Goal-Driven Inference for Web Knowledge Based System

Roman Simiński and Agnieszka Nowak-Brzezińska

Abstract Traditional knowledge based systems were developed as the desktop applications. Meanwhile, web applications have grown rapidly and have had significant impact on the application of such systems. In the presented work, we introduce the modified goal-driven inference algorithm which allow us to divide some parts of them into the client and server layers of the web application. Proposed approach assumes that the rule knowledge base is decomposed into the decision oriented group of rules. We argue that the knowledge base in the form of such rules group contains enough information, which allows to divide inference into the client and server side, ensuring the convenience and the effectiveness.

Keywords Knowledge base · Goal-driven inference · Decision oriented partitions

1 Introduction

The migration of information systems from the classic desktop software to the web application can be observed as a permanent trend. This trend also applies to the knowledge based systems. The "webalisation" of information systems causes many practical and implementation problems and challenges, but we can also identify in this field a number of interesting research problems. In this paper we present a modified goal-driven inference algorithm for web knowledge based systems.

A goal-driven algorithm is a one of the two popular strategies of inference in the knowledge based systems it started from a goal and ended with a fact that leads to the goal. Since it is easy to implement, a goal-driven inference is a key to building many practically used domain expert systems. In the modern web applications goal-driven inference could be divided between the client and server part of the web

R. Simiński (✉) · A. Nowak-Brzezińska
Institute of Computer Science, University of Silesia, Sosnowiec, Poland
e-mail: roman.siminski@us.edu.pl

A. Nowak-Brzezińska
e-mail: agnieszka.nowak@us.edu.pl

Z. Wilimowska et al. (eds.), *Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part IV*, Advances in Intelligent Systems and Computing 432, DOI 10.1007/978-3-319-28567-2_9

application. In contrast to that, the data-driven inference can be implemented entirely on the server side, without any conversation with user, only obtaining starting facts is required.

In the presented work we introduce the modified goal-driven inference algorithm, which allow us to divide some parts of them into the two layers of the web application. Proposed approach assumes that the rule knowledge base is decomposed into the decision oriented group of rules. We argue that the knowledge base in the form of such rules group contains enough information which allows to divide inference into the client and server side, ensuring the convenience and the effectiveness.

The first part of the work briefly presents a problem description and related work. The following part of the work describes the rules partitioning approach, then the utilization of this approach in optimization of inference algorithm is described and the modified version of algorithm is presented. Next, a simple case study is presented and the preliminary evaluation of modified algorithm concludes the presented work.

2 Problem Description and Related Works

A goal-driven inference always has a single goal or goals list to confirmation, this approach starts with the desired rule's conclusion matching to the current goal and works backward to find supporting facts [1]. If this rule requires additional information before it can succeed, the inference can execute additional rules, recursively if necessary. An inference engine will search the rules until it finds one which has a conclusion that matches a desired goal. If all conditions in the rule's premise are facts, the current goal is confirmed. If some of the rule's premise conditions are not known to be a fact, this conditions are added to the list of goals as new goals, pushing the other goals down in the list. At any time the algorithm only works on the one top goal [2]. If no rule is available to confirm whether the condition is a fact, the algorithm asks the environment about the truth of the considered condition. The environment may vary depending on the system application. Typically the user is the source of fact, but in the context of embedded systems, facts can be provided by the technical equipment [3].

The disadvantages of goal-driven inference follow from the inefficiency of searching in the large knowledge bases with rules that are not organized in any kind of structure and from the fact that recursive algorithms are difficult to follow [1, 2]. In large rule bases recursive calls are very often misguided, but they take time and consume memory resources. When we consider inference process distributed over the multilayer web application, missed recursive calls and large search space become a significant problem. When we consider classical inference algorithm, all above described operations are realized within the single function/class/module, implemented in the particular programming language. In the context of web-based implementation, specified operations have to be implemented in different way.

Traditional rule based systems were developed as the desktop applications and a number of development tools are available for developing traditional systems. Meanwhile, web applications have grown rapidly and have had significant impact on the application of traditional expert system. Several tools and languages are available for developing web-based expert systems—these tools use traditional expert system techniques and offer in addition the capacity for Web-based development [4, 5].

