete happens in the same search input where the user is writing his query and has to press either "enter key" or "right arrow key" to accept it; whereas auto-suggestion, usually appears as a list in the form of a drop-down from where the user can either press the ”down-arrow-key” or perform a mouse click to select it the desired suggestion. Both of these approaches are a real boost to the usability of search engines. Basically, we can say that auto-completion is the first item from the query suggestions list. In [4], they proposed a context aware query suggestion approach by mining click-through data and session data. First, they group similar queries into concepts and represent them on a bipartite graph. After this offline step, in an online step they will take the user query and find the concept for it in the graph and return the queries from that concept as suggestions. Another paper where click through data was analysed and used for ordering the suggestions is [14], where they demonstrate that the higher a suggestion is present in a suggestions list, tends to attract more click. In [6] Jiang et al. are reformulating the query by analysing how users previously reformulate their queries then adding words in the query and define a set of features which were applied using the LambdaMart [12] learning to rank algorithm. Others [13] have tried to apply probabilistic models, like Markov Process to predict what user's query will be immediately after he starts typing.

Personalized search. All the above papers do not consider the recent browsing history of the user when offering query suggestions to the user. Our main focus of this paper is to analyse the usefulness of the user's recent browsing history for query suggestions which will allow us to create a personal profile for each user and use that profile when ranking query suggestions. Personalized search, in general attracted attention of a lot of researchers, [18, 19, 20, 21, 22]. Each and every study showed that user's personal query history is very important in search systems. Let's take for example the very well known query ”ajax”. This query has three meanings that we are aware of: one would be the Dutch football team ”Ajax”, another one would be the cleaning product ”Ajax” and the last one would be ”Asynchronous JavaScript and XML” used in web development. In [1, 24, 25] we can see that these kind of queries are used by users pretty often. If we do not know anything about user's previous searches and interests, we could not know which result represents user's information need. In general, the way personalized search applies in auto-completion and query suggestion is by saving each query that a user used at a particular point in time, then use all this history in ranking query suggestions. In [17], we can see how Bennett et al. demonstrated that the long term query history is very useful when the users starts his search session and the short term query history is more relevant when the search session evolves. Matthijs and Radlinski, in [18] used a browser plugin to collect browsing history and used that history to re-rank search results and demonstrated that the returned results are more relevant to the user. Others, like Shokouhi in [23], went even further with personalisation and divided users into categories based on their age, gender and region and demonstrated that all these features have an impact on the suggestion that a user is waiting for when trying to search. For example, after typing letter 'i' in a search input, the most selected suggestion by male users is 'imdb' whilst female users were choosing 'indeed.com'. Another interesting example, from [23], is that users below 20 years mostly selected 'full house' after typing letter 'f' whilst the users above 50, selected 'florida lottery'.

All the above papers either consider the global or personal query history (measured at the search engine) or they use a form of browsing history, but for re-ranking search results returned by the search engine [18]. In contrast, we consider the personal browsing history of the user in order to provide better query suggestions. In this paper, we will present an experiment that validates the hypothesis that the user's recent browsing history is important for new query suggestions and a significant number of new queries can be predicted from the user's recent browsing history.

3. Architecture of the browser extension

In order to collect browsing history and submitted queries to Google search we have built a Chrome extension, named ***User History Collector***. The reason for collecting only Google searches is the fact that according to ***comScore*** in [26], in February 2016, out of the total explicit core searches performed on web, 64% were Google searches. The other part of 36% is divided between Bing, Yahoo, Ask Network etc. We choose to build a Chrome extension, and not an extension for other browsers, because according to latest *Browser Statistics* [27] from October 2017, made by *www.w3schools.com*, 76.1% of users are using a Chrome browser.

The entire extension is written in *javascript* which makes REST calls to some APIs that are persisting all user information in a MySQL database for later offline analysis. In Fig. 2 we can see the components of the extension and how data flows from one component to another. The *background script* and *content script* are actually components of a Chrome extension. The *content script* is a way of the extension to interact with webpages that are opened in a tab; it can be viewed as a part of the webpage, which is executed after the page is loaded. We use this component to extract the content of webpages. The *background script* is a component that holds the logic of the extension. We use this component to parse the data and send it to a server by making REST calls. The way our extension functions is, whenever a new page is loaded, the *content script* will be executed and based on the webpage, it will do the

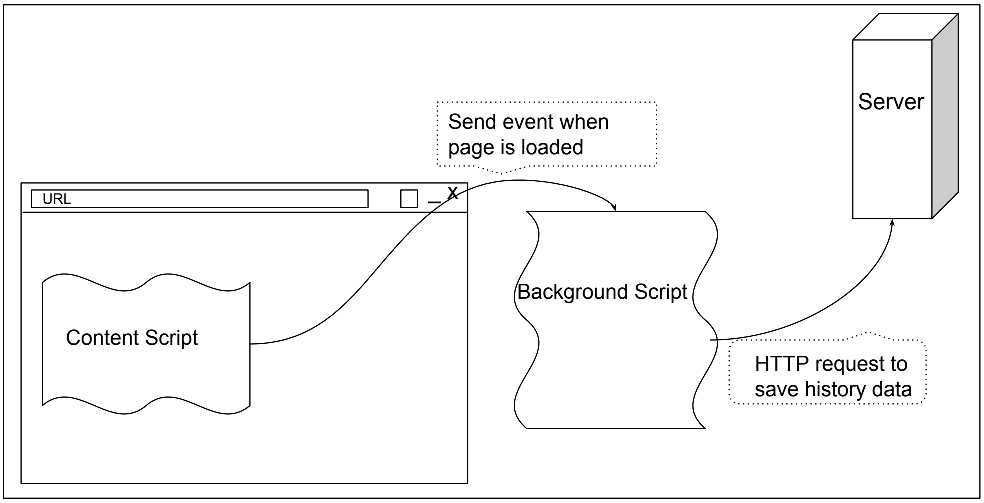

Server
Send event when page is loaded
URL
x
Background Script
Content Script
HTTP request to save history data

FIGURE 2. User History Collector components diagram

following (all webpages that are email pages, facebook pages and other pages that may contain personal information will not be analysed, will be ignored):

(1) If the URL of the page does not start with *"www.google."*, it will interpret it as a new webpage that was viewed and will extract the actual text from the HTML document and, alongside with page URL and page title, it will pass it to the *background script.* The *background script* will split the text in terms, will eliminate stop words and will calculate the term frequency for each unique word. After this step, it will make an Ajax HTTP request to a server which will store all the history data for later analysis.
(2) If the URL of the page, does match *"www.google."*, it means the user trying to perform a Google search. In this case:
   (a) For each key pressed in Google's search input, the *content script* will extract the value of the search input and the list of suggestions provided by Google for the written subquery. This information is passed to the *background script* which will send it to the server.
   (b) When user finishes to type the desired query and submits it to Google, that particular query is passed to *background script* which will send it to the server.

For all information that is passed to the *background script*, this will associate a unique identifier (which is generated once when the user installs the extension) before making the request to the server for persisting it.

4. Analysing collected data

Time period	Total number of clients	Total number of visited pages	Total number of Google queries
1 Month	14	14571	1847

Figure 3. Collected data

After having the extension running and collecting data for over a month, in table from Fig. 3 we can see that it gathered 14571 visited pages and 1847 queries that were submitted to Google from a number of 14 unique users that had the extension installed on their Chrome browser.

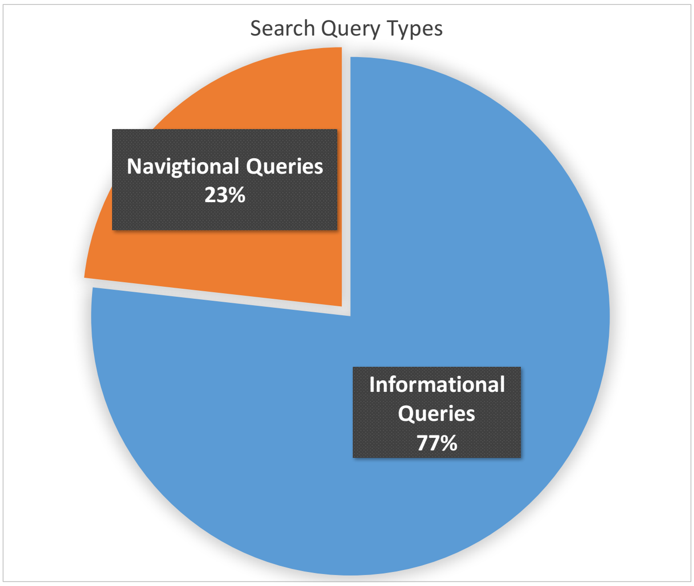


Figure 4. Query types

It is commonly accepted that search queries can be divided into two main types: *navigational queries* and *informational queries*. A *navigational query* is a search query entered by the user with the intent of finding a very particular webpage. For example, a user might type ”facebook” into Google's search input in order to find and navigate to ”Facebook” website. Another example would be if the user wants to go to ”Yahoo Mail”, he might search for ”yahoo mail” on Google, instead of directly typing the full address in the address bar. We can say that whenever a user submits a query to Google, and the URL of the first page that he navigates to contains all the terms from the query, the query is a *navigational query*. An *informational query* is a search query that can cover a very large topic, for which, the search engine can return a very large number of relevant results. When a user submits such a query to Google, he is looking for some information and not a particular website. For example, if the user submits the query ”einstein birthdate”, he is clearly looking for some information without caring the website he gets this from. In Fig. 4, which is built from the data collected by our Chrome extension, we can see that 77% of the queries are *informational queries* and 23% are *navigational queries*. We considered a query to be a *navigational query* if the URL of the first page, that is visited by the user, contains all query terms; all other queries that do not follow this rule are considered as *information queries*.

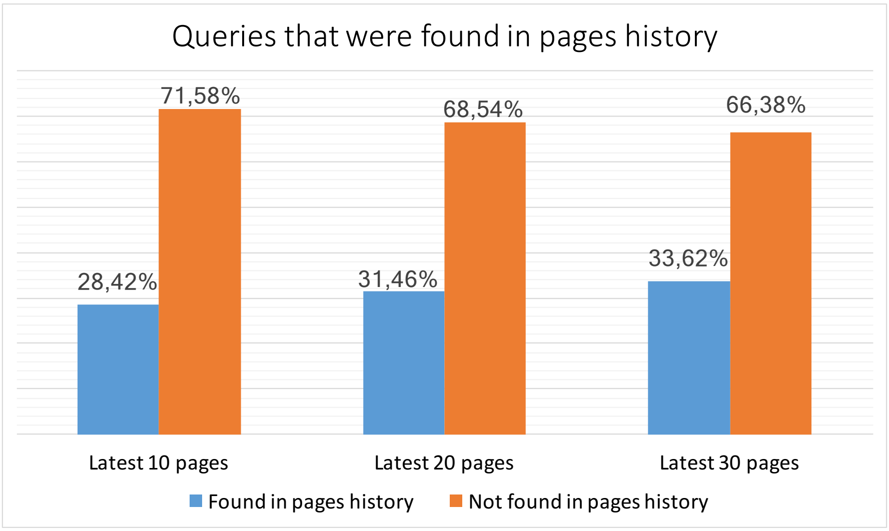


FIGURE 5. Relevance of browsing history

In Fig. 5 we analysed how many of the query terms of a query, can be found in webpages that were previously visited. If the query term appear in the URL of the page or in the title of the page, we no longer look within the actual content of the page because we have already found them. We made several

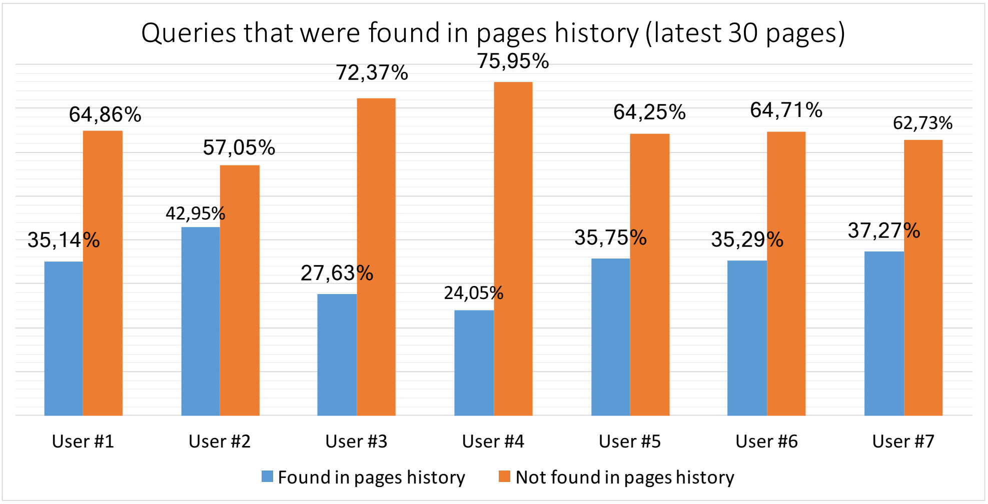


FIGURE 6. Relevance of browsing history (latest 30 pages) for particular users

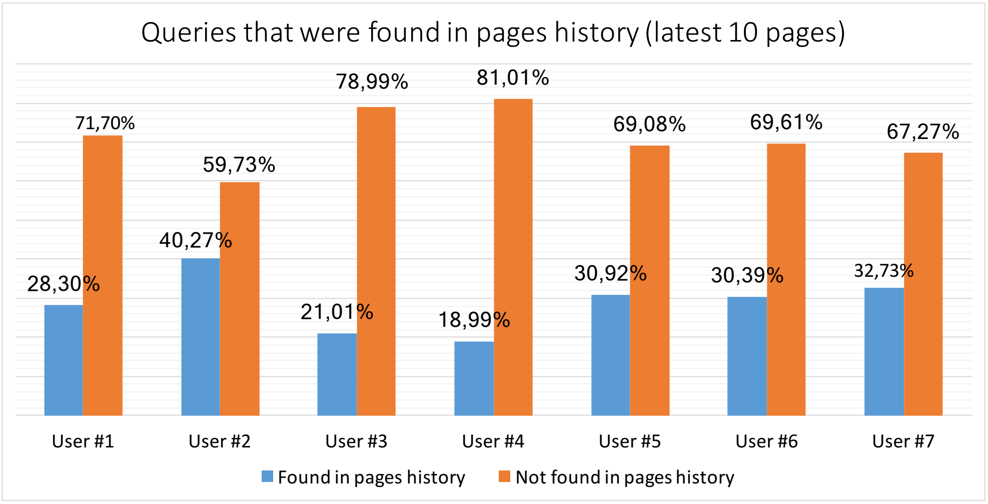


FIGURE 7. Relevance of browsing history (latest 10 pages) for particular users

tests related to the length of the recent history. First we considered the most recent history as latest 10 visited webpages. After this we increased this length to 20 pages, respectively 30 pages. We can observe how the number of the queries, that have been found in the recent history, increases as the length of the history increases.

In Fig. 6 and Fig. 7 we divided these results, and display how many query terms, can be found in webpages that were previously visited for particular randomly selected users that have installed our extension. Calculating the 75th percentile for these values, we can say that 37.27% of the queries that a user submits to Google, are found in the recent 30 pages long history and 32.73% for a history containing only the latest 10 webpages visited.

5. Conclusions and future work

In this paper we have studied how recent browsing history of users can have an impact on the next queries that the user will submit to Google search. In order to do this, we created a Chrome extension, that collects data about all pages that a user visits, the queries that he submits to Google and also each subquery and the entire list of suggestions that Google returns for the subquery. After having the extension installed on users browsers and collecting data for a month, we analysed the data and concluded that, in lots of cases, this history can be used to extract suggestions and display them for the next time the user will want to submit a search query. A way to extract suggestions from previously visited pages would be to take the most recent and very short browsing history (most recent 2 - 3 pages which were visited in the last couple of minutes), calculate a weight for each word in the page and based on these weights and the prefix that the user will type next, extract the most representative words and offer them as personal suggestions.

References

[1] Ryen W. White and Steven M. Drucker. Investigating behavioral variability in web search. In Proceedings of the 16th International Conference on World Wide Web, WWW '07, pages 21-30, New York, NY, USA, 2007. ACM.

