Reasoning with Data Flows and Policy Propagation Rules

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Enrico Daga [a], Aldo Gangemi [b], Enrico Motta [a]
[a] *Knowledge Media Institute, The Open University*
Walton Hall, Milton Keynes, United Kingdom
E-mail: {enrico.daga,enrico.motta}@open.ac.uk
[b] *Université Paris13, France and Institute of Cognitive Sciences and Technologies - CNR, Italy*
E-mail: aldo.gangemi@{univ-paris13.fr,cnr.it}

Abstract.

Data oriented systems and applications are at the centre of current developments of the World Wide Web. In these scenarios, assessing what policies propagate from the licenses of data sources to the output of a given data-intensive system is an important problem. Both policies and data flows can be described with Semantic Web languages. Although it is possible to define Policy Propagation Rules (PPR) by associating policies to data flow steps, this activity results in a huge number of rules to be stored and managed. In a recent paper, we described how it is possible to reduce the size of a PPRs database by using an ontology of the possible relations between data objects, the Datanode ontology, and applying the (A)AAAA methodology, a knowledge engineering approach that exploits Formal Concept Analysis (FCA). In this article we check whether this reasoning is feasible in realistic scenarios. To this purpose, we study the impact of compressing a rule base associated with an inference mechanism on the performance of the reasoning process. Moreover, we report on an extension of the (A)AAAA methodology that includes a *coherency check* algorithm, that makes this reasoning possible. We show how this compression, in addition to being beneficial to the management of the rule base, also has a positive impact on the performance and resource requirements of the reasoning process for policy propagation.

Keywords: Data Hub, Data Flows, Policies, Rules, Formal Concept Analysis, RDF Licenses

1. Introduction

Data oriented systems and applications are at the centre of current developments of the World Wide Web. Developers can access a large variety of (open) data, and publish the result of their processing on the Web. Emerging enterprises focus their business model on providing value from data collection, integration, processing and redistribution. For example, Data Hubs collect a large variety of data sources and process them in order to satisfy the needs of users through remote applications [9]. Differently from a closed enterprise environment, on the WWW the ownership and licensing of the data do not belong to the owner of the end user application, and sometimes even to the entity responsible of the data management and processing infrastructure.

In this complex scenario, assessing what policies propagate from the data sources to the output of a given data-intensive process is an important problem. Altough it is possible to define Policy Propagation Rules (PPR) by associating policies and data flow steps, this activity results in a huge number of rules to be stored and managed [8]. However, it is possible to compress a PPRs database by using an ontology of the possible relations between data objects, the Datanode ontology, and applying the (A)AAAA methodology, a knowledge engineering approach that exploits Formal Concept Analysis (FCA) [8]. In this article we study how this reasoning can be practically performed. We report on an extension of the (A)AAAA methodology that includes a *coherency check* between the hierarchy of the FCA lattice and the Datanode ontology, which allows to perform reasoning with a compressed rule base. However, while this activity would reduce the size of the input of the reasoner, it requires to compute more inferences. Therefore, we study the impact of rule base compression on the performance of a PPR reasoner.

The article is structured as follows. Section 2 traverses the relevant literature. Section 3 presents an exemplary use case, and introduces the ingredients for reasoning on policies propagation, going through the description of the data flow, the representation of policies, and the notion of Policy Propagation Rule (PPR) [8]. Section 4 provides a comprehensive overview of the (A)AAAA methodology, integrated with a novel *Analysis* phase that includes a *coherency check* algorithm that allows effective reasoning with a compressed rule base. We also evaluate the impact of this evolved methodology on the compression factor of the rule base. In Section 5 we report on experimental results on the performance of reasoning on PPRs with a compressed rule base, before closing the article with conclusions and persectives on future work.

2. Related Work

In the Web of (open) data, developers can access a large variety of information, and often publish the results of their processing. In this scenario, application developers need to know what are the usage constraints attached to a data source. Policies are therefore necessary in order to enable a correct exploitation of the resources.

Data repositories and registries are growing, spanning from data cataloguing services (Datahub[1]), data collections (Wikidata[2], Europeana[3]), to platforms that manage the collection and redistribution of data (Socrata[4]). Providers of Smart City Data Hubs need to support the developers on the assessment of the policies associated with data resulting from complex pipelines [9,2]. Emerging enterprises focus their business model on providing value from data collection, integration, processing and redistribution (Treasure Data[5]).

It is important to develop technologies that allow policies to be negotiated [3], and research is ongoing on how the association between data and policies can be shared and enforced on the web [21]. In this paper we concentrate on the problem of reasoning with propagating policies.

Policies can be represented on the Web. The W3C ODRL Community Group work on the development of a set of specifications to enable interoperability and transparent communication of policies associated with software, services and data. The Open Digital Rights Language (ODRL)[6] is an emerging information model to support the exchange of formal descriptions of policies [15].

The RDF Licenses Database [22] is an attempt to establish a database of licence descriptions based on RDF and the ontology provided by ODRL (among others).

Process executions can be described in the Semantic Web using the Provenance Ontology (PROV-O) [19]. PROV-O describes workflow executions in terms of *agents*, *actions* and *assets* involved. The Datanode ontology has been designed to describe Semantic Web applications by means of the relations between the data involved in their processes [7]. The ontology is a taxonomy of possible relations that may occur between data objects, which might be part of a process execution, such as one described with PROV-O. It can therefore be used to further qualify the implications of the actions performed in such a process. Datanode can describe process implications in a data-oriented way: as network of data objects. While policies and process executions can be represented, what we aim to do here is to study the reasoning on the propagation of policies across a data flow.

Rule based representation and reasoning on policies is required in order to enable secure data access and usage in distributed environments, particularly the Semantic Web [10,18,4]. Defeasable logic is necessary to

[1]Datahub. `https://datahub.io/`
[2]Wikidata. `https://www.wikidata.org`
[3]Europeana. `http://labs.europeana.eu/`
[4]Socrata. `https://www.socrata.com/`
[5]Treasure Data. `https://www.treasuredata.com/`
[6]`https://www.w3.org/community/odrl/`

reason with deontic statements, for example to check compatibility of licenses or to validate constraints attached to compontents on multi-agent systems [23]. This problem has been extensively studied in the literature [12,13] and tools that can perform such assessment do exist [17]. Our previous work introduces a form of policy reasoning, namely *policy propagation* [8]. A Policies Propagation Rule (PPR) is a Horn clause defined by associating a Datanode relation with a ODRL policy. Reasoning on Horn rules is one way of dealing with policies, particularly because they allow a tractable defeasable reasoning [1].

Formal Concept Analysis (FCA) [26] has the capability of classifying collections of objects depending on their *features*. We apply FCA to detect a common behavior of relations in terms of policy propagation in order to compress the rule base, in conjunction with the Datanode ontology. We refer the reader to [6] for a description of the Contento tool, that implements FCA as well as other functionalities for evolving concept lattices in Semantic Web ontologies, also part of the approach we present here.

The approach described in this paper clearly relates to principles and methods of knowledge engineering [25]. In [20], knowledge acquisition is considered as an iterative process of model refinement, and this is exactly how we decided to tackle our problem here. More recently, problem solving methods have been studied in relation to the task of understanding process executions [11].

The problem of compression of propositional knowledge bases has been deeply studied, and focused on the optimization of a Horn minimization process to boost rule execution [14,5]. In this work, we deal with compression as a mean to reduce the number of minimal rules to be managed (each PPR being already an atomic rule), by the means of an additional knowledge base (the Dataode ontology).

It is worth noting that our problem is not one of policy enforcement but one of providing the right information about policies that might affect the terms of use of a given asset resulting from a complex computation in an application. Moreover, in this article we do not focus on the quality of the descriptions, and assume a machine readable description of the policies of the input asset as well as the existence of an accurate data flow.

3. Reasoning on policies propagation

In this Section we describe the approach for reasoning on policy propagation, and we present a sample use case.

We define the problem of policies propagation as the one of identifying the set of policies associated with the output of a process, implied by the policies associated with the input data source. In order to perform reasoning on policy propagation, we need:

a) description of policies attached to data sources;
b) knowledge about the actions performed on the data, and
c) knowledge about what actions do propagate a given policy.

Description of policies. We assume the policies of data sources are described as licenses or "terms and conditions" documents, and they have an RDF representation by the means of the ODRL ontology. A policy expressed with the ODRL model includes a deontic aspect - `odrl:duty`, `odrl:permission` or `odrl:prohibition`, associated to a set of `odrl:Actions`. Set of policies can be associated with *assets*. For example, the RDF Licenses Database [22] is a source of such descriptions. In our work, we also developed ad-hoc RDF documents to satisfy this requirement, when necessary.

Description of the data flow. Data flows are represented with the Datanode ontology [7]. The ontology defines a unique type - *Datanode* - and 114 relations, starting from a single top property: relatedWith, having *Datanode* as `rdfs:domain` and `rdfs:range`.

A *datanode* is any data object that can be the input or output of a process. The class groups datasets, individuals, schema elements such as classes and properties, as well as identifiers under the same umbrella. Datanode relations can express meta-level aspects (*describes/describedBy*, *hasAnnotation/isAnnotationOf* and *hasStatistic/isStatisticOf*), containmnet (*hasPart/isPartOf*; *hasSection/isSectionOf*) as well as a properties like derivation (*hasExtraction/isExtractionOf*; *hasInference/isInferenceOf*; *cleanedFrom/cleanedInto*; *optmizedFrom/optimizedInto*), among others.

