TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 62, Issue III, September, 2019

INFERENCE SYSTEM TO ACHIEVE SATURATION OF (F, R, N) KNOWLEDGE BASES

Andrei ZAMFIRA, Horia CIOCARLIE

Abstract: This article aims to present a system for making inference (derivation) on any logical knowledge base with the goal to achieve its saturation. Typical knowledge bases from literature are comprised of facts and deduction rules. In our work was considered an extra layer of this ontology: the one of constraints. So, in our case, the KB is a triple: $(\mathcal{F},\mathcal{R},\mathcal{N})$. The concrete implementation of the system for derivation was made in the object-oriented language C#. The KB is represented in OO environment as a collection of objects of classes corresponding to each type of knowledge. The system takes at input the knowledge base (in OO representation), applies the derivation algorithm and produces at output the saturated set of facts. The testing was done on three types of KBs: small, medium and large. Results and comparisons are presented to the reader in tabular form.

Key words: *knowledge base, derivation algorithm, inference engine, saturation, forward chaining*

1. INTRODUCTION

In this paper is proposed an inference engine for achieving the saturation of logical knowledge bases. The system uses at its basis an algorithm for making derivation that applies rules on facts in order to deduce new knowledge from the factual part and at the same time checks not to violate any negative constraint stated in the set. In ordinary logics this is known the *Modus Ponens* principle. Saturation of the knowledge base is the state when, after an application of rules on the facts no new knowledge can be deduced.Our system differs from the existing state-of-art solutions in the fact that it adds an extra layer to the ontology, which is that of (negative) constraints.

In what follows we will make an introduction into logical formalisms, presenting the basic theoretical notions related to the domain.

1.1.Basic Terms - Definitions

We will considere here only the positive existential syntactic fragment of First Order Logics, i.e. FOL(∃,∧), (Baget et al.) [2] which is the one comprised of formulas built with the quantifiers (∃,∧) and only the connectors implication (→) and conjunction (∧). No disjunction and negation is implied, and a special constant is used to denote falsity (⊥).

A ***Vocabulary*** is a triple consisting of thee disjoint sets:

$$Voc=(C,V,P) \qquad (1)$$

where C is a finite set of constants, V is an infinite set of variables and P a finite set of predicates. A function $ar:P\rightarrow N$ associates a natural number to a predicate p∈P that defines its arity. A term over *Voc* is a constant t∈C, or a variable t∈V.

An ***atomic formula*** (atom) over *Voc* is a predicate: $p(t_1,\ldots t_n)$, where p∈P, ar(p)=n, and $t_1,\ldots t_n$ are terms of *Voc,* that are constants or variables; in general former are represented in uppercase and latter in lowercase letters. A *ground atom* is a predicate that contains only constants. A *conjunct* is named a conjunction of atoms, and a *ground conjunct* is one of ground atoms. A variable of a formula is called *free* if it isn't found in the scope of any quantifier. A

formula is *closed* if it has no free variables (also named sentence).

The most primitive form to represent knowledge are *facts.* A fact can be represented by a ground atom because this signifies a primitive form of knowledge. To represent incomplete knowledge are implied existentially quantified variables. For example: *"Teacher John teaches a student we don't know the name of"* is represented in First Order Logics by formula:

$$\exists x(teacher(John) \wedge student(x) \wedge teaches(John,x)) \quad (2)$$

The second main formalism for knowledge representation are *rules.* Rules are logical formulas that allow to infer (derive, produce) of new facts from the explicit stated ones. They encode the so-called *intentional knowledge*, regarded as an ontology layer that reinforces the expressiveness of the knowledge base. Uses variables and unknown individuals and are known in other places as *existential rules,* such as in (Baget et al)[2]. An existential rule is a closed formula of the form:

$$R = \forall \vec{y}((\forall \vec{x} B) \rightarrow (\exists \vec{z} H)) \quad (3)$$

where B and H are called the rule's *body* and *head*, respectively. These are facts (conjuncts of atoms) in which vars(B)= $\vec{x} \cup \vec{y}$ and vars(H)= $\vec{z} \cup \vec{y}$, where the notations *vars* and *terms* are used to denote the set of variables and terms from a fact. Notation $\vec{x}$ is used as a shorthand for a sequence of variables $(x_1,x_2,\ldots x_n)$.

