STUDIA INFORMATICA 2014
Volume 35 Number 2 (116)

Jarosław BĄK, Maciej NOWAK, Czesław JĘDRZEJEK
Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej

# WNIOSKOWANIE REGUŁOWE Z ONTOLOGIAMI OWL 2 RL

**Streszczenie**. W pracy zaprezentowano metodykę transformacji ontologii OWL 2 RL do zestawu reguł, które mogą zostać użyte w regułowym silniku wnioskującym. Zaproponowano podział wnioskowania na dwie części: terminologiczną i asercyjną. Opracowano mechanizm transformacji ontologii do Abstrakcyjnej Składni Reguł i Faktów. Umożliwia to zamianę ontologii na język dowolnego silnika wnioskującego. W artykule opisano implementację dla silników Jess i Drools. Przeprowadzone eksperymenty pokazują, że opracowane narzędzie radzi sobie dużo lepiej we wnioskowaniu na danych niż w narzędziu HermiT.

**Słowa kluczowe**: ontologie, systemy regułowe, wnioskowanie

# RULE-BASED REASONING WITH OWL 2 RL ONTOLOGIES

**Summary**. In this paper we present a method of transforming OWL 2 RL ontologies into a set of rules which can be used in a rule engine. We use HermiT reasoner to perform the TBox reasoning. The ontology is transformed into an Abstract Syntax of Rules and Facts and then into any forward chaining reasoning engine. We present an implementation of our method using Jess and Drools engines. The evaluation performed on benchmark ontologies shows that we can perform the ABox reasoning with considerably better performance than HermiT.

**Keywords**: ontologies, rule-based systems, reasoning

## 1. Wprowadzenie

W ostatniej dekadzie wykorzystanie ontologii w systemach informatycznych stało się coraz bardziej popularne w różnych dziedzinach, takich jak technologie internetowe, integracja baz danych, systemy wieloagentowe, przetwarzanie języka naturalnego i w wielu innych. Jednym z najbardziej popularnych sposobów wyrażania ontologii jest użycie języka Web

Ontology Language (OWL) [20]. Opiera się on na logikach deskrypcyjnych (ang. *Description Logics*), które są rodziną języków reprezentacji wiedzy. Powstające aplikacje, wykorzystujące język OWL, ujawniły wiele ograniczeń i wad tego języka [6]. W rezultacie została opracowana nowsza jego wersja – OWL 2 [18]. Zawiera ona kilka ulepszeń w stosunku do OWL 1.0 oraz OWL 1.1. Co więcej, język OWL 2 definiuje trzy profile, które oferują istotne korzyści w różnych zastosowaniach. Te profile to: OWL 2 EL, OWL 2 QL oraz OWL 2 RL. Profil OWL 2 EL opiera się na rodzinie *EL*++ logik deskrypcyjnych, które są przeznaczone do efektywnego wnioskowania na dużych terminologiach [1]. Drugi profil, OWL 2 QL, opiera się na logice deskrypcyjnej DL-Lite, która została zaprojektowana w celu umożliwienia sprawnego wnioskowania na dużych zestawach danych o stosunkowo prostym schemacie [4]. Ostatni profil, OWL 2 RL, został zaprojektowany do przeprowadzania wnioskowania w systemach wnioskowania regułowego przez wdrożenie zestawu predefiniowanych reguł. Opiera się na różnych logikach deskrypcyjnych, takich jak Horn-SHIQ [14] i Description Logic Programs [7]. Każdy z profili jest zdefiniowany jako odpowiednie ograniczenie składni (dopuszczalnych konstrukcji) języka OWL 2 [18].

W celu wykorzystania wszystkich możliwości, jakie zapewnia ontologia, należy zastosować silnik wnioskujący, który umożliwia wnioskowanie w części terminologicznej (TBox) oraz w części asercyjnej (ABox). Można zastosować różne silniki wnioskowania dla ontologii wyrażonych w różnych profilach OWL 2 [18] (jak również w różnych fragmentach OWL 1.1[1], np. Horn-SHIQ). Na przykład dla ontologii w profilu OWL 2 możemy użyć silnika HermiT[2], ale dla ontologii w profilu OWL 2 QL, której siła ekspresji jest stosunkowo ograniczona, możemy użyć silnika REQUIEM[3]. Wynika z tego, że odpowiednie wybranie silnika wnioskującego dla danej ontologii jest istotne, gdyż bezpośrednio wpływa na efektywność procesów wnioskowania, jak również przetwarzania zapytań i udzielania odpowiedzi.

Wykorzystując silnik wnioskujący, możemy wykonywać zadania wnioskowania w dwóch składowych częściach ontologii będącej bazą wiedzy KB: TBox i ABox. TBox jest to część terminologiczna zawierająca klasy i relacje oraz aksjomaty opisujące, w jaki sposób są one ze sobą powiązane. Typowe zadania wnioskowania dla części TBox to sprawdzenie spełnialności danej deskrypcji (klasy lub relacji) oraz tego, czy deskrypcja subsumuje inną deskrypcję (czy jedna jest bardziej ogólna niż druga). Natomiast ABox to część asercyjna, która zawiera aksjomaty stwierdzające, że indywiduum jest instancją danej klasy, a para indywiduów jest instancją danej relacji. Typowe zadania wnioskowania dla części ABox to ustalenie, czy jest ona spójna oraz czy indywiduum jest wystąpieniem danej deskrypcji. Zadania rozstrzygalności i zgodności są wykonywane w celu określenia, czy baza wiedzy jest spójna i nie

[1] http://www.w3.org/Submission/owl11-overview/
[2] http://www.hermit-reasoner.com/
[3] http://www.cs.ox.ac.uk/isg/tools/Requiem/

zawiera sprzeczności. Dzięki temu część TBox może być postrzegana jako schemat KB, który wyraża wiedzę dziedzinową, podczas gdy część ABox stanowi zbiór danych, który przechowuje informacje o poszczególnych obiektach. Przy użyciu odpowiedniego silnika możemy przeprowadzić wnioskowanie w ontologii w celu uzyskania dodatkowych informacji w postaci nowej (sklasyfikowanej) części TBox oraz nowej (wywnioskowanej) części ABox.