System Acquire [6], which allows the development of web-based user interfaces, is supported through a client–server development kit that supports Java and ActiveX controls, unfortunately, detailed information is enigmatic. System ExSys [7] provides the Corvid Servlet Runtime implements the Exsys Corvid Inference Engine as a Java Servlet. In this mode, the user interface is defined by HTML templates. Corvid systems can be also integrated with Adobe Flash. The Exsys Corvid Servlet Runtime uses Java Servlet technology, allowing the proven Corvid Inference Engine to be run on a server with only HTML pages sent to the client machine running the system.

The JESS is a rule engine and scripting language [8], which provides console for programming and enables basic input and output, it cannot be used directly in the web-based application but it is possible to use JESS within the JSP platform [9, 10]. The XpertRule KBS interfaces over the Web with a thin client using Microsoft's Active Server Page technology. Web Deployment Engine is a JavaScript rules runtime engine which runs within a browser [11]. Applications developed using the Knowledge Builder Rules Authoring Studio can be generated as Java Script/HTML files for deployment as Web applications. The JavaScript engine runs the rules, calculations and the JS/HTML user interface. The eXpertise2Go's Rule-Based Expert System provides free expert system building and delivering tools that implement expert systems as Java applets, Java applications and Android apps [12].

The goal-driven inference is available in Prolog, some of implementations allow to run interpreter within the web application. The SWI-Prolog [13] interpreter can be run in script mode, the script runs the SWI-Prolog interpreter with suppressed interactive output and the script file will produce a result of the query only. It is possible to run inference by CGI program on the server which builds the appropriate Prolog query and execute interpreter. This approach works on the server side and is not tailored to the specific of web application. It requires the usage of relevant program which builds a Prolog query, and executes the Prolog script and generates HTML [5].

Existing tools described above allow us to develop web-based expert systems, but these tools use traditional expert system techniques and offer in addition the capacity for Web-based development. Inference techniques are usually server oriented, front-end layer are used typically for visualization and simple interaction with user. It is hard to find new, modern approaches to web-based expert systems and new implementation of inference algorithms. We argue that in the modern web applications goal-driven inference must be divided between the client and server part of the web application, the data-driven inference can be implemented entirely on the server side. In the literature we can find some attempts to build web-based domain expert systems, e.g. [14, 15]. In general, our proposal is similar, but we are

focused on the implementation of a domain independent system. We propose the following architecture of the knowledge based system:

- Server side—management of rule base, which is decomposed and stored in the relational data base. All selections of applicable rule or rules are performed by the server-side services available via specialized API. The resulting information is transferred into the client side in the form of JSON objects. Server side is passive and is focused on rule-oriented services for inference.
- Client side—initialize inference when the goal is known, realizes the main event loop, including:
  - confirmation of the rule's condition against a dynamically created fact set,
  - acquisition of the fact from application environment (usually from the user),
  - initialization of recursive inference calls for sub-goals.

Client side utilizes JavaScript functions embedded in the HTML document, generated by the proper server side scripts from the application layer. The JavaScript functions use rules obtained from the server services via asynchronous AJAX requests. This distributed environment was used in the system described in the [16, 17].

3 Methods

Modified goal-driven inference algorithm is based on the proposed method of rule knowledge base partitioning, which allow us to obtain modular knowledge base. A significant part of this work contains a detailed description of proposed approach. introduced approach differs from other methods of the modularization. The presented solution is an extension of the research presented in [18, 19].

3.1 Knowledge Base and Rules Partition

The knowledge base is a pair $KB = (RS, FS)$ where RS is a non-empty finite set of rules and FS is a finite set of facts. $RS = \{r_1, \ldots, r_n\}$, each rule $r \in RS$ will have a form of Horn's clause: r: $p_1 \wedge p_2 \wedge \cdots \wedge p_m \rightarrow c$, where m—the number of literals in the conditional part of rule r, and $m \geq 0$, p_i—i-th literal in the conditional part of rule r, $i = 1 \ldots m$, c—literal of the decisional part of rule r. For each rule $r \in RS$ we define following functions: $concl(r)$—the value of this function is the conclusive literal of rule r; $cond(r)$—the value of this function is a set of conditional literals of rule r. We will also consider the facts as clauses without any conditional literals. The set of all such clauses f will be called *set of facts* and will be denoted by FS: $FS = \{f: \forall_{f \in FS}\ cond(f) = \{\} \wedge f = concl(f)\}$.