In this work we use the representatons of data flows extracted from the descriptions of several Semantic Web applications prepared in [7].

Policy Propagation Rules (definition). A Policy Propagation Rule (PPR) establishes a binding between a Datanode relation and a policy: when a policy holds for a data object, and this is linked to another with that relation, then the policy will also hold for the second one. A PPR is a Horn Clause of the following form:

$$has(X, P) \land propagates(P, R) \land relation(R, X, Y) \rightarrow has(Y, P)$$

Where X and Y are data objects, P is a policy and R a Datanode relation between the two. For example a PPR could be used to represent the fact that downloading a file F distributed with an attribution requirement will result in a local copy D, which also needs to be used according to the attribution requirement. Therefore, the above abstract rule could be instantiated as follows:

$$has(F, attribution) \land propagates(attribution, isCopyOf) \land relation(isCopyOf, F, D) \rightarrow has(D, attribution)$$

In fact, we can reduce a PPR to a more compact form, i.e. a binary association between a policy and a relation:

$$propagates(policy, relation)$$

as the other parts can be generated automatically from the above representation.

EventMedia [16] is a a web-based system that exploits real-time connections to enrich content describing events and associate it with media objects[7]. The application reuses web data exposed by third parties in order to collect event data and multimedia objects. The information is then presented using the LODE ontology [24]. Figure 1 displays how data in EventMedia is processed from event directories and is enriched with data from sources like the BBC or Foursquare [16] according to the Datanode description elaborated in [7].

Table 1 lists the licenses or terms of use documents associated with the data sources. In the present case we were able to collect information from five out of six sources, being the *Upcoming* service not available at the time of writing[12]. Listing 1 lists the set of policies extracted from the Flickr APIs Terms of Use.

[7]See `http://eventmedia.eurecom.fr/`.

[12]The Technical Report has been produced in 2014, when the EventMedia dataset description article was firstly submitted to the Semantic Web Journal. The description produced refers to the submitted version, and could be partly changed in the published version.

Table 1

Sources of Terms and conditions associated with the data sources exploited by EventMedia.

Source	T&C
Flickr	Flickr APIs Terms of Use[8]
Dbpedia	Creative Commons CC-BY-SA 3.0
Eventful	Eventful API Terms of Use[9]
LastFM	LastFM Terms of Service[10]
Upcoming	?
Musicbrain	Creative Commons CC0
Foursquare	Foursqaure Developers Policies[11]

Listing 1: Sample of policies representation extracted from the Flickr APIs Terms of Use.

```
:FlickrTC a odrl:Policy ;
 odrl:asset :Flickr;
 a odrl:Agreement ;
 rdfs:label "Flickr APIs Terms of Use" ;
 rdfs:seeAlso
  <https://www.flickr.com/services/api/tos/>;
 odrl:prohibition odrl:sell;
 odrl:prohibition odrl:sublicense;
 odrl:prohibition cc:CommercialUse;
 odrl:duty odrl:attribute
  .
```

A PPR reasoner receives as input the dataflow and the policies as RDF representations compliant with Datanode and ODRL, and produces as output an ODRL set like the one in Listing 2.

Listing 2: Example of policies associated with the output of EventMedia.

```
:outputPset a odrl:Set ;
  odrl:asset :output ;
  odrl:prohibition odrl:modify;
  odrl:prohibition cc:ommercialUse;
  odrl:duty odrl:attribute;
  odrl:prohibition odrl:sell
  .
```

Having a description of policies and data flow steps implies a huge number of propagation rules to be managed and computed (number of policies times number of actions). In this article we want to study to what extent it is possible to reduce the number of rules without loss of information. Our hypothesis is that compressing the size of the rule base by enabling some sort of inference mechanism would not negatively impact the efficiency of the computation.

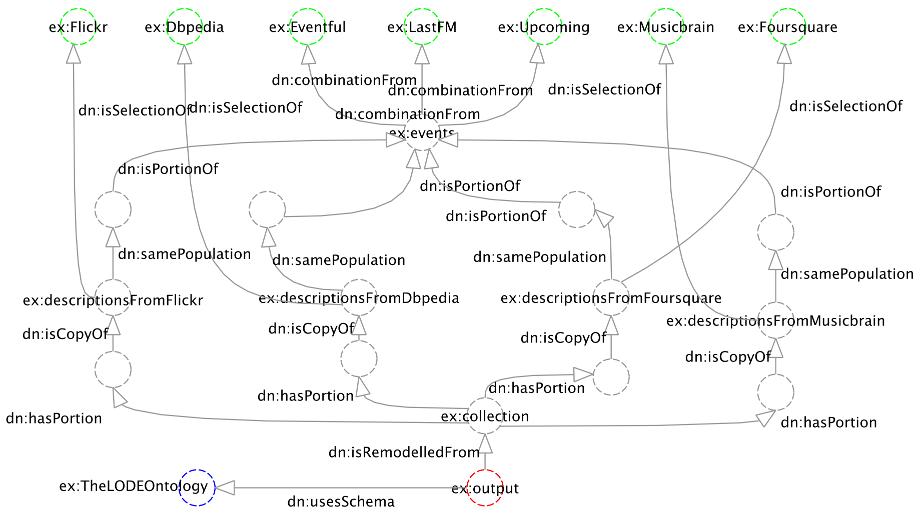


Fig. 1. Description of the reuse of sources in the EventMedia mash-up [16] (Figure from [7]). The input sources are the top nodes. The node at the bottom depicts the output data, which is a remodelling of the data collected from various sources according to a specific schema.

4. (A)AAAA Methodology

The approach for compressing a knowledge base of policy propagation rules relies on the Datanode ontology, that organizes the possible dataflow steps in a hierarchy. Firstly introduced in [8], the (A)AAAA methodology covers all the phases necessary to set up a compact knowledge base of PPRs.The (A)AAAA methodology is based on two assumptions: 1) Policy Propagation Rules are associations between policies and data flow steps, and 2) an ontology is available to organise data flow steps in a semantic hierarchy - the Datanode ontology. For example, this ontology would tell us that the relation *isCopyOf* is a kind of *isDerivationOf*. We provide here a journey through the methodology, alongside concrete examples on how it has been applied. The resulting compressed rule base will be the basis for our experiments in the remaining part of the article. With respect to the methodology presented in [8], the novel contribution of this article is the introduction of a *coherency check* method in the *Assessment* phase.

The methodology is composed of the following phases:

A1 **Acquisition.** The initial task is to set up a knowledge base of PPRs.

A2 **Analysis.** The FCA algorithm is performed on the knowledge base of PPRs. The output of the process is an ordered *lattice* of concepts: policies that propagate with the same set of relations.

A3 **Abstraction.** In this phase we search for matches between the ontology and the FCA lattice. When a match occurs, we subtract the rules that can be abstracted through the ontology's taxonomy.

A4 **Assessment.** We check to what extent a hierarchical organization of the relations matches the clusters produced by FCA (developing measures). This step

   a) performs a *coherency check* between the lattice and the ontology (i.e. number of mismatches);
   b) identifies what are the partial matches between the clusters and the ontology, and
   c) evaluates how much the ontology compresses the knowledge base (i.e. the compression factor)

A5 **Adjustment.** Observing the measures produced in the previous phase, particularly about mismatches and partial matches, in this phase we perform operations changing the rule base or the ontology in order to repair mismatches, correct inaccuracies, evolve the ontology, and improve the compression factor as a consequence.

The (A)AAAA methodology constitutes an iterative process, as shown in Figure 2.

In the remaining part of this Section, we illustrate each phase of the methodology, by first describing the general approach and then explaining how it has been applied in practice.

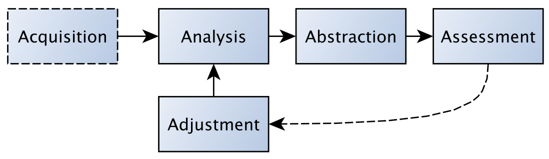


Fig. 2. (A)AAAA Methodology.

4.1. Acquisition

The initial task is to set up a knowledge base of Policy Propagation Rules (PPRs). As presented in Section 3, a PPR can be conceived as an association between a policy and a possible relation between two data objects. The rule base can be represented as a binary matrix, where each row is a possible relation between two data objects, and each column a policy. The cells in the matrix can be 1 or 0, depending on whether the policy propagates or not with the given relation. From each positive cell in the matrix, we can generate a PPR, and populate the set of rules R.

This approach has been applied as follows. In order to setup the knowledge base of Policy Propagation Rules we relied on the RDF Licenses Database [22], and extracted 113 possible policies. Each policy is an association of one deontic element (permission, prohibition or duty) and one action.[13]. We used the Datanode Ontology [7] to extract a list of 115 possible relations between data objects.

This phase required a manual supervision of all associations between policies and relations in order to establish the initial set of propagation rules[14]. Listing 3 displays a sample of binary relations that can be true or false in the matrix.

Listing 3: Example of cells of the binary matrix associating relations with policies

```
dn:hasPortion ,permission  odrl:copy ,1
dn:identifiersOf ,
            prohibition  cc:DerivativeWorks ,1
dn:isDependencyOf ,permission  odrl:derive ,0
dn:usesSchema ,permission  cc:Reproduction ,1
dn:processedInto ,duty  odrl:shareAlike ,1
dn:isVocabularyOf ,prohibition  odrl:use ,1
dn:metadata ,prohibition  odrl:transform ,0
```

At the end of this process, the matrix had 3363 cells marked as true. The initial knowledge base was then composed of 3363 Policy Propagation Rules (Listing 4)[15].