W niniejszym artykule przedstawiamy narzędzie, które jest w stanie wykonać wnioskowanie opierające się na ontologii za pomocą standardowego silnika wnioskującego w przód. Praca opisuje następujące elementy:

- metodę transformacji ontologii OWL 2 do zestawu reguł i zbioru faktów (jeśli ontologia zawiera ABox),
- Abstrakcyjną Składnię Reguł i Faktów (ASRF),
- schemat wnioskowania zgodnego z zaproponowaną metodyką,
- implementację opracowanego podejścia z wykorzystaniem dwóch silników wnioskujących w przód: Jess [8] i Drools[4],
- eksperymentalną ocenę zaproponowanego podejścia z wykorzystaniem ontologii zgodnych z OWL 2.

W niniejszym artykule koncentrujemy się na wnioskowaniu opierającym się na ontologii przy użyciu standardowego silnika wnioskowania w przód. W związku z tym zajmujemy się przede wszystkim profilem OWL 2 RL, jednakże przedstawiona metodyka może posłużyć do obsługi ontologii wykraczających poza OWL 2 RL. Jest to możliwe dzięki zastosowaniu narzędzia HermiT służącego do wnioskowania w TBox (terminologicznej części ontologii). W rezultacie możemy wykorzystać mechanizm wnioskowania opierający się na regułach do przeprowadzenia wnioskowania w ABox (asercyjnej części ontologii). Jest to zgodne z ideą OWL 2 RL (możliwość skalowalnego wnioskowania bez znacznej utraty siły wyrazu).

Pozostała część niniejszej pracy jest zorganizowana w następujący sposób. Kolejna sekcja przedstawia motywacje, które kierowały autorami. Rozdział 3 opisuje metodę transformacji ontologii OWL 2 do zestawów reguł i faktów. W rozdziale 4 została zaprezentowana Abstrakcyjna Składnia Reguł i Faktów. W rozdziale 5 zawarte są szczegóły dotyczące implementacji. Eksperymenty oraz ich wyniki zostały przedstawione w rozdziale 6. Rozdział 7 to zbiór prac o podobnej tematyce, natomiast rozdział ostatni zawiera wnioski i dalsze kierunki badań.

[4] http://www.jboss.org/drools

## 2. Motywacje

Z praktycznego punktu widzenia, terminologiczna część ontologii rzadko jest modyfikowana, w przeciwieństwie do części asercyjnej. Dzięki temu można oddzielić TBox od ABox. W rezultacie można zastosować różne schematy wnioskowania, jak również silniki wnioskujące. Co więcej, wnioskowania w części TBox należy wykonywać tylko raz (lub z każdą zmianą tej części), np. przy użyciu silników wnioskujących w logikach deskrypcyjnych. Następnie można wykonać wnioskowanie w części ABox za pomocą silnika regułowego (za każdym razem, gdy pojawią się nowe asercje). Tego typu podejście zostało przedstawione w [17], gdzie zastosowano silnik Pellet[5] z systemem regułowym Jena[6].

Według oficjalnej listy[7] silników wnioskujących dla OWL 2 niewiele z nich wspiera profil OWL 2 RL. Stanowi to dodatkową motywację, aby zapewnić metodykę efektywnego wnioskowania w OWL 2 RL w systemie regułowym (ponieważ silniki wnioskujące w DL obsługują wnioskowanie w TBox efektywniej niż w ABox, które z kolei może być wykonywane efektywniej przez silniki regułowe [16]). Niemniej jednak prosta implementacja silnika wnioskującego w OWL 2 RL słabo radzi sobie z dużymi zbiorami danych [10]. Ponadto niniejsza praca nie ogranicza się do konkretnego silnika lub konkretnej implementacji. Zamiast tego stanowi próbę zaproponowania schematu działania systemu wnioskującego w OWL 2 RL z dowolnym silnikiem wnioskującym w przód. Dodatkowo opracowana Abstrakcyjna Składnia Reguł i Faktów może być łatwo zastosowana w różnych narzędziach. Ostatnim motywem podejmowanych działań jest interoperacyjność z powszechnie stosowanym językiem Semantic Web Rule Language (SWRL) [13], która jest wskazana w wielu zastosowaniach praktycznych. W związku z tym składnia ASRF obejmuje zbiór funkcji wbudowanych języka SWRL (SWRL Built-ins).

Prezentowana praca jest wstępną wersją systemu RuQAR (ang. *Rule-based Query Answering and Reasoning*) mającego służyć do wnioskowania i realizacji zapytań. W niniejszym artykule przedstawiono jedynie funkcjonalność związaną z wnioskowaniem.