Existential rules bring with them new features, besides normal rules, some of them are: allow many atoms in the head of the rule (as shown in def.3), possibility to represent existential variables and unrestricted arity for predicates. This increase in expressiveness does not come alone but also with an increase in complexity of reasoning, some problems in the general existential rules framework are undecidable (e.g. implication, entailment). For classes of existential rules that ensure decidability while keeping expressiveness to be seen works of (Baget et al)[2], (Chein and Mugnier)[3].

The third form of knowledge is one used to represent negativity, that is to show how things are not ought to be; these are called *negative constraints*. This is the counterpart notion of integrity constraints from database systems, used to place restrictions on data and preserve its semantics. A negative constraint is a rule that has no head, only body:

$$N = \forall \vec{x}(B \rightarrow \bot) \quad (4)$$

Negative constraints play an important role in logical knowledge bases, they are devices used to detect inconsistencies in the factual part. Their triggering is interpreted as presence of inconsistency. An example would be:

$$N=retiredFrom(x,y) \wedge worksIn(x,y) \rightarrow \bot \quad (5)$$

that states that it is impossible for a person to be retired from a place and yet to work there.

After having specified the main logical formalisms of knowledge representation, we can now say what a knowledge base is.

A *knowledge base* (KB) over a vocabulary *Voc* is an entity: $\mathcal{K}=(\mathcal{F},\mathcal{R},\mathcal{N})$ consisting of a finite set of facts, rules and negative constraints.

Let V be a set of variables and T a set of terms T on a vocabulary. A *substitution* σ of V by T (denoted σ:V→T) is a mapping from V to T. Let F be a fact; σ(F) denotes the fact obtained from F by replacing each occurrence of x∈V∩ vars(F) by σ(x). A *homomorphism* from a fact F to a fact F' is a substitution σ of vars(F) by terms(F') such that σ(F) ⊆F'.

A rule R = B → H is applicable to a fact F if there is a homomorphism σ from B to F. The rule's application produces a new fact α(F,R, σ) = F∪σ^{safe}(H), where σ^{safe} is called safe substitution because it replaces existential variables with fresh (not introduced) ones. These are important not to attribute used variables to new facts, which causes problems when rules are reapplied on the new facts.

Rules can follow an order for application, i.e. it is possible that a rule R_2 is not applicable to any fact from the set but to become so after applying other rules. This gives birth to the concepts of *derivation sequence* and *R-derivation.*

Let F be a fact and $\mathcal{R}$ a set of rules. A fact F' is called an R-derivation of F if there is a finite sequence called *derivation sequence* $(F_0,F_1,\ldots F_n)$ in which F_0=F, F_n=F' and for $i=\overline{0,..n}$ there is a rule $R_i \in \mathcal{R}$ applicable to F_i and F_{i+1} is its immediate derivation. For more about

these notions reader is referred to (Baget et al.)[2] and Abdallah [1].

When having a set of facts $\mathcal{F}$ and one of rules $\mathcal{R}$ we are interested in unfolding all implicit knowledge from facts by applying the rules. This process is called *saturation procedure* and relies on a breadth-first search with forward chaining scheme [x],[y]. It begins from a derivation sequence in which F_0 is the initial set and at each step a new fact F_i is produced from the current fact F_{i-1} by computing all homomorphisms from the bodies of all rules to F_{i-1} and then make the rule applications. The fact obtained after step k is called *k-saturation* of $\mathcal{F}$. Next we will try to give a formal definition of these notions, as is shown in (Baget et al)[2].