Listing 4: Example of Policy Propagation Rules.

```
prop(dn:hasPortion ,permission  odrl:copy)
prop(dn:identifiersOf ,prohibition
                          cc:DerivativeWorks)
prop(dn:usesSchema ,permission  cc:Reproduction)
prop(dn:processedInto ,duty  odrl:shareAlike)
prop(dn:isVocabularyOf ,prohibition  odrl:use)
prop(dn:metadata ,prohibition  odrl:transform)
```

4.2. Analysis

The objective of the second phase is to detect common behaviors of relations with respect to policies propagation. We achieve this by applying FCA, providing as input the binary matrix representation of the rule base R. The output of the FCA algorithm is an ordered set of *concepts* C. In FCA terms, each concept groups a set of objects (the concept's *extent*) and maps it to a set of attributes (the concept's *intent*). An FCA Concept groups a set of objects all having a given set of attributes (and vice-versa). In our case, each concept maps a group of relations propagating a group of policies. These concepts are organized hierarchically in a *lattice*, ordered from the top concept T, that includes all the objects and potentially no attributes, to the bottom concept B, including all the attributes with potentially an empty extent (set of objects). All other concepts are ordered from the top to the bottom. For example, usually a first layer of concepts right below T would include large groups of objects all having few attributes in common. Layers below would have more attributes and less objects, until the bottom B is reached. In our case, the top concept T would include all relations and no policies, while the bottom concept B all the policies but no relations. The concepts identified by FCA collect relations that have a common be-

[13]We could have generated the policies by combining any ODRL action with any deontic component. However, this would have led to a large number of meaningless policies (eg: `duty odrl:use`). The adoption of the RDF License Database permitted to obtain a list of meaningful policies only.

[14]Thanks to the Contento tool, it was possible to edit manually the matrix with a reasonable effort [6,8].

[15]The reader can deduce that a large part of Datanode included relations that do not propagate any policy, for example the top relation `dn:relatedWith`, but also `dn:overlappingCapabilityWith`, `dn:about`, among others.

havior in our rule base R, as they propagate the same policies.

Following this approach, we applied the FCA algorithm and obtained 80 concepts (in the first iteration of the methodology). Listing 5 shows one example. All the relations in the extent of this concept propagate the policies in the intent.

Listing 5: Example of a Concept.

```
Concept 71
Ancestors: 75
Descendants: 17,52,68,69
Extent [42]                     Intent [9]
dn:cleanedInto                  duty cc:Attribution
dn:combinedIn                   duty cc:Copyleft
dn:duplicate                    duty cc:Notice
dn:hasAddition                  duty cc:ShareAlike
dn:hasAnonymized                duty cc:SourceCode
dn:hasAttributes                duty odrl:attachPolicy
dn:hasCache                     duty odrl:attachSource
dn:hasChange                    duty odrl:attribute
dn:hasComputation               duty odrl:shareAlike
dn:hasCopy
dn:hasDeletion
dn:hasDerivation
dn:hasExample
dn:hasExtraction
dn:hasIdentifiers
dn:hasInference
dn:hasInterpretation
dn:hasPart
dn:hasPortion
dn:hasReification
dn:hasSample
dn:hasSection
dn:hasSelection
dn:hasSnapshot
dn:hasStandIn
dn:hasStatistic
dn:hasSummarization
dn:hasTypes
dn:hasVocabulary
dn:identifiersOf
dn:isChangeOf
dn:isExampleOf
dn:isPartOf
dn:isPortionOf
dn:isSampleOf
dn:isSectionOf
dn:isSelectionOf
dn:isVocabularyOf
dn:optimizedInto
dn:processedInto
dn:refactoredInto
dn:remodelledTo
```

4.3. Abstraction

In this phase we apply a method for subtracting rules in order to reduce the size of the rule base. The abstraction process is based on applying an ontology that organizes the relations in a hierarchy. For instance, the relation $hasCopy$ is a sub-relation of $hasDerivation$. Intuitively, a number of policies propagated by $hasDerivation$ should be also propagated by $hasCopy$ and all the others sub-relations in that branch of the hierarchy. By grouping all the relations below $hasDerivation$ in a transitive closure, we obtain a group of relations similar to the ones in the FCA concepts that we call the $hasDerivation$ *branch*, for example. We expect the branches of the ontology to be reflected in the clusters of relations obtained by FCA, thus we search for matches between the ontology and the FCA lattice. When a match occurs, we subtract the rules that can be abstracted.

Listing 6: Abstraction algorithm.

```
R = Rules()
C = FCAConcepts()
H = ComputeBranches()
ForEach (c,h) in (C,H)
        (pre,rec) = Match(c,h)
        when  pre == 1.0
            Subtract(R, Policies(c), Branch(h))
```

The process is summarized in Listing 6, and described as follows:

1. **Concepts.** From the result of the FCA algorithm we obtain a set of concepts C including relations r (extent of the concept) all propagating a set of common policies p (intent of the concept):

$$C = ([r_1, p_1], [r_2, p_2], \ldots, [r_n, p_n])$$

2. **Branches.** For each relation in the ontology, we compute the transitive closure of its sub-relations, obtaining a set of mappings H:

$$H = ([t_1, b_1], [t_2, b_2], \ldots, [t_n, b_n])$$

where t is a relation in the ontology and b the related branch, i.e. all the sub-relations of t.
3. **Matching.** We search for (partial) overlaps between branch h in H and relations in the extent of c (for simplicity, *in* c). The function $Match$ returns a measure of the matching between a branch and a concept, evaluated as *precision* and *recall*. In the case a branch is fully in the concept, we say

the match has full precision (1.0). Inversely, recall indicates how much a branch covers the concept. A concept including only relations in the branch would result in a match with recall 1.0.

4. **Compression.** When a match has precision 1.0, we can use the relation h as abstraction for all the policies in the concept c. In other words, we can remove the rules referring to a subsumed relation, as they are implied by a more abstract one.

A general estimation of the effectiveness of the approach is given by the *compression factor* (CF). We calculate the CF as the number of abstracted rules divided by the total number of rules:

$$CF = \frac{|A|}{|R|}$$

with R the set of rules, and A the set of rules that can be subtracted.

This approach has been applied as follows. Listing 7 shows the output of the abstraction algorithm, returning the branches that are (partially) matched by a given concept.

For example, the extent of Concept 74 is a cluster of 43 relations, matching several branches in Datanode in different ways. At the beginning there is the Top Property of Datanode: `dn:relatedWith`. Its branch size (bs) is 115 relations, constituting the whole hierarchy in the ontology. Clearly, the size of the intersection is the same as the size of the extent (es), as all the 43 relations of the concept are in the branch. However, the precision (pre) of the matching is pretty low: 0.37. This branch obviously matches the whole extent of Concept 74 with full recall (rec) 1.0 - like with any other concept.

A more interesting case is the branch associated with `dn:hasPart`, whose intersection with the concept is made of all 7 sub-relations. In fact, looking at the intent of Concept 74 in Listing 5, it sounds reasonable that all the parts of a given data object inherit all the cited duties[16]. The full precision enables the rules reduction process. A similar case is with the relation `dn:isVocabularyOf`.

Listing 7: Example of the matches between a concept and the branches in the Datanode hierarchy.

```
c    es   is   bs   pre   rec   f1    branch
74   43   43   115  0.37  1     0.54  dn:relatedWith
[...]
74   43   7    8    0.87  0.16  0.27  dn:sameCapabilityAs
74   43   7    7    1     0.16  0.28  dn:hasPart
74   43   6    7    0.86  0.14  0.24  dn:isPartOf
74   43   3    6    0.5   0.07  0.12  dn:hasVocabulary
74   43   6    6    1     0.14  0.25  dn:isVocabularyOf
[...]
```

[16]They inherit many other policies as well, and those are considered by other concepts in a lower layer of the FCA lattice.

By only considering the branches matched with full precision by each concept, we can substract 1925 rules, for a compression factor CF of 0.572, in the first iteration of our methodology.

It is worth to note that this process only aims to identify matches between the rule base and the subsumption hierarchy of the ontology, and does not account for mismatches. In principle, it is possible that some policies propagated by a relation are not propagated by one of its subrelations. In this situation, a reasoner that want to benefit from a compressed rule base will return also wrong results. We deal with this problem in the *Assessment* phase.

4.4. Assessment

Objective of this phase is to assess to what extent the ontology and the FCA lattice are coherent. In particular, we want:

1. to detect mismatches (*coherency check*) to be resolved before using the compressed rule base with a reasoner, and
2. to identify *quasi matches* that could be boosted to become a full match performing changes in the rule base or the ontology

Coherency check. The abstraction process is based on the assumption that it is possible to replace asserted rules with inferences implied by subsumed relations in the ontology. This implies that all policies propagated by a given relation must be propagated by all subrelations in the original (uncompressed) rule base. A coherency check process is necessary to identify whether this assumption does hold for all the relations. In case it does not, we want to collect and report all these mismatches in order to be able to fix them at a later stage in the methodology. Listing 8 shows the algorithm used to detect such problems on a given concept in the lattice.