## 3. Transformacja ontologii OWL 2

Użycie regułowego silnika wnioskującego wykorzystującego ontologiczną bazę wiedzy wymaga metody transformacji ontologii do zestawu reguł. Ze względu na zastosowanie głównie profilu OWL 2 RL mechanizm wnioskowania został podzielony na dwa procesy

[5] http://clarkparsia.com/pellet/
[6] http://jena.apache.org/
[7] http://www.w3.org/2001/sw/wiki/OWL/Implementations

składowe: wnioskowanie w części TBox i ABox. Zgodnie z tym założeniem opracowano metodę transformacji ontologii OWL 2 do zestawu reguł i zestawu faktów. W tym przypadku wnioskowanie można wykonać oddzielnie dla każdej z części. Ze względu na fakt, że niniejsza metoda ma mieć zastosowanie w wielu regułowych silnikach wnioskujących, zaproponowano Abstrakcyjną Składnię Reguł i Faktów (ASRF), umożliwiającą prostą transformację ontologii OWL 2 na język silnika wnioskującego.

Rys. 1. Schemat transformacji ontologii OWL 2
Fig. 1. Scheme of OWL 2 ontology transformation

Schemat transformacji ontologii OWL 2 na zestaw reguł i zbiór faktów wyrażonych w ASRF przedstawiono na rys. 1. Najpierw ontologia OWL 2 jest ładowana do silnika HermiT, przy założeniu że jest ona spójna. Następnie wykonywane jest wnioskowanie w części TBox. W efekcie uzyskuje się nową wersję (nowy TBox) ontologii. Wreszcie ontologia jest przekształcana na dwa zestawy: zbiór reguł i zbiór faktów (jeśli ontologia zawiera część asercyjną). Oba zestawy są wyrażone w notacji ASRF. W ten sposób została oddzielona część TBox (zbiór reguł) od części ABox (zbiór faktów). W wyniku takiego podziału można wykonać wnioskowanie w części ABox z wykorzystaniem regułowego silnika wnioskującego.

Transformacja wywnioskowanej części TBox do zestawu reguł odbywa się zgodnie z regułami OWL 2 RL/RDF [18]. Wygenerowane reguły dzielą się na dwa rodzaje: reguły równości i reguły zależne od ontologii (tzw. reguły ABox). Reguły równości są zaczerpnięte bezpośrednio z tabeli 4 profilu OWL 2 RL. Reguły te są wyrażone w ASRF i są aksjomatyzacją semantyki równości w OWL 2. Są one niezależne od ontologii w przeciwieństwie do reguł Abox, które odzwierciedlają semantykę transformowanej ontologii.

Transformacja ontologii (po klasyfikacji wykonanej przez narzędzie HermiT) do zestawu reguł ASRF jest wykonywana w następujący sposób. Dla każdej deskrypcji klasy lub relacji, która jest zgodna z odpowiednim aksjomatem OWL 2 RL i odpowiednią regułą z tabeli 1, tworzona jest reguła odzwierciedlająca semantykę tej deskrypcji. Oznacza to, że zamiast zamieniać semantykę języka OWL 2 do reguł, tworzone są reguły odpowiadające tej semantyce i danej ontologii. Przykładowo, mając relację *hasSibling*, która jest zdefiniowana jako relacja symetryczna, można utworzyć regułę, która odzwierciedla fakt, że w przypadku wystąpienia instancji tej relacji symetryczna instancja również powinna wystąpić (użyto następujących skrótów: *S* dla podmiotu, *P* dla orzeczenia i *O* dla obiektu):

$$\begin{aligned} &\textit{If} \quad (\textit{Triple}\ \ (S\ ?x)\ \ (P\ \text{“}\textit{hasSibling}\text{”})\ \ (O\ ?y)) \\ &\textit{Then} \quad (\textit{Triple}\ \ (S\ ?y)\ \ (P\ \text{“}\textit{hasSibling}\text{”})\ \ (O\ ?x)) \end{aligned} \tag{1}$$

Reguła (1) jest zgodna z semantyką reguły *prp-symp* z tabeli 5 w specyfikacji profilu OWL 2 RL. Dla każdego aksjomatu z OWL 2, który występuje w danej ontologii, generowana jest semantycznie równoważna reguła zawierająca bezpośrednie odniesienie do ontologii. Generowana reguła jest instancją odpowiedniej reguły OWL 2 RL/RDF dla konkretnego zbioru TBox. Reguły te mogą być postrzegane jako reguły związane z instancją ontologii (tzw. reguły ABox). Są one zależne od ontologii, ponieważ wyrażają jej semantykę i są przeznaczone do wnioskowania na faktach (części ABox). W rezultacie uzyskuje się zbiór reguł, które mogą być stosowane w silniku wnioskującym w przód po wcześniejszej translacji składni ASRF na język tego silnika. Prezentowane podejście umożliwia tylko i wyłącznie wnioskowanie z częścią asercyjną ontologii. Ma to pozytywny wpływ na wydajność wnioskowania, ponieważ semantyka TBox jest bezpośrednio wyrażona w regułach ASRF. Ponadto dodatkowe korzyści wynikają z faktu, że liczba warunków w ciele każdej reguły jest mniejsza niż w odpowiadającej jej regule OWL 2 RL/RDF.