Let F be a fact and $\mathcal{R}$ a set of rules. With $\Pi(\mathcal{R},F)$ we denote the set of homomorphisms from bodies of all rules to fact F.

$$\Pi(\mathcal{R}, F) = \{(R,\sigma) \mid R \in \mathcal{R} \text{ and } \sigma \text{ homorph. from body}(R) \text{ to } F\} \quad (6)$$

The *direct saturation* of an arbitrary fact F with $\mathcal{R}$ is defined:

$$Cl_R(F) = \bigcup_{(R,\sigma)\in\Pi(R,F)} \sigma^{\text{safe}}(\text{head}(R)) \quad (7)$$

The *k-saturation* of F with $\mathcal{R}$ is denoted by $\text{Cl}_{\mathcal{R}}^{k}(F)$ and is inductively calculated:

$$\text{Cl}_R^0(F) = F,$$

$$Cl_R^i(F) = Cl_R\left(Cl_R^{i-1}(F)\right), for\ all\ i > 0 \quad (8)$$

The saturation procedure is known in the database community as *naïve chase* (Cali et al.)[3]. It has been used in database reparations that do not respect its functional dependencies. Other types of chases had been proposed in literature, differing only on how they handle existential variables and redundancy. For more about this to be seen (Deutsch et al.)[7] and (Marnette)[10].

1.1.Saturation: Example

Overall, what we can do when having a set of facts supplied by a set of rules and one of negative constraints is to use the rules in order to derive all implicit knowledge from the facts and in the same time be careful not to violate any negative constraints stated in the set. The result of this procedure is a set of facts extending the initial one. In ordinary logic this is called application of the Modus Ponens principle, and in Datalog is referred as Elementary production Principle (Ceri et al)[4]. In order for it to work, the body of the rule should map on some facts from the set, other said, there is a substitution of variables that makes the body of the rule resemble to some facts from the set.

Next we will present an example of how the saturation procedure works on a knowledge base, for the reader to better understand since this is the main scope of this research.

Let's consider the knowledge base $\mathcal{K}=(\mathcal{F},\mathcal{R},\mathcal{N})$:

- $\mathcal{F}$ ={p(A),q(B),s(B)}
- $\mathcal{R}$ ={p(x) →r(x,y), p(x)∧s(y) →p(y), q(x) →r(x,y), r(x,y) →t(x)}
- $\mathcal{N}$ = φ

The derivation algorithm is a breadth-first search which uses a forward chaining scheme that works by applying each rule on the initial set of facts and after one iteration the set of deduced facts is added to the initial one.In our example this is:

$\boldsymbol{Step1}$: $Cl_R^0(\boldsymbol{F}) = \boldsymbol{F}$

$\boldsymbol{Step2}$: $Cl_R^1(\boldsymbol{F}) = \boldsymbol{F} \cup \{r(A,y_1), \text{p(B)}, \text{r(B,}y_2\text{)}\}\ using\ R_1 with\ \sigma_1\{(x,A)\},\ R_2\ with\ \sigma_2 = \{(x,A),(y,B)\}, R_3\ with\ \sigma_3 = \{(x,B)\}$

$\boldsymbol{Step3}$: $Cl_R^2(F)$

$= Cl_R^1(F) \cup \{t(A)\}\ using\ R_4\ with\ \sigma_4$

$= \{(x,A),(y,y_1)\} and\ R_4 with\ \sigma_5$

$= \{(x,B),(y,y_2)\}$

$\boldsymbol{Step4}$: $no\ new\ facts\ are\ produced\ from\ rule\ application$

$\rightarrow procedure\ halts\ after\ 3\ steps$

2. RELATED WORKS

All the work presented in this paper has been done to implement a private project developed for the needs of an institution, Université de La Rochelle, France, in collaboration with some of the staff there.

For the theoretical part of this article, in order to familiarize with the notions of the domain of KR logical formalisms, 2 main resources studied by authors worth mention. First is the phd thesis of Arioua Abdallah [1] (one of the staff from Univ. LR), which is about knowledge representations formalisms, reasoning services, query answering in presence

of inconsistency. The second is the book of Dung [8], which was used by [1] to instantiate an argumentation framework using for representation the logical language Datalog+.

Some other important works in this field worth mention are those of (Poggi et al.)[11] and (Croitoru et al.)[6], which are all about advanced notions and concepts of this field.

Rule-based engines are frameworks that have been created in object-oriented languages in the scope to add a logic layer into the applications and to separate it from data, which brings a series of advantages. We will enumerate here the ones from C#, since this is the language used to implement also our inference engine system: NRules, SimpleRules.Net, NxBRE, DROOLS.Net, etc. For more information about these reader is referred to [12], [13], [14], [15].