For each rule set RS with n rules, there is a finite power set 2^{RS} with cardinality 2^n. Any arbitrarily created subset of rules $R \in 2^{RS}$ will be called a group of rules. In

this work we will discuss specific subset $PR \subseteq 2^{RS}$ called partition of rules. Any partition PR is created by partitioning strategy denoted by PS, which defines specific content of groups of rules $R \in 2^{RS}$, creating a specific partition of rules PR. We may consider many partitioning strategies for a single rule base, but in this work we will only present a few selected strategies. Each partitioning strategy PS for rules set RS generates the partition of rules $PR \subseteq 2^{RS}$: $PR = \{R_1, R_2, \ldots, R_k\}$, where: k—the number of groups of rules creating the partition PR, R_i—i-th group of rules, $R \in 2^{RS}$ and $i = 1,\ldots, k$.

Rules partitions terminologically correspond to the mathematical definition of the partition as a division of a given set into the non-overlapping and non-empty subset. The groups of rules which create partition are pairwise disjoint and utilize all rules from RS. The partition strategies for rule based knowledge bases are divided into two categories: s*imple* and *complex strategies.* For simple strategies, the membership criterion decides about the membership of rule r in a particular group $R \subseteq PR$ according to the membership function mc, time complexity not higher than $O(n \cdot k)$, where $n = |RS|$ and $k = |PR|$. For complex strategies, the particular algorithm decides about the membership of the rule r in some group $R \subseteq PR$, with time complexity typically higher than any simple partition strategy. An example of a complex strategy is described in the [20].

3.2 Simple Partitioning Strategy

Creation of simple partition for rules set requires the definition of the membership criteria which assigns particular rule $r \in R$ to the given group of rules $R \subseteq PR$. Proposed approach assumes that the membership criteria will be defined by the mc function, which is defined individually for every simple partition strategy. The function: $RS \times PR \rightarrow [0\ldots1]$ has the value 1 if the rule $r \in RS$ with no doubt belongs to the group $R \subseteq PR$, 0 in the opposite case. The value of the function from the range $0 < mc < 1$ means the partial membership of the rule r to the group R. Let us assume that threshold value $0 \leq T \leq 1$ exists. The value of the mc (r, R) function can be higher, higher or equal, equal, less, less or equal to the T value. Generally we can define simple partition of rule based knowledge base PR as follows: $PR = \left\{R\text{: } R \in 2^{RS} \wedge \forall r \epsilon R\ mc(r,R) \geq T\right\}$. Special case of the simple strategies is the strategy called selection. The selection divides the set of rules RS into the two subsets R and RS–R. Thus we achieve the partition $PR = \{R, RS{-}R\}$. In practical sense, selection is the operation with linear time complexity O(n) where n denotes the number of all rules in the knowledge base.

The algorithm of creating the partition which bases on simple strategy is presented in the pseudo-code below. The input parameters are: knowledge base RS, the function mc that defines the membership criteria and the value of the threshold T. Output data is the partition PR. Time complexity of such algorithm is $O(n \cdot k)$, where $n = |R|$, $k = |PR|$. For each rule $r \in RS$ we have to check whether the goal

partition PR contains the group R with rule r (the value of the mc function has to be at least T: $mc(r, R) \geq T$). If such a rule doesn't exist the given rule r becomes the seed of a new group which is added to the created partition PR. The simple partitioning and selection algorithm are simple, were described in [20], and for this reason will be omitted.

3.3 Decision Oriented Partitioning Strategies

Let us consider the following partitioning strategy PS_1, which creates groups of the rules from R by grouping rules with the same attribute in conclusions. The membership criteria for rule r and group R is given by the function mc defined as follows: $mc(r, R) = 1$ if $\forall r_i \in R\ concl(r_i) = concl(r)$, 0 otherwise. When we use the simple partition algorithm (Alg01:createPartitions) with the mc function defined in this way, we obtain *decision oriented partitions*. Each group of the rules generated by this algorithm will have the following form: $R = \{r \in R: \forall r_i \in R\ concl(r_i) = concl(r)\}$. The number of groups in the partition k: $1 \leq k \leq$ n depends on the number of different decisions included in conclusions of such rules. When we distinguish different decision, by the different conclusions appearing in the rules—we get one group for each conclusion. In every group we have rules with the same conclusion: a fixed (a, v) pair. Such partition PR_1 will be called *basic decision based partition*. Basic decision partition contains the set of rules, each rule in the every set contains the same attribute-value pair in the conclusion. All rules grouped within a rule set take part in an inference process confirming the goal described by the particular attribute-value—for each $R \in PR_1$ the conclusion set $|Concl(R)| = 1$. If we consider the specified (a, v) pair, we think about particular kind of concept or about particular property state. From pragmatic point of view we can say that for each group R of basic decision based partition, single pair $(a, v) \in Concl(R)$ represent concrete. Basic decision partition represent the basic relations occurring in rule base—we can say that this partition defines the scope of the knowledge about concrete from real-word concepts within particular rule base.