Tabela 1

Reguły wynikowe obecnie obsługiwane przez OWL 2 RL

| **Tabele ze specyfikacji OWL 2 RL** | **Wspierane reguły** |
|---|---|
| Table 4.<br>The Semantics of Equality | eq-sym, eq-trans, eq-rep-p,<br>eq-rep-s, eq-rep-o |
| Table 5.<br>The Semantics of Axioms<br>about Properties | prp-dom, prp-rng, prp-fp,<br>prp-ifp, prp-symp, prp-trp,<br>prp-eqp1, prp-spo1, prp-eqp2,<br>prp-inv1, prp-inv2 |
| Table 6.<br>The Semantics of Classes | cls-int1, cls-int2, cls-uni,<br>cls-svf1, cls-svf2, cls-avf,<br>cls-hv1, cls-hv2, cls-maxc2 |
| Table 7.<br>The Semantic of Class Axioms | cax-sco, cax-eqc1,<br>cax-eqc2 |

Tabela 1 przedstawia reguły obsługiwane w naszej aktualnej implementacji. Zestaw pochodzi ze specyfikacji OWL 2 [18] i zawiera mniejszą liczbę reguł niż w oryginale. Zdecydowano się użyć najprostszego podzbioru reguł OWL 2 RL/RDF, który jest łatwy do zaimplementowania w dowolnym silniku wnioskującym. Co więcej, zrezygnowano z reguł typu „ograniczenia" (np. CLS – nothing2 z tabeli 6 w profilu OWL 2 RL) oraz z każdej reguły, która nie ma wpływu na wnioskowanie w ABox (np. wszystkie reguły z tabeli 9 w profilu OWL 2 RL). Jednakże część reguł wymaga jeszcze implementacji, np. CLS – maxqc3 z tabeli 6 specyfikacji OWL 2 RL.

Przedstawiona metoda transformacji podczas wnioskowania może generować więcej rezultatów niż w tradycyjnym zastosowaniu reguł OWL 2 RL/RDF. Jest to spowodowane faktem wcześniejszego zastosowania wnioskowania w terminologii z użyciem narzędzia opierającego się na DL. W dużej mierze zależy to od ekspresywności danej ontologii. Niemniej stosując przedstawioną metodę do ontologii wykraczającej poza profil OWL 2 RL, nie uzy-

ska się tych samych rezultatów co w silniku opierającym się na DL (np. HermiT). W tym przypadku wyniki zastosowania reguł ASRF są poprawne, ale nie kompletne. Wspomniana sytuacja wystąpiła podczas testów LUBM, w przypadku których wszystkie wyniki uzyskane w ramach prezentowanej transformacji były wśród wyników pochodzących z narzędzia HermiT, jednak ich liczba była większa w przypadku drugiego narzędzia. Jest to poprawne działanie ze względu na fakt, że ekspresywność LUBM wykracza poza profil OWL 2 RL.

## 4. Abstrakcyjna Składnia Reguł i Faktów

Jak wspomniano w poprzednim rozdziale, wnioskowanie w TBox przeprowadza się z wykorzystaniem silnika HermiT. W kolejnym kroku ontologia jest automatycznie przekształcana do Abstrakcyjnej Składni Reguł i Faktów. Głównym celem opracowanej składni jest podniesienie poziomu abstrakcji i zapewnienie tym samym uniwersalnej reprezentacji reguł i faktów. Zastosowanie wyrażeń ASRF wymaga mapowania pomiędzy elementami ASRF a językiem wybranego silnika wnioskującego.

Do wyrażenia Abstrakcyjnej Składni Reguł i Faktów użyto notacji Extended Backus-Naur Form (EBNF) [21]. Gramatyka bezkontekstowa została przedstawiona na rys. 2. Symbole nieterminalne zapisano w nawiasach trójkątnych (< i >), natomiast pozostałe symbole są symbolami terminalnymi.

```
<Rule>              ::=   If <ConditionalAtom>+
                          Then <Atom>+
<ConditionalAtom>   ::=   <Atom> |
                          <Logic-operator> <ConditionalAtom> |
                          <Comparison>
<Atom >             ::=   ( Triple ( Subject <Argument> )
                              ( Predicate <Constant> )
                              ( Object <Argument> ) ) |
                          ( Triple <Term>+ )
<Logic-operator>    ::=   AND | OR | NOT
<Term>              ::=   ( <TermType> <Argument> )
<TermType>          ::=   Subject | Predicate | Object
<Comparison>        ::=   ( <Argument> <Comparator> <Argument> )
<Argument>          ::=   <Constant> | <Variable>
<Comparator>        ::=   equal | not equal | greater than |
                          greater than or equal | less than |
                          less than or equal | different from
<Fact>              ::=   ( Triple ( Subject <Constant> )
                          ( Predicate <Constant> )
                          ( Object <Constant> ) )
<Constant>          ::=   Skończona sekwencja znaków.
<Variable>          ::=   Skończona sekwencja znaków poprzedzona '?'.
```

Rys. 2. Abstrakcyjna Składnia Reguł i Faktów w notacji BNF
Fig. 2. Abstract Syntax of Rules and Facts in EBNF notation

Notacja składni ASRF opiera się na logice pierwszego rzędu. Każdy fakt to atom, który składa się z termów. Każdy term to zmienna (poprzedzona znakiem zapytania) lub stała. Ponadto każdy term jest jednym z następujących typów: *podmiot*, *orzeczenie*, *obiekt* lub *argument*. Podobnie każdy atom może być rodzaju: *Trójka* lub *Porównanie* (<, >, ≠ itd.). Każda reguła składa się z ciała *B* (części warunkowej – jeżeli) oraz głowy *H* (części wynikowej – to) reprezentowanych jako implikacja ($B \rightarrow H$). Oba elementy składają się z atomów. Wszystkie zmienne są kwantyfikowane uniwersalnie. Co więcej, można użyć dodatkowych operatorów, takich jak „*OR*", by wyrazić alternatywę (tylko w ciele reguły), co jest zgodne z profilem OWL 2 RL. Zarówno ciało, jak i głowa reguły mogą zawierać stałe i/lub zmienne, natomiast w reprezentacji faktów nie jest to dozwolone. Pozwalając na użycie dodatkowych operatorów, dodano obsługę typów wbudowanych w SWRL, które mogą być stosowane w celu porównywania wartości.