[2] Holger Bast and Ingmar Weber. Type less, find more: Fast autocompletion search with a succinct index. In Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '06, pages 364-371, New York, NY, USA, 2006. ACM.

[3] Ziv Bar-Yossef and Naama Kraus. Context- sensitive query auto-completion. In Proceedings of the 20th International Conference on World Wide Web, WWW '11, pages 107-116, New York, NY, USA, 2011. ACM..

[4] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, pages 875-883, New York, NY, USA, 2008. ACM.

[5] Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient interactive fuzzy keyword search. In Proceedings of the 18th International Conference on World Wide Web, WWW '09, pages 371-380, New York, NY, USA, 2009. ACM.

[6] Jyun-Yu Jiang, Yen-Yu Ke, Pao-Yu Chien, and Pu-Jen Cheng. Learning user reformulation behavior for query auto-completion. In Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, pages 445-454, New York, NY, USA, 2014. ACM.

[7] Mario Arias, Jose Manuel Cantera, Jesus Vegas, Pablo de la Fuente, Jorge Cabrero Alonso, Guido Garcia Bernardo, Cesar Llamas, and Alvaro Zubizarreta. Context-based personalization for mobile web search. In PersDB, pages 33-39, Auckland, New Zealand, 2008.

[8] Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. Clustering user queries of a search engine. In Proceedings of the 10th International Conference on World Wide Web, WWW '01, pages 162-168, New York, NY, USA, 2001. ACM

[9] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic query expansion using query logs. In Proceedings of the 11th International Conference on World Wide Web, WWW '02, pages 325-332, New York, NY, USA, 2002. ACM

[10] Holger Bast, Debapriyo Majumdar, and Ingmar Weber. Efficient interactive query expansion with complete search. In Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM '07, pages 857-860, New York, NY, USA, 2007. ACM.

[11] Ryen W White and Gary Marchionini. Examining the effectiveness of real-time query expansion. Information Processing and Management, 43(3):685-704, 2007.

[12] Christopher J. C. Burges, Krysta M. Svore, Paul N. Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. In Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14, YLRC'10, pages 25-35. JMLR.org, 2010.

[13] Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, Hongyuan Zha, and Ricardo Baeza-Yates. Analyzing user's sequential behavior in query auto-completion via markov processes. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15, pages 123-132, New York, NY, USA, 2015. ACM.

[14] Yanen Li, Anlei Dong, Hongning Wang, Hongbo Deng, Yi Chang, and ChengXiang Zhai. A two-dimensional click model for query auto-completion. In Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, pages 455-464, New York, NY, USA, 2014. ACM.

[15] Bernard J Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. Information processing and management, 36(2):207-227, 2000.

[16] Mark Sanderson. Ambiguous queries: Test collections need more sense. In Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '08, pages 499-506, New York, NY, USA, 2008. ACM.

[17] Paul N. Bennett, Ryen W. White, Wei Chu, Susan T. Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. Modeling the impact of short and long-term behavior on search personalization. In Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '12, pages 185-194, New York, NY, USA, 2012. ACM.

[18] Nicolaas Matthijs and Filip Radlinski. Personalizing web search using long term browsing history. In Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, WSDM '11, pages 25-34, New York, NY, USA, 2011. ACM.

[19] Mariam Daoud, Lynda Tamine-Lechani, Mohand Boughanem, and Bilal Chebaro. A session based personalized search using an ontological user profile. In Proceedings of the 2009 ACM Symposium on Applied Computing, SAC '09, pages 1732-1736, New York, NY, USA, 2009. ACM.
[20] Zhicheng Dou, Ruihua Song, and Ji-Rong Wen. A large-scale evaluation and analysis of personalized search strategies. In Proceedings of the 16th International Conference on World Wide Web, WWW '07, pages 581-590, New York, NY, USA, 2007. ACM.
[21] Ahu Sieg, Bamshad Mobasher, and Robin Burke. Web search personalization with ontological user profiles. In Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM '07, pages 525-534, New York, NY, USA, 2007. ACM.
[22] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '05, pages 449-456, New York, NY, USA, 2005. ACM.
[23] Milad Shokouhi. Learning to personalize query auto-completion. In Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '13, pages 103-112, New York, NY, USA, 2013. ACM.
[24] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Potential for personalization. ACM Trans. Comput.-Hum. Interact., 17(1):4:1-4:31, New York, NY, USA, 2010.
[25] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Implicit user modeling for personalized search. In Proceedings of the 14th ACM International Conference on Information and Knowledge Management, CIKM '05, pages 824-831, New York, NY, USA, 2005. ACM.
[26] https://www.comscore.com/Insights/Rankings/comScore-Releases-February-2016-US-Desktop-Search-Engine-Rankings
[27] https://www.w3schools.com/browsers/

Faculty of Mathematics and Computer Science, Babeş–Bolyai University, Cluj-Napoca, Romania
E-mail address: ionutb@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.06

IMPROVING PROGRAM COMPREHENSION THROUGH DYNAMIC CODE ANALYSIS

ROBERT FRANCISC VIDA

Abstract. Most of the software that is currently being developed in the industry tends to be very complex and it is safe to assume that a lot of future software projects will keep escalating in their complexity. This means that developers need to keep track of numerous aspects of their system at all time and that new additions to the team will have difficulty adjusting to projects. In this paper, we will propose a set of techniques that combine already existing dynamic code analysis concepts to resolve the aforementioned problems, which can be summarized as being program comprehension difficulties. To do this we introduce a few novel software analysis and visualization techniques that facilitate program comprehension. The approach proposed within this paper allows for easy identification of semantic information, data available at execution time, which might be difficult or even impossible to portray in an easy to understand representation using existing software visualization techniques. During this paper we will be considering traditional object oriented programming languages, however these ideas should be useful in the context of other programming paradigms as well.

1. Introduction

It is safe to say that currently, all over the world, there are many software systems being developed for different purposes. Obviously, the size and scope of these projects may vary but all of them have a certain degree of complexity, which, as stated in [10], is one of the leading causes of failure in the industry. That being said, we believe that if the developers have a good understanding of the system at all times, they will be able to minimize complexity growth

Received by the editors: November 30, 2017.

2010 *Mathematics Subject Classification.* 68N01, 68M20.

1998 *CR Categories and Descriptors.* K.6.3 [**Management of computing and information systems**]: Software Management – *Software maintenance*; D.2.5 [**Software Engineering**]: Testing and Debugging – *Tracing*.

Key words and phrases. Dynamic code analysis, program comprehension, software visualization.

as new features are added or maintenance is performed, this in turn reducing the risk of project failure.

The goal of this paper is to explore new techniques that might reduce the change of developers needlessly adding complexity to their software system. This is done by aiding them to better understand the behavior of their system while also helping new additions to the team gain knowledge about the system in an easier manner. This is called program comprehension. Most of the techniques that we will present rely on dynamic program analysis and have been designed specifically for object oriented programming, however they might be relevant for other programming paradigms as well.

The remainder of this paper is organized as follows. Explaining what program comprehension is and why we need it in Section 2. Section 3 presents dynamic program analysis and why it was chosen over its static counterpart, as well as describing the importance of software visualization and how it can be used. In Section 4 we review existing solutions for software visualization and list a few tools which can be used in order to enhance program comprehension and discuss their strengths and shortcomings. We present our original approach and provide a few detailed examples to better explain the techniques along with their concepts in Section 5. In Section 6 we mention a prototype that facilitates some of the techniques discussed and finally, in Section 7, we have the conclusion of this paper along with a few ideas for potential future work.

2. Program Comprehension

One of the most important things to consider before altering code is how much of it is actually understood by the person performing the change. It is imperative for the developer to fully understand, in as much detail as possible, how the program works before attempting to modify it since the chances of damaging the existing system are direct proportional with the size of the system, usually very high. This is also explained in [2]. The damage can range from the obvious bugs and inconsistencies to the more annoying and hard to identify ones such as performance, security and reliability.

In [11], the author presents the usefulness of program comprehension by describing what activities are required during maintenance and evolution tasks. The common activity across these tasks implies understanding the system. This should speak volumes about the importance program comprehension at all stages of the development cycle.

One of the most obvious methods of keeping a program easy to understand is to keep the code clean by the means of code review and constant refactoring sessions. The problem is that even if the code is clean, sometimes it can be very difficult to see the bigger picture, like how some components interact with each other on a bigger scale or in stretched out scopes. In this case, the obvious answer is to keep the system well documented. This is relatively easy to achieve since all you waste is a bit of time and can work wonders in a lot of cases. That being said, having too many system descriptors, be it explicit (documentation) or implicit (clean code) and still expect high program comprehension from the developers is a bit too much to ask, especially if they are relatively new to the project. It is very difficult for people to understand a system just by reading about it.

The solutions proposed to solve this problem, or at least alleviate its effects, are based on the fact that people are more susceptible to understand something if they are actively involved in the process of researching [9]. This is in contrast to more passive methods like looking through code or reading documentation, where the person researching can only think what is happening at any given moment. That being the case, we will look over different techniques through which, by using dynamic program analysis, one can gain a better insight over what happens within the program.

3. Dynamic program Analysis

Dynamic program analysis is a technique through which one can analyze the different properties of a program while it is executing. At times, static analysis has been used to analyze the dynamic behavior of programs since it was easier and did not require the execution of the program either. This, however, does not yield results as precise as actually doing dynamic analysis would. Also, as programming languages have evolved to running in more dynamic environments, the presence of features like dynamic binding, polymorphism and threads have made static analysis quite ineffective. This is because static analysis can only check what is present. The packages that may be loaded dynamically when running the application might not be present until the execution, so this shortcoming is understandable.

The two techniques mentioned earlier, dynamic program analysis and static program analysis are complementary, each having their own strengths and weaknesses. The static analysis examines the source code rigorously at compile-time. Relevant techniques include data-flow analysis, which analyzes how variables change their values through the execution flow, symbolic execution, which determines what input values case each part of an application to execute and dependence analysis. Dynamic analysis looks at an application while it is running and analyzes data obtained from that. Two commonly used techniques are assertions, which are simple checks inserted within the source code itself in order to check various things and coverage analysis which analyzes the flow of the execution while an application is running [4]. In this paper we will look at dynamic program analysis and explore to some extent the two methods mentioned earlier.

Dynamic analysis can help guide the development process towards producing a solution that behaves, within the realm of possibility, as intended, as well as aiding the developers in enhancing and optimizing an already working system. By analyzing the relationships between independent threads or duration of method calls as well as the context they are called from we can easily devise solutions that might improve overall performance. This is also clearly stated by Thomas Ball in his article [3].

3.1. **Software Visualization at Runtime.** Software visualization refers to displaying information about or related to the software system in a visual-oriented manner so that it is easier to understand and interpret [6]. The information type that can be used can range from the architecture of the system, how the code is structured, to its runtime behavior, algorithm behavior.

In order to extract runtime information without altering the source code, one could use profiling, which is a type of dynamic program analysis. Through it we can obtain information like space or time complexity, method calls and other statistics that are generated while running the application. Using this information it is fairly easy to construct an execution graph or calculate the frequency and duration of subroutines. It is on this type of analysis that the techniques presented in this paper are based on.

After the desired information has been obtained, the next important step is deciding on how the information shall be displayed. One obvious solution would be to display the flow of execution in the form of an UML Sequence Diagram. This would be quite suitable since this type of diagram is very intuitive and exposes the information and component interactions nicely, however there are a two fatal drawbacks. The first one is that if the execution flow complexity is too great, it will become very difficult for a user to follow through, and the second is that this type of diagram was not created with the idea of multiple threads of execution.

In order to identify the most suitable form of visualizing software runtime data we need to take a look at what developers usually use to gain knowledge regarding the system. That being said, the built-in debugger that most IDEs (Integrated Development Environment) have would fit the description perfectly. Representing the execution flow in a tree-like manner, which if going from a leaf to the root will look like the common execution stack, would be ideal since the user will already be familiar with the format.

4. Related Work

Trying to gain program comprehension by analyzing the execution flow is not a new idea, however the approach that we took in this paper is. In [12] we see an approach to compare different execution traces to identify changes within execution code, order and duration. The difference between this paper and ours, is that we try to focus on providing as much information as possible regarding a execution trace and present it in a easy to understand manner. Comparing two execution flows is also presented here, however it is not the main issue we want to tackle.

In [8] we see a similar approach to ours, applying different program analysis techniques on the software in order gain some understanding of it. Technically speaking, it takes it a step further by using both static and dynamic code analysis, whereas we only use the latter. The techniques that we propose in this paper can be used not only to understand code someone else wrote but to gain insight into ones own code as well. The other slight difference is the manner in which data is presented, we chose to present the data in a tree-like manner while in the aforementioned article they seem to have chosen to go with graphs.

Along the years there have been numerous tools that have aided developers in analyzing the software they develop, assuring them that they are on the right track and reducing the possibility to introduce faulty or algorithmically incompatible features into their system.

One might argue that a debugger can be considered a tool for facilitating program comprehension by observing the execution flow, however it has a very big flaw. Since their aim is to inspect the code by stopping the execution at certain points called breakpoints, they often bring the application in a state in which it couldn't naturally be in. This is an especially serious matter in situations where there are multiple threads or scheduling components.

VisualVM is a visual tool written in and for Java that uses lightweight profiling to extract statistics regarding an application during its runtime [13]. The application is very easy to use and can connect to running applications at any time. The main drawback of this tool is that it only uses a flat profiler. This means that it only gathers data regarding memory consumption, execution duration and execution time. This is good if someone needs a quick overview of the system during its execution or if they are looking for memory leaks, however it lacks context. That being said, it is impossible for the developer to reason the behavior of the system against these statistics.

Gprof specializes on call graph executions. Call graphs can be both static, which takes into account all possible routes and does not require the application to be running, and dynamic, which takes into account only executed methods. This means that it is able to assess the cost of routines accurately [7]. Because of this it is easy for the people running the analysis to see the methods that were called during the execution and also their ordering, giving us a context of them. Seeing the path the execution took, can give developers a few hints where something went wrong or where optimizations might be possible.

Aprof is a Valgrind tool designed to help developers identify inefficiencies in code [5]. It is input-sensitive, that means that on top of call graph, it takes into account the input for methods and measures their performance based on the workload received. This is very important because the analysis allows the developers to pinpoint the exact location where the execution ran off track.

5. Proposed Concepts and Techniques

The techniques presented within this paper are very straightforward and previous knowledge regarding dynamic program analysis is not required in order fully understand the ideas behind them. Before we discuss the techniques themselves we should first define a few concepts which will be used. Some of these concepts are not new by any means, while the others can be seen as being built on top of existing ideas. Either way, it is important to understand them since they are the foundation for the techniques that we will discuss later on.

Since we will be looking at techniques through which to extract and present the execution flow of an application we will need to analyze the simplest component that we can relate to. That being said, considering that we are targeting object oriented programming, this would be the class method.

5.1. **The Method Structure.** An application is usually composed of multiple classes. Each of these classes have methods of the format $Method = (L_{input}, L_{instructions}, V_{output})$, where: L_{input} is the list of input arguments which can be empty, $L_{instructions}$ is the list of instructions within which can also be empty and V_{output} is the optional return value from the method. It is important to keep in mind that the instructions may contain calls to other methods. The best format to express the method execution, considering our needs, is a version that offers the following components: input arguments L_{input} with call time t_0, output value V_{output} with return time t_f and a set of the method instructions that contains only other method calls $L_{method} = (i|i \in L_{instructions}$ such that i is a method called during execution). The format for the method execution will look like this:

$$Method_{execution} = (t_0, L_{input}, L_{method}, t_f, V_{output})$$

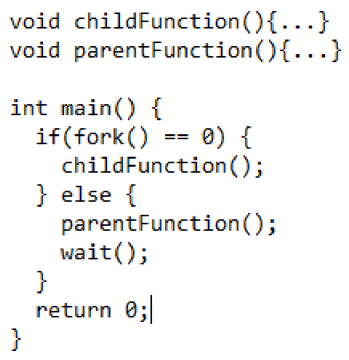
5.2. **Concepts.** Each of these concepts are independent to each other, this is important since it means that they can be used separately or in combination to each other. This allows us to gather more specialized information regarding our system, information that is more relevant to our goal. The concepts that we considered in our approach are:

> Call stack or Call tree - sequence of calls presented in a stack or tree layout. The main reason for choosing the approach of using a stack layout is because of the familiarity developers have with stack traces used when debugging.
> Selective focus - in order to minimize the impact on the running application it would be optimal to focus attention on only parts of it.
> Context information - sometimes, having information on what each method call starts with and produces is for the process of understanding what exactly is happening to the application while the executing.
> Selective focus using context information - this is selective focus enhanced with the knowledge obtained from analyzing the context. Basically only recording methods when certain conditions are met.

