Towards the Automatic Detection and Correction of Errors in Automatically Constructed Ontologies

Towards the automatic detection and correction of errors in automatically constructed ontologies

Andriana Gkaniatsou
School of Informatics
University of Edinburgh
Edinburgh, UK
Email: agkaniat@inf.ed.ac.uk

Alan Bundy
School of Informatics
University of Edinburgh
Edinburgh, UK
Email: a.bundy@ed.ac.uk

Fiona McNeill
School of Informatics
University of Edinburgh
Edinburgh, UK
Email: f.j.mcneill@ed.ac.uk

***Abstract*—The Open Information Extraction Project[1] is one of the most ambitious attempts in the area of automatically constructing ontologies by harvesting information from the web. What we will call their KnowItAll Ontology contains about 6 billion items, consisting of triples and rules. The downside of such automatically constructed ontologies is that they contain a vast number of errors: some arising from errors in the original web data and some from errors in extracting the data. In this project we explore whether techniques we have developed in the domain of ontology repair can be used to detect and correct some of these errors. In particular, we explore whether the errors in their ontology can be automatically detected by using a theorem prover. We also present a manual classification of the errors as a preliminary feasibility exploration, and discuss our future work towards automatically correcting the ontology based on the error classification.**

I. Introduction

Ontology construction has long been a subject of extensive research. Some of the goals in that line of work are capturing as much detail as possible within a specific domain, reducing the cost of ontology construction and minimising the likelihood of inconsistencies. The traditional way of constructing ontologies is to utilise domain experts for the conceptualisation of the domain. The work of the Cyc project [1] has emphasised the importance and applications of a wide-coverage, commonsense ontology. At the same time, however, the Cyc project also illustrates the immense resource investment required to construct such an ontology manually and the limitations of such an approach.

The need to lower the cost of ontology construction and create flexible ontologies capable of coping with continuously changing data has resulted in a growing interest in automating the ontology construction process as much as possible. Several methods have been proposed, among them the automatic construction of ontologies by mining the web. One of the most successful projects based on that approach is the KnowItAll Project[2], an outcome of the Open Information Extraction Project, which obtains large-scale knowledge from the web, structures it and presents in the form of an ontology. However, the biggest drawback of such an approach is that erroneous data that already exists in the web is transferred in the ontology.

Ontology consistency checking is one of the most fundamental steps towards ontology construction. Common approaches exploit Description Logic (DL) reasoners, as most ontology languages, such as OWL [2], are based on description logics. However, the development of languages based on the more expressive first-order logic, *e.g.*, [3], [4], dictates the use of first-order theorem provers for consistency checking. This technique was only recently introduced and has gathered significant attention. For example, in [5] different theorem provers were tested for finding inconsistencies in Cyc; whereas in [6], the first-order general purpose theorem prover Vampire was used to find inconsistencies in large ontologies such as SUMO [7].

Most first-order provers use resolution techniques for their proofs: given a goal ϕ and a set of axioms Ψ, prove that ϕ follows from Ψ by proving $\neg\phi \wedge \Psi$ unsatisfiable, under the premise that Ψ is satisfiable. If $\neg\phi \wedge \Psi$ is unsatisfiable, then there does not exist a contradiction within the ontology. Such a technique, to be successful, requires the language of that axiom set to contain negation. The KnowItAll Ontology lacks negation, thus, a new approach is needed. This paper presents our approach to automatically detecting functionality and type clash inconsistencies within the KnowItAll Ontology using the E theorem prover [8], and our techniques to handle the lack of negation. E is a full first-order theorem prover with equality. Given a set of formulas, E saturates it by systematically applying a number of inference rules until all possible rules have been applied, or the empty clause has been derived.

If the ontology contains inconsistencies, one way to repair them is to discard from the ontology the minimal set of data necessary to block the proof of inconsistency. This approach secures the consistency of the ontology, but also results in information loss. We propose repairing the inconsistent data rather than discarding it, to minimise possible information loss. By classifying the occurring errors we can identify the links between the correct and the erroneous data, and decide whether the erroneous data should be repaired or discarded

[1] http://ai.cs.washington.edu/projects/open-information-extraction
[2] http://www.cs.washington.edu/research/knowitall

from the ontology. Using first-order logic allows us to express and apply the repairs. Note here that alternative logics like Description Logic are not expressive enough for the repairs we have identified, as we will show in Section IV-B2.

To summarise, our main contribution is that we present ways to *automatically* detect functionality and type clash inconsistencies in a ontology that lacks negation, by using a theorem prover. Furthermore, we present a manual classification of the errors that cause the inconsistencies, and we discuss ways to map the inconsistencies to the appropriate repairs. The rest of the paper is organised as follows. In Section II we present our methodology towards the automatic detection of inconsistencies. In Section III we present a preliminary evaluation of our approach and the results. A manual classification of the errors and our proposals to repair the inconsistencies using ontology repair techniques is discussed under Future Work in Section IV. Finally, we draw conclusions in Section V.