Domyślnym znaczeniem głowy każdej reguły jest dodanie (wywnioskowanie) nowej trójki (faktu). Aby reguła została użyta, wszystkie elementy warunkowe, które występują w jej ciele, muszą być prawdziwe. Na przykład reguła (1) jest zgodna z ASRF, podobnie jak przykładowe fakty (2) i (3). Fakt (3) został wywnioskowany przy zastosowaniu reguły (1) do faktu (2).

(*Triple* (*S* "*Person*10") (*P* "*hasSibling*") (*O* "*Person*20")) (2)

(*Triple* (*S* "*Person*20") (*P* "*hasSibling*") (*O* "*Person*10")) (3)

Składnia ASRF jest podobna do składni dobrze znanych języków silników regułowych, takich jak Jess czy Clips[8]. Jednakże ma ona mniejszą siłę wyrazu, ponieważ jest ograniczona do wyrażeń dostępnych w profilu OWL 2 RL. Przykładowo nie można wnioskować o niespójnościach w bazie wiedzy.

## 5. Implementacja

RuQAR stanowi implementację opracowanej metody przekształcania ontologii OWL 2 do zestawu reguł i zbioru faktów wyrażonych w składni ASRF. Narzędzie jest implementowane w Javie i pozwala wykonywać wnioskowanie ABox w dwóch silnikach: Jess oraz Drools. RuQAR to biblioteka, która może być użyta w zastosowaniach wymagających wydajnego wnioskowania w ABox. Narzędzie wykorzystuje OWL API [11] do obsługi plików ontologii oraz do wyodrębniania aksjomatów języka OWL. Stosowane wersje silników wnioskujących to: Drools w wersji 5.5 i Jess w wersji 7.1.

[8] http://clipsrules.sourceforge.net/

RuQAR zawiera interfejs *RulesInto*, którego implementacja ma odwzorowywać notację ASRF na język wybranego silnika wnioskującego. Implementacja tego interfejsu wymaga zdefiniowania odwzorowań pomiędzy ASRF a elementami języka docelowego. Obecna implementacja dostarcza zbiór mapowań dla narzędzi Jess oraz Drools. Silniki używają różnych języków, a także różnego podejścia do reprezentacji reguł i faktów. Jess używa własnego języka skryptowego i reprezentuje reguły oraz fakty jako wewnętrzne obiekty Java, natomiast reguły w Drools są reprezentowane jako stwierdzenia *when… then…*, w których jako fakt może wystąpić dowolna klasa Javy. Co więcej, dane mogą być reprezentowane jako zwykłe obiekty. W rezultacie opracowano dodatkową klasę Triple, która może być stosowana w ciele każdej reguły, jak również do reprezentowania faktów jako trójki. W ten sposób składnia ASRF może być wykorzystywana w różnych silnikach wnioskowania regułowego. Ponadto, według naszej wiedzy, jest to pierwsze tego typu użycie narzędzi Drools oraz Jess (z wyjątkiem pracy zaprezentowanej w [19], która bezpośrednio implementuje semantykę ontologii wyrażonej w profilu OWL 2 RL).

Opisywana metoda transformacji tworzy dwa oddzielne zestawy: reguł i faktów. Oznacza to, że wygenerowany zbiór reguł ABox można zastosować do faktów pochodzących ze źródeł innych niż przekształcona ontologia. Ponadto można używać wygenerowanych faktów z innymi zbiorami reguł. Jednakże w tym przypadku należy pamiętać, że wyniki wnioskowania mogą być niezgodne z zamierzeniami twórców ontologii.

## 6. Eksperymenty

Narzędzie RuQAR zostało przetestowane za pomocą ontologii pobranych ze strony KAON2[9]: Vicodi[10] – ontologia zawierająca informacje o historii Europy, Semintec[11] – ontologia opisująca dziedzinę finansową i LUBM[12] – ontologia o strukturach uczelni. W ramach testów użyto różnych zestawów danych dla każdej z ontologii (Semintec_0, Semintec_1 itd.), gdzie wyższy numer oznacza większy zestaw danych ABox.

Schemat testowania dla każdej ontologii był następujący. Najpierw przeprowadzano wnioskowanie TBox z wykorzystaniem narzędzia HermiT. Następnie sklasyfikowana ontologia została załadowana do silnika wnioskującego i uruchamiano wnioskowanie ABox. W każdym przypadku rejestrowano czas wnioskowania i uzyskaną wielkość części asercyjnej. Przeprowadzono wnioskowanie ABox za pomocą następujących silników: Jess, Drools i HermiT. Zweryfikowano, że silniki generują identyczne wyniki (podobne, empiryczne po-