5.2.1. *Call Stack/Call Tree.* The call stack or call tree is purely a visual concept through which one can depict the execution flow of an application. Having a clear and intuitive way of checking the execution steps of an application is a crucial aspect. It is mainly though this that the users observe how the program unfolded, thus it is crucial to the process of understanding the system.

The decision to organize and present the method calls in a tree-like manner was made with the purpose of having the developers already be familiar with the representation since it works similar to how a stack trace works in debug mode, just a bit more hierarchical.

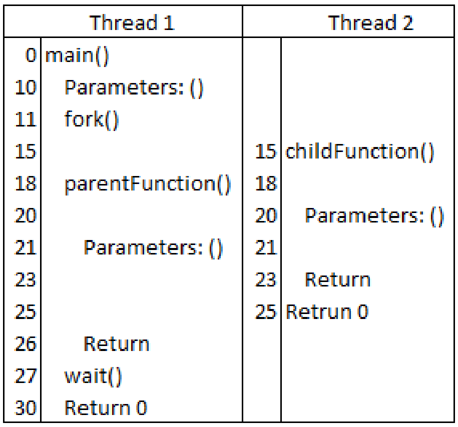
As the root node we will have the signature of the method being called. This will include the method name and the types of parameters it accepts. The first child will be composed out of the parameters sent when calling the method. If no arguments were sent, then this node should not be present at all since it would be irrelevant. The children of the node will be the elements of L_{input} component mentioned earlier. The last node will represent the return statement of the method, this node should always be shown because this means that the method finished successfully. If the method returns a value, the value will also be shown. This would be V_{output}. In between the first and last node will be the nodes of all methods called from within the current method, sorted chronologically. These nodes will be method nodes themselves and there should be one for each element of L_{method}.



```
void childFunction(){...}
void parentFunction(){...}

int main() {
  if(fork() == 0) {
    childFunction();
  } else {
    parentFunction();
    wait();
  }
  return 0;
}
```

FIGURE 1. Call stack code



Thread 1		Thread 2	
0	main()		
10	Parameters: ()		
11	fork()		
15		15	childFunction()
18	parentFunction()	18	
20		20	Parameters: ()
21	Parameters: ()	21	
23		23	Return
25		25	Retrun 0
26	Return		
27	wait()		
30	Return 0		

FIGURE 2. Call timeline visual representation

All of the nodes will have a time associated with them, this will be represented by a number in a column on the right hand side. This will be very useful since it offers information on the duration of the calls.

In case there are multiple threads running, there should be separate trees for each thread. The nodes of the threads should be intertwined, with empty nodes representing that something happened on another thread at that time.

In Figure 1 we can see a sample code of a program that has two threads and each of them goes on to call a different function. Figure 2 depicts an execution of this code. We can see where each function call starts for each thread. It is clear where the function execution overlapped and where they stopped.

A stacktrace displays the order in which methods were executed and a common profiler can provide insight into the context of the execution environment. With this format we gain both at the same time. We can see contextual information integrated within the execution order of the methods.

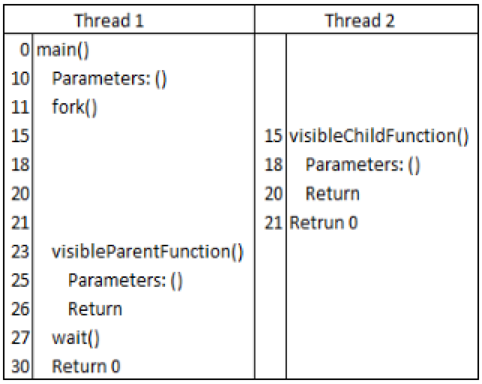
5.2.2. *Selective Focus.* Instead of capturing the execution of every function call on every thread, it may sometimes be desired to only focus on a certain thread, or a certain group of functions. Applying such filters on the profiler will drastically improve performance and reduce the impact that the profiler has on the analyzed application.

The selective focus concept provides a good solution for reducing unnecessary analysis on portions of code that are of no interest to the developer. This also allows for other analysis concepts that consume more processing power to be used without the fear that they might disturb the natural flow of execution

```
void visibleChildFunction(){...}
void visibleParentFunction(){...}
void ignoredParentFunction(){...}

int main() {
  if(fork() == 0) {
    visibleChildFunction();
  } else {
    ignoredParentFunction();
    visibleParentFunction();
    wait();
  }
  return 0;
}
```

FIGURE 3. Selective call timeline code



Thread 1		Thread 2	
0	main()		
10	Parameters: ()		
11	fork()		
15		15	visibleChildFunction()
18		18	Parameters: ()
20		20	Return
21		21	Retrun 0
23	visibleParentFunction()		
25	Parameters: ()		
26	Return		
27	wait()		
30	Return 0		

FIGURE 4. Selective call timeline visual representation

of the software too much. It is undeniable that if the developer adds a lot of resource consuming analysis concepts and if the application is multithreaded, then there is a good possibility that the execution will go into a unique state which would not be possible under normal circumstances.

This is not a new concept, there are a lot of tools that have ways to selectively choose what parts of the application to analyze, however this method differs from the way you define these parts and the manner in which the report is generated at the end of the analysis.

In Figure 3 we have a sample code of a program that has two threads and each of these threads will call their own functions. Figure 4 depicts a potential execution of the code previously stated. In this certain representation, the person running the analysis decided to ignore the method ignoredParentFunction() and so it is not represented within the timeline.

To take full advantage of the capabilities of this analysis method, we strongly suggest implementing the filtering system in a dynamic fashion, by this we mean being able to specify target methods through a mechanism similar to regular expressions. This is not hard to do and one would gain the ability to mark the functions that are to be analyzed at runtime. This is imperative if the developer wants to intercept function calls even if they were declared through reflection.

5.2.3. *Context Information.* This concept is concerned with recording the data that is used in inter-method communication. This covers both input and output data, however the amount of how much data to record should be kept in mind. What this means is that if for example we have a class as input data, we need to specify how deep within its fields we will record. If the class has another class as field, and so on, there should be a stopping point to reduce the stress of the analysis.

All of the data can be stored in a context along with their reference id so we can observe the changes made to an entity through the entire execution. This would allow the users to easily follow data modification during the application execution. It is important to note, that this concept is similar to dynamic program slicing [1], which is a technique that gathers all the statements that changed the value of a variable during an execution. On the other hand, our technique is able to identify the changes done to the object itself. So they are similar in aim, which is to study the evolution of the application state, but have different approaches on how to do this.

The context information concept is one of the most resource-consuming of all of the ones proposed within this paper, however it also is the one that gives the most detailed insight about what happened within the code because it can clearly display all input and output values for each function.

5.2.4. *Conditional Focus.* This concept builds on top of the previous mentioned concept, selective focus, by adding awareness regarding the context. This means that after we filter the parts of the program we want to focus on, we can go even further and add that only when specific input values are passed should we inspect the section. The overhead added to the execution might not be very appealing, however it is well worth the sacrifice in order to have a way to add this sort of flexibility.

5.3. **Techniques.** Next, we will use the concepts defined earlier and combine them so that they will aid us in our goal of understanding the program better. All of these techniques assume that we use a call stack as a way of presenting the information.

5.3.1. *Basic Analysis Technique.* The first combination will be very straightforward and somewhat predictable, we will use selective focus and context information. By using selective focus, we reduce the strain put on the application by the analyzer and by using context information we expose detailed information about the piece of code that we are interested in. This combination is important since with this the developer can gain in-depth knowledge over the part of the application that he desires. Illustrating a call stack/tree of the methods that were executed and the data received, changed and produced. This combination can also be used as an advanced form of logging.

5.3.2. *History Technique.* This technique is actually built on top of the previous one, however the step is in a horizontal direction. What this means is that it does not go deeper down into extracting more data or filtering the inspected scope of the program, but simply keeps track of multiple executions and attempts to compare them. This is important since this way the developers can examine the evolution of the behavior of a program. They can also execute the same steps over and over again, in order to check the consistency and reliability of multithreaded sections. The visual representation is not difficult to understand, this makes it is easy to present to non-technical users and explain how it all works. It eases the communication bridge between two groups of people that usually have difficulty explaining their point of view to the other.

5.3.3. *Checking Technique.* This technique would more likely be used for testing purposes rather than program comprehension. It is very possible to use a slightly altered version of the conditional focus concept to check at all stages that different values do not pass through certain areas of the code. Using such a method on a system would seem as though it was attempted to add formal verification on top of an already existing system. This is a strange approach since formal methods are performed before any code is written, however this method is worth mentioning since there are some situations where such approaches might be needed.

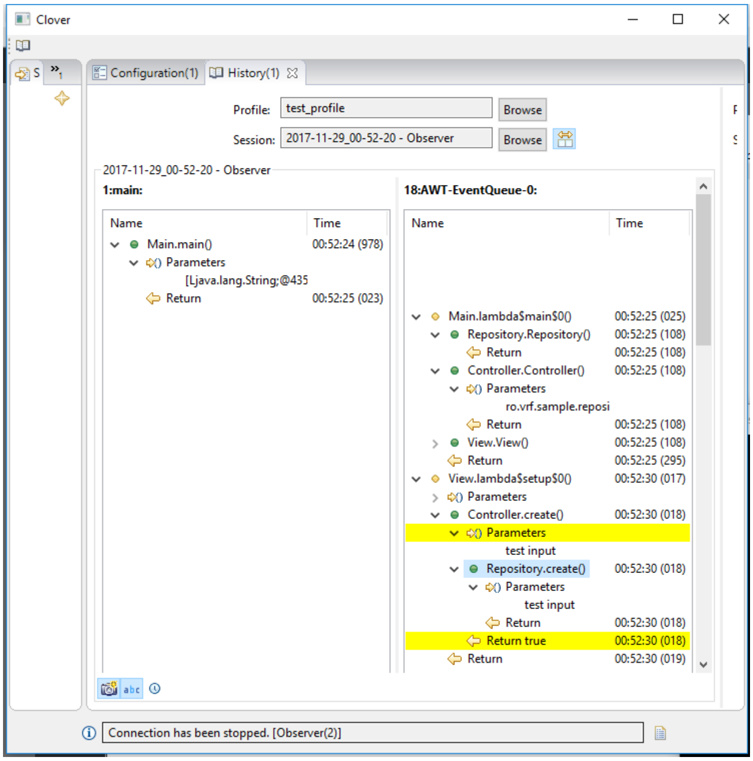
5.4. **Threats to validity.** It is important to keep in mind that the concepts and techniques presented have not been proven to be a definite improvement over other similar tools nor do we claim them to be. The techniques were devised in order to explore new ideas in the domain of program comprehension and are still in an experimental state at this moment.

An aspect that is quite concerning to the validity of these techniques is scalability. These concepts were only tested in environments of small sized applications that did not make use of too many execution threads of the same. This concern relates to both execution and visualization issues. By this we mean that the tool might behave faulty when a larger application is analyzed, but also that the visualization mechanism might prove to be less suited when too many points of interest need to be shown at the same time.

Another thing to keep in mind is that different programming languages will have different instrumentation limitations. In a few cases these techniques might actually be impossible to implement.

6. Working Prototype

Most of the concepts and techniques presented within this paper have already been implemented into a stable prototype. Written in Java, by the use of instrumentation it is able to observe other Java applications while they are being executed without affecting the normal execution flow too much. The



Profile: test_profile

Session: 2017-11-29_00-52-20 - Observer

2017-11-29_00-52-20 - Observer

1:main:

Name	Time
Main.main()	00:52:24 (978)
Parameters	
[Ljava.lang.String;@435	
Return	00:52:25 (023)

18:AWT-EventQueue-0:

Name	Time
Main.lambda$main$0()	00:52:25 (025)
Repository.Repository()	00:52:25 (108)
Return	00:52:25 (108)
Controller.Controller()	00:52:25 (108)
Parameters	
ro.vrf.sample.reposi	
Return	00:52:25 (108)
View.View()	00:52:25 (108)
Return	00:52:25 (295)
View.lambda$setup$0()	00:52:30 (017)
Parameters	
Controller.create()	00:52:30 (018)
Parameters	
test input	
Repository.create()	00:52:30 (018)
Parameters	
test input	
Return	00:52:30 (018)
Return true	00:52:30 (018)
Return	00:52:30 (019)

Connection has been stopped. [Observer(2)]

FIGURE 5. Screenshot of the prototype

only impact on the inspected application is that the overhead added by the analysis itself, by this we mean the mechanism through which we extract the data, so the execution threads might slow down a bit. The prototype is able to handle multiple execution threads and structures the flow into a hierarchical manner.

A few of the concepts described in this paper were only partially implemented or do not have the flexibility previously described, the reason behind this is that it is only a prototype meant to show the appeal of such a tool. A noteworthy but not necessarily critical flaw for the tool is the fact that it is unable to inspect the core classes because they are being used in order to extract the data from the analyzed application. The reason we say it is not critical is because one would normally use this application to analyze their own code. In order to gather as much data as possible, we recommend that the instrumentation process starts as soon as possible, exactly when the target application is started would be ideal. The reason behind this is that although the tool is able to analyze already running applications, it is limited to classes that have not been loaded, by this we mean those that have not yet been used.

In Figure 5 we can see the execution flow of an application that has two threads. The two tree structures depict the methods called from each thread as they are called, each having the identifying name of the thread above them. The root nodes represent the first methods called that respect the filtering conditions set before the analysis began. Whenever there are parameters sent to the methods, a child node containing a list of parameters will be present. Next, if there are other methods called from this method, they will be indicated through separate suggestive nodes. The last child node of a method node will be the return statement that will also indicate the return value if there is any. All nodes, except for parameter nodes and their children, display the time at which they occurred, this way one could easily tell how long the method took to execute. We believe that this way it is easy to see crucial information regarding methods, such as access control modifiers, input parameters, entry and exit time points as well as method calls performed within. However, when there are many chained methods it might be difficult to keep track of the exact location within execution tree. In order to aid the user in orienting themselves within the execution tree, we highlighted with yellow background the entry and exit point of the currently selected method.

7. Conclusion and Further work

From all the information presented in this paper, it is easy to understand the importance of program comprehension and why it is imperative for it to be as high as possible. The method through which this is done is not particularly important, however by using dynamic program analysis you get to observe the application in its most crucial state, at runtime. By directly observing how the application behaves during the execution you get to see how it reacts, no need for speculation, we can see exactly how all parts come together and work with each other.

We have presented various techniques through which one might enhance the experience of gaining or maintaining program comprehension regarding an application along with a working prototype that makes use of these methods. It is important to note that all of these techniques require the user be engaged in the analysis task so that the process of understanding the application can progress more naturally.

In the future we plan to further refine the concepts and techniques previously mentioned as well as extend them to provide more customizable and relevant information to the developers or the interested users. Pursuing other techniques is not of the table. There were a few other ideas that did not make it into this paper for various reasons. For example a technique through which one would be able to identify hidden dependencies, detect places where design patterns should be implemented or determine if two classes belonging to different components are connected to each other to tightly when they shouldn't (high coupling).

References

[1] Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. *SIGPLAN Not.*, 25(6):246–256, June 1990.

[2] Usman Akhlaq and Muhammad Usman Yousaf. Impact of software comprehension in software maintenance and evolution. Master's thesis, Blekinge Institute of Technology, 2010. Chapter 8.

[3] Thomas Ball. The concept of dynamic analysis. *SIGSOFT Softw. Eng. Notes*, 24(6):216–234, October 1999.

[4] Mario Barrenechea. Program analysis. `https://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/barrenecheamario.pdf`, . [Online; accessed 5-September-2017].