Let us consider the second partitioning strategy PS_2, the membership criterion for rule r and group R is given by the function mc defined as follows: $mc(r, R) = 1$ if $\forall r_i \in R\ attrib(concl(r_i)) = attrib(concl(r))$, 0 otherwise. When we utilize the simple partition algorithm with the mc function defined in such way, we obtain different *ordinal decision oriented partitions*. Each group of the rules generated by this algorithm may have the following form: $R = \{r \in R: \forall r_i \in R\ attrib(concl(r_i)) = attrib\ (concl(r))\}$. The number of groups in the partition k: $1 \leq k \leq n$ depends again on the number of different decisions included in conclusions of these rules. Currently we distinguish decisions by the different attribute appearing in the conclusion part of the rules—we obtain one group for each decision attribute. This kind of partitioning strategy is called *ordinal decision partitioning strategy*. Partitions produced by the ordinal decision partition can be constructed as the composition of the basic decision partitions. Ordinal decision partition represents the relations occurring in a

rule base—we can say that this partitioning strategy can be considered as a model of decision about concepts from real-word.

3.4 Modified Goal-Driven Inference

Modification of the classical goal-driven inference algorithm is based on extracting information of internal rules dependencies. This information allows to perform only promising recursive calls of backward inference algorithm, optimization relies on reducing the number of rules searched for each run of inference and reducing the number of unnecessary recursive calls.

Modified algorithm as input data takes *PR*—the decision partition, *FS*—the set of facts and *g*—the goal of the inference. As the output data it takes *F*S—the set of facts, including possible new facts obtained through inference, the function's result as boolean value, true if the goal *g* is in the set of facts: $g \in FS$, false otherwise.

```
function goalDrivenInference( PR, g, var FS ) : boolean
begin
  if g∈FS or ¬g∈FS then return g∈F
  else
    truePremise ← false;
    select R∈PR where g∈Concl(R)
    while ¬truePremise ∧ R≠ϕ do
      select r∈{R} according to the selection strategy
      forall w ∈ cond(r) do
        truePremise ← (w∈FS)
        if ¬truePremise ∧ w∈In_C(R) then
          truePremise ← goalDrivenInference (PR, w, FS)
        elseif ¬truePremise then
          truePremise ← environmentConfirmsFact(w)
        elseif !truePremise then
          break
        endif
      endfor
      if ¬truePremise then
        R = R-{r}
      endif
    endwhile
  endif
  if truePremise then FS = F∪{g}
  return truePremise
end function
```

Only promising groups of rules are selected for further processing (select $R \in PR$ where $g \in Concl(R)$), where $Concl(R)$ is the set of conclusions for the group of rule R, containing literals appearing in the conclusion parts of the rules r from R. Only the selected subset of the not activated rules of R is processed in each iteration. Finally, only the promising recursive calls are made ($w \in In_C(R)$). $In_C(R)$ denotes connected inputs of the rules group, defined in the following way: $In_C(R) = \{(a, v) \in Cond(\mathrm{R}): \exists_{r \in R}\, (a, v) = concl(r)\}$, where $Cond(R)$ is the set of conditions for the group of rule R, containing literals appearing in the conditional parts of the rules r from R. In each iteration the set R contains only proper rules matching to the currently considered goal. It completely eliminates the necessity of searching for rules with conclusion matching to the inference goal, it is not necessary to search within the whole set of rules R—this information is simply stored in the decision-partitions and does not have to be generated.