[9] http://kaon2.semanticweb.org/
[10] http://www.vicodi.org
[11] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

dejście zastosowano w [5 i 19] w celu porównania silników wnioskowania dla OWL 2 RL z Pellet/RacerPro i HermiT). HermiT wygenerował więcej wyników w przypadku ontologii LUBM. Jest to poprawny wynik, ponieważ tylko Vicodi jest ontologią w pełni zgodną z profilem OWL 2 RL. Jednakże stanowi to główną przyczynę bardzo dużych różnic w czasach wnioskowania w stosunku do Jess i Drools (w przypadku LUBM). Niemniej jednak wszystkie wyniki wywnioskowane przez Jess i Drools znajdowały się wśród wyników uzyskanych przez silnik HermiT. Wynika to z faktu, że w przypadku narzędzia HermiT możemy wywnioskować dodatkowe przynależności instancji do klas, co nie jest możliwe w profilu OWL 2 RL (ontologia LUBM zawiera konstrukcje OWL 2 DL). Jednakże w każdym przypadku uzyskano lepszą wydajność we wnioskowaniu ABox z Jess oraz Drools niż z HermiT. Przykładowo dla ontologii Semintec_4.owl czasy wnioskowania w ABox dla Jess, Drools i HermiT były następujące (wyniki były identyczne): odpowiednio powyżej 3 sekund, w ciągu 5 sekund i ponad 16 sekund. Rysunki 3, 4 oraz 5 przedstawiają wyniki testów. Jess wykonuje obliczenia szybciej niż Drools, podczas gdy HermiT zawsze znajdował się na ostatnim miejscu. Uzyskane wyniki potwierdzają, że opracowana metoda zwiększa wydajność wnioskowania w ABox w porównaniu z wnioskowaniem opierającym się tylko na silniku DL.

Tabela 2

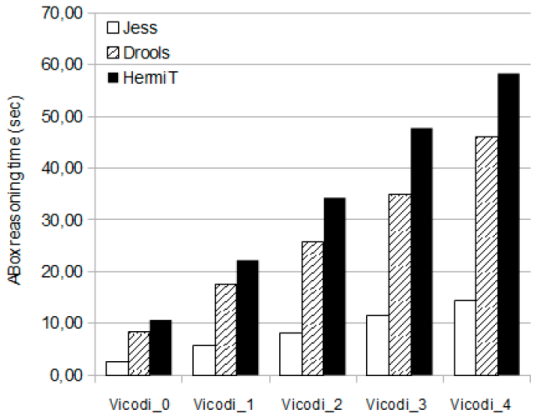
Rozmiary części Abox przed wnioskowaniem i po wnioskowaniu

| Ontologia | ABox | Jess/Drools | Liczba uruchomionych reguł | HermiT |
|---|---|---|---|---|
| **Vicodi_0** | 53 653 | 212 746 | 399 036 | 212 746 |
| **Vicodi_1** | 107 306 | 425 492 | 798 072 | 425 492 |
| **Vicodi_2** | 160 959 | 638 238 | 1 197 108 | 638 238 |
| **Vicodi_3** | 214 612 | 850 984 | 1 596 144 | 850 984 |
| **Vicodi_4** | 268 265 | 1 063 730 | 1 995 180 | 1 063 730 |
| **Semintec_0** | 65 189 | 90 781 | 132 236 | 90 781 |
| **Semintec_1** | 130 378 | 181 562 | 264 472 | 181 562 |
| **Semintec_2** | 195 567 | 272 343 | 396 708 | 272 343 |
| **Semintec_3** | 260 756 | 363 124 | 528 944 | 363 124 |
| **Semintec_4** | 325 945 | 453 905 | 661 180 | 453 905 |
| **LUBM_1** | 100 543 | 155 637 | 157 056 | 155 652 |
| **LUBM_2** | 230 061 | 354 219 | 358 769 | 354 253 |
| **LUBM_3** | 337 127 | 518 333 | 525 938 | 518 383 |
| **LUBM_4** | 477 784 | 733 846 | 745 991 | 733 917 |

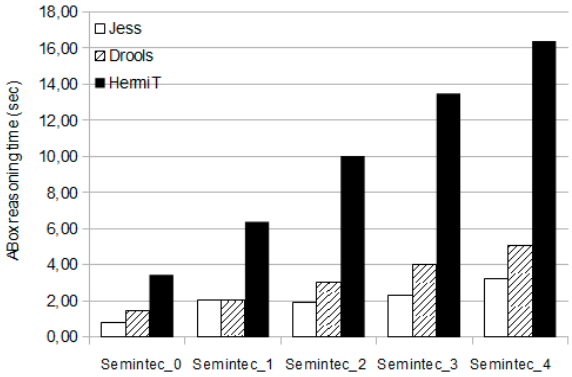
W tabeli 2 przedstawiono dane dotyczące wnioskowania w ABox. Kolumna *ABox* oznacza liczbę trójek w asercyjnej części ontologii przed wnioskowaniem. Kolumna *Jess/Drools* pokazuje rozmiar ABox po procesie wnioskowania. Kolumna *Liczba uruchomionych reguł*

[12] http://swat.cse.lehigh.edu/downloads/index.html

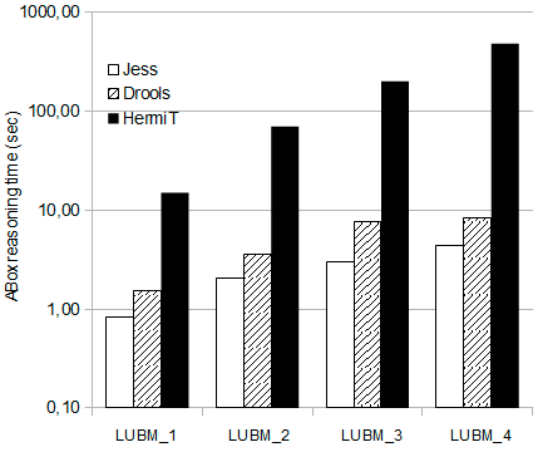
przedstawia liczbę reguł, które były uruchomione podczas wnioskowania. Liczby w tych dwóch kolumnach były identyczne w przypadku obu testowanych silników regułowych. Ostatnia kolumna zawiera informacje o rozmiarze ABox po przeprowadzeniu wnioskowania z silnikiem HermiT. We wszystkich przypadkach (z wyjątkiem LUBM) wynikowe zestawy trójek były dokładnie takie same.