[5] Emilio Coppa, Camil Demetrescu, and Irene Finocchi. Input-sensitive profiling. *SIGPLAN Not.*, 47(6):89–98, June 2012.

[6] Denis Gracanin, Kresimir Matkovic, and Mohamed Eltoweissy. Software visualization. *Innovations in Systems and Software Engineering, A NASA Journal*, 1(2):221–230, September 2005.

[7] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. Gprof: A call graph execution profiler. *SIGPLAN Not.*, 17(6):120–126, June 1982.

[8] Wilhelm Kirchmayr, Michael Moser, Ludwig Nocke, Josef Pichler, and Rudolf Tober. Integration of static and dynamic code analysis for understanding legacy source code. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 543–552, 2016.

[9] Michael Prince. Does active learning work? a review of the research. *Journal of Engineering Education*, 93(3):223–231, 2004.

[10] Roger Sessions. The it complexity crisis: Danger and opportunity. Technical report, ObjectWatch, 2009.

[11] Priyadarshi Tripathy and Kshirasagar Naik. *A Practitioner's Approach, Software Evolution and Maintenance*, chapter 8. John Wiley & Sons, Inc., New York, NY, USA, 2014.

[12] Jonas Trmper, Jrgen Dllner, and Alexandru C. Telea. Multiscale visual comparison of execution traces. In *Proceedings of the 21st International Conference on Program Comprehension*, pages 53–62, 2013.

[13] Visual vm. https://visualvm.github.io/ [Online; accessed 12-December-2017].

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: `robertv@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.07

ROBBY: A NEUROROBOTICS CONTROL FRAMEWORK USING SPIKING NEURAL NETWORKS

CĂTĂLIN V. RUSU, TIBERIU BAN, AND HOREA ADRIAN GREBLĂ

Abstract. The variety of neural models and robotic hardware has made simulation writing time-consuming and error prone, forcing thus scientists to spend a substantial amount of time on the implementation of their models. We developed a framework called “Robby” that allows the quick simulation of large-scale neural networks designed for robotic control by spiking neural networks. It provides both mechanism for robotic communication and tools for building and simulating neural controllers. We present the basic building blocks of “Robby” and a simple experiment to show its practical value.

1. Introduction

As hardware becomes more diverse and affordable the need for controlling different hardware platforms within similar contexts becomes more prominent. The difficulty lies in the fact that each robot has a different underlying physical layout with different programming interfaces. Thus, similar control programs would have different implementations depending on the robotic platform. In this context neural simulators, that are able to simulate large-scale neural networks efficiently, and robotic frameworks, that allow them to interact with robotic devices, are highly desirable. Such frameworks: (i) allow the facile control of physical cognitive agents; (ii) enable scientist to spend less time on programming details and more on detailing experiments; (iii) provide a basis to easily explore theoretical principles in the context of real computational tasks involving physical autonomous agents; (iv) help increase our understanding of how large neural networks mediate cognitive functions. Popular frameworks either provide a collection of software and algorithms focused on robot communication, sensing and navigation while leaving the development of control

Received by the editors: October 20, 2017.

2010 *Mathematics Subject Classification.* 68T40.

1998 *CR Categories and Descriptors.* I.2.9 [**Artificial Intelligence**]: Robotics – *Operator interfaces*.

Key words and phrases. neural simulators, robotic frameworks, cognitive robotics, spiking neural networks.

programs (neural networks) to the user, like "Player/Stage" [1] (biased towards wheeled robots) or "YARP" [2] (biased towards humanoid robotics), or either provide limited support like "Pyro" [3] or "Orocos" [4]. Thus, it would be of interest to have a system that provides both the abstraction layer for robot communication and the logic to support the development of neural controllers. We introduce "Robby", a flexible and distributed framework for robotic control with spiking neural networks, ideal for large-scale simulations. It enables the control of robotic platforms occupying different physical locations by multiple types of neural networks. In the framework, controllers are primarily neural networks, but in principle they can be any user-defined controller. Additional support for joystick controllers is provided to allow direct manipulation of devices. While this setup might seem restrictive it is sufficient for common simulations in neurobotics while keeping the architecture of the system simple. Since "Robby" makes easy to simulate and explore spiking neural networks with different architectures and properties with the aim of training autonomous robots it could be of interest to the scientific community interested in cognitive robotics. In the following we present the basic principles behind "Robby" and provide several future development directions together with a simple evaluation to show its use.

2. Architecture and implementation details

Low level programming is tedious because it requires a deep understanding of the underlying hardware platform and knowledge of complex languages and programming interfaces. As complex behavior generally requires complex hardware, a large amount of time is spent on writing even the most basic simulations. Essentially, "Robby" is a fast and lightweight platform aimed at simulating spiking neural networks and facilitating the control of various types of robotic devices. From a software development point of view it is written to promote reusability, extensibility and flexibility. The time spent writing code is thus minimized and scientists are able to spent more time modeling rather than setting up complex environments and debugging. The framework is written in C++ and adheres to the POSIX standards. Even if C++ is considered to be a high level language, it offers the means to interact at low level with the hardware in an efficient and portable way.

The architecture of "Robby" is modular. It consists of a control structure (the server), a behavioral component (the client) and a commons component (Fig. 1). The server is in strict relation with devices through an instantiation of corresponding drivers. It forwards commands received from clients, and awaits and reads replies from devices. Besides providing the communication functionality it also provides an interface to plot the raw sensory data received

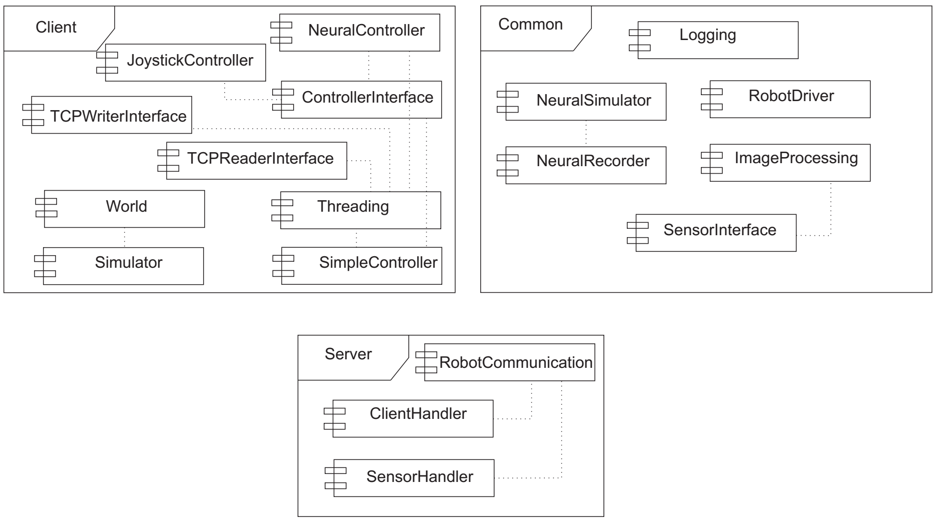
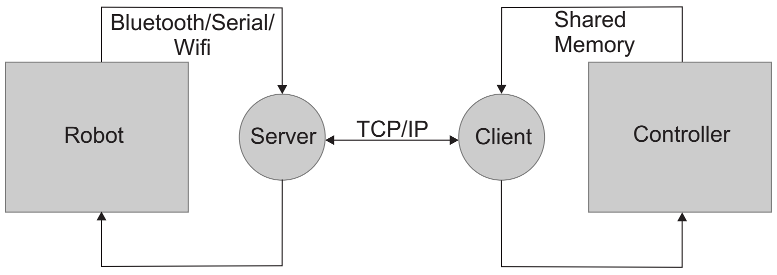


FIGURE 1. "Robby" architecture.

from the devices together with basic communication cycle parameters. The client reads data from the controller and maps it into robot commands which are later sent to the server where they are processed and forwarded. After it sends them to the server it awaits a reply before notifying the controller that the communication cycle is over. A controller implements the *ControllerInterface* and runs in a separate thread. Currently the available controller types are: a *NeuralController* which is a spiking neural network and a *JoystickController* which is useful for direct manipulation of devices. To allow flexibility, other user-defined controller types can be added with the restriction that they must implement the *ControllerInterface*. The common component contains the neural simulator which facilitates the creation and simulation of spiking neural networks [5], device drivers and various image processing algorithms like Laplacian of Gaussian and log-polar filters [6, 7] used to process device video data. Other sensorial controllers can be added provided that they implement the *ControllerInterface*.

This server/client strategy acts as proxy between the client controller and server driver entities to increase flexibility in control and allow the controller and robot to be in different locations with server and client communication mediated through Ethernet. Besides this obvious geographical benefit, such a separation allows the decoupling between the computationally inexpensive communication process and the highly time consuming simulations performed by the controller. What is actually transmitted through the TCP channel are robotic commands embedded into packets. Their aim is to keep communication uniform and enable a seamless control of many types of robots. Each package contains the command together with parameters and optional sensor values. Thus, as long as a physical connection between the server and the robotic device can be initiated multiple controllers and devices can co-exist. "Robby" makes no assumption about the connection medium between the server and the robotic devices, but, as stated previously, a driver needs to be supplied. Efforts are made to increase the number of supported connection types, but currently only drivers for devices with serial connections are provided. Fig. 2 depicts the communication processes inside "Robby". The client and server communicate by using TCP sockets. At the server side, communication between devices and the server could be achieved via bluetooth, wireless or serial depending on the device capabilities. At the client side, communication between the client and the controller is achieved by using a common memory buffer guarded by a critical section. This setup allows the existence of multiple controllers at the same time. Any of the controllers can be replaced by other controllers if they comply with the *ControllerInterface*.



1

FIGURE 2. "Robby" Communication. Communication between the server and client is implemented using the TCP/IP protocol.

While the speed of the simulation itself arguably is not important for detailed modeling of complex biophysical entities and small simulations, in the case of large-scale simulations for the purpose of robotic control it still remains an important constraint. Such simulations of large neural systems consisting of thousands of neurons are heavily time consuming because of the amount of interaction in the network which needs to be evaluated. As memory becomes an inexpensive commodity the trade-off between memory usage and simulation speed needs to be carefully investigated. Recent computing optimization techniques [8, 9] propose the usage of lookup tables to avoid the repeated computation of a value. Thus the runtime computation of what might be expensive is replaced with a simple indexing operation with constant complexity. These approaches increase code size and memory consumption, but the speed gain outweighs the cost. "Robby" implements lookup tables to improve the performance when simulating large neural networks. They are used when computing postsynaptic responses, a process that involves, for some neural models, repetitive evaluations of exponential functions. In addition, when simulating neural networks some operations are independent and can be executed in parallel (for example the update of a neuron membrane potential). These operations are implemented using OpenMP directives [10] to allow multi-threading.

3. Robby as a framework for robot learning

As outlined in previous sections, "Robby" is designed for the simulation of large-scale neural networks for robotic control in a computationally efficient way with as little code as possible. It is able to simulate different types of spiking neurons at different levels of detail. In the current implementation, the available models are the integrate-and-fire [5] and Izhikevich [11]. Because of its simplicity the integrate-and-fire neuron is commonly used in large-scale simulations [12, 13, 14] while the Izhikevich neuron can reproduce the complex behavior observed only at more detailed models while at the same time allowing an efficient implementation [11]. Thus, this selection of neuron models albeit small is sufficient, since complex models are computationally expensive and networks composed out of them would not be feasible as robotic controllers. Different types of static and dynamic synapses together with various plasticity rules (short-term plasticity [15], spike-timing-dependent plasticity (STDP) [16], synaptic scaling [17] or intrinsic neuronal plasticity [18]) are available in order to facilitate learning. In the case of static synapses a fixed current is injected into the postsynaptic neuron at the time of the presynaptic activation while dynamic synapses feature facilitation or depression mechanisms. In addition, different supervised learning rules for spiking neural networks [19, 20] together with reward modulated spike-timing-dependent plasticity [21] are implemented in order to create a framework for reinforcement learning [22].

In the following we present a simple experiment to demonstrate some of the features of “Robby” and their application. Consider the setup presented

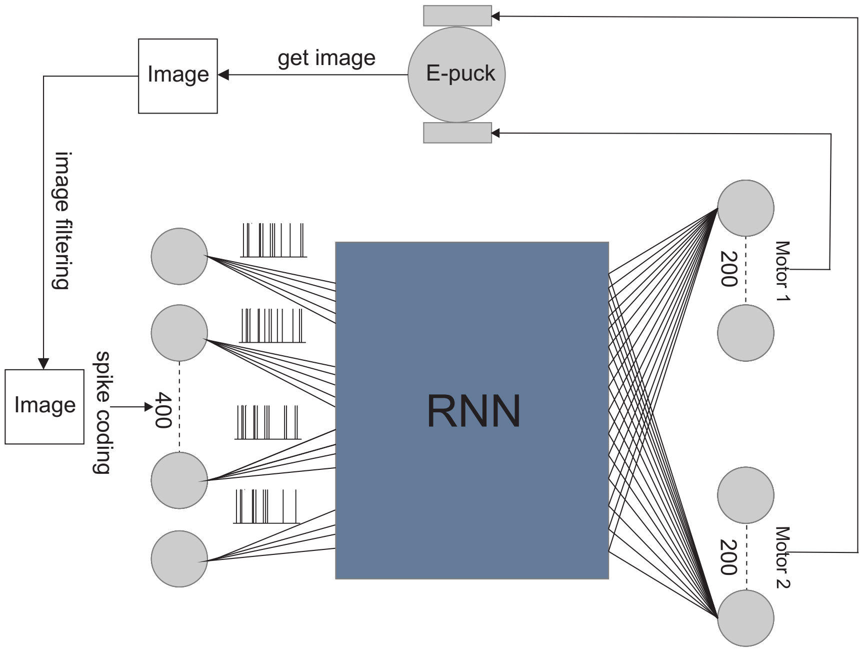


Figure 3. Experiment setup. A spiking neural network controlled an E-puck robot located in a rectangular arena. The network received as input the video information from a camera mounted on top of the robot and controlled the motor activation.

in Fig. 3. An E-puck robot with an externally mounted camera navigated through a rectangular arena which contained no obstacles. The controller of the robot was a spiking neural network with three layers of neurons. The first and third served as sensor and motor neurons respectively.

The input neurons conveyed video information about the environment. More precisely, the pixels of the image received from the camera were averaged to provide input for the 400 input neurons. These activation values were normalized between 0 and 1. The input neurons fired Poisson spike trains with rate proportional to the activation, between 10 and 50 Hz. The 400 output neurons served as motor neurons, 200 for each motor (left and right). Each of these neuron populations is further divided into two 100 neuron pools. These two pools of neurons are assigned to one motor with its speed proportional to their corresponding average firing rate. The spikes were converted to effector activation by integrating them with a leaky accumulator of time constant $\tau = 500$ ms. The activation of two 100 motor neurons populations were averaged to yield the activation of one effector at a timestep. In this antagonistic setup given the activation a_+ and a_- of the two 100 neuron populations, the motor was given a relative command $(a_+ + a_-)$. The network was thus composed out of 400 input neurons and 400 motor neurons. The network also had 1500 hidden neurons. All the neurons were modeled as integrate-and-fire. Each non-input neuron in the network sent connections to 30% of hidden and motor neurons. Input neurons projected onto 45% of the hidden neurons. All the connections were chosen randomly from the uniform distribution. The connections between the first and second layer together with the recurrences within the hidden layer were static spiking synapses while to ones that projected onto the motor neurons were static STDP synapses. In this simple experiment the goal of the robot was to freely explore the environment for 15 seconds. The distance traveled by the robot from the initial point is depicted in Fig. 4. It was computed from the information received from a camera located on top of the environment which recorded every position of the robot. The distribution of synaptic weights of a randomly selected motor neuron at the beginning and at the end of the simulation together with the activation values of the motors received at each timestep are also depicted in Fig. 4. The resulted bimodal weight distribution is consistent with experimental results [16]. The duration of a simulation step which includes the timestep of the neural controller together with the time required to send data to and from the robot has on average the value of 70 ms (see Fig. 4). Due to the low data transfer rates of the robot connection the duration of the communication cycle increases to an average of 400 ms if the video from the E-puck on-board camera is transmitted instead from the external camera on top of the robot. This value is dependent upon the settings of the camera like for example number of pixels or color depth. Although this experiment is simplistic it presents some of the basic features of “Robby” and how they can easily be used in the context of robot learning and control with spiking neural networks.