Listing 8: Coherency check algorithm.

```
c = // a concept
M = []
S=SuperConcepts(c)
ForEach s in S
```

```
    E=Extent(c)
    E1=Extent(s)
    ForEach(e1 in E1)
       if not(Contains(E,e1))
          ForEach e in E
             if Contains(Branch(e),e1)
               M = [e,e1]|P // Mismatch detected
return M
```

We know from the definition of a FCA lattice that superconcepts will include a larger set of relations propagating a smaller number of policies. Given a concept c, the algorithm extracts the relations (extent) of each of any superconcept (S). In case these relations are not also present in (the extent of) c, it is mandatory that they are not subrelations of any relation in the extent of c. In case they are, this means that a subrelation is not inheriting all the policies of the parent one, thus invalidating our assumption. Mismatches M are identified and reported. Listing 9 shows the result of the algorithm for a concept. In this example, a number of subrelations of `dn:isVocabularyOf` do not propagate some of the policies of Concept 71.

Listing 9: Coherency check result for concept 71: mismatches.

```
Concept  Branch             Relation
71       dn:isVocabularyOf  dn:attributesOf
71       dn:isVocabularyOf  dn:datatypesOf
71       dn:isVocabularyOf  dn:descriptorsOf
71       dn:hasVocabulary   dn:hasDatatypes
71       dn:hasVocabulary   dn:hasDescriptors
71       dn:hasVocabulary   dn:hasRelations
71       dn:isChangeOf      dn:isAdditionOf
71       dn:isChangeOf      dn:isDeletionOf
71       dn:isVocabularyOf  dn:relationsOf
71       dn:isVocabularyOf  dn:typesOf
```

Quasi matches. The result of the *Abstraction* phase can be represented as a set of measures between concepts and portions of the ontology. The measures we are interested in are:

– Extent size (ES). Number of relations in a concept.
– Intersection size (IS). Number of relations of the branch that are also present in the concept.
– Branch size (BS). Number of relations in the branch.
– Precision (Pre). It is calculated as $Pre = IS/BS$, meaning the degree of matching of a branch with the extent of the concept.
– Recall (Rec). Recall, calculated as $Rec = IS/ES$, i.e. how much the extent of the concept is covered by the branch.
– F-Measure (F1). F-Measure, the well-known fitness score, calculated from precision and recall as $F1 = 2 * ((Pre * Rec)/(Pre + Rec))$.

These measures are now considered to quantify and qualify the way the ontology aligns with the propagation rules: precision and recall indicate how much a relation is close to being a suitable abstraction for policy propagation.

Table 2 shows an example of a concept obtained applying the approach in the context of the Datanode ontology. Here it is worth mentioning some general considerations that can be made by inspecting these measures.

Table 2
Excerpt from the table of measures computed by the abstraction algorithm in Listing 6.
c=Concept ID, ES=Extent Size, IS=Intersection Size, BS=Branch size, Pre=Precision, Rec=Recall, F1=F-Measure.

c	ES	IS	BS	Pre	Rec	F1	Branch
79	52	52	115	0.45	1	0.62	relatedWith
77	46	19	21	0.9	0.41	0.56	hasDerivation
75	44	8	11	0.73	0.18	0.29	samePopulationAs
67	35	7	7	1	0.2	0.33	hasPart
67	35	6	7	0.86	0.17	0.28	isPartOf
36	16	3	3	1	0.19	0.32	hasCopy
36	16	3	3	1	0.19	0.32	isCopyOf
24	12	6	6	1	0.5	0.67	hasVocabulary
9	8	1	1	1	0.12	0.21	hasReification
0	4	4	115	0.03	1	0.06	relatedWith

When $Rec = 1$, the whole extent of the concept is in the branch. The branch might also include other relations, that do not propagate the policies included in the concept. When $Pre = 1$, we can perform the subtraction of rules, as in Listing 6. A low recall indicates that a high number of exceptions still need to be kept in the rule set. It also reflects a high ES, from which we can deduce a low number of policies in the concept. As a consequence of that, inspecting a partial match with high precision and low recall highlights a problem that might be easy to fix, as the number of relations and policies to compare will be low. For example, row 2 of Table 2 relates to a relation with $BS - IS = 2$, so we need only to check whether 2 relations in the `hasDerivation` branch might also propagate the policies in concept 77. The perfect match between a concept and a branch of the ontology would be $F1 = 1$. However, when this does not happen we can try to improve the approximation.

At this stage we can make the following considerations:

– The presence of mismatches between the lattice and the ontology will make the reasoner return wrong results, thus they must be eliminated.
– The size of the matrix that was prepared in the *Acquisition* phase is pretty large, and it is possible that errors have been made at that stage of the process.
– The Datanode ontology has not been designed for the purpose of representing a common behavior of relations in terms of propagation of policies. It is probably possible to refine the ontology in order to make it cover this use case better (and reduce the number of rules even more).

4.5. *Adjustment*

In this phase, we try to make adjustments in order to (a) repair mismatches identified in the assessment phase and (b) evolve the rule base R or the ontology hierarchy H to improve the compression factor CF. Six operations can be performed: *Fill*, *Wedge*, *Merge*, *Group*, *Remove*, *Add*. Here we briefly illustrate the property of each operation, referring to [8] for the details of it, and provide three examples.

Fill. The Fill operation makes a branch b be fully in concept c, attempting to push Pre up to 1. This is achieved by adding to R all the rules generated from the association between the policies in concept c and the relations in branch b. This change affects the PPR knowledge base R, increasing the number of rules.

Wedge A new relation is wedged between a top relation and all its direct subproperties. The new branch will allow to perform a *Fill* operation, in the next iteration.

Merge Two top relations are abstracted by a new common relation. The new branch will allow to perform a *Fill* operation, in the next iteration.

Group A set of relations are all together in the extent of a concept, but belong to different branches. We want to create a common parent relation for them. Again, the new branch enables a *Fill* operation, to be executed in the next iteration.

Remove and Add We can *Remove* a relation as a subproperty of another (and possibly cut a sub-branch). This operation removes a single subsumption relation in the ontology. After that, we might relocate it elsewhere with a *Add* operation.

After a change to the ontology, normally the *Fill* operation is performed on the newly created branch, in order to populate the rule base accordingly. Except for the *Fill* operation, all the operations are performed on the ontology. As shown in Figure 2, after the Adjustment phase we restart a new iteration.

In what follows we illustrate three examples of changes performed during our application of the methodology.

Example 1. The *Assessment* phase of the methodology reported possible mismatches between the FCA output and the ontology hierarchy. These errors must be repaired if we want the compressed rule base to be used by a reasoner. For example, Listing 9 shows the set of mismatches detected for concept (71). In this list, the `dn:isVocabularyOf` branch contains a number of relations that do not propagate the related policies, breaking the assumption that all the policies of `dn:isVocabularyOf` are also propagated by all the other relations in his branch. With a *Fill* operation, we can add all the necessary rules to remove this mismatch.

Example 2. In Section 4.4 we described a method to catch possible errors in the rule base, based on the identification of partial matches with high precision and low recall. Such cases highlight a branch that is close to be fully included in a concept. As example, we can pick branch `dn:isPartOf` from Listing 7. Listing 10 shows the details about how the concept matches this branch. It happens that all relations except `dn:isSelectionOf` are part of this concept. In other words, they propagate the policies listed in the intent of Concept 74 (Listing 5). However, this is a mistake that happened during the *Acquisition* phase, as `dn:isSelectionOf` should behave in a similar way to `dn:isExampleOf`.

Listing 10: Example of the matches between a concept and the branches in the Datanode hierarchy.

```
c     es    is    bs    pre   rec   f1    branch
74    43    6     7     0.86  0.14  0.24  dn:isPartOf
                              + dn:isPartOf
                              ! dn:isSelectionOf
                              + dn:isExampleOf
                              + dn:isSectionOf
                              + dn:identifiersOf
                              + dn:isPortionOf
                              + dn:isSampleOf
```

We decide then to perform a *Fill* operation, adding all the necessary rules to make the branch `dn:isPartOf` fully covering the intent of Concept 74.

Example 3. A branch with similar scores is `dn:sameCapabilityAs`. Listing 11 shows that the only missing relation is the top one. In Datanode, `dn:sameCapabilityAs` is defined as the relation between two objects having the same vocabulary and the same population (containing actually the same data). However, it is possible that two objects have the same "data" without having the same policies. For example, datasets like lists of cities or postcodes might be imported from different sources, and having different policies while containing the same data! In this case we opted for adding a new relation to Datanode that can abstract all the branches with a more focused semantic: `dn:sameIdentityAs`. `dn:sameIdentityAs` tries to capture exactly the fact that two data objects share the same origin, the same population and the same vocabulary. The operation performed to add this relation is *Wedge*, as the new property is injected between `dn:sameCapabilityAs` and its direct subrelations.

Listing 11: Example of the matches between a concept and a branch in the Datanode hierarchy.

```
c    es   is   bs   pre   rec   f1    branch
74   43   7    8    0.87  0.16  0.27  dn:sameCapabilityAs
                          ! dn:sameCapabilityAs
                          + dn:hasCopy
                          + dn:hasSnapshot
                          + dn:hasCache
                          + dn:isCopyOf
                          + dn:isSnapshotOf
                          + dn:isCacheOf
                          + dn:duplicate
```

After each operation we run our process again from the *Analysis* phase to the *Assessment*, in order to evaluate whether the change fixed the mismatch and/or how much the change affected the compression factor. The process is repeated until all mismatches have been fixed, and a reasonably good compression factor is reached, or no more meaningful changes are possible.