Rys. 3. Czasy wnioskowania na części ABox ontologii Vicodi
Fig. 3. Reasoning times on the ABox part of Vicodi ontology




Rys. 4. Czasy wnioskowania na części Abox ontologii Semintec
Fig. 4. Reasoning times on the ABox part of Semintec ontology




Rys. 5. Czasy wnioskowania na części Abox ontologii LUBM
Fig. 5. Reasoning times on the ABox part of LUBM ontology

## 7. Podobne prace

Najbardziej zbliżone do prezentowanego rozwiązanie zastosowano w pracy DLEJena [17]. Jednakże w naszej pracy nie ograniczamy się do jednego narzędzia wnioskującego (Jena w porównaniu z Jess i Drools). Dodatkowo stosujemy inną metodę transformacji – zamiast reguł szablonowych (ang. *template rules*) do tworzenia reguł ABox wykorzystujemy mechanizm generacji zaimplementowany na sztywno w Javie. Dzięki takiemu podejściu nie generujemy zbędnych (nadmiarowych) reguł, które występują w [17]. Ponadto w narzędziu DLEJena reguły służące do wnioskowania w ABox są generowane w czasie działania wnioskowania, podczas gdy w naszej metodyce reguły ABox są generowane przed tym procesem.

Zestaw dwóch silników wnioskujących dla OWL 2 RL przedstawiono w pracy [19]. W tym przypadku narzędzia Drools oraz Jess są używane do wnioskowania z regułami bezpośrednio odwzorowującymi semantykę profilu OWL 2 RL.

Praca zaprezentowana w [10] dostarcza mechanizmów wnioskowania dla OWL 2 RL i opiera się na tzw. częściowym indeksowaniu służącym do optymalizacji skalowalnej materializacji wyników z wykorzystaniem zestawu reguł szablonowych podobnie jak w narzędziu DLEJena.

Skalowalny silnik wnioskujący dla OWL 2 RL został przedstawiony w [15], gdzie silnik wnioskujący zaimplementowano jako wewnętrzne narzędzie systemu bazy danych Oracle. Praca ta wprowadza nowe techniki przetwarzania równoległego oraz specjalne optymalizacje obliczania relacji *owl:sameAs*.

W [5] przedstawiono metodę magazynowania asercyjnej i wywnioskowanej części bazy wiedzy w relacyjnej bazie danych. Zaproponowano również nową, opartą na bazach danych metodę wnioskowania w przód, która pozwala wykonywać skalowalne wnioskowanie w ontologii OWL 2 RL z dużymi zbiorami danych.

## 8. Podsumowanie i dalsze prace

W niniejszym artykule przedstawiono metodę przekształcania ontologii OWL 2 w zestaw reguł i zestaw faktów. Zaproponowano Abstrakcyjną Składnię Reguł i Faktów, ponadto opisano schemat wnioskowania, wstępną implementację oraz przeprowadzone eksperymenty.

Zaprezentowano pierwszą wersję narzędzia RuQAR, które jest wciąż rozwijane. Aktualna wersja, przy wykorzystaniu narzędzi Jess oraz Drools, jest w stanie przeprowadzić wnioskowanie ABox ze znacznie lepszą wydajnością niż w przypadku zastosowania narzędzia HermiT. W następnej wersji narzędzia RuQAR planujemy dodać obsługę relacyjnych baz danych, a także zoptymalizowany system zapytań (obecnie możemy skorzystać z nieefek-

tywnych metod zapytań dostępnych w Jess i Drools). Dodatkowo planujemy optymalizację procesu wnioskowania przez zastosowanie i rozszerzenie metod opisanych w [5 i 2]. Ze względu na składnię ASRF zaproponowane optymalizacje będą możliwe do wykorzystania zarówno w Jess, jak i w Drools oraz w innych silnikach wnioskujących w przód.

Według naszej wiedzy prezentowana praca jest pierwszą realizacją wnioskowania dla OWL 2 RL za pomocą Drools i Jess (z wyjątkiem [19]), które może być z powodzeniem zastosowane w dowolnej aplikacji wymagającej wydajnego wnioskowania z dużymi zbiorami danych. Planujemy również przeprowadzić testy z najnowszymi wersjami narzędzi Jess i Drools (odpowiednio 8,0 i 6,0). Naszym celem będzie sprawdzenie, czy wydajność wnioskowania wzrosła wraz z nową wersją; dzięki temu będziemy mogli porównać dwa różne algorytmy zaimplementowane w silniku Drools: ReteOO (Drools 5.5) i PHREAK (Drools 6.0).