The experiment was aimed to present the flexibility of the framework that had been developed around *Robby.* As previously stated, this framework has a major advantage of being able to include elements of supervised learning rules with respect to the goal of achieving results in reinforced learning.

The next step that is under development is to apply the neural network framework of Robby to a new business domain in gathering information from

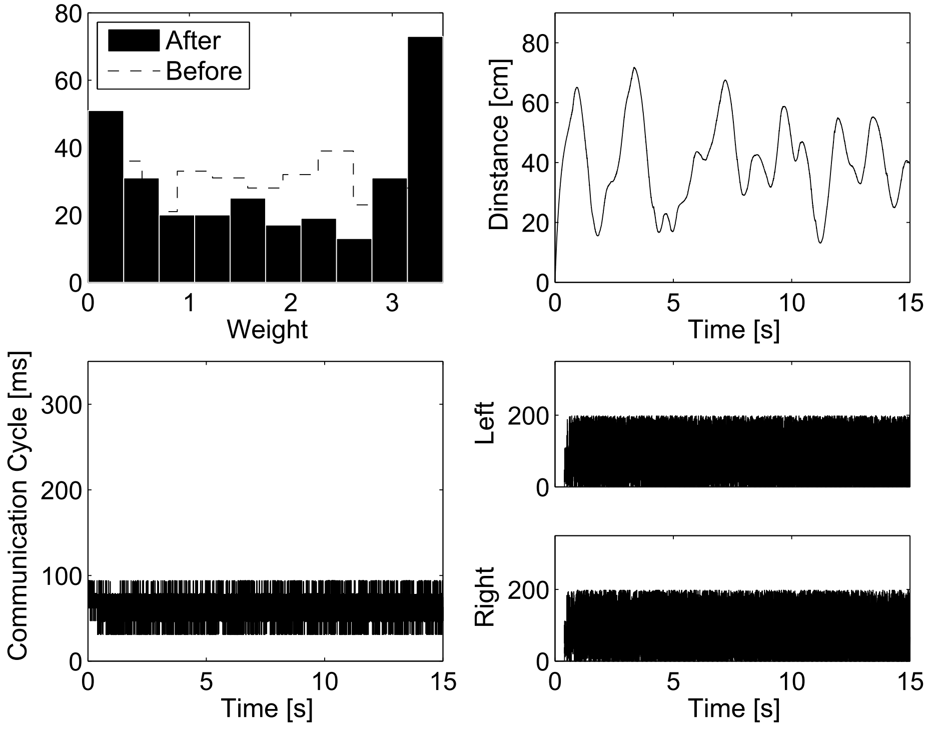


Figure 4. Experiment results. (a) The weight distribution of a randomly selected motor (output) neuron at $t = 0$ s (dashed line) and $t = 15$ s. (b) The distance traveled by the robot from the initial point. (c) The duration of a simulation step in ms. (d) The activation value of each motor.

patterns of mistakes found by analysing test papers, as presented as a theoretical approach in [23]. Results already obtained in developing a mathematical model [24] proved that in order to maximize the robustness of the framework, elements of unsupervised learning (i.e. generation of frequent item sets and determining association rules) need to be linked with elements of supervised learning, in order to prune out coincidental occurrences that are not relevant in terms of knowledge gathered as patterns of mistakes.

So far results from [24] as well as [23] were heavily based on rationales that followed strongly the mathematical model of association rules. This business domain can be extended with a new approach, incorporating elements of neural networks in addition to the existing mathematical model. Using the framework of Robby the goal is to include results gathered from association rules and frequent item set discovery in order to modify existing algorithms based on neural networks that are currently used in credit card fraud detection, in order to study the possibility of predicting frauds in evaluation tests, based on patterns of mistakes.

4. Conclusion

We have introduced a flexible distributed control framework for robotic interaction with spiking neural networks ideal for large-scale simulations. Our aim was to create a multi-threaded, flexible, lightweight framework which promotes code reuse. "Robby" is not intended to be a multi-purpose tool, but it proved to be a convenient tool for quickly exploring new ideas and write experiments with a small amount of code. At the current stage in the development the number of supported robotic platforms and neuron models is still limited. Future plans include support for further commonly used robotic devices and neuron models.

References

[1] B.P. Gerkey, R.T. Vaughan, K. Stoy, A. Howard, G.S. Sukhatme, and M.J. Mataric. *Most Valuable Player: A Robot Device Server for Distributed Control.* In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Wailea, Hawaii, 2001.

[2] P. Fitzpatrick, G. Metta, and L. Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, 2008.

[3] D.S. Blank, D. Kumar, L. Meeden, and H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. *Journal of Educational Resources in Computing*, 2004.

[4] Bruyninckx H. Open robot control software: the orocos project. 2001.

[5] W. Gerstner and W. Kistler. *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge University Press, 2002.

[6] R. Haralick and L. Shapiro. *Computer and Robot Vision.* Addison-Wesley Publishing, 1992.

[7] G. Wolberg and S. Zokai. *Robust image registration using log-polar transform.* IEEE International Conference on Image Processing, 2000.

[8] M. Hall and J. Mayfield. *Improving the Performance of AI Software: Payoffs and Pitfalls in Using Automatic Memoization.* Proceedings of the Sixth International Symposium on Artificial Intelligence, Monterrey, Mexico, 1993.

[9] M. Hall and J.P. McNamee. Improving software performance with automatic memoization. *Johns Hopkins APL Technical Digest*, 18(2), 1997.

[10] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP.* Morgan Kaufmann, 2000.

[11] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14:1569–1572, 2003.

[12] Brunel N. The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, 3:195–206, 2000.

[13] Cessac B and Vieville T. On dynamics of integrate-and-fire neural networks with adaptive conductances. *Frontiers in Computational Neuroscience*, 2, 2008.
[14] Soula H. Alwan A. and Belson G. Learning at the edge of chaos: Temporal coupling of spiking neuron controller for autonomous robotics. 2005.
[15] L.F. Abbott and W.G. Regehr. Synaptic computation. *Nature*, 431:796–803, 2004.
[16] S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3:919–926, 2000.
[17] G.G Turrigiano and S. B. Nelson. Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5:97–107, 2004.
[18] W. Zhang and D. J. Linden. The other side of the engram: Experience-driven changes in neuronal intrinsic excitability. *Nature Reviews Neuroscience*, 4:885–900, 2003.
[19] F. Ponulak. *ReSuMe-new supervised learning method for Spiking Neural Networks*. International Conference on Machine Learning, ICML, 2005.
[20] R. Florian. The chronotron: a neuron that learns to fire temporally-precise spike patterns. *PLoS ONE*, 7(8), 2012.
[21] R. Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502, 2007.
[22] Sutton RS and Barto AG. Reinforcement learning. 1998.
[23] Ban T. Fuzzy computing for complexity level of evaluation tests. *Studia Universitatis Babes-Bolyai, Seria Informatica*, LVIII:81–93, 2013.
[24] Ban T. Generating and assessing test papers complexity using predictions in evolutionary algorithms. 2009.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `rusu@cs.ubbcluj.ro`
E-mail address: `tiberiu@cs.ubbcluj.ro`
E-mail address: `horea@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 2, 2017
DOI: 10.24193/subbi.2017.2.08

METRIC DENOTATIONAL SEMANTICS FOR REMOTE PROCESS DESTRUCTION AND CLONING

ENEIA NICOLAE TODORAN

Abstract. We present a denotational semantics designed with continuations for a concurrent language providing a mechanism for synchronous communication, together with constructions for process creation, remote process destruction and cloning. We accomplish the semantic investigation in the mathematical framework of complete metric spaces.

1. Introduction

We study the semantics of a concurrent language $\mathcal{L}^{pc}_{syn}$ providing a mechanism for synchronous communication, together with constructions for process creation, remote process destruction and cloning. We design a denotational semantics for $\mathcal{L}^{pc}_{syn}$ by using the *continuation semantics for concurrency (CSC)* technique [16]. Following [4], we accomplish the semantic investigation in the mathematical framework of complete metric spaces.

The central characteristic of the CSC technique is the modeling of continuations as application-specific structures of computations, where by computation we understand a partially evaluated denotation (meaning function). The CSC technique was introduced and developed in a series of works [16, 7, 8]. A comparison between CSC and the classic direct approach to concurrency semantics [4] is provided in [16, 7].

To illustrate how synchronous interactions can be modeled with CSC, in Section 3 we start with a language $\mathcal{L}_{syn}$ that is very simple but provides a synchronization mechanism between concurrent components. In Section 3 we provide a denotational semantics designed with CSC for $\mathcal{L}_{syn}$. $\mathcal{L}_{syn}$ is a *uniform* language in the sense that its elementary statements are uninterpreted

Received by the editors: November 30, 2017.

2010 *Mathematics Subject Classification.* 68Q55, 68Q85, 68N15, 68M14.

1998 *CR Categories and Descriptors.* D.3.2 [**Software**]: Programming Languages – *Concurrent, distributed, and parallel languages*; F.3.2 [**Theory of Computation**]: Logics and Meanings of Programs – *Denotational semantics*.

Key words and phrases. metric semantics, continuation semantics for concurrency, remote process destruction, remote process cloning.

symbols taken from a given alphabet. $\mathcal{L}_{syn}^{pc}$ is a *non-uniform* language: in general, in $\mathcal{L}_{syn}^{pc}$ the behavior of an elementary statement depends upon the current state of a program. The terminology *uniform* vs *non-uniform* language is also used, e.g., in [4, 18].

The language $\mathcal{L}_{syn}^{pc}$ is studied in Section 4. $\mathcal{L}_{syn}^{pc}$ provides CSP-like synchronous communication [10]. $\mathcal{L}_{syn}^{pc}$ also provides constructions for process creation, process destruction and process cloning. In $\mathcal{L}_{syn}^{pc}$ a process can not only commit suicide or clone itself, but it can also kill or clone any other process in the system. Process creation is a well known control concept encountered both at operation system level and in concurrent programming. Process destruction and process cloning are operations that can be encountered at operating system level, in some coordination languages [11], or in distributed object oriented and multi agent systems such as Obliq [5] and IBM Java Aglets [12, 19]. The former operation kills a parallel running process and is similar to the "`kill -9`" system call in Unix. The latter operation creates an identical copy of a (parallel) running process.

For the development of our ideas we have chosen the mathematical framework of metric semantics [4], where the main mathematical tool is Banach's fixed point theorem. We need the theory developed in [2] for solving reflexive domain equations as continuations in the CSC approach are elements of a complete space which is the solution of a domain equation where the domain variable occurs in the left-hand side of a function space construction.

1.1. **Contribution.** We present a denotational (mathematical) semantics for a concurrent language $\mathcal{L}_{syn}^{pc}$ incorporating advanced control mechanisms for remote process creation, destruction and cloning. The denotational semantics is designed with metric spaces [4] and continuation semantics for concurrency (CSC) [16], a technique providing sufficient flexibility for handling the advanced control concepts incorporated in $\mathcal{L}_{syn}^{pc}$. Various semantic models for languages with process creation are presented, e.g., in [1, 3, 4, 15]. However, as far as we know, this is the first paper presenting a denotational (mathematical) semantics for remote process destruction and cloning.

2. Preliminaries

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. For X a set we denote by $\mathcal{P}_\pi(X)$ the collection of all subsets of X which have property π. For example, $\mathcal{P}_{finite}(X)$ is the set of all finite subsets of X. If $f : X \to X$ and $f(x) = x$ we call x a *fixed point* of f. When this fixed point is unique (see 2.1) we write $x = fix(f)$. The notions of *partial order* and *total* or *simple order* are assumed to be known. We recall that, given a partially ordered set $(X, \leq_X)$, an element $x \in X$ is said to be *maximal* if there are no elements strictly greater than x in X, that is if $x \leq_X y$ then $y \leq_X x$ in which case $x = y$.

Let $(x \in)X, (y \in)Y, (z \in)Z$, $f \in X \to Y$ and $g \in X \to Y \to Z$. The functions $(f \mid x \mapsto y) : X \to Y$ and $(g \mid x, y \mapsto z) : X \to Y \to Z$ are defined as follows:

$$(f \mid x \mapsto y)(x') = \begin{cases} y & \textsf{if} \quad x' = x \\ f(x') & \textsf{if} \quad x' \neq x \end{cases}$$

$$(g \mid x, y \mapsto z) = (g \mid x \mapsto (g(x) \mid y \mapsto z))$$

$(f \mid x \mapsto y)$ is a variant of the function f which behaves like f almost everywhere, except for point x where $(f \mid x \mapsto y)$ yields y. Instead of $((f \mid x_1 \mapsto y_1) \cdots \mid x_n \mapsto y_n)$ we write $(f \mid x_1 \mapsto y_1 \mid \cdots \mid x_n \mapsto y_n)$.

Following [4] the study presented in this paper takes place in the mathematical framework of 1-bounded complete metric spaces. We assume known the notions of metric and ultrametric space, isometry (distance preserving bijection between metric spaces; we denote it by '$\cong$') and completeness of metric spaces. If (X, d_X), (Y, d_Y) are metric spaces we recall that a function $f : X \to Y$ is a *contraction* if $\exists c \in \mathbf{R}$, $0 \leq c < 1$: $\forall x_1, x_2 \in X$: $d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2)$. Also, f is called *non-expansive* if $d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2)$. We denote the set of all c-contracting (nonexpansive) functions from X to Y by $X \xrightarrow{c} Y$ ($X \xrightarrow{1} Y$).

Theorem 2.1. *(Banach) Let (X, d_X) be a non-empty complete metric space. Each contracting function $f : X \to X$ has a* unique *fixed point.*

For any set $(a, b \in)A$ the so-called *discrete metric d_A* is defined as follows: $d_A(a, b) =$ $\textsf{if } a = b \textsf{ then } 0 \textsf{ else } 1$. (A, d_A) is a complete ultrametric space.

Definition 2.1. *Let $(X, d_X), (Y, d_Y)$ be (ultra) metric spaces. On $(x \in)X$, $(f \in)X \to Y$ (the function space), $((x, y) \in)X \times Y$ (the cartesian product), $(u, v \in)X \sqcup Y$ (the disjoint union) and on $(U, V \in)\mathcal{P}(X)$ (the power set of X) we can define the following metrics:*

(a) $d_{\frac{1}{2} \cdot X} : X \times X \to [0, 1]$, $\quad d_{\frac{1}{2} \cdot X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2)$

(b) $d_{X \to Y} : (X \to Y) \times (X \to Y) \to [0, 1]$
$d_{X \to Y}(f_1, f_2) = sup_{x \in X} d_Y(f_1(x), f_2(x))$

(c) $d_{X \times Y} : (X \times Y) \times (X \times Y) \to [0, 1]$
$d_{X \times Y}((x_1, y_1), (x_2, y_2)) = max\{d_X(x_1, x_2), d_Y(y_1, y_2)\}$

(d) $d_{X \sqcup Y} : (X \sqcup Y) \times (X \sqcup Y) \to [0, 1]$*:*
$d_{X \sqcup Y}(u, v) =$
$\textsf{if } u, v \in X \textsf{ then } d_X(u, v) \textsf{ else if } u, v \in Y \textsf{ then } d_Y(u, v) \textsf{ else } 1$

(e) $d_H : \mathcal{P}(X) \times \mathcal{P}(X) \to [0, 1]$ *is the Hausdorff distance defined by:*

$$d_H(U,V) = max\{sup_{u\in U} d(u,V), sup_{v\in V} d(v,U)\}$$
$$\textit{where } d(u,W) = inf_{w\in W} d(w,w) \textit{ (by convention } sup\emptyset = 0, inf\emptyset = 1\textit{)}.$$

Given a metric space (X, d_X) a subset A of X is called *compact* whenever each sequence in A has a convergent subsequence with limit in A. We will use the abbreviations $\mathcal{P}_{co}(\cdot)$ $(\mathcal{P}_{nco}(\cdot))$ to denote the power set of compact (non-empty and compact) subsets of '$\cdot$'.