None of these systems can be used to achieve our objective, that is to deduce new facts by inference while checking not to cross any negative constraints from the set, and the process repeats until is obtained saturation. This is why we had to design and implement from the scratch our own inference engine for this special purpose that respects the stated requirements.

The novelties brought by our derivation system compared to the rule engines from the literature are:

- knowledge base, in its pure logical form that is stored in files, is formalized in pure FOL syntax, in order to be understood by reader who is familiar with this syntax (and not some unknown other); thus our system can be seen as an FOL syntax parser
- adds a new class of knowledge that is used to impose restrictions on the factual part (constraints) in order to preserve consistency of the knowledge base: facts newly produced are validated against the set of constraints not to violate them and if they do then are not added to the memory
- increased expressivity of knowledge: are represented also the variables used in KR formalisms (facts, rules) and even their logical quantifiers, which can be useful to capture and show complex First Order Logics formulae
- use optimization strategies: forward chaining and Backtracking algorithm for the process of finding facts to match a rule's body in an iterative, progressive and ordered fashion, which simplify computations especially in case of large knowledge bases (sets of fact and rule)

3. SYSTEM DESIGN

In order to be able to process it and apply the algorithm for derivation we must find a way to represent the logical knowledge as an object-oriented structure. Next we will show our considered design approach.

3.1. Object Structure

Each logical KR formalism (fact, rule, negative constraint) is represented by a type class that defines instance variables and methods specific to that formalism.

An atom, which is the most basic form of data, was represented in OO by a class Atom that has as attributes its name and a collection of its values. Defines 2 constructors and 2 methods (besides get/set) to print the atom in the normal form to the user: $p(x_1,x_2,\ldots x_n)$.

A fact, as it was stated in the definition from introduction, is the existential closure of a conjunction of atoms. This was represented in OO by a class Fact that has data a collection of Atom objects and, similar with Atom class, defines 2 constructors and 2 methods for the pretty-print of the fact in its usual form to us.

A rule was represented by a class Rule that has attributes two facts: one is the body (premise) and another is the head (conclusion). A negative constraint is a rule that has no head, so this was specified by a class that has as data only one fact (the body). This structure is presented in Fig.2, that represents the UML class diagram of the inference system.

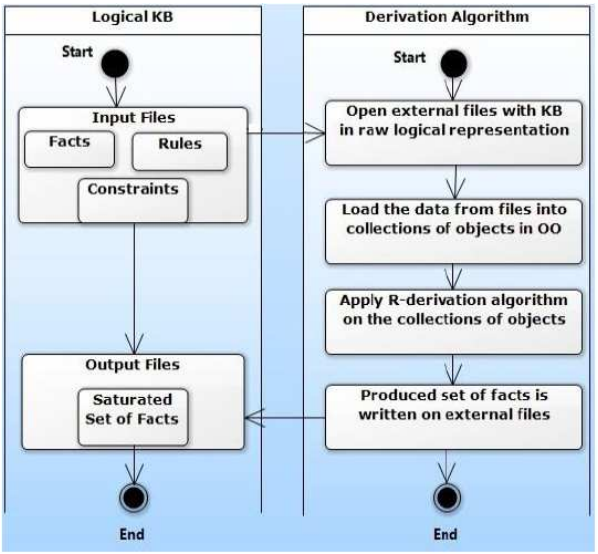

Fig.1: UML activity diagram of the derivation system

The logical knowledge base is stored into external files, one for each KR type (fact, rule, constraint) and is represented in the raw First Order Logic syntax. The main operations performed by the derivation system are the following:

i. read the external files of the KB, loads data into objects of the system and creates collections to store all the objects
ii. applies the derivation algorithm over the knowledge base stored in objectual form (collections of objects) in order to deduce all implicit knowledge from the explicit ones; the process goes recursively until the KB saturation is achieved
iii. after the set of all new facts (saturated) has been computed, translate them into FOL formulae and append them on the file with the initial set of facts

This process is shown in Fig.1, which represents the UML activity diagram of system.