4 A Simple Case Study and Discussion

To illustrate the conception of inference modification, we consider an example rule base:

r_1: $(a, 1) \wedge (b, 1) \rightarrow (c, 1)$	r_4: $(b, 3) \wedge (d, 3) \rightarrow (e, 1)$	r_7: $(d, 4) \rightarrow (f, 1)$
r_2: $(a, 1) \wedge (b, 2) \rightarrow (c, 2)$	r_5: $(b, 3) \wedge (d, 2) \rightarrow (e, 1)$	r_8: $(d, 4) \wedge (g, 1) \rightarrow (f, 1)$
r_3: $(a, 1) \wedge (b, 3) \rightarrow (c, 1)$	r_6: $(b, 3) \rightarrow (e, 2)$	r_9: $(c, 1) \rightarrow (d, 4)$

The different variants of the partitions could be build and stored by the server side services after any knowledge base modification or could be created in the inference initialization phase. In the Table 1 we present two decision oriented partitions. In Case II the asterisk '*' means any attribute value, only attributes are considered. We use basic decision partition for small or "flat" rules bases (without sub-goals). The ordinal decision partitions are useful when we consider large set with a high probability of occurrence of the sub-goals confirmation.

The modified algorithm proposed in this work extracts information of internal rules dependencies from partitioned knowledge base. Important role in the modification plays the information obtained from the set of conclusions for the group of

Table 1 Decision oriented partitions

Case I: basic decision partitions		Case II: ordinal decision partitions	
R1 = {r1, r3}	Concl(R1) = {(*c*, 1)}	R1 = {r1, r2, r3}	Concl(R1) = {(*c*, *)}
R2 = {r2}	Concl(R2) = {(*c*, 2)}	R2 = {r2, r4, r6}	Concl(R2) = {(*e*, *)}
R3 = {r4, r5}	Concl(R3) = {(*e*, 1)}	R3 = {r7, r8}	Concl(R3) = {(*f*, *)}
R4 = {r6}	Concl(R4) = {(*e*, 2)}	R4 = {r9}	Concl(R4) = {(*d*, *)}
R5 = {r7, r8}	Concl(R5) = {(*f*, 1)}		
R6 = {r9}	Concl(R6) = {(*d*, 4)}		

rule: Concl(R). It is possible to determine the rules set matching to the given fact only by searching within conclusions sets. When we consider the goal (f, 1), we can determine matching rules set R5 (Case I) or R3 (Case II) through the single search within the conclusions sets. This searching operation can be done by the server service—resulting rules set can be transferred into the client side as the XML or JSON data for further processing. Typically the matching group of rules has a significantly lower cardinality than the entire set of rules.

The main inference loop could be done be the JavaScript code in the client-side. Client-side code selects the rule from the rules set transferred from the server service, confirms condition from selected rule's premise, manages the dynamically gathered facts. When the algorithm have to confirm sub-goal, it can determine the usefulness of each recursive call by examining whether the sub-goal is in the set In_C(R). When we again consider the goal (f, 1), we have to analyze rules r_7 and r_8. The literal (d, 4) becomes a new sub-goal and recursive call is necessary. This call is promising, because there is another decision partition R_6 (Case I) and R_4 (Case II) supporting sub-goal (d, 4). When we consider sub-goal (g, 1), it is possible to immediately reject potential recursive call—there are no connected rules subset supporting sub-goal (g, 1). This can be done through a single asynchronous call of proper server service via AJAX. The classic version of the algorithm does not known whether the call is promising.

5 The Preliminary Experimental Results

The complexity of decision partition is $O(n \cdot k)$, where $n = |RS|$, $k = |PR|$, where the number of groups in the partition k: $1 \leq k \leq$ n typically is significantly smaller than the number of rules n. We have to store additional information for created rules partitions, however additional memory or disk space occupation for data structures seems acceptable. For n rules and k rules group we need approximately $is \cdot n + ps \cdot m$ bytes of additional memory for data structures (is—size of integer, ps—size of a pointer or reference).

Therefore, for n = 1000 rules, k = 100 rules group we need approximately 2.5 KB (precise amount of memory or disk space depends on used programming language, the conception of organization the data structures and designated system platform).

The proposed algorithm has been tested on artificial knowledge bases prepared for tests and on all real-word knowledge bases available for authors. We present only summary of the results for real-word bases—optimistic and pessimistic results with relation to the results of classic goal-driven algorithm (100 %). A limited scope of this work does not allow us to present a detailed information about test methodology and results of tests for recursive calls (Table 2).