4.6. *Evaluation of the CF*

In the previous sections we described the phases of the (A)AAAA methodology, and how we applied it to the task at hand. Figure 3 shows how the compression factor CF increases with the number of adjustments performed, while Figure 4 illustrates the progressive reduction of mismatches. Details about the changes performed are provided in Table 3 ($+$). This includes statistics about number of mismatches ($\neq$), the impact on number of rules (R), number of concepts generated by FCA (C), number of rules abstracted (A), remaining rules (R_+), and compression factor (CF). Moreover, Table 3 highlights the improvements obtained before (published in [8]) and after the introduction of the *coherency check* method in the *Assessment* phase (after change 15).

Table 3

List of changes performed.

$+$	C	$\neq$	R	A	R_+	CF
0	80	15	3363	1925	1438	0.572
1	80	16	3370	1953	1417	0.58
2	80	16	3370	1953	1417	0.58
3	80	16	3480	2283	1197	0.656
4	80	18	3482	2299	1183	0.66
5	78	12	3500	2376	1124	0.679
6	78	14	3608	2484	1124	0.688
7	78	16	3716	2592	1124	0.698
8	96	16	3822	2698	1124	0.706
9	93	15	3824	2706	1118	0.708
10	93	15	3824	2706	1118	0.708
11	93	15	3824	2706	1118	0.708
12	93	15	3824	2706	1118	0.708
13	76	15	3837	2765	1072	0.721
14	76	15	3844	2778	1066	0.723
15	78	15	3865	2817	1048	**0.729**
16	78	13	3866	2818	1048	0.729
17	78	13	3874	2826	1048	0.729
18	63	11	3878	2830	1048	0.73
19	63	11	3882	2834	1048	0.73
20	63	9	3892	2844	1048	0.731
21	55	9	3897	2849	1048	0.731
22	60	8	3898	2850	1048	0.731
23	60	3	3908	2860	1048	0.732
24	54	0	3914	2870	1044	0.733
26	34	0	4225	3451	774	**0.817**

Table 3

The first column identifies the change performed (starting from the initial state).
C=Number of concepts in the FCA lattice
$\neq$=Number of mismatches between the FCA lattice and the ontology
R=Number of rules before the process
A=Number of rules abstracted (subtracted)
R_+=Size of the compressed rule base (without the abstracted rules)
CF=Compression Factor

Apart from being mandatory to be able to use the compressed knowledge base with a reasoner, the application of this approach allowed us to reduce the size even more. As final result we obtained: 4225 rules in total, 34 concepts, 3451 rules abstracted and 774 rules remaining, boosting the CF up to **0.817**.

Thanks to this methodology we have been able to fix many errors in the initial data, to refine Datanode by clarifying the semantics of many properties and adding new useful ones. The version of the ontology at the beginning of this work can be found at `http://purl.org/datanode/0.3/ns/`. The current

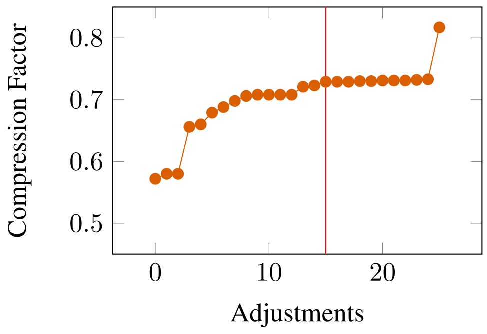


Fig. 3. Progresss of the CF.

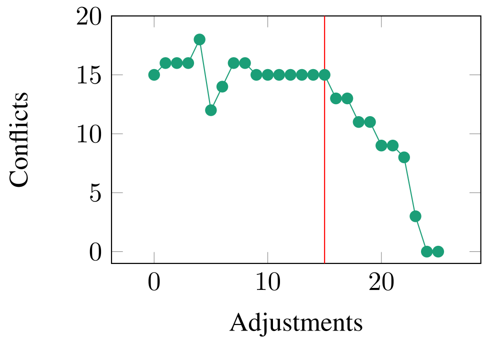


Fig. 4. Progresss in the number of conflicts.

version of the ontology is at `http://purl.org/datanode/0.5/ns/`.

5. Experiments

The methodology described in the previous Section allows to reduce the number of rules to be stored and managed. We will now report on the results of a set of experiments we performed on reasoning about policies propagation. In particular, we want to evaluate the impact of compression on reasoning performance. We took 15 data flows descriptions from a previous work [7], referring to 5 applications that rely on data obtained from the Web. Each data flow represents a data manipulation process, from an input data source (sometimes multiple sources), resulting in one principal output node. The task of the reasoner is to find all the policies associated with the output of the data flow, according with the ones associated with the input. This task is performed two times: the first providing the full set of propagation rules, the second providing the compressed rule base in conjunction with the hierarchy of relations.

The experiments have the objective to compare performance of the reasoner when using an *Uncompressed* or a *Compressed* rule base. To make the observations visible and not dependent on a given implementation, we run the experiments with two different reasoners: a *Naive* implementation and an *Optimized* implementation. Both reasoners have the capability of executing PPRs and expand the results according to the ontology hierarchy.

The *Naive* implementation is a Prolog program relying on JLog, a prolog interpreter written in Java[17]. The program incorporates a *meta rule* that traverses the set of PPRs, encoded as facts. At the same time, it supports the subsumption between relations. Listing 12 shows an excerpt of the program.

Listing 12: Excerpt of the Prolog *Naive* reasoner program.

```
i_rdfs_sub_property_of(X,X) .
i_rdfs_sub_property_of(X,Y) :-
  rdfs_sub_property_of(X,Z), i_rdfs_sub_property_of(Z,Y) .
i_propagates(X,Y) :- propagates(X,Y) .
i_propagates(X,Y) :- i_rdfs_sub_property_of(X,Z),
                       propagates(Z,Y) .
i_has_policy(T,P,_) :- has_policy(T,P) .
i_has_policy(T,P,L) :-
                    i_has_relation(S,T,R),
                      not(visited(S,L)),
                        i_propagates(R,P),
                          i_has_policy(S,P,[S|L]) .
i_has_policy(T,P) :- i_has_policy(T,P,[]) .
```

The *Optimized* reasoner is built on top of the RDFS reasoner of Apache Jena[18] in combination with SPIN[19], a rule engine that allows to define rules using the SPARQL language. The core part of the reasoner executes PPRs as a SPARQL meta query (Listing 13).

Listing 13: Construct meta-query of the *Optimied* SPIN based reasoner.

```
CONSTRUCT {
  ?this ppr:policy ?policy
} WHERE {
  ?int ?relatedWith ?this .
  ?int ppr:policy ?policy .
  ?relatedWith ppr:propagates ?policy
}
```

We performed the experiments with the data flows listed in Table 4. These use cases were formalized before the present work (in [7]). The related data flow descriptions were not altered for the task at hand, except that the part about the policies of the input was added.

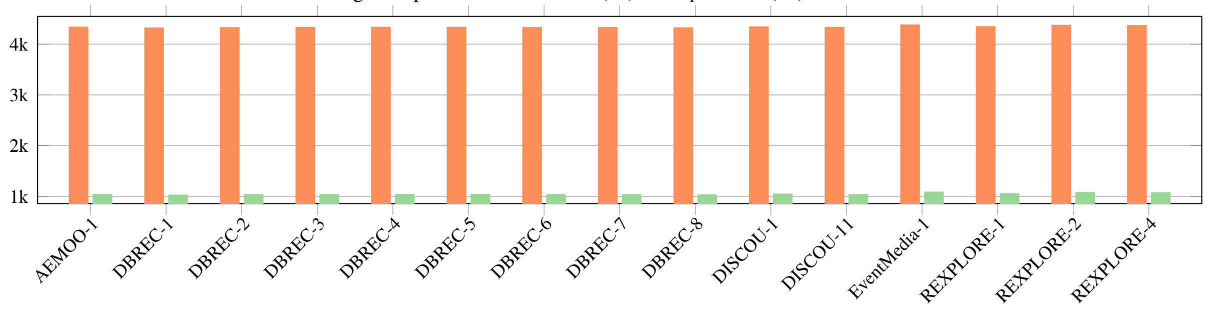
[17]`http://jlogic.sourceforge.net/`
[18]`http://jena.apache.org/`
[19]`http://spinrdf.org/`

Table 4

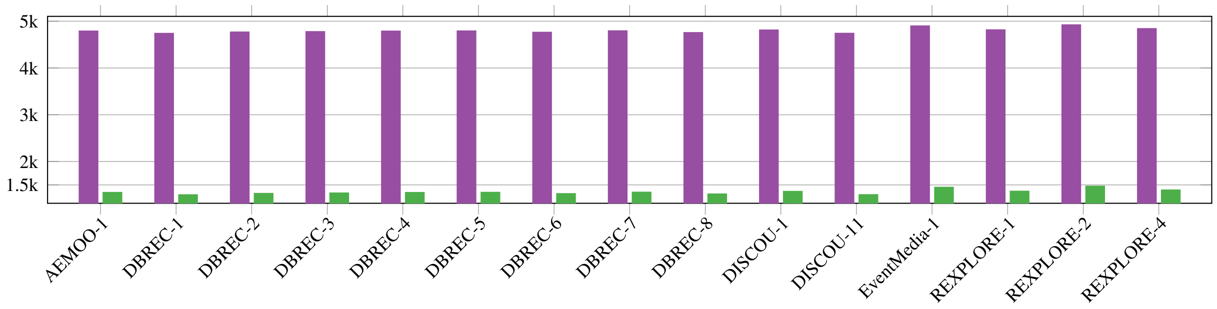
Data flows used in the experiments. The *has policy* and *has relation* columns report the number of statements about policies and relations between data objects in the dataflow. *relations* reports the number of distinct relations, the same applying to *data objects*, *policies*, *sources* and *output policies*. Highlighted are the maximum and minimum values for each of the data flow inputs. In one case (DISCOU-11), none of the policies attached to the source are propagated to the output.