II. Methodology

The KnowItAll Ontology. The KnowItAll system consists of two subsystems: the Reverb system [9], and the SHERLOCK system [10]. The Reverb system extracts binary relation phrases and the corresponding arguments, from the web. Only the extractions that match with some predefined syntactic and lexical constraints are added to the ontology. Then, the added relations are checked for partial functionality [11]. A relation Rel is functional, when given an element of the domain $i \in Domain(Rel)$, there exists a unique element of the range $y \in Range(R)$ such that $Rel(i, y)$ is true. Functionality restricts both the existence and the uniqueness of y. Whereas partial functionality restricts only the uniqueness of y: relation Rel is a partial function if given an element $i \in Domain(Rel)$ and if there exists $y \in Range(Rel)$ such that $R(i, y)$ is true, then y is unique. SHERLOCK classifies the extracted arguments and produces inference rules for each typed relation, based on the web extractions. The resulting ontology consists of the extracted relations and the corresponding instances, the classes of these instances and the inference rules.

Translation. The ontology has to conform to the TPTP (Thousands of Problems for Theorem Provers) syntax [12], a standard first-order format readable by the E prover. TPTP is a library of problems written in TPTP syntax, and is the basis for the CADE ATP System Competition [13]. In the following experiments we used the TPTP distinct object notation. In TPTP distinct object notation, all distinct objects are surrounded by double quotes and are not considered equal. We decided to translate subsets of the original ontology to limit the search space for the initial exploration. The KnowItAll Ontology is represented as plain lists, thus, we created semantic annotations between the instances and the classes *i.e.*, $instance\text{-}of$, and then constructed the sub-ontologies for the relations we were interested in testing. The sub-ontologies consist of sets of axioms and conjectures of the form: `fof(axiom_name, axiom, conjecture).` The general form of a conjecture describing a relation Rel into a human-readable first-order syntax between an instance i of class $class_i$ and an instance y of class $class_y$, is:

$$\exists i, y(instance\text{-}of(i, class_i) \wedge instance\text{-}of(y, class_y) \wedge Rel(i, y))$$

Finding Inconsistencies. Resolution based first-order theorem provers aim at proving a goal by showing that its negation leads to contradictions within the axiom set. However, lack of negation in the axiom set can block that proof. So, instead of constructing goals to prove the ontology consistent, we constructed goals to prove the ontology inconsistent. If these goals are satisfiable, which means that the goals follow from the axioms, we automatically prove the existence of such inconsistencies. For example, the following conjecture describes the partial functionality of a relation Rel:

$$\begin{aligned}\forall i, x, y(&instance\text{-}of(i, Domain(Rel)) \\ &\wedge instance\text{-}of(x, Range(Rel)) \\ &\wedge instance\text{-}of(y, Range(Rel)) \\ &\wedge Rel(i, x) \wedge Rel(i, y) \wedge x = y)\end{aligned}$$

The meaning of this conjecture is the following. Given i there exists a unique x such that $Rel(i, x)$. Instead of constructing this conjecture, we constructed its negation: given i there exists a non unique x such that $Rel(i, x)$. The conjecture in first-order format is:

$$\begin{aligned}\exists i, x, y(&instance\text{-}of(i, Domain(Rel)) \\ &\wedge instance\text{-}of(x, Range(Rel)) \\ &\wedge instance\text{-}of(y, Range(Rel)) \\ &\wedge Rel(i, x) \wedge Rel(i, y) \wedge x \neq y)\end{aligned}$$

To prove a goal satisfiable, the E prover applies that goal on all given axioms. If a proof is found it returns as answers all the instances that satisfy that proof.

III. Experiments and results

The experiments we conducted are concerned with partial functionality violation and type clash. First, we tested the $be\text{-}capital\text{-}of$ and $be\text{-}the\text{-}capital\text{-}of$ relations for partial functionality violations. Each sub-ontology consists of axioms that describe these relations, the corresponding instances and their classes. In total we tested $1,177$ axioms for the $be\text{-}the\text{-}capital\text{-}of$ sub-ontology, and $1,053$ axioms for the $be\text{-}capital\text{-}of$ sub-ontology. KnowItAll does not consider these relations as synonyms because they do not match syntactically. So, these two sub-ontologies overlap but are not exactly the same. For the type clashes experiment, we are only concerned with the $instance\text{-}of$ axioms. We tested a randomly generated set of $city$, $leader$, $member$, $club$, $tree$, $product$, $nutrition$, $medication$, man and $fruit$ instances, for a total of $5,788$ axioms.

