Proceedings of the Federated Conference on Computer Science and Information Systems pp. 419–424
DOI: 10.15439/2018F264
ISSN 2300-5963 ACSIS, Vol. 15

Visualization of logical formulas

Radosław Klimek
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Krakow, Poland
Email: rklimek@agh.edu.pl

***Abstract*—Visualization of logical formulas for classical propositional calculus gains a new meaning in the context of a huge development of SAT solvers which find solutions of satisfiability problems of bigger and more complex tasks including the ones of industrial meaning. The aim of this work is to create, using modern IT tools, a coherent and widely accessible system in a form of web application, which will be able to provide a coopertive system, as well as will help to visualize, analyze and transform logical formulas. The system itself is now in the initial, however, mature implementation phase. It is open for new ideas and methods. New functionalities have already been introduced but there are plans to create the new ones.**

***Index Terms*—SAT; visualization; graph.**

I. Introduction

SATISFIABILITY problem is the classical and fundamental problem of theoretical computer science in which there can be encoded many other problems [1]. Satisfiability problem is a testing whether exists of a logical value assignment (true/1, false/0) to variables in a particular formula which will satisfy it, namely its logical value will be true. Those types of problems can be applied to many branches and are commonly used, for example in combinatorial optimization [2], graph theory, automated theorem proving, verification of software models [3], [4], [5], [6], and artificial/ambient intelligence [7], [8], [9].

Tools which solve satisfiability problem are called SAT solvers. Their rapid development and routine solving of tasks, which consist of many thousands of variables and clauses, enables practical implementation of reasoning engines which are quite common and easily accessible [10], [11]. There was created a standard of input files recording, known as DIMACS CNF format or simply DIMACS, where the base DIMACS relates to the name of the research center where it was created, namely The Center of Discrete Mathematics and Theoretical Computer Science, and CNF is a reference to the conjunctive form of the normal formula (Conjunctive Normal Form). This form is used to save logical formula in leading solvers based on CDCL method where problem is enclosed in this form in a particular file having DIMACS form. Thanks to using one format it is easy to compare existing SAT solvers and to see new possibilities of creating new, interesting tools based on the same input files which can be related to SAT or similar concepts.

Nowadays, SAT solvers are commonly used in a formal verification of designed systems, especially embedded systems, cyber-physical systems and software drivers. Solvers become more and more popular. Their widespread use is visible in the processes described as Electronic Design Automation (EDA) because designing processes of those sets, especially embedded sets, need to be analyzed carefully as the errors done during their design and production can have detrimental effects and be very costly. The similar situation appears when we talk about drivers which unstable work can interfere with the working of the system core.

Visualization of logical formula can also be a useful, and to some extend helpful, tool in solving satisfiability problem as well as in analysis and preprocessing of formulas as input data for SAT solvers. The graphic form of formula, its shape, regularity or irregularity, consistency or inconsistency can provide us with many valuable pieces of information about the encoded problem, see for example [9, Table 11]. It can also suggest recommended formula preprocessing methods which can speed up the process of searching for a satisfiable solution. That is why visualization helps us to understand the encoded problem better and gives us an opportunity of carrying an additional and quicker analysis of the problem before our attempt to solve it. Last but not least, visual image of a formula can have an aesthetic meaning. The generated visualizations create interesting visual effects which may be attractive for people who have not dealt with SAT topic so far.

The aim of this work is to create widely accessible service in form of web application, built on the basis of modern IT tools, easy to handle and enabling the visualization of logical formulas of propositional calculus which are the input data for SAT solvers based on CDCL method (Conflict Driven Clause Learning). The present system is based on work [12], however, it has been adjusted to the web requirements[1].

The development of solvers which work on the basis of this method needs to be acknowledged as a spectacular one. Every formula can be visualized and analyzed before we start looking for its solution. Although in this work there were presented the well-known visualization methods but there also were proposed new functionalities of the system. The existing program called *DPvis* created by Carsten Sinz, see seminal and fundamental work [13], enables crating different types of graphs on the basis of formulas describing SAT problems. However, this program is hardly accessible and not compatible with demands of the modern and dynamic IT market which can discourage the present and potential users. Under those circumstances, creating the fully web application, see [12], which enables cooperative work in client-server

[1]The planned website for this service is http://forvis.agh.edu.pl.

architecture, built on the basis of modern IT tools and offering SAT problems visualizations, may encourage new users or specialists and provide them with tools which simplify their everyday work.

