International Journal of Advanced Computer Technology (IJACT)

ISSN:2319-7900

LOGIC-BASED CONSISTENCY CHECKING OF XRBL INSTANCES

Gianfranco D'Atri, University of Calabria, Italy

Abstract

The world leading standard for business reporting is XBRL, which stands for eXtensible Business Reporting Language. XBRL defines XML elements and attributes that can be used to encode business reports in a non-ambiguous way. Nonetheless, XBRL provides only basic validation capabilities. To face this issue, an approach to validating the semantic correctness of financial reports written in XBRL is proposed. The idea is to represent the data present in XBRL reports by means of the logic-based formalism OntoDLP, which roots its semantics in the well–known paradigm of Answer set Programming. The expressive power of OntoDLP then used to model both simple and complex numerical validations on XBRL instances. The availability of efficient engines for evaluating OntoDLP programs enables the definition of a system architecture, where the all the methods proposed and discussed can be concretely implemented to support sophisticated forms of reasoning over XBRL documents.

Introduction

The eXtensible Business Reporting Language (short: XBRL) is the world leading standard for business reporting. From the syntactical viewpoint, it belongs to the family of XML-based languages. From the semantic viewpoint, data in XBRL is reported according to a number of different *conceptual* dimensions, and a domain *ontology* is concretely made available in the specification as a reference metamodel [3]. In principle, the ontological annotations of the XBRL data might be used to support advanced forms of reasoning over the financial reports, such as in particular to check the correctness and the consistency of the results of arithmetic formulas involving financial variables. However, XBRL lacks of these advanced features and only very limited forms of semantic validation are available.

Moving from the above observation, the paper investigates the possibility of supporting validation of XBRL reports by exploiting well-known reasoning methods and systems that have been developed to deal with enterprise/corporate ontologies, though not specifically XBRL. In fact, the use of ontologies to conceptualize business enterprise information has attracted much research in recent years. In particular, it has been observed that, in this context, standard ontology-based mechanism based on the *open world assumption* might be not appropriate, so that specific onology languages founding on the *closed world assumption* have been developed. Accordingly, we will hereinafter focus on the OntoDLP language, which roots its semantics in the Answer Set Programming paradigm (and hence on the closed world assumption) [5], and which is supported in a powerful environment named OntoDLV providing a user-friendly visual environment and a robust persistency-layer.

OntoDLP supports all major ontology features including classes, inheritance, relations and axioms. OntoDLP strongly typed, and includes also complex type constructors, like lists and sets. Moreover, OntoDLV a powerful interoperability mechanism with OWL, allowing the user to retrieve information from OWL ontologies, and build rule-based reasoning on top of OWL ontologies. The system is already used in a number of real-world applications including agent-based systems, information extraction, and text classification. Therefore, OntoDLP is a rather solid candidate to constitute the basis of a system supporting automatic validation of XBRL data in concrete application domains. This topic is addressed in the rest of the paper.

In particular, in order to end up with a self-contained discussion, the next section will take a closer look at the main features of OntoDLP, then we will focus on the salient ingredients of the XBRL specifications. These two separate worlds will be eventually merged together in Section, where an approach to encode XBRL instances (and validation constraints defined over them) in terms of OntoDLP specification is presented. An appealing feature of the encoding is that the resulting OntoDLP specification can be fed to the OntoDLV system to check consistency of XBRL instances. This paves the way for the definition of a concrete system architecture to reason about XBRL instances on top of the OntoDLV system, which is illustrated in Section. Finally, conclusions and a few remarks on interesting avenues for further research are drawn in Section.

The Onto DLP language

In this section we overview OntoDLP, an ontology representation and reasoning language based on Disjunctive Logic Programming (DLP). We assume the reader familiar with DLP syntax and semantics [6], a nice introduction to DLP can be found in [5, 4]. In the following OntoDLP presented by means of a running example inspired by the one presented in [8]. The description is limited to constructs that will be used in the remainder of this paper. An in-depth description

ISSN:2319-7900

of OntoDLP be found in [8, 9].