Remark 2.1. *Let $(X, d_X), (Y, d_Y), d_{\frac{1}{2}\cdot X}, d_{X\to Y}, d_{X\times Y}$, $d_{X\sqcup Y}$ and d_H be as in definition 2.1. In case d_X, d_Y are ultrametrics, so are $d_{\frac{1}{2}\cdot X}, d_{X\to Y}$, $d_{X\times Y}$, $d_{X\sqcup Y}$ and d_H. If in addition $(X, d_X), (Y, d_Y)$ are complete then $(X, d_{\frac{1}{2}\cdot X})$, $(X \to Y, d_{X\to Y})$, $(X \xrightarrow{1} Y, d_{X\to Y})$, $(X\times Y, d_{X\times Y})$, $(X\sqcup Y, d_{X\sqcup Y})$, $(\mathcal{P}_{co}(X), d_H)$ and $(\mathcal{P}_{nco}(X), d_H)$ are also complete metric spaces. In the sequel we will often suppress the metrics part in domain definitions. In particular we will write $\frac{1}{2}\cdot X$ instead of $(X, d_{\frac{1}{2}\cdot X})$.*

2.1. **Structure of Continuations.** In the CSC approach a continuation is an application-specific structure of computations. Intuitively, a CSC-based model is a semantic formalization of a process scheduler which repeatedly selects and activates computations contained in a continuation [16]. Let $(x \in)\mathbf{X}$ be a complete metric space. Following [16, 7, 8] we define the domain of CSC with the aid of a set $(\alpha \in)Id$ of *(process) identifiers* and we use the following notation:

$$\{\!|\mathbf{X}|\!\} \overset{not.}{=} \mathcal{P}_{finite}(Id) \times (Id \to \mathbf{X})$$

We let π range over $\mathcal{P}_{finite}(Id)$. Let $(\pi, \phi) \in \{\!|\mathbf{X}|\!\}$, where ϕ ranges over $Id \to \mathbf{X}$. We define $id : \{\!|\mathbf{X}|\!\} \to \mathcal{P}_{finite}(Id)$, $id(\pi, \phi) = \pi$. We use the following abbreviations: $(\pi, \phi)(\alpha) \overset{not.}{=} \phi(\alpha)$, $(\pi, \phi) \setminus \pi' \overset{not.}{=} (\pi \setminus \pi', \phi)$, $((\pi, \phi) \mid \alpha \mapsto x)$ $\overset{not.}{=} (\pi \cup \{\alpha\}, (\phi \mid \alpha \mapsto x))$. The operations $id, (\cdot)(\alpha)$, $(\cdot) \setminus \pi$ and $(\cdot \mid \alpha \mapsto x)$ are further explained in [7, 8].

We treat (π, ϕ) as a 'function' with finite graph $\{(\alpha, \phi(\alpha)) \mid \alpha \in \pi\}$, thus ignoring the behaviour of ϕ for any $\alpha \notin \pi$ (π is the 'domain' of the 'function'). Essentially, a structure (π, ϕ) is a finite partially ordered *bag* (or multiset)[1] of computations.

We also use the following notation:

$$[\mathbf{X}] \overset{not.}{=} \mathbf{N} \times (\mathbf{N}^+ \to \mathbf{X})$$

We let ι range over $(\iota \in)\mathbf{N}$ and φ range over $(\varphi \in)\mathbf{N} \to \mathbf{X}$. $\mathbf{N}$ is the set of natural numbers, and $\mathbf{N}^+ = \mathbf{N} \setminus \{0\}$ (the set of positive natural numbers).

[1]A *partially ordered multiset* is a more refined structure; see, e.g., chapter 16 of [4].

We use a structure $(\iota, \varphi) \in [\mathbf{X}]$ to model a *stack* of elements of the type $\mathbf{X}$. We define $[]_{(\cdot)} : [\mathbf{X}]$, $\mathsf{empty}(\cdot) : [\mathbf{X}] \to \mathit{Bool}$, $(\cdot : \cdot) : \mathbf{X} \times [\mathbf{X}] \to [\mathbf{X}]$, $\mathsf{hd}(\cdot) : [\mathbf{X}] \to (\mathbf{X} \cup \{\uparrow\})$ and $\mathsf{tl}(\cdot) : [\mathbf{X}] \to ([\mathbf{X}] \cup \{\uparrow\})$ as follows

$$[]_x = (0, \lambda\iota.x)$$

$$\mathsf{empty}(\iota, \varphi) = (\iota = 0)$$

$$x : (\iota, \varphi) = (\iota + 1, (\varphi \mid \iota + 1 \mapsto x))$$

$$\mathsf{hd}(\iota, \varphi) = \begin{cases} \uparrow & \text{if} \quad \iota = 0 \\ \varphi(\iota) & \text{if} \quad \iota > 0 \end{cases}$$

$$\mathsf{tl}(\iota, \varphi) = \begin{cases} \uparrow & \text{if} \quad \iota = 0 \\ (\iota - 1, \varphi) & \text{if} \quad \iota > 0 \end{cases}$$

Remarks 2.1.

(a) *If we endow Id, $\mathcal{P}_{finite}(Id)$ and $\mathbf{N}$ with discrete ultrametrics, then $\{\!|\mathbf{X}|\!\}$ and $[\mathbf{X}]$ are also a complete ultrametric space. $\{\!|\mathbf{X}|\!\}$ and $[\mathbf{X}]$ are composed spaces built up using the composite metrics of definition 2.1.*

(b) *We use the set $(\alpha \in)Id$ (of process identifiers) together with a function $\nu : \mathcal{P}_{finite}(Id) \to Id$, defined such that $\nu(A) \notin A$, for every $A \in \mathcal{P}_{finite}(Id)$. A possible example of such a set Id and function ν is $Id = \mathbf{N}$ and $\nu(A) = max(A) + 1$, with $\nu(\emptyset) = 0$.*

(c) *Throughout this paper the symbol $\uparrow$ denotes an* undefined value.

(d) $[]_x$ *is an* empty stack*, for any $x \in \mathbf{X}$.*

3. A simple uniform language with synchronization

In this section we apply the CSC technique in the definition of a denotational semantics for a simple concurrent language $\mathcal{L}_{syn}$ with *synchronization.* $\mathcal{L}_{syn}$ is essentially based on Milner's CCS [14]. The language $\mathcal{L}_{syn}$ provides atomic actions, recursion, action prefixing (in the form $a;s$), nondeterministic choice ($s_1 + s_2$) and parallel composition ($s_1 \| s_2$). We assume given two sets $(c \in)Sync$ and $(\overline{c} \in)\overline{Sync} = \{\overline{c} \mid c \in Sync\}$ of *synchronization actions*, and a set $(b \in)IAct$ of *internal actions.* We define $(a \in)Act = IAct \cup Sync \cup \overline{Sync}$, and let τ be a special symbol, $\tau \notin Act$. We also assume given a set $(x \in)PVar$ of procedure variables. Synchronization in $\mathcal{L}_{syn}$ is achieved by the execution of a pair $\overline{c}, c$. First, the $\overline{c}$-step is executed. It is followed immediately by the corresponding c-step. *There are no actions interspersed between $\overline{c}$ and c.* The order is important here: the $\overline{c}$-step is always executed first. A $\overline{c}$-step is an abstract model of a send operation, while a c-step is an abstract model of a receive operation. The approach to recursion in $\mathcal{L}_{syn}$ is based on declarations and guarded statements [4].

Definition 3.1. *(Syntax of $\mathcal{L}_{syn}$)*

(a) *(Statements)* $s(\in Stat) ::= a \mid a;s \mid x \mid s+s \mid s\|s$
(b) *(Guarded statements)* $g(\in GStat) ::= a \mid a;s \mid g+g \mid g\|g$
(c) *(Declarations)* $(D \in)Decl = PVar \to GStat$*; following* [4] *we assume a fixed declaration D!*
(d) *(Programs)* $(\rho \in)Prog = Decl \times Stat$

The denotational semantics function $\mathcal{D}$ is of the type $(\mathcal{D} \in)Sem_D = Stat \to \mathbf{D}$, where $\mathbf{D}$ is defined by the following system of domain equation (isometry between complete metric spaces):

$$\mathbf{D} \cong (Id \times \mathbf{Kont}) \xrightarrow{1} \Gamma \to \mathbf{P}$$

$$(\gamma \in)\Gamma = \{\uparrow_\Gamma\} \cup \overline{Sync}$$

$$(\kappa \in)\mathbf{Kont} = \{\!\!|\, \frac{1}{2} \cdot \mathbf{D} \,|\!\!\}$$

$$(p \in)\mathbf{P} = \mathcal{P}_{nco}((IAct \cup \{\tau\})^\infty)$$

The construction $\{\!\!|\, \frac{1}{2} \cdot \mathbf{D} \,|\!\!\}$ was explained in Section 2.1. In the 'equations' above, the sets Id, $\{\uparrow_\Gamma\} \cup \overline{Sync}$ and $IAct \cup \{\tau\}$ are endowed with the discrete metric (which is an ultrametric). The elements γ of the set $(\gamma \in)(\{\uparrow_\Gamma\} \cup \overline{Sync})$ contain *synchronization information.*

The space $(IAct \cup \{\tau\})^\infty$ contains all finite (possibly empty) and infinite sequences over $(IAct \cup \{\tau\})$. $(IAct \cup \{\tau\})^\infty$ is an instance of the following:

Definition 3.2. *Let $(x \in)\mathbf{X}$ be a nonempty complete space. The space $\mathbf{X}^\infty$ is defined by the equation $\mathbf{X}^\infty \cong \{\epsilon\} \sqcup (\mathbf{X} \times \frac{1}{2} \cdot \mathbf{X}^\infty)$. ϵ models the* empty sequence*. The elements of $\mathbf{X}^\infty$ are finite or infinite sequences over $\mathbf{X}$. Instead of $(x_1, (x_2, \ldots, (x_n, \epsilon) \ldots))$, and $(x_1, (x_2, \ldots))$ we write $x_1 x_2 \ldots x_n$, and $x_1 x_2 \ldots$, respectively. We use the symbol "$\cdot$" as a concatenation operator over sequences. In particular we write $x \cdot q = (x, q)$, for any $x \in \mathbf{X}$ and $q \in \mathbf{X}^\infty$; we also write $x \cdot p = \{x \cdot q \mid q \in p\}$ for any $x \in \mathbf{X}$ and $p \in \mathcal{P}_{nco}(\mathbf{X}^\infty)$.*

Definition 3.3. *We let q range over $\mathbf{Q} = (IAct \cup \{\tau\})^\infty$. We define $+, \oplus : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$ as follows:*

$$p_1 + p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \tau\} \cup \{\tau \mid \tau \in (p_1 \cap p_2)\}$$
$$p_1 \oplus p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in (p_1 \cap p_2)\}$$

For any $\gamma \in (\{\uparrow_\Gamma\} \cup \overline{Sync})$ we also define $\oplus^\gamma : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$ by:

$$p_1 \oplus^\gamma p_2 = \text{if } \gamma = \uparrow_\Gamma \text{ then } p_1 + p_2 \text{ else } p_1 \oplus p_2$$

Definition 3.4. *(Denotational semantics for $\mathcal{L}_{syn}$) Let $C_+ : \mathbf{Kont} \to \mathbf{P}$ and $C_\oplus : \mathbf{Kont} \to (\overline{Sync} \times Id) \to \mathbf{P}$ be given by:*

$$\begin{aligned} C_+(\kappa) &= \text{if } (id(\kappa) = \emptyset) \text{ then } \{\epsilon\} \text{ else } +_{\alpha \in id(\kappa)} \kappa(\alpha)(\alpha, \kappa \setminus \{\alpha\})(\uparrow_\Gamma) \\ C_\oplus(\kappa)(\overline{c}, \overline{\alpha}) &= \text{if } (id(\kappa) \setminus \{\overline{\alpha}\} = \emptyset) \text{ then } \{\epsilon\} \\ &\qquad \text{else } \oplus_{\alpha \in (id(\kappa) \setminus \{\overline{\alpha}\})} \kappa(\alpha)(\alpha, \kappa \setminus \{\alpha\}, \overline{c}) \end{aligned}$$

We define the denotational semantics function $\mathcal{D} : Stat \to \mathbf{D}$ as follows:

$$\begin{aligned} \mathcal{D}(b)(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \tau \cdot b \cdot C_+(\kappa) \text{ else } \{\epsilon\} \\ \mathcal{D}(b;s)(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \tau \cdot b \cdot C_+(\kappa \mid \alpha \mapsto \mathcal{D}(s)) \text{ else } \{\epsilon\} \\ \mathcal{D}(\overline{c})(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \tau \cdot C_\oplus(\kappa)(\overline{c}, \alpha) \text{ else } \{\epsilon\} \\ \mathcal{D}(\overline{c};s)(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \tau \cdot C_\oplus(\kappa \mid \alpha \mapsto \mathcal{D}(s))(\overline{c}, \alpha) \text{ else } \{\epsilon\} \\ \mathcal{D}(c)(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \{\tau\} \\ &\qquad \text{else if } \gamma = \overline{c} \text{ then } \tau \cdot C_+(\kappa) \text{ else } \{\epsilon\} \\ \mathcal{D}(c;s)(\alpha,\kappa)(\gamma) &= \text{if } \gamma = \uparrow_\Gamma \text{ then } \{\tau\} \\ &\qquad \text{else if } \gamma = \overline{c} \text{ then } \tau \cdot C_+(\kappa \mid \alpha \mapsto \mathcal{D}(s)) \text{ else } \{\epsilon\} \\ \mathcal{D}(x)(\alpha,\kappa)(\gamma) &= \mathcal{D}(D(x))(\alpha,\kappa)(\gamma) \\ \mathcal{D}(s_1 + s_2)(\alpha,\kappa)(\gamma) &= \mathcal{D}(s_1)(\alpha,\kappa)(\gamma) \oplus^\gamma \mathcal{D}(s_2)(\alpha,\kappa)(\gamma) \\ \mathcal{D}(s_1 \parallel s_2)(\alpha,\kappa)(\gamma) &= \mathcal{D}(s_1)(\alpha_1, (\kappa \mid \alpha_2 \mapsto \mathcal{D}(s_2)))(\gamma) \oplus^\gamma \\ &\qquad \mathcal{D}(s_2)(\alpha_2, (\kappa \mid \alpha_1 \mapsto \mathcal{D}(s_1)))(\gamma) \end{aligned}$$

where in the last clause $\alpha_1 = \nu(id(\kappa)), \alpha_2 = \nu(id(\kappa) \cup \{\alpha_1\})$.

Let $b_0 \in IAct$ be a distinguished internal action. Let $\kappa_0 = (\emptyset, \lambda\alpha.\mathcal{D}(b_0))$ and $\alpha_0 = \nu(\emptyset)$. We define $\mathcal{D}[\![\cdot]\!] : Stat \to \mathbf{P}$ by

$$\mathcal{D}[\![s]\!] = \mathcal{D}(s)(\alpha_0, \kappa_0)(\uparrow_\Gamma)$$

Following [16], we use the term *process* to denote a computation (partially evaluated denotation) contained in a continuation. Synchronization is modeled as in [16]. Some explanations may help.

- In the yield of $\mathcal{D}$ successful synchronization is modeled by two consecutive τ-steps ($\tau\tau$), which correspond to some pair $\overline{c}, c$ of synchronization actions. Single τ steps are used to model *deadlock*. They can only be produced by unsuccessful synchronization attempts and they are removed from the yield of $\mathcal{D}$ as long as there are alternative computations. This is expressed in the definition of the operator $+$. The operator $\oplus$ describes the behavior of the system in those states where a synchronization attempt occurred. Thus a τ-step has been produced and its pair is expected. No other action is possible. If some process produces a τ step then the computation continues. The computation stops only if all processes are unable to produce the expected τ-step. This is marked by the empty sequence ϵ in the yield of $\mathcal{D}$. In those states where synchronization succeeds by the contribution of some concurrent process, $\oplus$ removes the eventual ϵ's from the final yield of $\mathcal{D}$. It is easy to check that the operators $+, \oplus$ and $\oplus^{\gamma}$ (for any $\gamma \in (\{\uparrow_\Gamma\} \cup \overline{Sync})$) are well-defined, nonexpansive, associative, commutative and idempotent [4].