In the future the service will be extended to provide fluent transition between different methods of formula visualizations or different methods of encoding logical formulas. There are also plans to open the service for related problems according to classical SAT, namely *MaxSAT* (finding the maximal number of clauses, if not all, of the entire formula to be satisfied) and *weighted MaxSAT* (finding an assignment that maximizes the total weight of the satisfied clauses), *Partial MaxSAT*(some clauses are deemed hardinfinite weights, clauses with finite weight are soft), and other problems, which will enable the analysis of fragmentary problem solutions in the context of the whole problem.

II. Preliminaries

Supported file format is a commonly recognized DIMACS CNF format. System should offer a possibility of storing saved files and exporting created visualizations to graphic files within a wide range of formats.

Logical formula F is in CNF form because it is built from conjunction of clauses, which can be presented as $F = \bigwedge_{i=1}^{m} C_i$ that is a conjunction of clauses C_1, C_2, ..., C_m and every clause is built from disjunction of literals $C_i = \bigvee_{j=1}^{n} l_i, j$ where literal l can be a variable x or its negation $\neg x$. The example of logical formula in CNF form is the formula:

$$(x \vee \neg y) \wedge (z \vee y \vee \neg x)$$

DIMACS CNF file is saved as ASCII text file which enables easy transmission between different operational systems or working environments. The file can have lines of comments, so-called line of problem, but its most important part are the following clauses, whereby every of them finishes with number 0. Every clause is built from a sequence of natural numbers where every number clearly identifies one variable (can be from thesaurus of variables). Moreover, every number may be predeceased by minus or not be predeceased, where symbol of minus signifies negation of this variable. Therefore, the particular numbers signify literals. The following file is an example of that:

```
c
p cnf 212 118098
-169 -177 -184 -182 -174 -187 -196 -201 -199
-191 -17 -26 -32 -28 -23 -35 -41 -48 -46 -38
-53 -60 -65 -62 -56 -68 -77 -83 -79 -73 -86
-93 -100 -97 -89 -103 -109 -117 -113 -108
-122 -126 -132 -129 -125 -139 -144 -151 -146
-142 -154 -162 -166 -164 -157 12 13 14 15 0
-169 -177 -184 -182 -174 -187 -196 -201 -199
-191 -17 -26 -32 -28 -23 -35 -41 -48 -46 -38
-53 -60 -65 -62 -56 -68 -77 -83 -79 -73 -86
-93 -100 -97 -89 -103 -109 -117 -113 -108
-122 -126 -132 -129 -125 -139 -144 -151 -146
-142 -154 -162 -166 -164 -157 12 13 14 15
170 171 172 173 178 179 184 185
181 182 175 176 0
```

The size of logical formula within a file does not prove difficulty of the problem itself but, of course, in practice, the time spent on searching for the solutions of the big formulas is longer, sometimes very long. There are tools which aim is to minimize logical formulas which can reduce time of solving the problem significantly. One of the better known tools is *SatELite* program which implements the following techniques [14]:

1) elimination of variables by resolutions – deleting unnecessary variables and equivalent literals;
2) fast subsumption – C_1 clause subsumes C_2 when $C_1 \subseteq C_2$, therefore C_1 may be deleted;
3) self-subsumption – C_1 clause almost fully subsumes C_2 with exception of one literal x which appears in C_2 with a different value. For example, $C_1 = \{x, a, b\}$, and $C_2 = \{\neg x, a\}$. Resolution for x gives $C_1' = \{a, b\}$, which subsumes C_1, therefore C_1 can be added to logical formula and C_1 can be removed.

III. Visualization methods

There are used three basic methods of visualization in this work . It does not cover all planned implementation methods. The present set should be interpreted as an initial proposition for system functionalities. The currently used methods are [13]:

- factor graph,
- interaction graph,
- resolution graph.