Use case	has policy	has relation	relations	data objects	policies	sources	output policies
AEMOO-1	6	18	13	10	6	1	6
DBREC-1	6	2	2	2	6	1	3
DBREC-2	6	8	7	5	6	1	6
DBREC-3	6	12	7	8	6	1	6
DBREC-4	6	14	8	9	6	1	3
DBREC-5	6	14	10	10	6	1	6
DBREC-6	6	10	10	6	6	1	3
DBREC-7	6	9	6	10	6	1	6
DBREC-8	6	5	4	5	6	1	6
DISCOU-1	7	22	10	14	7	1	5
DISCOU-11	5	13	9	12	5	1	0
EventMedia-1	37	25	8	24	25	6	4
REXPLORE-1	16	14	8	14	8	3	3
REXPLORE-2	32	23	4	18	14	6	6

Fig. 5. Input size for the *Naive* (5a) and *Optimized* (5b) reasoners.



(a) *Naive* reasoner: input size (number of Prolog facts) with and without compression for each use case. It is visible that the size of the data flow has a small impact on the general size of the input.



(b) *Optimized* reasoner: input size (number of RDF triples) with and without compression for each use case. It is visible that the size of the data flow has a small impact on the general size of the input.

Each data flow describes a process executed within one of five applications. The table lists the basic properties of these use cases. The size of the data flow is reported in the *has relation* column of the table, indicating the number of Datanode relations asserted in the input. The *has relation* column reports the number of statements about policies. *relations* reports the number of distinct relations, the same applying to *data objects*, *policies*, *sources* and *output policies*. Highlighted are the maximum and minimum values for each of the data flow inputs. In one case (DISCOU-11), none of the policies attached to the source are propagated to the output.

Each experiment takes the following arguments:

– *Input*: a data flow description
– *Compression*: $True/False$
– *Output*: the output resource to be queried for policies

In case *compression* is $False$, we feed the reasoner with the complete set of PPRs and no information on subsumption between the relations described in the dataflow. Conversely, when *compression* is set to $True$, the compressed set of PPRs is used in conjunction with the Datanode ontology. It is worth noting that the (A)AAAA Methodology is also an evolution method, and we are considering here the evolved rule base (and ontology), that has been harmonized by fixing mismatches between the rule set and the ontology.

The experiments was executed on a MacBook Pro with processor Intel Core i7/3 GHz Dual Core and 16 GB of RAM. In case a process was not completed in five minutes, it was forcely interrupted. Each process was monitored and information about CPU usage and RAM (RSS memory) registered at intervals of half a second. We compared the results of the experiments with and without compression, and we always obtained the same result set. When terminating, the experiment output would include: total execution time (t), resources load time (l), setup time (s), and query execution time (q). The size of the input for each experiment is reported in the diagrams in Figure 5.

We consider performance on two principal dimensions: time and space.

Time performance is measured under the following dimensions:

L Resources load time.
S Setup time. Includes L, in addition to any other operation performed before being ready to receive queries (eg, materialization)
Q Query time.
T Total duration: $T = S + Q$

Space is measured as follows:

Pa Average *CPU* usage.
M Maximum memory required by the process

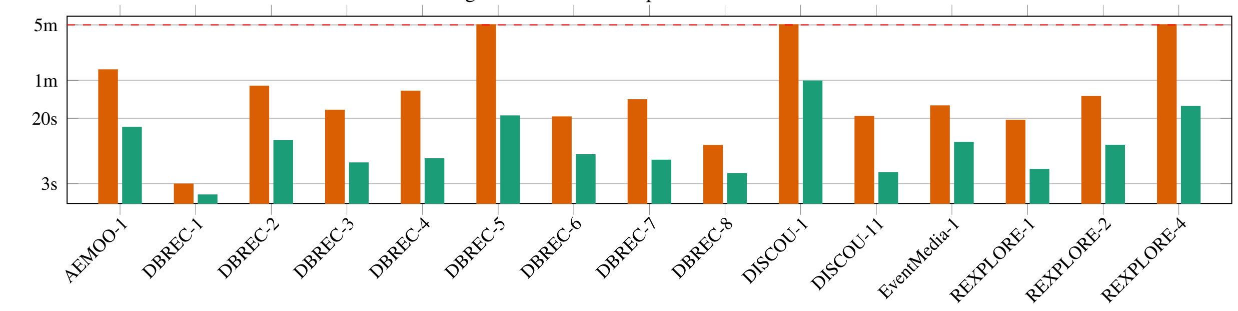
Each experiment was executed 20 times. What we are showing here is the average of the measures obtained in the different executions. In order to evaluate the accuracy of the average measure computed from the twenty executions of the same experiment, we calculated the related *Coefficient of Variation* (CV)[20]. In almost all the cases the CV for the *Naive* reasoner was below 0.1, with the exception of memory (RSS) usage, that in many cases showed a fluctuation between 0.2 and 0.4. With the *Optimized* reasoner the experiments reported a much more stable behaviour in terms of consumed resources, the CV being assessed below 0.1 in almost all the cases, except the Query time of some experiments. However, these were fluctuating around an average of 10ms. Finally, we consider the results of these experiments to be very accurate.

Before discussing the results, it is worth reminding the reader that this evaluation is not targeted to compare the two implementations of a PPR reasoner, but to observe the impact of our compression methodology on a *Naive* and an *Optimized* implementations, assuming that a hypothetical implementation would perform between these two extremes.

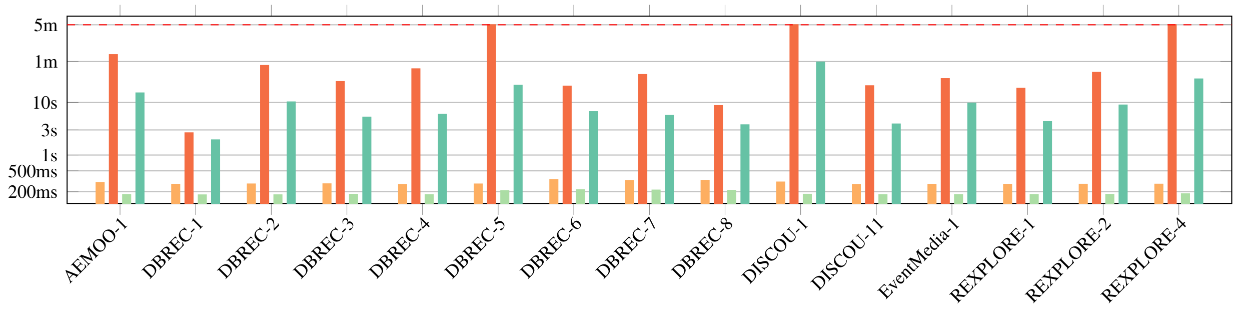
Figures 6 and 7 illustrate the results of the experiments performed with the *Naive* and the *Optimized* reasoner respectively. Figure 6a displays a comparison of the execution time between an *Uncompressed* and *Compressed* input with the *Naive* reasoner. For each use case, the bar on the left displays the time with an *Uncompressed* input, the one on the right the time with a *Compressed* input. We will follow this convention in all the other diagrams as well. In all cases there has been a significant increase in performance. In three cases (DBREC-5, DISCOU-1, REXPLORE-4) the *Uncompressed* version of the experiment could not complete in five minutes, while the *Compressed* version returned results in less then a minute. The execution time of the experiments with the *Optimized* reasoner (Figure 7a) is much smaller (fractions of a second), having the maximum execution time of approximately 2 seconds (EventMedia-1). However, also

[20]Coefficient of Variation. `https://en.wikipedia.org/wiki/Coefficient_of_variation`

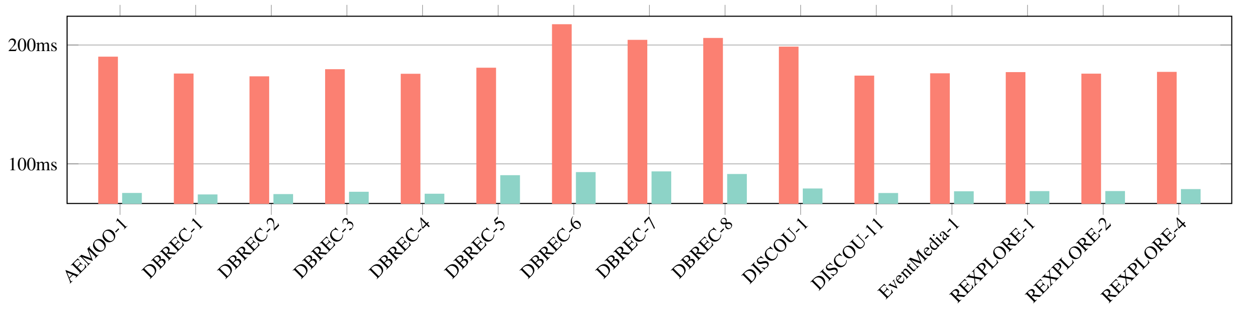
Fig. 6. *Naive* reasoner: performance measures.