## BIBLIOGRAFIA

1. Baader F., Brandt S., Lutz C.: Pushing the el envelope further. [w:] Clark K., Patel-Schneider P. F. (red.): Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions, 2008.
2. Bak J., Brzykcy G., Jedrzejek C.: Extended rules in knowledge-based data access. Proceedings of the 5th international conference on Rule-based modeling and computing on the semantic web, RuleML'11, Springer-Verlag, Berlin, Heidelberg 2011, s. 112÷127.
3. Boris M.: On the properties of metamodeling in OWL. Journal of Logic and Computation, Vol. 17, Issue 4, 2007, s. 617.
4. Calvanese D., Giacomo G., Lembo D., Lenzerini M., Rosati R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. J. Autom. Reason., Vol. 39, Issue 3, 2007, s. 385÷429.
5. Faruqui R., MacCaull W.: Owlontdb: A scalable reasoning system for OWL 2 RL ontologies with large aboxes. [w:] Weber J., Perseil I. (red.): Foundations of Health Information Engineering and Systems. Lecture Notes in Computer Science, Vol. 7789, Springer, Berlin, Heidelberg 2013, s. 105÷123.
6. Grau B. C., Horrocks I., Motik B., Parsia B., Patel-Schneider P., Sattler U.: OWL 2: The next step for OWL. Web Semant. 2008, Vol. 6, Issue 4, 2008, s. 309÷322.
7. Grosof B. N., Horrocks I., Volz R., Decker S.: Description logic programs: combining logic programs with description logic. Proceedings of the 12th international conference on World Wide Web, WWW '03, ACM, New York, NY, USA 2003, s. 48÷57.

8. Hill E. F.: Jess in Action: Java Rule-Based Systems. Manning Publications Co. Greenwich, CT, USA 2003.
9. Hogan A., Decker S.: On the ostensibly silent w in OWL 2 RL. [w:] Polleres A., Swift T. (red.): Web Reasoning and Rule Systems, Lecture Notes in Computer Science, Vol. 5837, Springer, Berlin, Heidelberg 2009, s. 118÷134.
10. Hogan A., Pan J., Polleres A., Decker S.: SAOR: Template rule optimisations for distributed reasoning over 1 billion linked data triples. [w:] Patel-Schneider P., Pan Y., Hitzler P., Mika P., Zhang L., Pan J., Horrocks I., Glimm B. (red.): The Semantic Web ISWC 2010, Lecture Notes in Computer Science, Vol. 6496, Springer, Berlin, Heidelberg 2010, s. 337÷353.
11. Horridge M., Bechhofer S.: The OWL API: A Java API for working with OWL 2 ontologies. OWLED, 2009.
12. Horrocks I., Patel-Schneider P. F.: Knowledge representation and reasoning on the semantic Web: OWL, 2010.
13. Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B., Dean M.: SWRL: A semantic web rule language combining OWL and RuleML, 2004.
14. Hustadt U., Motik B., Sattler U.: Data complexity of reasoning in very expressive description logics. Proc. IJCAI 2005, Professional Book Center, 2005, s. 466÷471.
15. Kolovski V., Wu Z., Eadon G.: Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system. [w:] Patel-Schneider P., Pan Y., Hitzler P., Mika P., Zhang L., Pan J., Horrocks I., Glimm B. (red.): The Semantic Web ISWC 2010, Lecture Notes in Computer Science, Vol. 6496, Springer, Berlin, Heidelberg 2010, s. 436÷452.
16. Meditskos G., Bassiliades N.: Combining a DL reasoner and a rule engine for improving entailment-based OWL reasoning. Proceedings of the 7th International Conference on The Semantic Web, ISWC ’08, Springer-Verlag, Berlin, Heidelberg 2008, s. 277÷292.
17. Meditskos G., Bassiliades N.: Dlejena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet. J. Web Sem., Vol. 8, Issue 1, 2010, s. 89÷94.
18. Motik B., Grau B. C., Horrocks I., Wu Z., Fokoue A., Lutz C.: OWL 2 Web ontology language profiles (second edition). W3C Recommendation, 2012.
19. O’Connor M. J., Das A.: A pair of OWL 2 RL reasoners. [w:] Klinov P., Horridge M. (red.): OWLED, CEUR Workshop Proceedings, Vol. 849, CEUR-WS.org, 2012.
20. Patel-Schneider P. F., Horrocks I.: OWL 1.1 Web ontology language, http://www.w3.org/Submission/owl11-overview/, 2006.
21. Pattis R. E.: EBNF: A Notation to Describe Syntax, 1980.

Wpłynęło do Redakcji 7 lutego 2014 r.

**Abstract**

In this paper we present a method of transforming OWL 2 RL ontologies into a set of rules which can be used in any forward chaining rule engine. By applying a reasoning engine we can perform reasoning tasks with two parts of ontology as a knowledge base KB: the TBox and the ABox. The transformation schema of OWL 2 ontology into a set of rules and a set of facts expressed in Abstract Syntax of Rules and Facts is presented in Figure 1. Firstly, OWL 2 ontology is loaded into the HermiT engine to perform the TBox reasoning and to produce inferred ontology. We assume that the ontology is consistent. As a result of TBox reasoning we obtain a new classified version of the ontology (new TBox). After that the ontology is transformed into two sets: a set of rules and a set of facts (if it contains the assertional part). In that way we separate the TBox part (set of rules) from the ABox part (set of facts). The inferred ontology is automatically transformed into its ASRF syntax counterpart. This context-free grammar is presented in Figure 2. Then, ontology can be transformed into any forward chaining reasoning engine language in order to conduct ABox reasoning. We present an implementation of our method using two engines: Jess and Drools. We evaluate our approach by performing the ABox reasoning on the number of benchmark ontologies originating from KAON2 project. Additionally, we compare obtained results with inferences provided by the HermiT reasoner. The evaluation presented in Table 2 shows that we can perform the ABox reasoning with considerably better performance than HermiT.

**Adresy**

Jarosław BĄK: Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, Polska, jaroslaw.bak@put.poznan.pl.
Maciej NOWAK: Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, Polska, maciej.wojciech.nowak@gmail.com.
Czesław JĘDRZEJEK: Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, Polska, czeslaw.jedrzejek@put.poznan.pl.