3.2. Algorithm for Derivation

The core algorithm that is used by our inference engine to deduce new facts can be specified using the following pseudo-code:

Algorithm R-derivation:

Input: facts set, rules set, constraints set

Output: set of deduced facts

Variables: Dictionary[] (stores the variables of a rule together with their values found during the matching on facts)

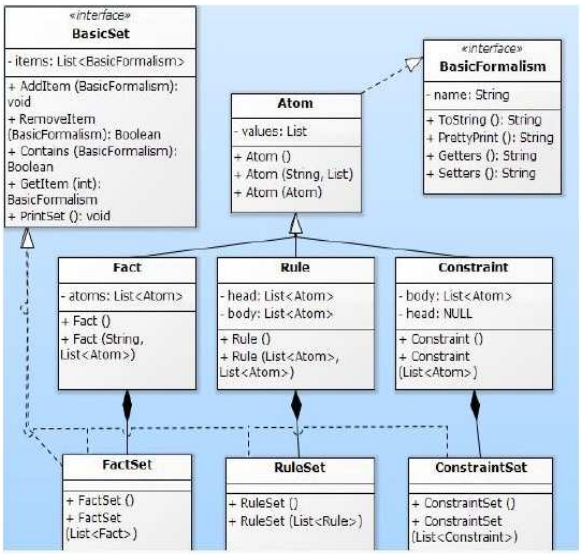

Fig.2: UML class diagram of the system

Begin

while (new facts are produced from rule applications)

begin

1. *Take each rule from the set and try to find a set of facts that match its body*
2. *Take each atom (conjunct) from the rule's body and try to find a fact in the set that match it: 3 steps process*
   - *2.1. Check to see if has the same name with the fact*
   - *2.2. Check to see if has same arity with the fact*
   - *2.3. Check into the Dictionary[] if the atom has variables that have been assigned (this is the case in previous conjunct matches); if Yes then check if their values are same with the corresponding ones from the fact*
3. *If all the above 3 conditions have been filled then the fact matched the conjunct, so the conjunct's variables are substituted in the Dictionary[] with the corresponding values from the fact*

4. *If there was a conjunct from the crt. rule that was not matched by any fact from the set then break the search of next conjuncts, since the crt. rule cannot be matched*
5. *If all conjuncts of the rule have been matched on facts from set then the rule was matched, so is computed the fact from the head based on the values of variables from Dictionary (the substitution of the rule)*
6. *Validate the newly produced fact on the set of constraints in order not to violate some*
7. *If the fact is valid then add it to the set of produced facts, else discard it*

end_w

End.

4. IMPLEMENTATION

For the actual implementation of our system we headed towards the C# programming language. We chose this option because of the numerous advantages that the .NET platform has, such that is a much younger platform (e.g. than Java), portability (projects can be run on multiple platforms), scalability and reliability, non-verbose and easy to code syntax.

The development environment that was used to build the project is Microsoft Visual Studio 2012.

The idea was to represent the knowledge base as an object-oriented structure in order to apply the derivation algorithm on it and deduce all implicit knowledge by matching rules on facts. As it was shown in the design phase, each logical form of data was specified by a class, and each set of logical knowledge was represented by a collection of objects instances of a particular class type. In what follows we'll dive into more details about the implementation and inner workings of the core component of this engine, the inference algorithm.

In the implementation of the algorithm for derivation we considered two case scenarios regarding the structure of facts, which lead each one to a different strategies of implementation. The first version considers that facts are single atoms of data, such as $p(x_1,x_2,x_3)$ and the second version considers facts in their generic form, conjuncts of atoms (as it was shown in def. of section 1).

Version 1

In this case, in order to find a subset of facts that match the body of a rule, the algorithm takes each atom (conjunct) of the rule and look up in the set of facts to find one that matches it based on all necessary 3 conditions: name, arity and values of variables (if assigned in previous conjuncts matches). If all atoms from the rule's body have been iterated and found facts for them then the rule matched. The next step is to deduce the conclusion, for that is created the new fact from the rule's head that has the same name with the rule's conjunct and as values it has those of the variables of conjunct as have been found during the matching process of the rule and that are stored into the Dictionary[] data structure.