Undirected factor graph $G_F = (V, E)$ is a graph where a set of vertices V is created by variables and clauses, therefore $V = X \cup C$, where X is a set of variables and C is a set of clauses. E symbol means a set of edges. The edge is drawn between variable x and clause c when $x \in C$ or $\neg x \in C$. Provided that there exists a problem described by formula: $(\neg x \vee \neg y \vee z) \wedge (y \vee z) \wedge (x \vee \neg z \vee u)$ it is possible to determine the following clauses:

1) $C_1 = (x \vee \neg y)$
2) $C_2 = (\neg x \vee z \vee u)$
3) $C_3 = (x \vee z)$

Construction of factor graph for the problem presented above should start from drawing variables and clauses in form of vertices, see Fig. 1. Later on we draw edges which connect variables with clauses in which they appear, starting from clause C_1. Finally, the graph looks like the last structure in Fig. 1.

In contrast to undirected factor graph, directed factor graph includes the difference between a variable in positive and negative form. It is usually presented using colors, where green means not negated variable and red a negated one. For the analyzed problem, the graph looks like the following one: Fig. 2.

Variable interaction graph $G_I = (V, E)$ is a graph in which a set of vertices V is equal to a set of variables X which is equivalent to $V = X$. E means a set of edges. The edge

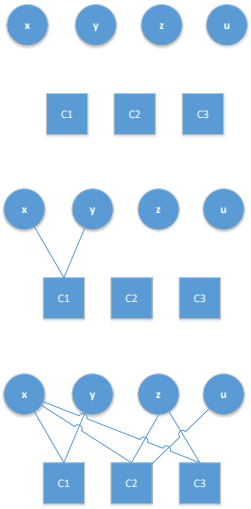

Fig. 1. Structure of a factor graph, and successive construction steps

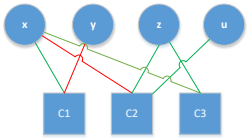

Fig. 2. Directed factor graph

connects two variables: x and y when they appear together in at least one clause ($x \in C_i$ and $y \in C_i$). On the basis of the example from the previous section, there are determined the following clauses:

1) $C_1 = (x \vee \neg y)$
2) $C_2 = (\neg x \vee z \vee u)$
3) $C_3 = (x \vee z)$

The first step to create interaction graph is drawing variables in the form of vertices, see Fig. 3. Later on, starting from C_1 clause, the appearing variables are connected together. When analyzing next clauses, if the particular edge already exists for a pair of variables, it is omitted. Finally, the graph looks like the last structure in Fig. 3.

Resolution graph $G_R = (V, E)$ is a graph in which a set of vertices V is equal to a set of clauses, therefore $V = C$. E means a set of edges. The edge connects two variables C_1 and C_2 only if there is a variable x where $x \in X$, in which X means a set of variables where $x \in C_1$ and $\neg x \in C_2$. Using the previously presented formula and having already determined clauses:

1) $C_1 = (x \vee \neg y)$
2) $C_2 = (\neg x \vee z \vee u)$
3) $C_3 = (x \vee z)$

Construction of the resolution graph starts from drawing clauses in form of vertices, see Fig. 4. Later on the following variables which appear in both C_1 and C_2 clauses are analyzed. Because $x \in C_1$ and $x \in C_2$ there can be drawn an

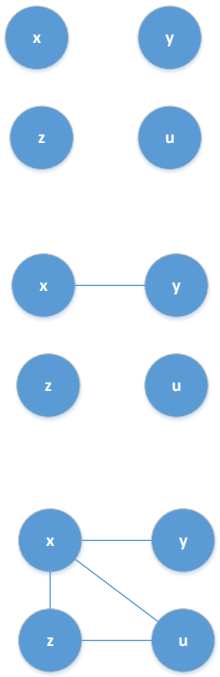

Fig. 3. Structure of interaction graph, and the succseeding, and successive construction steps

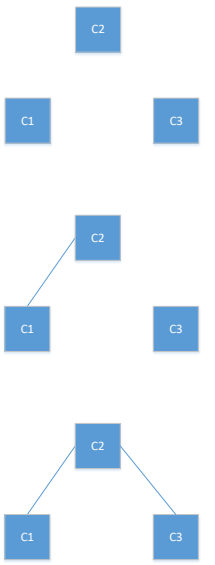

Fig. 4. Structure of resolution graph, and the succseeding, and successive construction steps

edge between the clauses. The redundant edges are omitted. Finally, the graph looks like the last structure in Fig. 4.