- Note that the execution of an internal action $b \in IAct$ is also preceded by a τ-step. In the CSC approach this is necessary only if we want to obtain a denotational model which is *correct* with respect to a corresponding operational model; further explanations are provided in [16].
- A process can not synchronize with itself. The function $C_\oplus$ receives as parameter the process identifier of the current process, and chooses some other process for synchronization.

Remark 3.1. *$\mathcal{D}$ can be formally defined as fixed point of an appropriate higher order contraction. In Section 4 we give the details of such a proof for a more complex language. For $\mathcal{L}_{syn}$, the proof can proceed by induction on the following complexity measure: $c : Stat \to \mathbf{N}$, $c(a) = c(a;s) = 1, c(x) = 1 + c(D(x))$, $c(s_1 + s_2) = c(s_1 \| s_2) = 1 + max\{c(s_1), c(s_2)\}$; the mapping c is well-defined due to our restriction to guarded recursion* [4].

Examples 3.1.

- $\mathcal{D}[\![\bar{c} \parallel c]\!] = \mathcal{D}(\bar{c} \parallel c)(\alpha_0, \kappa_0)(\uparrow_\Gamma)$
$= \mathcal{D}(\bar{c})(\alpha_1, (\kappa_0 \mid \alpha_2 \mapsto \mathcal{D}(c)))(\uparrow_\Gamma) \oplus^{\uparrow_\Gamma} \mathcal{D}(c)(\alpha_2, (\kappa_0 \mid \alpha_1 \mapsto \mathcal{D}(\bar{c})))(\uparrow_\Gamma)$
$= \tau \cdot C_\oplus(\kappa_0 \mid \alpha_2 \mapsto \mathcal{D}(c))(\bar{c}, \alpha_1) + \{\tau\}$
$= \tau \cdot \mathcal{D}(c)(\alpha_2, \kappa'_0)(\bar{c}) + \{\tau\} = \tau \cdot \tau \cdot C_+(\kappa'_0) + \{\tau\}$ $\quad$ *[$id(\kappa'_0) = \emptyset$]*
$= \{\tau\tau\} + \{\tau\} = \{\tau\tau\}$
*where $\alpha_0 \in Id$ and $\kappa_0 \in$ **Kont** are as in Definition 3.4, $\alpha_1 = \nu(id(\kappa_0))$, $\alpha_2 = \nu(id(\kappa_0) \cup \{\alpha_1\})$, and $\kappa'_0 = (\kappa_0 \mid \alpha_2 \mapsto \mathcal{D}(c)) \setminus \{\alpha_2\}$.*
- $\mathcal{D}[\![b_1 + b_2]\!] = \{\tau b_1, \tau b_2\}$
- $\mathcal{D}[\![(\bar{c} + b) \parallel c]\!] = \{\tau\tau, \tau b \tau\}$

4. Remote Process Destruction and Cloning

The CSC technique can be used to design denotational (compositional) semantics for various advanced control concepts, including: synchronous and asynchronous communication [16], multiparty interactions [17, 8], maximal parallelism [6, 9] and systems with dynamic hierarchical structure [9]. In this work we use CSC to design a denotational semantics for an imperative concurrent language $\mathcal{L}^{pc}_{syn}$ providing synchronous CSP-like synchronous communication [10] together with constructions for process creation, and remote process destruction and cloning.

We assume given a class of *variables* $(v \in)Var$, a set $(e \in)Exp$ of *expressions*, and a set $(x \in)PVar$ of *procedure variables.* We also assume given a class

$(c \in)Chan$ of *communication channels.* We assume that the evaluation of an expression $(e \in Exp)$ always terminates and delivers a value in some set $(\alpha \in)Val$. The set Val of values is assumed to be countably infinite.

Remark 4.1. *The constructs for process control (*new, kill, clone*) operate with* process identifiers*, which are elements of the given countably infinite set* $(\alpha \in)Id$ *introduced in Section 2.1. For simplicity (and without loss of generality), in the rest of this section* we assume that the class $(\alpha \in)Id$ of process identifiers coincides with the class Val of vales: $Id = Val$.[2]

Definition 4.1. *We define the syntax of* $\mathcal{L}^{pc}_{syn}$ *by the following components:*

(a) *(Statements)* $s(\in Stat) ::= a \mid x \mid s + s \mid s;s$

(b) *(Guarded statements)* $g(\in GStat) ::= a \mid g + g \mid g;s$

(c) *(Declarations)* $(D \in)Decl = PVar \to GStat$

(d) *(Programs)* $(\rho \in)Prog = Decl \times Stat$

where $a(\in AStat)$ *is given by:*

$$a ::= \mathsf{skip} \mid v := e \mid c!e \mid c?v \mid v := \mathsf{new}(s) \mid \mathsf{kill}(e) \mid v := \mathsf{clone}(e)$$

We assume an approach to recursion based on declarations and guarded statements (as in the previous section). Without loss of generality [4], in the rest of this section *we assume a fixed declaration $D \in Decl$ and in all contexts we refer to this fixed D.* In $\mathcal{L}^{pc}_{syn}$ we have assignment $(v := e)$, recursion, sequential composition $(s; s)$, nondeterministic choice $(s + s)$, CSP-like synchronous communication (given by the statements $c!e$ and $c?v$) and constructions for process creation $(v := \mathsf{new}(s))$, remote process destruction $(\mathsf{kill}(e))$ and remote process cloning $(v := \mathsf{clone}(e))$. The net effect of a construct $v := \mathsf{new}(s)$ is to create a new process with body s that runs in parallel with all other processes in the system. A new *process identifier* is automatically generated and assigned to v. In a $\mathsf{kill}(e)$ or $v := \mathsf{clone}(e)$ statement, the expression e is evaluated to some value α, which is interpreted as a process identifier.[3] The execution of a statement $\mathsf{kill}(e)$ kills the parallel runing process with identifier α. When a $v := \mathsf{clone}(e)$ statement is executed, a new process - identical to the one with identifier α - is created and its identifier is assigned to v. The constructs $c!e$ and $c?v$ are as in Occam [13]. Synchronized execution of two actions $c!e$ and $c?v$ occurring in two parallel processes, results in the transmission of the current value of e along the channel c from the process executing

[2]For example, we could put $Id = Val = \mathbf{N}$ ($\mathbf{N}$ is the set of natural numbers) in which case Exp would be a class of numeric expressions. However, it is straightforward to extend the semantic model by using different support sets for the class of values and the class of process identifiers.

[3]The statements $\mathsf{kill}(e)$ and $v := \mathsf{clone}(e)$ are inoperative if the value of the expression e (in the current state) is not a valid process identifier.

the $c!e$ (send) statement to the process executing the $c?v$ (receive) statement. The latter assigns the received value to the variable v.

4.1. **Denotational Semantics.** In the definition of the denotational semantics for $\mathcal{L}_{syn}^{pc}$ we use the set $(\alpha \in)Id$ of process identifiers and the constructions $\{\!| \cdot |\!\}$ and $[\cdot]$ introduced in Section 2.1. In $\mathcal{L}_{syn}^{pc}$ each process has its own local data. Values can be communicated between processes but there is no shared memory area. We define a class $(\sigma \in)State = Id \to Var \to Val$ of (distributed) *states*. The meaning of (the local) variables of a process with identifier α is given by $\sigma(\alpha)$. The evaluation of expressions in $\mathcal{L}_{syn}^{pc}$ is modeled by a given valuation $V : Exp \to (Var \to Val) \to Val$. We recall that we take $Val = Id$.

We design a denotational semantics function $\mathcal{D}$ for $\mathcal{L}_{syn}^{pc}$. The type of $\mathcal{D}$ is $\mathcal{D} : Sem_D = Stat \to \mathbf{D}$, where:

$$(\psi \in)\mathbf{D} \cong (Id \times \mathbf{Kont}) \xrightarrow{1} (\Omega \times State) \to \mathbf{P}_D$$

$$(\kappa \in)\mathbf{Kont} = \{\!|\ [\frac{1}{2} \cdot \mathbf{D}]\ |\!\}$$

$$(\omega \in)\Omega = \{\uparrow_\Omega\} \cup (Chan \times Val)$$

$$(q \in)\mathbf{Q} = (\{\tau\} \cup State)^\infty$$

$$(p \in)\mathbf{P}_D = \mathcal{P}_{nco}(\mathbf{Q})$$

The construction $(\{\tau\} \cup State)^\infty$ was introduced in Definition 3.2. We assume that $\tau \notin State$. For easier readability, we denote typical elements (c, α) of $Chan \times Val (\subseteq \Omega)$ by $c!\alpha$.

Remark 4.2. *In the definition of the domain of continuatins* **Kont** *we use the constructions* $\{\!| \cdot |\!\}$ *and* $[\cdot]$ *introduced in Section 2.1. A continuation of type* $\mathbf{Kont} = \{\!|\ [\frac{1}{2} \cdot \mathbf{D}]\ |\!\}$ *is essentially a bag (multiset) of stacks of computations. An element of the type* $[\frac{1}{2} \cdot \mathbf{D}]$ *is a stack of computations (denotations). In this section we use the term* process *when referring to an element of the type* $[\frac{1}{2} \cdot \mathbf{D}]$. *A stack of computations of the type* $[\frac{1}{2} \cdot \mathbf{D}]$ *represents a process (an execution thread with a local state) executed in parallel with all the other processes contained in a continuation.*

The domain of computations (denotations) $\mathbf{D}$ is given by a recursive domain equation. In the domain equations given above, the sets Id, Ω, and $State$ (and $\{\tau\} \cup State$) are endowed with discrete metrics (which are ultrametrics).

According to [2] the solutions for **D** and **Kont** are obtained as complete ultrametric spaces.

The denotational semantics function is defined below as the fixed point of an appropriate higher-order mapping. In Definition 4.2 the operators presented in Definition 3.3 are adapted to $\mathcal{L}^{pc}_{syn}$.

Definition 4.2. 4.2.1 Definition $+, \oplus : \mathbf{P}_D \times \mathbf{P}_D \to \mathbf{P}_D$ *are defined by:*

$$
\begin{aligned}
p_1 + p_2 &= \{q \mid q \in p_1 \cup p_2, q \neq \tau\} \cup \{\tau \mid \tau \in (p_1 \cap p_2)\} \\
p_1 \oplus p_2 &= \{q \mid q \in p_1 \cup p_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in (p_1 \cap p_2)\}
\end{aligned}
$$

For any $\omega \in \Omega$ *we also define* $\oplus^{\omega} : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$ *by:*

$$p_1 \oplus^{\omega} p_2 = \textsf{if } \omega = \uparrow_{\Omega} \textsf{ then } p_1 + p_2 \textsf{ else } p_1 \oplus p_2$$

The operators $+$, $\oplus$ and $\oplus^{\omega}$ are well-defined, nonexpansive, associative, commutative and idempotent [4].

Definition 4.3. *(Denotational semantics* $\mathcal{D}$ *for* $\mathcal{L}^{pc}_{syn}$*) We define* $C_+ : \mathbf{Kont} \to$ *State* $\to \mathbf{P}_D$ *and* $C_{\oplus} : \mathbf{Kont} \to (\Omega \times Id \times State) \to \mathbf{P}_D$ *as follows:*

```
$C_+(\kappa)(\sigma) =$
        let $\overline{\kappa} = \kappa \setminus \{\alpha' \mid \textsf{empty}(\kappa(\alpha'))\}$ in
        if $id(\overline{\kappa}) = \emptyset$ then $\{\epsilon\}$
        else $+_{\alpha \in id(\overline{\kappa})}$ $\textsf{hd}(\overline{\kappa}(\alpha))\,(\alpha, (\overline{\kappa} \mid \alpha \mapsto \textsf{tl}(\overline{\kappa}(\alpha))\,))(\uparrow_{\Omega}, \sigma)$
$C_{\oplus}(\kappa)(\omega, \overline{\alpha}, \sigma) =$
        let $\overline{\kappa} = \kappa \setminus \{\alpha' \mid \textsf{empty}(\kappa(\alpha'))\}$ in
        if $id(\overline{\kappa}) \setminus \{\overline{\alpha}\} = \emptyset$ then $\{\epsilon\}$
        else $\oplus_{\alpha \in (id(\overline{\kappa}) \setminus \{\overline{\alpha}\})}$ $\textsf{hd}(\overline{\kappa}(\alpha))\,(\alpha, (\overline{\kappa} \mid \alpha \mapsto \textsf{tl}(\overline{\kappa}(\alpha))\,))(\omega, \sigma)$
```

Let $u \in UStat$ *be given by:*

$$u ::= \textsf{skip} \mid v := e \mid c!e \mid v := \textsf{new}(s) \mid \textsf{kill}(e) \mid v := \textsf{clone}(e)$$

The statements of the subclass $UStat \subseteq AStat$ *can not be executed in those states where a communication attempt occurred.*

Let α^* *be some distinguished value* $(\alpha^* \in) Val$. *Let* ψ^* *be some distinguished computation* $\psi^* \in \mathbf{D}$.

We define $\Psi \in Sem_D \to Sem_D$ *for* $S \in Sem_D (= Stat \to \mathbf{D})$ *by:*

$$\begin{array}{rcl}
\Psi(S)(u)(\alpha,\kappa)(c!\alpha,\sigma) & = & \{\epsilon\} \quad \text{for any } \ u \in UStat \\
\Psi(S)(\mathsf{skip})(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot\sigma\cdot C_+(\kappa)(\sigma) \\
\Psi(S)(v := e)(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot\sigma_{assign}\cdot C_+(\kappa)(\sigma_{assign}) \\
\Psi(S)(c!e)(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot C_\oplus(\kappa)(c!V(e)(\sigma(\alpha)),\alpha,\sigma) \\
\Psi(S)(c?v)(\alpha,\kappa)(\omega,\sigma) & = & \begin{cases} \{\tau\} & \text{if} \quad \omega = \uparrow_\Omega \\ \sigma_{rcv}\cdot C_+(\kappa)(\sigma_{rcv}) & \text{if} \quad \omega = c!\alpha' \\ \{\epsilon\} & \text{if} \quad \omega = c'!\alpha' \\ & \qquad\quad c' \neq c \end{cases} \\
\Psi(S)(v := \mathsf{new}(s))(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot\sigma_{new}\cdot C_+(\kappa \mid \alpha_\nu \mapsto S(s) : []_{\psi^*})(\sigma_{new}) \\
 & & \text{where } \alpha_\nu = \nu(id(\kappa)) \\
\Psi(S)(\mathsf{kill}(e))(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot\sigma\cdot C_+(\kappa \setminus \{\overline{\alpha}\})(\sigma) \\
 & & \text{where } \overline{\alpha} = V(e)(\sigma(\alpha)) \\
\Psi(S)(v := \mathsf{clone}(e))(\alpha,\kappa)(\uparrow_\Omega,\sigma) & = & \tau\cdot\sigma_{clone}\cdot C_+(\kappa \mid \alpha_\nu \mapsto \kappa(\overline{\alpha}))(\sigma_{clone}) \\
 & & \text{where } \alpha_\nu = \nu(id(\kappa)) \\
 & & \qquad\quad \overline{\alpha} = V(e)(\sigma(\alpha)) \\
\Psi(S)(x)(\alpha,\kappa)(\omega,\sigma) & = & \Psi(S)(D(x))(\alpha,\kappa)(\omega,\sigma) \\
\Psi(S)(s_1 + s_2)(\alpha,\kappa)(\omega,\sigma) & = & \Psi(S)(s_1)(\alpha,\kappa)(\omega,\sigma) \oplus^\omega \\
 & & \Psi(S)(s_2)(\alpha,\kappa)(\omega,\sigma) \\
\Psi(S)(s_1; s_2)(\alpha,\kappa)(\omega,\sigma) & = & \Psi(S)(s_1)(\alpha,(\kappa \mid \alpha \mapsto S(s_2) : \kappa(\alpha)))(\omega,\sigma)
\end{array}$$

where $\sigma_{assign} = (\sigma \mid \alpha, v \mapsto V(e)(\sigma(\alpha)))$ *in the semantic equation for* $v := e$, $\sigma_{rcv} = (\sigma \mid \alpha, v \mapsto \alpha')$ *in the (second) semantic equation for* $c?v$, $\sigma_{new} = ((\sigma \mid \alpha, v \mapsto \alpha_\nu) \mid \alpha_\nu \mapsto \lambda v'.\alpha^*)$ *in the semantic equation for* $v := \mathsf{new}(s)$*, and* $\sigma_{clone} = ((\sigma \mid \alpha, v \mapsto \alpha_\nu) \mid \alpha_\nu \mapsto \sigma(\overline{\alpha}))$ *in the equation for* $v := \mathsf{clone}(e)$*.*

We recall that ψ^* *is a distinguished computation* $\psi^* \in \mathbf{D}$*. The notation* $[]_{\psi^*}$ *was introduced in Section 2.1 as a mean to construct an empty stack.*

We put $\mathcal{D} = fix(\Psi)$*. Let* $\alpha_0 = \nu(\emptyset)$ *and* $\kappa_0 = (\{\alpha_0\}, \lambda\alpha.[]_{\psi^*})$*. We define* $\mathcal{D}[\![\cdot]\!] : Stat \to State \to \mathbf{P}_D$ *by*

$$\mathcal{D}[\![s]\!](\sigma) = \mathcal{D}(s)(\alpha_0,\kappa_0)(\uparrow_\Omega,\sigma)$$

The technique used to model synchronous interactions was introduced in [16] and was already illustrated in this paper in Section 3. In the semantic equation for process creation $v := \mathsf{new}(s)$ a new process executing the computation $\mathcal{D}(s)$ is started in parallel with all processes contained in the continuation. Process destruction is handled in the sematic equation for $\mathsf{kill}(e)$ by removing the process with identifier $\overline{\alpha} = V(e)(\sigma(\alpha))$, where α is the identifier of the process which executes the statement $\mathsf{kill}(e)$ in the current state σ. Process cloning is handled in the sematic equation for $v := \mathsf{clone}(e)$ by starting a clone of the process with identifier $\overline{\alpha} = V(e)(\sigma(\alpha))$, where α is the identifier of the process which executes the statement $v := \mathsf{clone}(e)$ and σ is the current state of the distributed system.