The specific internal structure of knowledge bases, goals specification, and facts set configurations causes the significant differences between the optimistic and pessimistic case. Experiments for the artificial rules bases randomly generated also

Table 2 The results of the experiments

	invest.kb	media.kb	credit.kb	finanalysis.kb
The number of attributes	34	12	46	43
The number of rules	66	135	171	800
The number of groups (number of rules per group)	10 (7, 5, 13, 9, 6, 4, 11, 7, 3, 1)	2 (21, 114)	8 (6, 4, 3, 4, 109, 30, 12, 3)	24 (4, 10, 91, 7, 11, 3, 4, 2, 8, 5, 72, 8, 4, 3, 4, 126, 6, 6, 3, 47, 303, 5, 65, 3)
The number of searched rules, optimistic case	31 %	16 %	2 %	4 %
The number of searched rules, pessimistic case	89 %	85 %	74 %	82 %

confirm that when the number of groups increase, the number of searched rules decreases. The optimistic case let to reduce the number of searched rules to the 2 % (in pessimistic case we still reduce the number of rules with relation to the classical algorithm). We understand that we need more experiments on the real knowledge bases, presented results are preliminary. The implementation works are still in progress [16, 17], but the next stage of research will focus on the experiments on two real bases counting over 1200 and 4000 rules.

6 Summary

We introduced a modified goal-driven algorithm and the conception of distribution of such algorithm over the web-based software architecture. Modification of the classical inference algorithm is based on information extracted from the rules of groups generated by the decision partition. The proposed modification consists of the reduction of the search space by choosing only the rules from particular rule group, according to a current structure of decision oriented rules partition and the estimation of the usefulness for each recursive call for sub-goals. Therefore, only promising recursive call of the classical backward algorithm will be made. Every rule base already contains the information necessary to achieve modification steeps mentioned above. We only have to discover and utilize these information. The goal-driven inference proposed in this work is currently used in the two experimental versions of web-based expert system described in [16, 17].

Acknowledgements This work is a part of the project "Exploration of rule knowledge bases" founded by the Polish National Science Centre (NCN: 2011/03/D/ST6/03027).

References

1. Grzymala-Busse, J.W.: Managing uncertainty in expert systems, vol. 143. Springer Science & Business Media, Berlin (1991)
2. Walton, D.N.: Practical reasoning: goal-driven, knowledge-based, action-guiding argumentation, vol. 2. Rowman & Littlefield, Lanham (1990)
3. Smith, D.E.: Controlling Inference. Stanford University, Stanford (1985)
4. Grove, R.: Internet-based expert systems. Expert systems 17.3 (2000)
5. Dunstan, N.: Generating domain-specific web-based expert systems. Expert systems with applications 35 (2008)
6. Acquired Intelligence Home Page. http://aiinc.ca
7. Exsys Home Page. http://www.exsys.com
8. JESS Information. http://herzberg.ca.sandia.gov
9. Canadas, J., Palma, J., Túnez, S.: A Tool for MDD of rule-based web applications based on OWL and SWRL. Knowl Eng Softw Eng **1** (2010)
10. Ho, K.K.L., Lu, M.: Web-based expert system for class schedule planning using JESS. In: Information Reuse and Integration, IRI-2005 IEEE International Conference (2005)
11. XpertRule Home Page. http://www.xpertrule.com
12. eXpertise2Go's Rule-Based Expert System. http://expertise2go.com
13. The SWI-Prolog Home Page. http://www.swi-prolog.org
14. Li, D., Fu, Z., Duan, Y.: Fish-expert: a web-based expert system for fish disease diagnosis. Expert Syst Appl **23**(3) (2002)
15. Zetian, F., Feng, X., Yun, Z., XiaoShuan, Z.: Pig-vet: a web-based expert system for pig disease diagnosis. Expert Syst Appl **29**(1) (2005)
16. Simiński, R., Manaj, M.: Implementation of expert subsystem in the web application—selected practical issues. Studia Informatica **36**(1) (2015)
17. Nowak-Brzezińska, A.: KbExplorator a inne narzędzia eksploracji regułowych baz wiedzy. Studia Informatica **36**(1) (2015)
18. Nowak-Brzezińska, A., Simiński, R.: Knowledge mining approach for optimization of inference processes in rule knowledge bases, LNCS 7567, pp. 534–537. Springer, Berlin (2012)
19. Simiński, R.: Extraction of Rules Dependencies for Optimization of Backward Inference Algorithm, Beyond Databases, Architectures, and Structures, Communications in Computer and Information Science, Springer International Publishing, vol. 424, pp. 191–200. Springer, Berlin (2014)
20. Nowak-Brzezińska, A., Simiński, R.: New inference algorithms based on rules partition. In: CS&P 2014, Informatik-Berichte, vol. 245. Humboldt-University, Chemnitz, Germany (2014)