TABLE I
FUNCTIONALITY VIOLATION FOR THE *be-the-capital-of* RELATION: SAMPLE INSTANCES THAT CAUSE FUNCTIONALITY INCONSISTENCIES.

$City_x$	$City_y$	$Country_i$
bonn	*berlin*	*germany*
tokyo	*paris*	*japan*
hong-kong	*bejiing*	*china*
berne	*bern*	*switzerland*
prague	*rome*	*europe*

TABLE II
FUNCTIONALITY VIOLATION FOR THE *be-the-capital-of* RELATION: SAMPLE INSTANCES THAT CAUSE FUNCTIONALITY INCONSISTENCIES.

$Country_x$	$Country_y$	$City_i$
siam	*thailand*	*bangkok*
prc	*china*	*bejiing*
assyria	*babylonia*	*nineveh*
jamaica	*canada*	*kingston*
sindh	*pakistan*	*karachi*

TABLE III
FUNCTIONALITY VIOLATION FOR *be-capital-of relation*: INSTANCES THAT CAUSE FUNCTIONALITY INCONSISTENCIES.

$City_x$	$City_y$	$Country_i$
canberra	*melbourne*	*australia*
alquds	*jerusalem*	*palestine*
tokyo	*nara*	*japan*
brindisi	*rome*	*italy*
sindh	*pakistan*	*karachi*

A. Functionality Violation Results

Relation $be\text{-}the\text{-}capital\text{-}of(City,\ Country)$ is considered as partially functional in both arguments: a country can have only one capital city; a city can be capital for only one country. Our goal is to prove that there exists in the ontology a country with more than one capital city, and *vice versa*.[3] To prove these inconsistencies we constructed the goals 1 and 2 (described in first-order syntax).

$$\begin{aligned} &\exists City_x, City_y, Country_i \\ &(instance\text{-}of(City_x, city) \\ &\wedge instance\text{-}of(City_y, city) \\ &\wedge instance(Country_i, country) \\ &\wedge be\text{-}the\text{-}capital\text{-}of(City_x, Country_i) \\ &\wedge be\text{-}the\text{-}capital\text{-}of(City_y, Country_i) \\ &\wedge City_x \neq City_y) \end{aligned} \quad (1)$$

$$\begin{aligned} &\exists Country_x, Country_y, City_i \\ &(instance\text{-}of(Country_x, country) \\ &\wedge instance\text{-}of(Country_y, country) \\ &\wedge instance\text{-}of(City_i, city) \\ &\wedge be\text{-}the\text{-}capital\text{-}of(City_i, Country_x) \\ &\wedge be\text{-}the\text{-}capital\text{-}of(City_i, Country_y) \\ &\wedge Country_x \neq Country_y) \end{aligned} \quad (2)$$

The meaning of goal 1 is the following. There exists a country $Country_i$ and two different cities $City_x$ and $City_y$ such that both cities are related to that country with the $be\text{-}the\text{-}capital\text{-}of$ relation.

E found goal 1 provable and returned 134 unique answers. Some of the answers are shown in Table I. Goal 2 was formulated to prove that there exists a city $City_i$ which is capital for two different countries: $Country_x$ and $Country_y$.

E found goal 2 provable and returned 25 unique answers. A sample of these answers is presented in Table II.

We conducted the same experiments for the $be\text{-}capital\text{-}of$ relation. We found inconsistencies regarding only the uniqueness of the city instances. The instances that cause these violations are presented in Table III.

[3]Although in reality there exist countries with multiple capital cities, *e.g.*, South Africa, we are only concerned with the definition of functionality within the ontology.

B. Type Clashes Results

Type clashes refer to multiple typed instances. We are concerned with type clashes because in common ontologies with class hierarchies, an instance inherits multiple types from its taxonomical parents. So, it is unnecessary to define multiple times the types that an instance inherits. Also, type clashes can indicate cases that disjoint instances share the same types because of a similar representation. To test the existence of type clashes we constructed the following goal:

$$\exists i, x, y(instance\text{-}of(y, i) \wedge instance\text{-}of(y, x) \wedge i \neq x)$$

The goal describes that there exists an instance that belongs to two different classes. E found the goal satisfiable and returned the instances and the classes that satisfy that goal. Some of those instances are presented in Table IV.

TABLE IV
TYPE CLASHES INCONSISTENCY: RESULTS.