The denotational semantics $\mathcal{D}$ is defined as the (unique) fixed point of the higher-order mapping Ψ. Definition 4.3 is justified by Lemma 4.1, Lemma 4.2 and Banach's fixed point theorem 2.1. Similar lemmas are given in [16]. We omit the proof of 4.1. To illustrate a proof technique specific of metric semantics we present the proof of Lemma 4.2(c) by using and inductive argument. For inductive reasonings, in the case of the language $\mathcal{L}^{pc}_{syn}$ one can use the following complexity measure $c_s : Stat \to \mathbf{N}$ $c_s(a) = 1$, for any $a \in AStat$, $c_s(x) = 1 + c_s(D(x))$, $c_s(s_1; s_2) = 1 + c_s(s_1)$ and $c_s(s_1 + s_2) = 1 + max\{c_s(s_1), c_s(s_2)\}$. The mapping c_s is well defined due to our restriction to guarded recursion [4].

Lemma 4.1. *The mappings C_+ and $C_\oplus$ (as introduced in Definition 4.3) are well-defined. Also, for any $\kappa_1, \kappa_2 \in$ **Kont** we have:*

(a) $d(C_+(\kappa_1,)C_+(\kappa_2)) \leq 2 \cdot d(\kappa_1, \kappa_2)$ *and*
(b) $d(C_\oplus(\kappa_1), C_\oplus(\kappa_2)) \leq 2 \cdot d(\kappa_1, \kappa_2)$.

Lemma 4.2. *For any $S \in Sem_D, s \in Stat, \alpha \in Id, \kappa \in \mathbf{Kont}, \omega \in \Omega, \sigma \in State$:*

(a) $\Psi(S)(s)(\alpha, \kappa)(\omega, \sigma) \in \mathbf{P}_D$ *(it is well-defined),*
(b) $\Psi(S)(s)$ *is nonexpansive (in κ) and*
(c) Ψ *is $\frac{1}{2}$-contractive in S.*

Proof We only prove Lemma 4.2(c). It suffices to show that

$$d(\Psi(S_1)(s)(\alpha, \kappa)(\omega, \sigma), \Psi(S_2)(s)(\alpha, \kappa)(\omega, \sigma)) \leq \tfrac{1}{2} \cdot d(S_1, S_2).$$

We proceed by induction on $c_s(s)$. Two subcases.

Case $s = (v := \mathsf{clone}(e))$

$$\begin{aligned}&d(\Psi(S_1)(v := \mathsf{clone}(e))(\alpha, \kappa)(\omega, \sigma),\\ &\quad \Psi(S_2)(v := \mathsf{clone}(e))(\alpha, \kappa)(\omega, \sigma))\\ &= 0 \leq \tfrac{1}{2} \cdot d(S_1, S_2)\end{aligned}$$

Case $s = x$

$$\begin{aligned}&d(\Psi(S_1)(x))(\alpha, \kappa)(\omega, \sigma), \Psi(S_2)(x)(\alpha, \kappa)(\omega, \sigma))\\ &= d(\Psi(S_1)(D(x)))(\alpha, \kappa)(\omega, \sigma),\\ &\qquad \Psi(S_2)(D(x))(\alpha, \kappa)(\omega, \sigma))\\ &\qquad\qquad [c_s(D(x)) < c_s(x), \text{ ind. hypothesis}]\\ &\leq \tfrac{1}{2} \cdot d(S_1, S_2)\end{aligned}$$

□

Remark 4.3. *When the CSC technique is employed in the semantic design, (groups of) computations contained in a continuations can be manipulated as data. By expoiting this facility in this paper we offer a denotational (compositional) semantics for remote process destruction and cloning. Our attempts* *to model such remote control operations by using only classic compositional techniques have failed. In the classic direct approach to concurrency* [4] *the semantic designer defines the various operators for parallel composition as functions that manipulate final semantic values belonging to some power domain construction. The meaning of a process appears in the final yield of a denotational mapping interleaved with the meanings of other parallel processes. The problem with this approach is that the remote control operations considered in this paper may be executed long after the creation of the process. It it does not seem possible to extract somehow the meaning of a process from an element of a power domain, in order to remove (kill) it or to clone it. On the contrary, in the CSC approach the semantic designer operates with partially evaluated meaning functions (contained in continuations) which can easily be manipulated to model such remote control operations.*

Example 4.1. *In this example we assume that Exp is a class of numeric expressions and put* $Val = \mathbf{N}$*. Consider the* $\mathcal{L}^{pc}_{syn}$ *program statement* $s (\in Stat)$

$$s = v_{new} := \mathsf{new}(c!1); ((v_{clone} := \mathsf{clone}(v_{new});\ \mathsf{kill}(v_{new}); c?v) + v := 2)$$

Let $\sigma \in \Sigma$, $\alpha^{new}_{\nu} = \nu(id(\kappa_0)) = \nu(\{\alpha_0\})$ *and* $\alpha^{clone}_{\nu} = \nu(\{\alpha_0, \alpha^{new}_{\nu}\})$*, where* $\kappa_0 \in \mathbf{Kont}$ *and* $\alpha_0 \in Id$ *are as in Definition 4.3. Let* $\overline{\sigma}_{new} = ((\sigma \mid \alpha_0, v_{new} \mapsto \alpha^{new}_{\nu}) \mid \alpha^{new}_{\nu} \mapsto \lambda v'.\alpha^*)$, $\overline{\sigma}_{clone} = ((\overline{\sigma}_{new} \mid \alpha_0, v_{clone} \mapsto \alpha^{clone}_{\nu}) \mid \alpha^{clone}_{\nu} \mapsto \overline{\sigma}_{new}(\alpha^{new}_{\nu}))$, $\overline{\sigma}_{kill} = \overline{\sigma}_{clone}$, $\overline{\sigma}_{rcv} = (\overline{\sigma}_{kill} \mid \alpha_0, v \mapsto 1)$, $\overline{\sigma}_{assign} = (\overline{\sigma}_{new} \mid \alpha_0, v \mapsto 2)$*. It is easy to check that:*

$$\mathcal{D}[\![s]\!](\sigma) = \mathcal{D}(s)(\alpha_0, \kappa_0)(\uparrow_\Omega, \sigma) = \tau \cdot \overline{\sigma}_{new} \cdot \mathcal{D}(s_1)(\alpha_0, \kappa_1)(\uparrow_\Omega, \overline{\sigma}_{new})$$

where $s_1 = (v_{clone} := \mathsf{clone}(v_{new});\ \mathsf{kill}(v_{new}); c?v) + v := 2$*, and* $\kappa_1 = (\kappa_0 \mid \alpha_0 \mapsto [\,]_{\psi^*} \mid \alpha^{new}_{\nu} \mapsto \mathcal{D}(c!1) : [\,]_{\psi^*})$*. We have:*

$$\begin{aligned}
&\mathcal{D}(s_1)(\alpha_0, \kappa_1)(\uparrow_\Omega, \overline{\sigma}_{new})\\
&= \mathcal{D}(v_{clone} := \mathsf{clone}(v_{new});\ \mathsf{kill}(v_{new}); c?v)(\alpha_0, \kappa_1)(\uparrow_\Omega, \overline{\sigma}_{new}) \oplus^{\uparrow_\Omega}\\
&\quad \mathcal{D}(v := 2)(\alpha_0, \kappa_1)(\uparrow_\Omega, \overline{\sigma}_{new})\\
&= \mathcal{D}(v_{clone} := \mathsf{clone}(v_{new}))(\alpha_0, \kappa_2)(\uparrow_\Omega, \overline{\sigma}_{new}) + \{\tau \overline{\sigma}_{assign} \tau\}
\end{aligned}$$

where $\kappa_2 = (\kappa_0 \mid \alpha_0 \mapsto \mathcal{D}(\,\mathsf{kill}(v_{new}); c?v) : [\,]_{\psi^*} \mid \alpha^{new}_{\nu} \mapsto \mathcal{D}(c!1) : [\,]_{\psi^*})$*. Note that a deadlock is detected after the execution of the assignment statement* $v := 2$ *because (when* $v := 2$ *is selected for execution) the computation* $\mathcal{D}(c!1)$ *contained in the continuation has no synchronization counterpart .*

The execution of the statement $v_{clone} := \mathsf{clone}(v_{new})$ *creates an copy of the process with identifier* α^{new}_{ν} *which will run in parallel with the other processes.*

$$\mathcal{D}(v_{clone} := \mathsf{clone}(v_{new}))(\alpha_0, \kappa_2)(\uparrow_\Omega, \overline{\sigma}_{new}) = \tau \cdot \overline{\sigma}_{clone} \cdot C_{+}(\kappa_3)(\overline{\sigma}_{clone})$$

where $\kappa_3 = (\kappa_0 \mid \alpha_0 \mapsto \mathcal{D}(\,\mathsf{kill}(v_{new}); c?v) : [\,]_{\psi^*} \mid \alpha^{new}_{\nu} \mapsto \mathcal{D}(c!1) : [\,]_{\psi^*} \mid \alpha^{clone}_{\nu} \mapsto \mathcal{D}(c!1) : [\,]_{\psi^*})$*.*

After the cloning operation, the execution of the statement $\mathsf{kill}(v_{new})$ *removes the process with identifier* α_v^{new} *from the continuation. Next, after a synchronization step the computation terminates. In the end we obtain:*

$$\mathcal{D}[\![s]\!](\sigma) = \{\tau\overline{\sigma}_{new}\tau\overline{\sigma}_{clone}\tau\overline{\sigma}_{kill}\tau\overline{\sigma}_{rcv}, \tau\overline{\sigma}_{new}\tau\overline{\sigma}_{assign}\tau\}$$

where $s = v_{new} := \mathsf{new}(c!1); ((v_{clone} := \mathsf{clone}(v_{new}); \mathsf{kill}(v_{new}); c?v) + v := 2)$.

5. Conclusion

The CSC technique can be used to design denotational (compositional) semantics for various advanced control concepts, including: synchronous and asynchronous communication [16, 7], maximal parallelism [6, 9], and multiparty interactions [17, 8]. In this work we present a denotational semantics designed with metric spaces and continuations for a concurrent language providing constructions for CSP-like synchronous communication in combination with constructions for process creation, and remote process control (remote process destruction and cloning). Various denotational models for process creation have been developed [1, 15, 3, 4]. However, we are not aware of any paper reporting a denotational model for remote process destruction and process cloning.

The use of continuation semantics for concurrency (CSC) technique [16, 7] proved to be fruitful. In the CSC approach the semantics of remote process destruction and cloning can easily be modeled by appropriate manipulations of the computations contained in continuations. We think that the CSC technique could be used to model remote process control operations in combination with various interaction mechanisms, including remote procedure call and ADA-like rendezvous. The rendezvous programming concept was studied by using techniques from metric semantics in several papers [15, 3, 4]. In the near future we intend to study the formal relationship between the denotational and the operational semantics of remote process destruction and cloning also by using techniques from metric semantics [4].

References

[1] P. America and J.W. de Bakker, Designing Equivalent Semantic Models for Process Creation, *Theoretical Computer Science*, vol. 60, 1988, pp. 109–176.

[2] P. America and J.J.M.M. Rutten, Solving Reflexive Domain Equations in a Category of Complete Metric Spaces, *Journal of Computer and System Sciences*, vol. 39, 1989, pp. 343–375.

[3] J.W. de Bakker and E.P. de Vink, Rendez-vous with Metric Semantics, *New Generation Computing*, vol. 12, 1993, pp. 53–90.

[4] J.W. de Bakker and E.P. de Vink, *Control Flow Semantics,* MIT Press, 1996.

[5] L. Cardelli, A Language with Distributed Scope, *Proceedings of the 22nd Annual ACM Symposium on Principles of Programming Languages*, pp. 286–297, ACM Press, 1995.

[6] G Ciobanu and EN Todoran, Relating Two Metric Semantics for Parallel Rewriting of Multisets, *Proceedings of 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*, pp. 273–280, IEEE Computer Press, 2012.

[7] G. Ciobanu and E.N. Todoran, Continuation Semantics for Asynchronous Concurrency, *Fundamenta Informaticae*, vol. 131(3–4), 2014, pp. 373–388.

[8] G Ciobanu and EN Todoran, Continuation Semantics for Concurrency with Multiple Channels Communication, *Formal Methods and Software Engineering - Proceedings of 17th International Conference on Formal Engineering Methods (ICFEM 2015)*, *Lecture Notes in Computer Science*, vol. 9407, 2015, pp. 400–416.

[9] G Ciobanu and EN Todoran, Continuation Semantics for Dynamic Hierarchical Systems, *Proceedings of 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2015)*, pp. 281–288, IEEE Computer Press, 2015.

[10] C.A.R. Hoare, *Communicating Sequential Processes,* Prentice Hall, 1985.

[11] A.A. Holzbacher, A Software Environment for Concurrent Coordinated Programming, *Proc. 1st Int. Conference on Coordination Languages and Systems*, *Lecture Notes in Computer Science*, vol. 1061, 1996, pp. 249–267.

[12] D.B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets,* Addison Wesley, 1998.

[13] INMOS Ltd, *Occam Programming Manual,* Prentice-Hall, 1984.

[14] R. Milner, *Communication and Concurrency,* Prentice-Hall, 1989.

[15] J.J.M.M. Rutten, Semantic Correctnes for a Parallel Object-Oriented Language, *SIAM Journal of Computing*, vol. 19, 1990, pp. 341–383.

[16] E.N. Todoran, Metric Semantics for Synchronous and Asynchronous Communication: a Continuation-based Approach, *Electronic Notes in Theoretical Computer Science*, vol. 28, 2000, pp. 101–127.

[17] E.N. Todoran and N. Papaspyrou, Experiments with Continuation Semantics for DNA Computing, *Proceedings of the IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP 2013)*, pp. 251–258, 2013.

[18] E.P. de Vink, *Designing Stream Based Semantics for Uniform Concurrency and Logic Programming*, Ph.D thesis, Vrije Universiteit Amsterdam, 1990.

[19] Aglets portal site, 2004, `http://aglets.sourceforge.net/`

Computer Science Department, Technical University, Cluj-Napoca, Romania
E-mail address: `eneia.todoran@cs.utcluj.ro`