It can be well thought that this problem of matching a rule on a set of facts does not have a single solution, but multiple, since multiple sets of facts can be found that match the body of a rule. As it is well known, in the case of problems that have multiple solutions is used the Backtracking algorithm in order to walk through all possibilities and find every possible solution to the given problem. I think the most well-known problem that relies on Backtracking algorithm in order to find its solutions is "Queens problem", which asks how to place N queens on a NxN chess board with condition not to attack one another.

The Backtracking algorithm involves searching through n sets S_i for elements of a result set X between which must exist a relation φ. Our currently studied problem is very similar, implying searching n times through a set for elements that must fulfill 3 conditions in order to be included in the final result set of facts X of n elements. After a solution was found the algorithm goes back to the previous conjunct being checked and reiterates through the set of facts starting from the point it was left at last found, in order to find another fact that could match the conjunct. The algorithm stops when its backtrack counter variable reaches value 0, which means it has been walked through all possibilities and all solutions found.

Version 2

This version of the algorithm considers the facts in their general form, as conjunctions of atoms. This is a much difficult to implement case since now it cannot be iterated through conjuncts of the rule and search for them facts in the set, since now facts are not single atoms of data. Now the strategy is somehow reversed, the set of facts being iterated and for each one we look-up the array of conjuncts of the rule for a subset that could match the fact currently checked. It ends when all conjuncts of the rule had been matched. This strategy also relies on the Backtracking algorithm for finding all solutions to the problem, but the search space and result vector are now others.

5. EXPERIMENTS AND RESULTS

For the testing phase of our inference system we used 3 knowledge bases of different sizes:

- small: 6 items in each set of knowledge (units)
- medium: 60 items in each set of knowledge (tens)
- large: 200 items in each set of knowledge (hundreds)

Our algorithm based on optimization techniques (presented earlier) has been compared with a more primitive solution who works by making all possible combinations of facts of length equal to that of the rule, then verify to see if each combination is a match. This way all solutions are guaranteed to be found since the whole space of solutions is searched, but the drawback is the high volume of time and computations required. Results obtained are presented in the next two tables, one for each version of the derivation algorithm, showing the number of operations and time required by each method.

The tests were performed on an Asus Vivobook laptop having an Intel Dual Core @ 1.5GHz with x64 architecture, 1.5GB DDR3 and 240GB hard-disk space running on a Windows 10 operating system. To be precised that the tests results may vary greatly on the machine on which are done, depending on its resources, architecture and many more.

TABLE 1

Tests results of version 1 of algorithm

Solution	Knowledge Base					
	Units		Tens		Hundreds	
	Ops. (10^3)	Tm. (ms)	Ops. (10^6)	Tm. (ms)	Ops. (10^9)	Tm. (s)
Optimal	1,585	10	2,556	690	0,444072	3,94
Basic	4,005	33	4,890	990	1,880432	7,95

TABLE 2

Tests results of version 2 of algorithm

Solution	Knowledge Base					
	Units		Tens		Hundreds	
	Ops. (10^3)	Tm. (ms)	Ops. (10^6)	Tm. (ms)	Ops. (10^9)	Tm. (s)
Optimal	1,858	15	3,506808	990	0,999548	6,94
Basic	5,115	43	5,670880	1880	3,770032	11,95

6. CONCLUSIONS

As it has been said in section 2, all the work presented in this article has been filed for the development of a private project for the needs of a university in France (Universite de La Rochelle), and everything presented here are original contributions, ideas and solutions of the authors.

The system presented here is intended an inference engine that can operate on any logical knowledge base. The main novelty brought by our work is that we consider knowledge bases which have an extra layer of security on data, represented by negative constraints, so our system not only does simple applications of rules on facts but also validates the produced knowledge with the stated restrictions. Another important feature brought is that was designed from scratch for the single goal of saturation of the KB. We could not find something similar in literature to which we can compare our work, the only systems related are the business rule engines frameworks from OO programming languages, but we cannot use these to achieve the goal of our system, and especially for our type of KB. A comparison has been done in section 2 of this article.