$Class_i$	$Class_x$	$Instance_y$
city	*fruit*	*minneola*
product	*medication*	*meprobamate*
leader	*man*	*barack-obama*
man	*city*	*obama*

IV. FUTURE WORK: AUTOMATION OF ANALYSIS AND REPAIR

Another aspect of our work is to address the feasibility of automatically repairing the ontology, through identifying errors and appropriate repairs. To automate the repair process, we need appropriate feedback on the nature of an error. Such feedback can be provided through error classification. We present a manual analysis of the errors we identified as an initial feasibility study into further work. We believe that, in the scope of this work, it is possible to apply existing repair techniques. We describe how the ontology repair techniques

presented in [14], [15] could be applied. Also, we present a methodology for identifying the semantically correct data, and we suggest a strategy towards automatically repairing the KnowItAll Ontology. This section corresponds to the identification of the appropriate methodology towards repairing the KnowItAll Ontology; the presented techniques have not yet been implemented.

A. Error Classification

We manually analysed and classified the errors to understand their nature. The results of this analysis can provide information on whether this procedure can be automated or not. For instance, we investigate whether the errors are general enough so we could apply *e.g.*, natural language techniques to automate the analysis, or are they very specific so that more sophisticated feedback is needed *e.g.*, user feedback. Another alternative would be to use standard feature-based classification techniques, *e.g.*, from machine learning.

1) Functionality Violation Results Analysis: We classify the causes of these inconsistencies into three major categories: (a) multiple representation of the same instance, (b) different instances share common characteristics, and (c) semantically wrong extractions.

Multiple Representations of the Same Instance. Among the different representations either only one of the representations is correct, or more than one representation is correct. When more than one representation is correct, we identify the following categories: (i) Old name and New name: a place which has been renamed with the passing of time, *e.g.*, Thailand's former name is Siam. (ii) Official and Common name: a place whose official name is different from its common name, *e.g.*, People's Republic of China and China. (iii) Official name and Abbreviation: a place which is known both with an official name and with the corresponding abbreviation, *e.g.*, People's Republic of China and PRC. (iv) More than one official name: a place with more than one official name, *e.g.*, Myanmar and Burma. (v) English and Non English name: a name represented both in English and in another language, *e.g.*, Moldavia and Moldova. (vi) Official and Colloquial name: a place whose official name differs from the name people use, *e.g.*, Thessaloniki, and Thessalonika.

We believe that any representation that falls into one of the above categories should be kept in the ontology. To avoid current and future inconsistencies, one of the representations must be thought of as the main one and the rest should be repaired according to their relation with that representation.

Different Instances that Share Common Characteristics. This category contains instances that are misinterpreted as the same because of a relation between them; *e.g.*, a part-of relation: Sindh is a province of Pakistan, however, they are treated as the same instance. Below we provide a classification of the causes of this misinterpretation.

Different Time Period. The KnowItAll system normalises the extracted data from the web, and then adds it to the ontology *i.e.*, extractions of the form *was the capital of* are converted to *be the capital of*. The normalisation process can cause functionality violations. Such violations are caused when: (i) A country has changed its capital city over the years, *e.g.*, Nara and Tokyo. (ii) An empire or a country that does not currently exist, has the same capital as a current country, *e.g.*, Roman Empire and Italy.

Part of a Country. Places that are parts of a country, for instance provinces, are specified as countries, *e.g.*, Sindh and Pakistan.

Same Relation With Different Meanings. The relations $be\text{-}the\text{-}capital\text{-}of$ and $be\text{-}capital\text{-}of$ can have a metaphorical meaning, in addition to the literal meaning *e.g.*, cultural capital, financial capital *etc*. However, both meanings are represented by the same relations, resulting in inconsistencies. For example, Islamabad is the capital of Pakistan, while Lahore is the country's cultural capital.

Different instances with similar names. A common mistake is to use the same representation for geographical places that are not the same. Such cases are: (i) Refer to a region of a country by using the name of that country, *e.g.*, calling the European part of Russia, Russia. (ii) Refer to two different countries using the same name, *e.g.*, calling Northern Ireland, Ireland.

Multiple Instances Specified as a Single. In the KnowItAll Ontology connective words, *e.g.*, 'and', have not been taken into account. The result is multiple definitions of countries, both individually and as pairs. For example, within the ontology co-exist the relations: $instance\text{-}of(scotland, country)$, $instance\text{-}of(sweden, country)$, $instance\text{-}of(scotland\text{-}and\text{-}\ sweden, country)$.

Semantically Wrong Extractions. The KnowItAll Ontology contains extractions that represent semantically wrong relations. We categorise such extractions into:(i) Completely wrong relations, *e.g.*, $be\text{-}the\text{-}capital\text{-}of(miami, cuba)$, $be\text{-}the\text{-}capital\text{-}of(paris, japan)$.[4] (ii) A continent defined as a country *e.g.*, $be\text{-}the\text{-}capital\text{-}of(brussels, europe)$, $be\text{-}the\text{-}capital\text{-}of(prague, europe)$. (iii) A block of neighbouring countries defined as a single country *e.g.*, Scandinavia is defined as a country. (iv) Extractions that result from bad human inference. For example, Glasgow is defined as the capital of Scotland which might have resulted from a misleading inference that the city with the highest population, or perhaps the most famous city, is also the capital city.