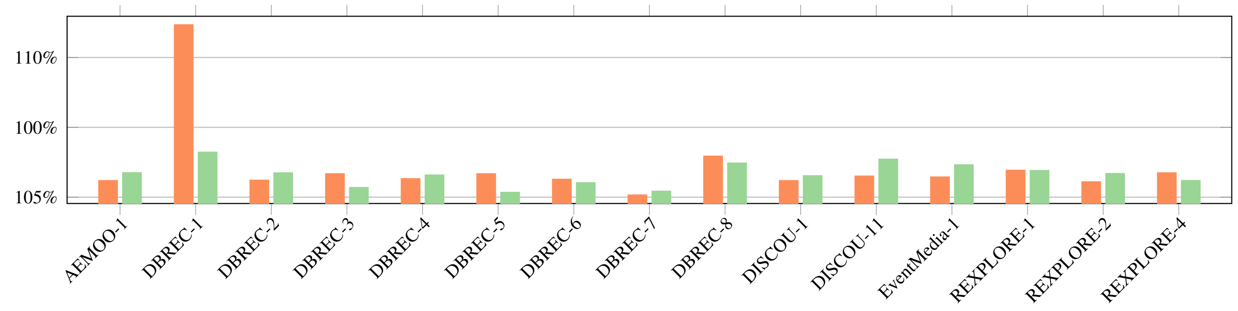
(a) *Naive* reasoner: total execution time (T). The bar on the left of each experiment shows the average duration of the experiment with the uncompressed input. The bar on the right the one with a compressed input.



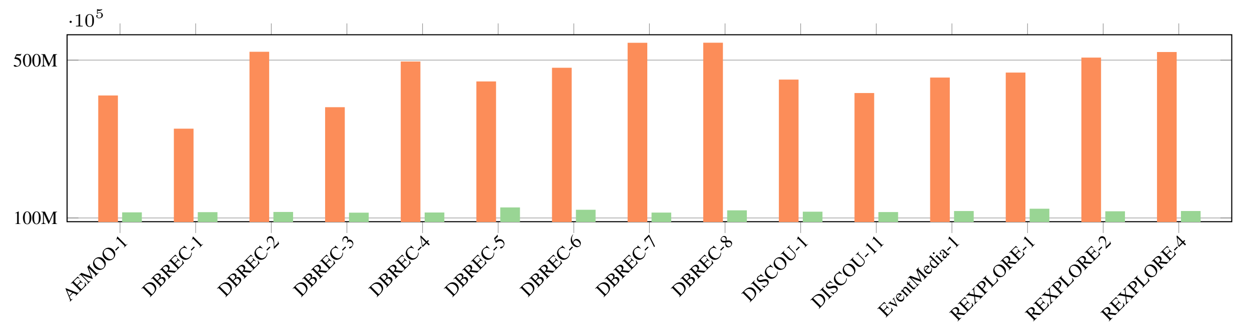
(b) *Naive* reasoner: Setup/Query execution time (S/Q).



(c) *Naive* reasoner: resources load time (L).

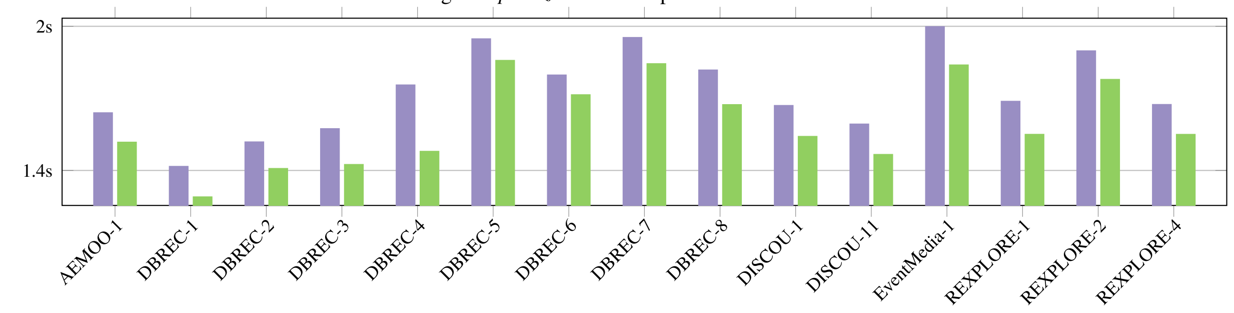


(d) *Naive* reasoner: average CPU (Pa).

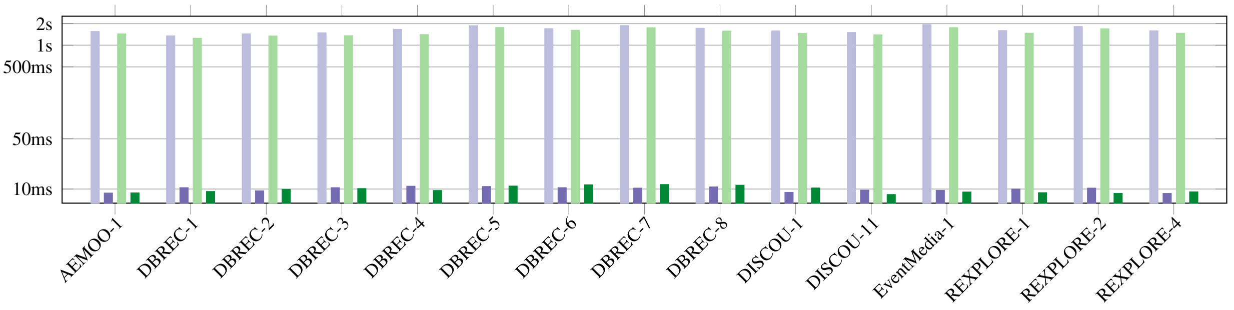


(e) *Naive* reasoner: max memory consumption (M).

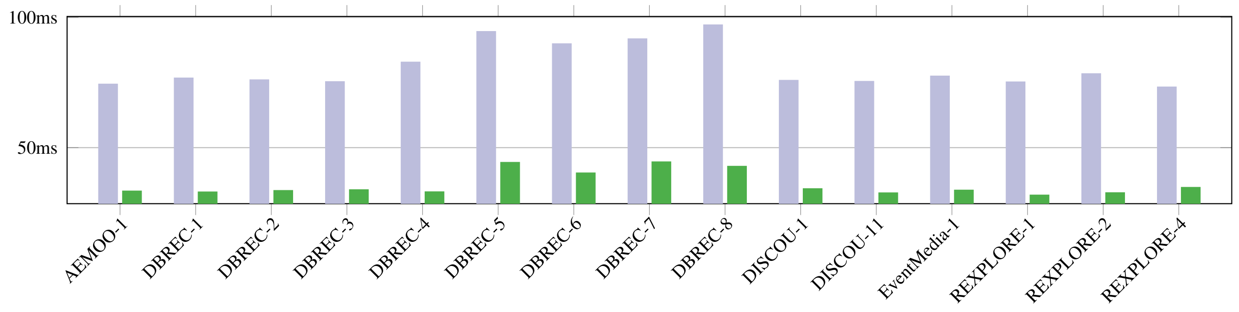
Fig. 7. *Optimized* reasoner: performance measures.



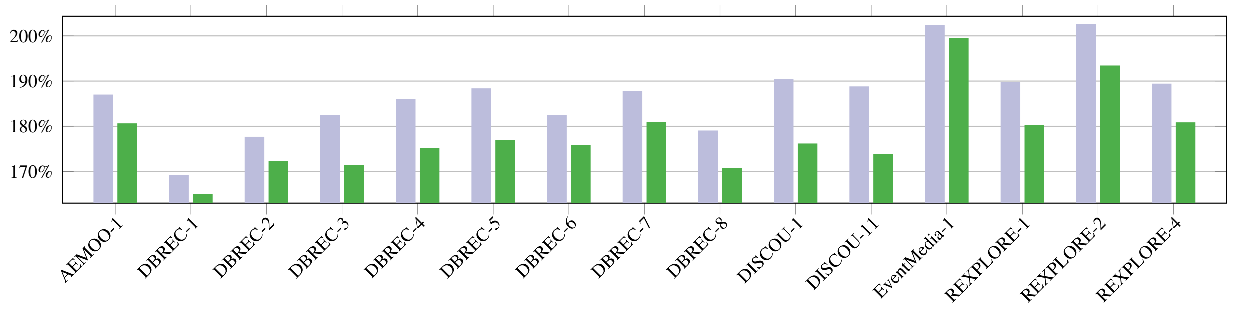
(a) *Optimized* reasoner: Total execution time (T).



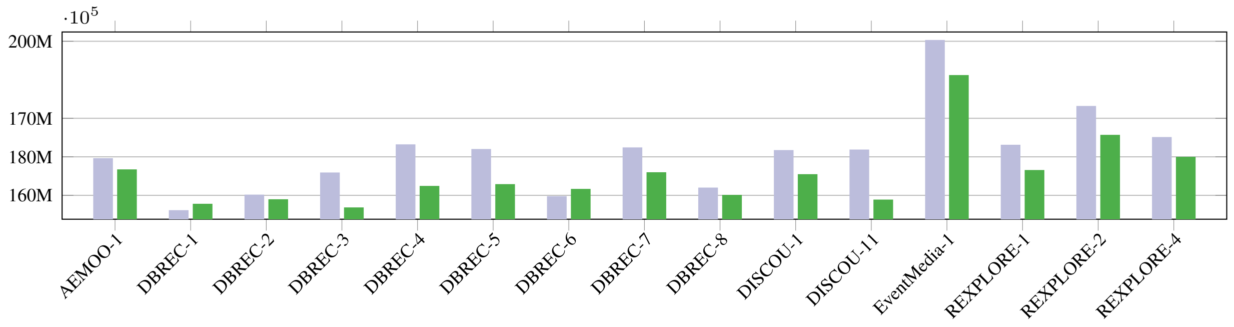
(b) *Optimized* reasoner: Setup/Query execution time.