The methods of visualization presented above are the basic set of methods. In the future there are plans to introduce the new ones taking into consideration needs and nature of SAT-related problems.

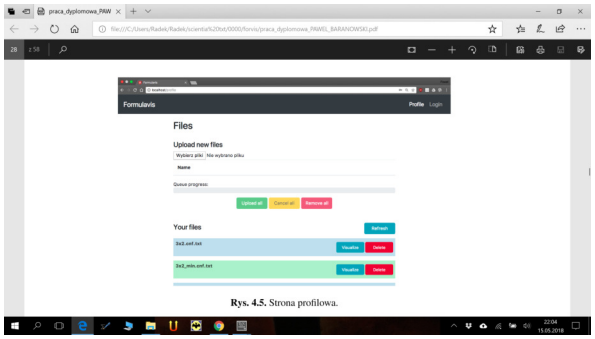

Fig. 5. The screenshot of the system – formula management

IV. Design of visualization system

The implemented system, see Fig. 5, is breifly described. Server application was made using *Python* language together with frame of *Django REST Framework* application [15]. For operation of asynchronous tasks, *Celery* task queue [16], together with *RabbitMQ* [17] message broker, are responsible. Data is stored in *PostgreSQL* database server. Client application, as well as user's interface, was created on the basis of *TypeScript* language using the frame of *Angular 4* application. There was used an effective visualization library based on *JavaScript* language named *vis.js* [18]. Server proxy *Nginx* [19] is responsible for managing the movement between user's graphic interface and server application. The whole project is containerized using *Docker* technology [20]. *Nginx* plays the role of Reverse-Proxy or Forward-Proxy server, see Fig. 6. Reverse-Proxy server helps in:

- hiding the current system which is behind proxy server,
- distributing the movement between application instances of the server,
- pointing the movement towards proper applications,
- compressing the content of data flow,
- manipulating with requests and answers.

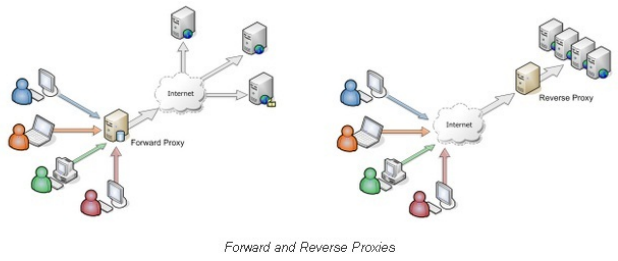

Fig. 6. Difference between Forward-Proxy i Reverse-Proxy server. Source: Vivek Srivastav: Proxy Pass. [In:] http://viveksrivastv.blogspot.com/p/apache-administration.html. Accessed on 1 May 2018.

Celery is a asynchronous task queue based on task distribution using messages. It concentrates on operations performed in the shortest possible time but also it supports planned tasks. Tasks are performed at the same time using one or many executive instances – workers and with the use of multiprocess transformation. Those tasks can be performed asynchronously (it enables the ordering application to perform further work) or synchronically (the ordering application waits for the result). In order to use *Celery*, it is required to use message broker which provides data for workers. The message broker recommended by *Celery* authors is *RabbitMQ*. It is an open source and easy to implement system in *Erlang* language. It supports and monitors asynchronous message sending and their deployment using clusters which enables further development of the system.

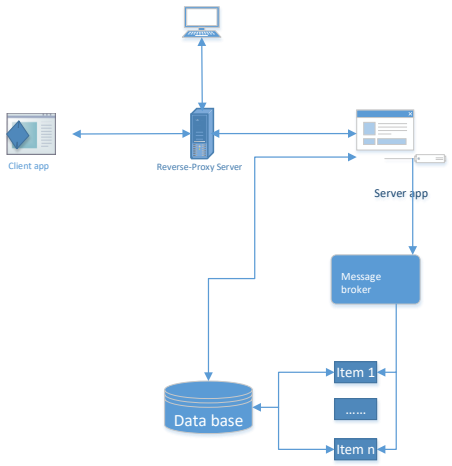

Fig. 7. Structure of logical formula visualization system

The created client application is an interface of the whole system because a target user has an access to it. Thanks to that, the user can log in and log out, upload and delete data from server and, most importantly, display visualizations and save them to file. Implemented application is in the form of browser application. Server application, implemented separately, organizes the whole data processing according to user's expectations. Detailed analysis of this issue exceeds the aim and size of this work, see also Fig. 7.