2) Type Clashes Results Analysis: Most type clashes result from the lack of class hierarchies. For example, $barack\text{-}obama$ instance is multiply defined: as a $leader$ and as a man. This entails lack of type inheritance. Type clashes also occur because of synonymy between disjoint instances. As a result the same instance belongs to two or more disjoint classes. For example, consider the relations $instance\text{-}of(obama, city)$[5], $instance\text{-}of(obama, man)$; two $obama$ instances with dis-

[4]Such relations usually come from unreliable sources *e.g.*, Paris is defined as the capital of Japan, as an example of false assertions in a website that describes Logic.

[5]Obama is a city located in Fukui Prefecture, Japan.

joint meanings.

3) Error Classification: Further Experiments: We conducted further experiments regarding the functionality violation of other relations that are considered partially functional in the KnowItAll Ontology. These relations are: $have\text{-}a\text{-}campus\text{-}in(University, City)$, $be\text{-}a\text{-}freelance\text{-}writer\text{-}in(Writer, City)$, $be\text{-}the\text{-}birthstone\text{-}for(Stone, Month)$. In case of inconsistencies, we analysed the errors that caused them, and checked whether they fall into the classification we previously presented.

We found 102 functionality violations of the $be\text{-}a\text{-}freelance\text{-}writer\text{-}in/2$, 74 functionality violations of the $have\text{-}a\text{-}campus\text{-}in/2$, and 10 of the $be\text{-}the\text{-}birthst\text{-}one\text{-}for/2$ relations.[6] All the errors that cause the inconsistencies, fall into the classification we previously presented. These results are encouraging towards the generality of our classification. However, further experiments need to be conducted and more categories need to be added, for a more concrete classification.

B. Suggested Repairs

1) Class Hierarchies: The first step towards repairing the KnowItAll Ontology is to construct class hierarchies. The KnowItAll Ontology currently lacks specification of taxonomic parents, resulting in ambiguity, repetition and inconsistencies. Consider the relations $instance\text{-}of(meprobamate, medication)$ and $instance\text{-}of(meprobamate, product)$, found within the KnowItAll Ontology. By creating a class hierarchy $medication \subset product$,we can infer that $meprobamate$ is an instance of $product$ by defining it as a $medication$. The next step is to fix semantically wrong types of instances, for example, to change the type of $europe$ from $country$ to $continent$.

2) Abstraction and Refinement Techniques: The importance of ontology repair and a set of possible repairs were firstly introduced in [14], [15] and implemented in various systems *e.g.*, [15], [16]. The ontology repair techniques that the authors suggest fall under one of two major categories: abstraction repairs and refinement repairs. Abstraction repairs reduce the detail of the ontology, leading to a more general form *e.g.*, $european\text{-}country \mapsto country$. Refinement repairs add detail to the ontology leading to more specific representations, *e.g.*, $european\text{-}country \mapsto west\text{-}european\text{-}country$. Most of the inconsistencies in the KnowItAll Ontology result from underspecification and generality of the defined concepts. Thus, we consider refinement repairs as the most appropriate, from which we identify splitting arguments, splitting predicates and adding extra arguments to the relations (note that the language of the KnowItAll Ontology is restricted to triples, so this is not possible unless the language is extended), to fit best with the problem we face. Different repairs are required depending on the nature of the inconsistency. Thus, with the appropriate reasoning the most suitable repair should be selected. Also, we consider abstraction techniques, such as merging relations, to deal with synonymy.

Splitting Arguments. We split an argument into two or more arguments of the same type. The resulting arguments aim to provide precision to the relation, by capturing more detail than the original one *e.g.*, the time period of this argument. For example, consider the following inconsistency: $be\text{-}the\text{-}capital\text{-}of(nara, japan)$, $be\text{-}the\text{-}capital\text{-}of(tokyo, japan)$. The first relation refers to ancient Japan while the second relation refers to current Japan. One appropriate repair is to split $japan$ into $current\_japan$ and $past\_japan$. By replacing the above relations with these two instances, the inconsistency is fixed, while the relation is more specific regarding the time period it refers to. Then, we need to apply the same technique for all such occurrences within the ontology. However, if the repair is based on domain-specific information extracted from some particular relation, then applying it on all occurrences within the ontology might result in semantic inconsistencies. For instance, consider a $country_i$ that has both a cultural and a financial capital. Then, we could split this $country_i$ into $cultural\_country_i$ and $financial\_country_i$. But splitting all occurrences of $country_i$ in that way is semantically inconsistent. Thus, this technique should be applied for general errors, or instead of repairing all such occurrences within the ontology, provide the appropriate mappings.