The system can be found on the author's drive: https://drive.google.com/open?id=1QzbIogncFL-b-zymGP2JhtXtQN9ZQwDT, together with the 3 knowledge bases which were used to perform the tests:

https://drive.google.com/open?id=13QZP8MhbpriTTI6F7Dgiu7uDwmdlcdZ-

REFERENCES

[1] A.Abdallah, *Formalizing and Studying Dialectical Explanations in Inconsistent Knowledge Bases* Phd Thesis, Universite de La Rochelle, October 2016.

[2] J.F.Baget, M.Leclre, M.Mugnier, E.Salvat, *On Rules with Existential Variables: Walking the Decidability Line*, Artificial Intelligence, vol.175, pp.1620-1654.

[3] A.Cali, G.Gottlob, M.Kifer, *Taming the Infinite Chase: Query Answering under Expressive Relational Constraints*, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (2008), pp.70-80.

[4] S.Ceri, G.Gottlob, L.Tanca, *What You Always Wanted to Know about Datalog*, IEEE Transactions on Knowledge and Data Engineering (1989)

[5] S.Ceri, G.Gottlob, L.Tanca, *Logic Programming and Databases*, Science and Business Media, Springer (2012).

[6] M.Croitoru, S.Vesic, *What can Argumentation do for Inconsistent Ontology Query Answering?,* Proceedings of the International Conference on Scalable Uncertainty Management, pp.15-29, Springer (2013).

[7] A.Deutsch, A.Nash, J.Rammel, *The Chase Revisited,* Proceedings of the 27th ACM-SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM 2008, pp.149-158.

[8] P.M.Dung, *On the Acceptability of Arguments and its Fundamental Role in Non-monotonic Reasoning, Logic Programming and n-Person Games*, Artificial Intelligence, vol.77, pp.321-357, (1995).

[9] D.Lembo, M.Lenzerini, R.Rosatti, D.Savo, *Inconsistency-Tolerant Semantics for Description Logics*, Proceedings of the International Conference on Web Reasoning and Rule Systems, pp.103-117, Springer-Verlag (2010).

[10] B.Marnette, *Generalized Schema Mappings: from Termination to Tractability*, Proceedings of the 28th ACM-SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM 2009, pp.13-22.

[11] A.Poggi, D.Lembo, D.Calvanese, G. De Giacomo, M.Lenzerini, *Linking Data to Ontologies*, Journal on Data Semantics X, pp.133-173, Springer (2008).

[12] DROOLS.NET: https://sourceforge.net/projects/drooldotnet/

[13] SimpleRules.NET: https://www.codeproject.com/Articles/1181151/SimpleRules-Net-Easy-to-use-Rules-Engine

[14] NRules: https://github.com/NRules/NRules/wiki

[15] NxBRE: https://sourceforge.net/projects/nxbre/

Un sistem de inferenta pentru obtinerea saturatiei bazelor de cunostinte (F, R, N)

***Rezumat**: Acest articol isi propune sa creeze un sistem pentru realizarea de inferente (derivatii) asupra oricarui tip de baza de cunostinte logica cu obiectivul final sa ii obtina saturatia. Bazele de cunostinte din literatura de specialitate sunt in general constituite dintr-un set de fapte si unul de reguli de deductie. In lucrarea noastra am mai adaugat un nivel la aceasta ontologie: unul de constrangeri. Deci, in cazul nostru, o baza de cunostinte este o tripla* $(\mathcal{F},\mathcal{R},\mathcal{N})$. Inplementarea concreta a sistemului de derivatii a fost facuta intr-un limbaj orientat-obiect, anume Java. Baza de cunostinte este reprezentata in mediul OO ca niste colectii de obiecte ale claselor corespunzatoare fiecarui tip de cunostinte. Sistemul primeste la intrare baza de cunostinte in reprezentare obiectuala, aplica algoritmul de derivare si produce setul de fapte saturat. Testele au fost efectuate asupra a 3 tipuri de baze: mica, medie si mare. Rezultatele si comparatiile sunt prezentate in forma de tabele.

Andrei ZAMFIRA, DrD, Politehnica University of Timisoara, Dept. of Computer Science, andreizamfira@gmail.com;

Horia CIOCARLIE, Prof., Politehnica University of Timisoara, Dept. of Computer Science, E-mail : horia.ciocarlie@cs.upt.ro