(c) *Optimized* reasoner: resources load time.



(d) *Optimized* reasoner: average CPU usage.



(e) *Optimized* reasoner: max memory consumption (RSS).

in this case we report a costant increase in performance for all the use cases, in some cases significant (DBREC-3, DBREC-4). We can divide the Total time (T) of the experiment and observing setup time S and query time Q. Setup time includes resources load time. This observation is depicted in Figures 6c and 7c, and in both cases the impact of the rules reduction process is evident. An interesting difference between the two implementations can be seen by comparing Figures 6b and 7b. The cost of the query time in the *Naive* reasoner is very large compared to the related setup time (S). The *Optimized* reasoner, conversely, showed a larger setup time (S) with a very low query execution (Q) cost. The reason is that the second materializes all the inferences at setup time, before query execution. This accounts for the lack of difference in query time between the *Uncompressed* and *Compressed* version of the experiments.

We did not observed changes in CPU usage (Pa) in the *Naive* implementation (Figure 6d), while changes in memory consumption (M) looks significant (Figure 6e). The boost in space consumption has been also observed in the *Optimized* reasoner (Figures 6d and 6e), even if smaller, and negative in only 2 cases with regard to memory consumption (DBREC-1 and DBREC-6).

A summary of the impact of the compression on the different measures is depicted in Figures 8 and 9. A serious improvement has been achieved in the case of the *Naive* implementation of a PPR reasoner. Such a reasoner could be implemented in several different ways, this is why we bring the experiments with the *Optimized* implementation as a counter proof of the validity of the hypothesis that the compression of the knowledge base do actually have a positive impact on the reasoning process, even if in different ways depending on the efficiency of the algorithms and implementations.

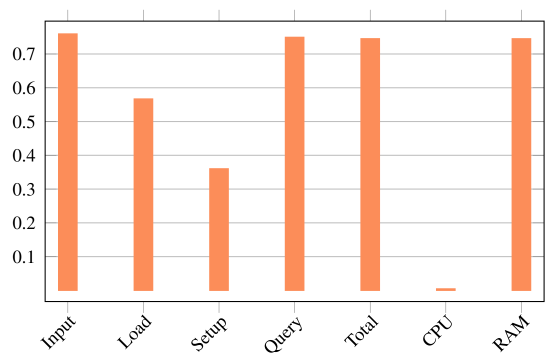


Fig. 8. *Naive* reasoner: boost analysis.

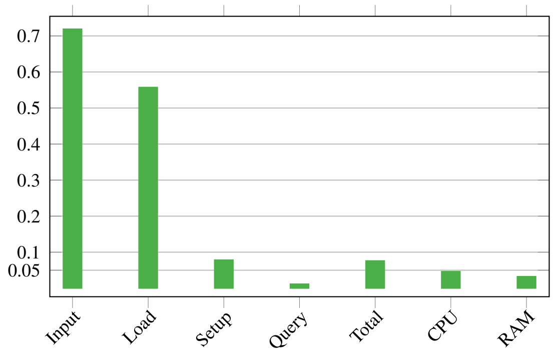


Fig. 9. *Optimized* reasoner: boost analysis.

6. Conclusions

In this article we presented an approach for reasoning on the propagation of policies in a data flow. This method is grounded on a rule base of Policy Propagation Rules (PPRs). Rules can easily grow in number, depending on the size of the possible policies and the one of the possible operations performed in a data flow. The (A)AAAA methodology can be used to reduce this size significantly, as demonstrated in [8], by relying on the inference properties of the Datanode ontology, applied to describe the possible relations between data objects. We presented an evolved version of the methodology, that was required to be sure the inferred policies were correct when using the compressed rule base. However, while this activity reduces the size of the input of the reasoner, it requires to compute more inferences. Therefore, we performed experiments to assess the impact of the compression on reasoning performance. The contributions of this article are two:

- the (A)AAAA methodology has been extended by including a *coherency check* algorithm, and
- experimental results demonstrating that a compressed knowledge base makes the reasoning on policies propagation more efficient.

This is a preliminary step on studying compression in knowledge management and its impact on reasoning in a more general point of view. Future work includes methods to support or automate the generation of data flow descriptions and to study the validation of data flows with respect to policies, particularly when multiple sources are used. Finally, we are setting up an experimental evaluation in the environment of the MK:Smart Data Hub [9].

References

[1] G. Antoniou and G. Wagner. Rules and defeasible reasoning on the semantic web. In Schröder, Michael and Wagner, Gerd, editor, *Rules and Rule Markup Languages for the Semantic Web*, volume 2876 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin Heidelberg, 2003.

[2] J.-M. Bohli, A. Skarmeta, M. Victoria Moreno, D. Garcia, and P. Langendorfer. Smartie project: Secure iot data management for smart cities. In *Recent Advances in Internet of Things (RIoT), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[3] O. Boissier, M. Colombetti, M. Luck, J.-J. Meyer, and A. Polleres. Norms, organizations, and semantics. *The Knowledge Engineering Review*, 28(01):107–116, 2013.

[4] P. A. Bonatti and D. Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Proceedings of the Third International Summer School Conference on Reasoning Web*, RW'07, pages 240–268, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] E. Boros, O. Čepek, and P. Kučera. A decomposition method for cnf minimality proofs. *Theoretical Computer Science*, 510:111–126, 2013.

[6] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. A bottom-up approach for licences classification and selection. In S. Villata and S. Peroni, editors, *Proc. of the International Workshop on Legal Domain And Semantic Web Applications (LeDA-SWAn) held during the 12th Extended Semantic Web Conference (ESWC 2015)*, pages 33–40. ACM, 2012.

[7] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. Describing semantic web applications through relations between data nodes. Technical Report kmi-14-05, Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, 2014.

[8] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. Propagation of policies in rich data flows. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, pages 5:1–5:8, New York, NY, USA, 2015. ACM.

[9] M. d'Aquin, A. Adamou, E. Daga, S. Liu, K. Thomas, and E. Motta. Dealing with diversity in a smart-city datahub. In T. Omitola, J. Breslin, and P. Barnaghi, editors, *Proceedings of the Fifth Workshop on Semantics for Smarter Cities, a Workshop at the 13th International Semantic Web Conference (ISWC 2014)*, Riva del Garda, Italy, 19 October 2014. CEUR-WS.org.

[10] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *The Semantic Web: Research and Applications*, pages 342–356. Springer, 2004.

[11] J. M. Gómez-Pérez and O. Corcho. Problem-solving methods for understanding process executions. *Computing in Science & Engineering*, 10(3):47–52, 2008.

[12] G. Governatori, H.-P. Lam, A. Rotolo, S. Villata, G. Atemezing, and F. Gandon. Checking licenses compatibility between vocabularies and data. In *Proceedings of the Fifth International Workshop on Consuming Linked Data (COLD2014)*, 2014.

[13] G. Governatori, A. Rotolo, S. Villata, and F. Gandon. One license to compose them all. In *The Semantic Web–ISWC 2013*, pages 151–166. Springer, 2013.

[14] P. L. Hammer and A. Kogan. Optimal compression of propositional horn knowledge bases: complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

[15] R. Iannella, S. Guth, D. PÃd'hler, and A. Kasten. ODRL: Open Digital Rights Language 2.1. Technical report, W3C Community Group, 2015.

[16] H. Khrouf and R. Troncy. Eventmedia: A lod dataset of events illustrated with media. *Semantic Web journal, Special Issue on Linked Dataset descriptions*, 7(2):193–199, 2016.

[17] H.-P. Lam and G. Governatori. The making of spindle. In *Rule Interchange and Applications*, pages 315–322. Springer, 2009.

[18] H. Li, X. Zhang, H. Wu, and Y. Qu. Design and application of rule based access control policies. In *Proc of the Semantic Web and Policy Workshop*, pages 34–41, 2005.

[19] D. McGuinness, T. Lebo, and S. Sahoo. PROV-o: The PROV ontology. W3C recommendation, W3C, Apr. 2013. http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[20] E. Motta, T. Rajan, and M. Eisenstadt. Knowledge acquisition as a process of model refinement. *Knowledge acquisition*, 2(1):21–49, 1990.

[21] J. Padget and W. W. Vasconcelos. Policy-carrying data: A step towards transparent data sharing. *Procedia Computer Science*, 52:59 – 66, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[22] V. Rodríguez-Doncel, S. Villata, and A. Gómez-Pérez. A dataset of RDF licenses. In R. Hoekstra, editor, *Legal Knowledge and Information Systems. JURIX 2014: The Twenty-Seventh Annual Conference.* IOS Press, 2014.

[23] M. Sensoy, T. J. Norman, W. W. Vasconcelos, and K. Sycara. Owl-polar: A framework for semantic policy representation and reasoning. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12:148–160, 2012.

[24] R. Shaw, R. Troncy, and L. Hardman. Lode: Linking open descriptions of events. In *The Semantic Web*, pages 153–167. Springer, 2009.

[25] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1):161–197, 1998.

[26] R. Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*, pages 1–33. Springer, 2005.