V. Examples of visualization

There were examined many formulas in terms of possibilities for system work as well as visualization effects. All figures (from 8 to 13) are provided by the designed system, see also [12]. Presentation starts from simple formulas and continues to the most complicated ones. We will start from a very simple example, see also Fig. 8:

```
p cnf 3 2
1 -3 0
2 3 -1 0
```

In interaction graph:

- blue vertices – represent variables,
- edge, shade of edge signifies number of connections between variables (the more of them, the darker the shade is). Interaction graph presents the structure of variables neighborhood within the examined logical formula. On the basis of that there can be performed the analysis if a

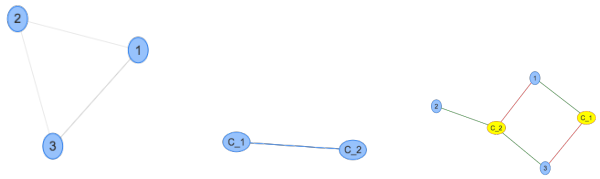

Fig. 8. First example: interaction, resolution and factor graphs

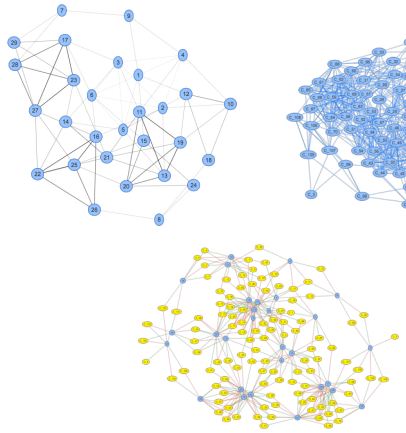

Fig. 9. Second example, interaction, resolution and factor graphs

problem is integral or if it can be divided into parts. What is more, the edges brightness can help to find frequent neighborhoods of variables.

In resolution graph:

- blue vertices – represent clauses.

Resolution graph visualizes the structure of clause dependencies. Vertices are connected by edge if they have one (or more) literals of a different logical value. In directed factor graph:

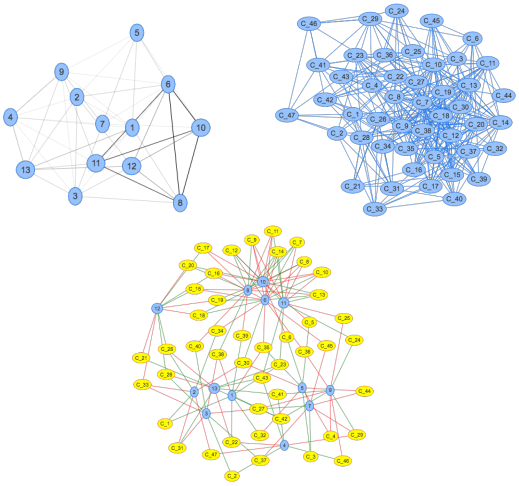

Fig. 10. Second example, after minimalization: interaction, resolution and factor graphs

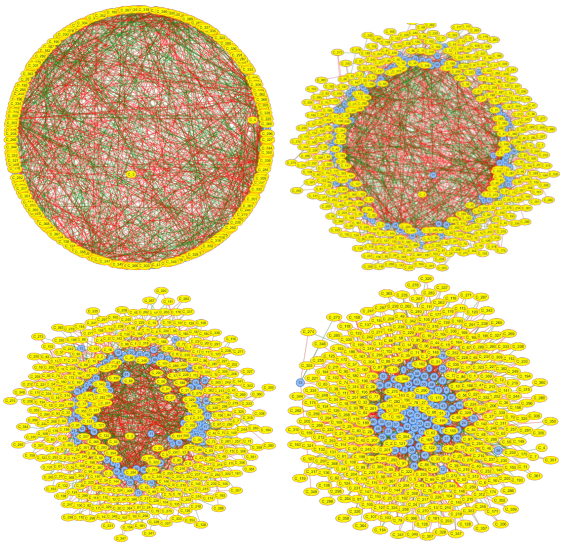

Fig. 11. Third example: stabilization, factor graphs – before stabilization, first screenshot, second screenshot, after stabilization

- yellow vertices – represent clauses,
- blue vertices – represent variables,
- red edge – represents a variable with negation,
- green edge – represents a variable without negation.