Splitting Relations. This technique is similar to splitting arguments: we split a relation into two or more specific relations. The resulting relations capture in more detail the relationship between the arguments. For example, split $be\text{-}the\text{-}capital\text{-}of$ into $be\text{-}the\text{-}current\text{-}capital\text{-}of$ and into $be\text{-}the\text{-}past\text{-}capital\text{-}of$. In this way we consider functional only the relation that refers to current situations *i.e.*, $be\text{-}the\text{-}current\text{-}capital\text{-}of$. This approach is suitable for most of the errors we previously presented, for example, split $be\text{-}the\text{-}capital\text{-}of$ into $be\text{-}the\text{-}cultural\text{-}capital\text{-}of$, $be\text{-}the\text{-}economical\text{-}capital\text{-}of$ *etc*. This repair does not affect the whole Ontology as we only repair a specific relation.

Adding extra arguments. We convert an inconsistent binary relation to a ternary by introducing a new argument. That argument specifies the exact semantics of the relation and results from the error classification. Also, it is that third argument that will specify whether a relation is functional or not. For that reason, appropriate rules that specify the conditions under which a relation is functional should also be constructed. In this way, we maintain the ontology's consistency while we do not change any of the original information. For example, we can add an argument that indicates time period:
$be\text{-}the\text{-}capital\text{-}of(tokyo, japan, current)$,
$be\text{-}the\text{-}capital\text{-}of(otsu, japan, past)$,
$be\text{-}the\text{-}capital\text{-}of(kamakura, japan, past)$,
$be\text{-}the\text{-}capital\text{-}of(saitama, japan, past)$,
$be\text{-}the\text{-}capital\text{-}of(kyoto, japan, past)$.

[6]These functionality violations raise the issue of not considering these relations as functional. For example, it is not unusual for a University to have more than one campus *e.g.*, Heriot-Watt University has 4 campuses: three in the UK and one in Dubai.

This repair is suitable for all kind of errors we previously presented; add an argument that indicates time, geographical location, culture *etc*. The degree by detail we add, depends on the available information. For example, if we knew the exact chronological period for each Japan's capital city, then instead of adding $past$ in the relation, we could add the exact time period *e.g.*, 710-784 AC.

Merging relations. We merge two or more relations into one, when these relations are considered as synonyms. For example, consider the $be\text{-}the\text{-}capital\text{-}of/2$ and $be\text{-}capital\text{-}of/2$ relations. They share the same semantics but they do not connect the same sets of instances: not all the instances that connect by the first relation, also connect by the second relation (and *vice versa*). In this way we have *hidden* inconsistencies, for instance, a country with two capital cities; each capital city is indicated with a different relation. This problem can be solved by merging these relations into one. Then, if new inconsistencies occur, we can resolve them by applying one of the above repairs.

C. Identifying the Correct Information

Repairing the ontology requires that we have enough information to locate the inconsistencies within the ontology, identify the cause of the inconsistency, distinguish the correct occurrence from the incorrect ones and find the corresponding repair. The degree of which we can repair the ontology depends on the available information.

The KnowItAll Ontology comes with the original extractions (the extractions before the normalisation process) and the number of *independent* sentences each extraction comes from. The number of *independent* sentences depends on how many times a relation has been extracted. If a relation with the same arguments appears more than once within a text, it is only extracted once. The same applies if a relation appears into related texts, for example aggregator pages. We believe that the number of the independent sentences each extraction comes from can indicate, in most cases, how correct that instance is likely to be: as the sources vary, it is rare to extract the same error from multiple sources.

To evaluate this idea, for each functionality violation of $be\text{-}the\text{-}capital\text{-}of$ and $be\text{-}capital\text{-}of$ we previously detected, we kept only the occurrence of the relation that came from the highest number of independent sentences, within a set S. After that, we manually checked whether S is semantically correct and found that 95% of the $be\text{-}the\text{-}capital\text{-}of$ and 92% of the $be\text{-}capital\text{-}of$ instances were semantically correct. This technique succeeds in identifying the correct data only when it comes from multiple independent sentences.

The un-normalised extraction of each relation can indicate the semantic meaning of the relation and can also indicate the appropriate repair. We have observed that, in many cases, the meaning of an extraction changes entirely after the normalisation process, leading to loss of information and ambiguity. For example, $was\text{-}the\text{-}capital\text{-}of(nara, japan)$, and $is\text{-}the\text{-}capital\text{-}of(tokyo, japan)$, which represent different semantics, after the normalisation process become the same relation: $be\text{-}the\text{-}capital\text{-}of(nara, japan)$, $be\text{-}the\text{-}capital\text{-}of(tokyo, japan)$. In 100% of the sub-ontologies we investigated, the un-normalised extractions of the $be\text{-}the\text{-}capital\text{-}of$ and $be\text{-}capital\text{-}of$ relations, indicate the time period (past - current) the relations refer to. These extractions can provide a metric on whether a relation within the ontology represents the correct semantic meaning. For instance, within the $be\text{-}capital\text{-}of$ inconsistent set, this metric distinguished 75% of the correct relations: the correct ones, in the un-normalised form were presented as $is\text{-}capital\text{-}of$, while the rest were presented by the relation $was\text{-}capital\text{-}of$. Using these metric, we were unable to identify 25% of the correct relations relations because they were all presented in the present tense, *e.g.*, $is\text{-}capital\text{-}of(kingston, jamaica)$, $is\text{-}capital\text{-}of(kingston, canada)$.