Factor graph shows clause dependencies and variables with their logical values.

The next analyzed formula has 29 variables and 109 clauses, and is shown in Fig. 9. After preprocessing, the minimization was performed which gave 13 variables and 47 clauses, see Fig. 10.

In a created system there was implemented graph/formula *stabilization* operation. It means, that the graph was rebuilt in such a way that the edges got a total minimal length. Stabiliza-

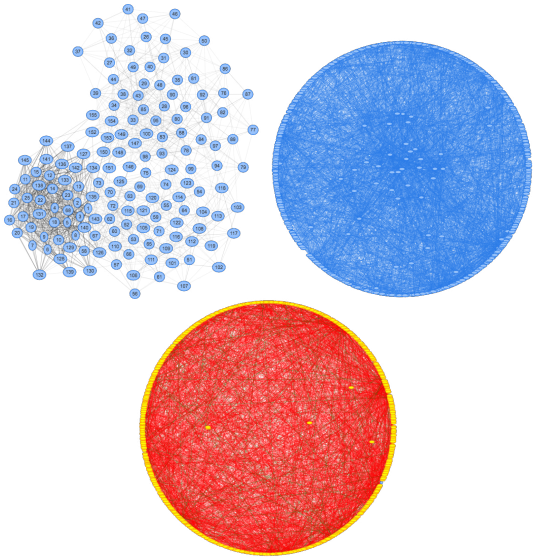

Fig. 12. Fourth example: interaction, resolution and factor graphs

tion helps to get more compact visualizations. The stabilization process can be observed in the case of logical formula of 83 variables and 369 clauses. The next images present the following visualization stages, until its accomplishment, see Fig. 11.

The fourth example: logical formula – 155 variables, 1135 clauses, see Fig. 12.

And the last example: logical formula after minimalization – 42 variables, 133 clauses, see Fig. 13.

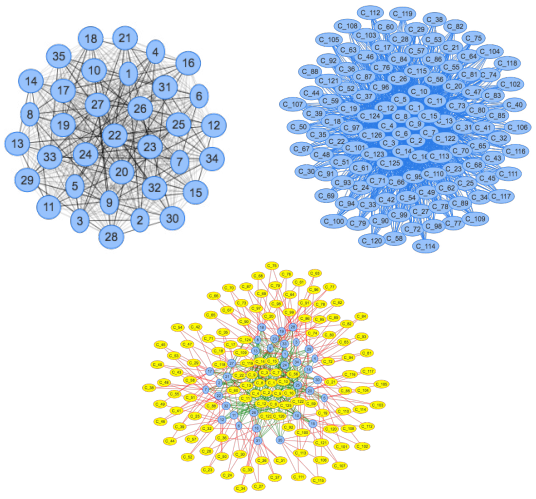

Fig. 13. The last example: interaction, resolution and factor graphs

VI. Conclusion

Visualization project of logical formulas has to be considered as a successful one, however, it is now at its initial stage and it will be continued. The system enables the elementary transformation of formulas and their preprocessing. The aim of the whole project was to create widely accessible tool helping to understand, analyze and examine the structures of problems presented with the use of logical formulas.

The system also offers storing files on the server which requires creating log-in system and the user's profile. It also offers the possibility of interaction with generated graphs, minimization of uploaded formulas and export of visualizations to the selected graphic formats. Its design and implementation aspects are very modern as a result of using the open approach as well as modern software technologies.

There are plans of a further system development by introducing new functionalities, another visualization methods and opening it for other SAT problems, for example MaxSAT.

References

[1] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009.