D. Repair Strategy

In this Section we present a potential strategy for repairing the ontology. The first step we identify is creating a consistent subset $\mathcal{M}^-$ from the original ontology $\mathcal{M}$, $\mathcal{M}^- \subset \mathcal{M}$. Then we continuously enrich it by repairing the inconsistencies occurring in $\mathcal{M}$ and adding them to $\mathcal{M}^-$.

Using a combination of the techniques presented in Section IV-C as heuristics for distinguishing the correct extractions, we can identify $\mathcal{M}^-$. Considering as the correct relation the one that comes from the highest number of independent sentences, can be such a heuristic. We posit that we can approximate the quality of an extraction by its number of independent sentences: an extraction of high quality will be supported by a high number of independent sentences. Also, we can set a threshold difference k between the number of independent sentences of the semantically correct extractions, and the inconsistent ones. This can guide us in identifying extractions that we are not interested in repairing *i.e.*, completely semantically wrong extractions. This is better explained through an example illustrated in Table V. Consider the $be\text{-}the\text{-}capital\text{-}of$ sub-ontology $\mathcal{M}$, and an empty set S. For each inconsistency, we consider as correct the one that comes from the maximum independent sentences. We set the threshold difference k to be equal to a support of 60% of independent sentences for the correct occurrence. If a relation meets the threshold we add it to S; else we will discard it. According to Table V, $be\text{-}the\text{-}capital\text{-}of(tokyo, japan)$ has the maximum number of independent sentences, equal to 12; we consider it as the correct one and we keep it to $\mathcal{M}$. Also, $k = \frac{60}{100}12 \simeq 7$. So, we discard $be\text{-}the\text{-}capital\text{-}of(paris, japan)$ because it comes from less than 7 independent sentences, and we add $be\text{-}the\text{-}capital(kyoto, japan)$ to S. The output of this procedure is an ontology $\mathcal{M}^- = \mathcal{M} - \{be\text{-}the\text{-}capital\text{-}of(paris, japan), be\text{-}the\text{-}capital\text{-}of(kyoto, japan)\}$, and $S = \{be\text{-}the\text{-}capital(kyoto, japan)\}$. We then select $be\text{-}the\text{-}capital(kyoto, japan)$ from S, check the un-normalised extractions, which indicate past time period, and repair it according to the repair techniques we identified in Section IV-B2. We repeat this procedure until $S = \{\emptyset\}$.

TABLE V
JAPAN CAPITAL CITIES AND THE CORRESPONDING INDEPENDENT SENTENCES.

relation	city	country	independent sentences
be-the-capital-of	*tokyo*	*japan*	12
be-the-capital-of	*kyoto*	*japan*	7
be-the-capital-of	*paris*	*japan*	1

V. DISCUSSION

In this report we have presented our strategy and our results on automatically detecting inconsistencies in large scale ontologies. We demonstrated our techniques on using a first-order theorem prover to identify inconsistencies within an ontology that lacks negation. Our strategy is to firstly identify small sub-ontologies and then translate them into a theorem-prover appropriate input language. Then, construct goals for the prover, that describe the inconsistencies; if the goals are provable, then the inconsistencies exist within the ontology.

Using query languages *e.g.*, SQL, SPARQL to retrieve the inconsistent data is another possible approach. SQL would require to transform the ontology into a relational representation. This implies either to use a relational table per predicate, which would result either in a vast number of small tables, or a denormalised representation that would use relational attribute values to denote predicate types. Both are cumbersome solutions and result in complicated queries to test the required goals either in terms of the number of tables in an SQL from-clause, or in terms of the number of extra predicates inlined in the SQL where-clause. A theorem prover is a more elegant solution in that respect. SPARQL could also be appropriate for the problem we dealt with. This technique could provide an insight into the inconsistencies, however, it is restricted to binary relations where first-order logic can handle relations of arbitrary arity. In the work we presented we tested only binary relations, however richly axiomatised foundational ontologies such as SUMO[7] express the requirement of being able to handle relations with arity greater than 2. We therefore chose to use a more flexible approach with extendibility in mind. Another approach would be to use a Description Logic (DL) reasoner, after translating the ontology into the appropriate format. However, DL does not use the Unique Name Assumption (UNA)—that is, treating different symbols as different objects. The problem that we are dealing with requires reasoning with equality, so, UNA is needed. For example, consider a binary relation R which is considered functional, and the following facts in the ontology: $R(a, b)$, $R(a, c)$. Without UNA these relations satisfy the functionality requirement with $b = c$. Finally, we chose to translate the KnowItAll ontology into first-order logic as it is the most expressive format for our repairs. For example, it would not be possible to express the *adding arguments* repair into DL, as all predicates have a fixed arity of two. Another problem is that DL does not allow mappings between instances. One of our future directions is to deal with synonymy, and First-Order logic allows us to define equalities between instances. For instance, in first-order logic we can define China and PRC to be equal and thus, resolve the inconsistencies by recognising that the two different names refer to the same object

We have reported the results of this strategy, and we have analysed the errors that cause these inconsistencies. We believe that the nature of these errors can indicate the appropriate repairs. Also, we identified and presented the repairs that fix and enrich the original ontology. Finally, we presented the repair strategy that can lead to successfully automatically repairing the ontology. Such strategy is not only suitable for resolving inconsistencies at once, but for dealing with ontology evolution in general. As ontologies change to adapt to the world, new inconsistencies may be introduced. This strategy can deal with this problem during the life cycle of an ontology.

The aim is for our repair techniques to be automated. Only a small part of the error classification we presented is currently done automatically; that part deals with temporal classification *i.e.*, past-present-future. The detailed categorisation is the outcome of manual work. However, the repairs we present are general enough to be adapted to different domains. It is therefore conceivable that, in a fully automated system that uses our techniques, the repairs would be expressed at a general level. Thus, one of our future directions is to fully automate error classification so that repairs can then add or remove the desired degree of detail.

[7]http://suo.ieee.org/

REFERENCES

[1] D. Lenat, "CYC: a large-scale investment in knowledge infrastructure," *Commun. ACM*, vol. 38, no. 11, pp. 33–38, November 1995.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "Owl Web Ontology Language 1.0 (r)eference," *W3C, Recommendation*, 2004.

[3] D. Berardi, M. Gruninger, R. Hull, and S. McIlraith, "Towards a First-Order Ontology for Semantic web Services," *http://www.w3.org/2004/08/ws-cc/mci-20040904*, 2004.

[4] I. Horrocks and P. F. Patel-Schneider, "A proposal for an owl rules language," in *Proceedings of the 13th international conference on World Wide Web*, ser. WWW '04, no. 9, 2004, pp. 723–731.

[5] D. Ramach, R. P. Reagan, and K. Goolsbey, "First-Orderized Research-Cyc: Expressivity and efficiency in a common-sense ontology," *In Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications*, 2005.

[6] I. Horrocks and A. Voronkov, "Reasoning Support for Expressive Ontology Languages Using a Theorem Prover," in *FoIKS*, 2006, pp. 201–218.

[7] I. Niles and A. Pease, "Towards a Standard Upper Ontology," in *Proceedings of the International Conference on Formal Ontology in Information Systems*, ser. FOIS '01. ACM, 2001, pp. 2–9.

[8] S. Schulz, "E - a brainiac theorem prover," *AI Commun.*, vol. 15, pp. 111–126, August 2002.

[9] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and Mausam, "Open Information Extraction: The Second Generation," in *IJCAI*, 2011, pp. 3–10.

[10] S. Stefan, O. Etzioni, D. S. Weld, and J. Davis, "Learning first-order Horn clauses from web text," *EMNLP 2010*, vol. 11, pp. 1088–1098, 2010.

[11] T. Lin, Mausam, and O. Etzioni, "Identifying Functional Relations in Web Text," in *EMNLP*, 2010, pp. 1266–1276.

[12] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0," *Journal of Automated Reasoning*, vol. 43, no. 4, pp. 337–362, 2009.

[13] F. J. Pelletier, G. Sutcliffe, and C. B. Suttner, "The development of CASC," *AI Commun.*, vol. 15, no. 2-3, pp. 79–90, 2002.

[14] A. Bundy and F. McNeill, "Representation as a Fluent: An AI Challenge for the Next Half Century," *IEEE Intelligent Systems*, vol. 21, pp. 85–87, 2006.

[15] F. McNeill and A. Bundy, "Dynamic, Automatic, First-order Ontology Repair by Diagnosis of Failed Plan Execution," *In IJSWIS, Special Issue on Ontology Matching*, vol. 3, pp. 1–35, 2007.

[16] M. Chan, J. Lehmann, and A. Bundy, "GALILEO: A System for Automating Ontology Evolution," in *In Notes of the IJCAI-11 Workshop on Automated Reasoning about Context and Ontology Evolution*, 2011.