[2] E. Kucharska, K. Grobler-Debska, and K. Raczka, "ALMM-based methods for optimization makespan flow-shop problem with defects," in *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology ISAT 2016 - Part I, Karpacz, Poland, September 18-20, 2016*, 2016, pp. 41–53. [Online]. Available: https://doi.org/10.1007/978-3-319-46583-8_4

[3] R. Klimek, "Towards formal and deduction-based analysis of business models for SOA processes," in *Proceedings of 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), 6–8 February, 2012, Vilamoura, Algarve, Portugal*, J. Filipe and A. Fred, Eds., vol. 2. SciTePress, 2012, pp. 325–330.

[4] R. Klimek and P. Szwed, "Verification of archimate process specifications based on deductive temporal reasoning," in *Proceedings of Federated Conference on Computer Science and Information Systems (FedCSIS 2013), 8–11 September 2013, Kraków, Poland*. IEEE Xplore Digital Library, 2013, pp. 1131–1138.

[5] R. Klimek, "From extraction of logical specifications to deduction-based formal verification of requirements models," in *Proceedings of 11th International Conference on Software Engineering and Formal Methods (SEFM 2013), 25–27 September 2013, Madrid, Spain*, ser. Lecture Notes in Computer Science, R. M. Hierons, M. G. Merayo, and M. Bravetti, Eds., vol. 8137. Springer Verlag, 2013, pp. 61–75.

[6] P. Wiśniewski, K. Kluza, A. Ligeza, and A. Suchenia, "Generation of synthetic business process traces using constraint programming," in *Proceedings of Federated Conference on Computer Science and Information Systems (FedCSIS 2018), 9–12 September 2018, Poznań, Poland*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds. IEEE Xplore Digital Library, 2018, pp. 441–449.

[7] R. Klimek, "Behaviour recognition and analysis in smart environments for context-aware applications," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2015), October 9–12, 2015, City University of Hong Kong, Hong Kong*. IEEE Computer Society, 2015, pp. 1949–1955.

[8] R. Klimek and L. Kotulski, "Towards a better understanding and behavior recognition of inhabitants in smart cities. a public transport case," in *Proceedings of 14th International Conference on Arificial Inteligence and Soft Computing (ICAISC 2015), 14–18 June, 2015, Zakopane, Poland*, ser. Lecture Notes in Artificial Intelligence, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., vol. 9120. Springer Verlag, 2015, pp. 237–246.

[9] R. Klimek, "Exploration of human activities using message streaming brokers and automated logical reasoning for ambient-assisted services," *IEEE Access*, vol. 6, pp. 27 127–27 155, 2018.

[10] C. P. Gomes, H. A. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," in *Handbook of Knowledge Representation*, ser. Foundations of Artificial Intelligence, F. van Harmelen, V. Lifschitz, and B. W. Porter, Eds. Elsevier, 2008, vol. 3, pp. 89–134.

[11] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*, 1st ed. Addison-Wesley Professional, 2015.

[12] P. Baranowski, "System for visualization of logical formulas, Engineering diploma thesis, supervisor: Radosław Klimek, AGH University of Science and Technology," 2018.

[13] C. Sinz, "Visualizing SAT Instances and Runs of the DPLL Algorithm," *Journal of Automated Reasoning*, vol. 39, no. 2, pp. 219–243, Aug. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10817-007-9074-1

[14] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *Theory and Applications of Satisfiability Testing*, F. Bacchus and T. Walsh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 61–75.

[15] T. Christie, "Website: Django rest framework," 2018, accessed on 6-Jan-2018. [Online]. Available: http://www.django-rest-framework.org/

[16] Celery Development Team, "Website: Celery: Distributed task queue," 2018, accessed on 6-Jan-2018. [Online]. Available: http://www.celeryproject.org/

[17] Rabbit Technologies Ltd., "Website: RabbitMQ documentation," 2018, accessed on 7-Feb-2018. [Online]. Available: https://www.rabbitmq.com/documentation.html

[18] vis.js Development Team, "Website: vis.js. dynamic browser based visualization library," 2018, accessed on 6-Jan-2018. [Online]. Available: vhttp://visjs.org/

[19] NGINX Development Team, "Website: Nginx products," 2018, accessed on 6-Jan-2018. [Online]. Available: https://www.nginx.com/resources/wiki/

[20] Docker Development Team, "Website: Docker - software containeraization platform," 2018, accessed on 6-Jan-2018. [Online]. Available: http://www.docker.com/