Classes. The main construct in OntoDLP the (base) *class*. A base class models a collection of individuals who belong together because they share some properties. Classes are defined by using the keyword `class` followed by its name. Class attributes modeling instance properties are typed. They are specified by means of pairs `attribute-name:attribute-type`, where `attribute-name` is the name of the property and `attribute-type` is the class the attribute belongs to. Attributes can also take values from the built-in classes `string` and `integer` (respectively representing the class of all alphanumeric strings and the class of non-negative integers). As an example the following class declarations model basic concepts of a (toy) banking ontology:

```
class bank(name:string, asset:integer).
class account(balance:integer).
class branch(bank:bank, location:place,
asset:integer).
class place(name:string).
class enterprise(name:string,
country:place).
class person(name:string, age:integer,
father:person, mother:person,
residence:place).
```

The above statements model that banks have a name and own an asset; the branches of a given bank are located into a given place and also have an asset; accounts have a balance; enterprises have a name and a country (which is a place); persons have name, age, residence (which is also a place), father and mother (which are other persons); and finally, each place has a name. Note that class definitions can be recursive (see attribute father of class person).

Individuals. Class instances, called *objects*, model the individuals of a domain. Objects are uniquely identified by a constant called the *object identifier* (oid), and are declared by asserting logic facts. For example, the following statements

```
rome:place(name:"Rome").
john:person(name:"John", age:34,
father:jack, mother:ann,
residence:rome).
```

.assert that `rome` and `john` are instances of the class `place` and `person`, respectively. The oid `rome` identifies a place named "Rome" that fills residence attributes. Thus `john` lives in Rome, and `jack` and `ann` are `father`, `mother` of `john`, respectively.

Referential integrity and correctness of types is mandatory to write well-formed instances. e.g., `rome` has to be declared as a place identifier to properly fill residence attribute.

Associations among objects. Relations define associations among objects. They are declared like classes, where the keyword relation (instead of class) precedes a list of attributes. As an example, we model a relationship between persons and their bank account as follows:

```
relation customerHoldsAccount(
customer:person,
account:account).
```

The instances of a relation are called *tuples*, and are declared by logic facts. For instance, to model that account `acc001` is held by John, which holds account `acc012` with Ann we write:

```
customerHoldsAccount(customer:john,
account:acc001).
customerHoldsAccount(customer:ann,
account:acc012).
customerHoldsAccount(customer:john,
account:acc012).
```

Logic Programs. In addition to the ontology specification, logic programs can be written to declaratively specify properties and reason on ontological data. Logic programs are sets of logic rules. Logic rules are written according to the well-known Prolog conventions, where variables begin with uppercase letter, and terms by start by lowercase letter. The implication symbol `:-` can be intuitively read as "if". Rules also may also feature disjunction `v`, negation as failure `not`, and aggregation functions (see [1] for more details). The rules can access the information present in the ontology. The programmer can introduce a number of *auxiliary* predicates (as `paymentsNumber`) which do not require an explicit schema definition.

Rules can be collected in *reasoning modules*. *reasoning modules* are the language components Reasoning modules are identified by a name and are defined by a set of (possibly disjunctive) logic rules. Syntactically, the keyword *module* precedes the name which is followed by a logic program enclosed in curly brackets. As an example consider the following module, which computes the number of payments (withdrawals + deposits) performed on a bank account:

```
module computePaymentsNumber {
paymentsNumber(A, PayN) :-
A:account(),
#count{ W:withdrawals(A,W) }=Wnum,
#count{ D:deposits(A,D) }=Dnum,
PayN=Wnum+Dnum.
}
```

The logic rule can be read as follows: the payment number `PayN` associated to account `A` is computed by summing the number of deposits `Dnum` with the number of withdrawals `Wnum` associated to `A`. Note that the aggregate

ISSN:2319-7900

`#count` is used to count withdrawals and deposits associated to a given account.

Intensional constructs. The notions of class and relation introduced above correspond, from a database point of view, to the *extensional* part of the OntoDLP language. Classes and relations can also be defined intensionally (as views in databases), in the sense that objects of a class can be "derived" (or inferred) from the information already stated in an ontology. This is obtained by means of logic rules. As an example with the following statements

```
class richPerson(name:string).
P:richPerson(name:N)
:- P:person(name:N),
   A:account(balance:B),
   holdsSavingsAccount(customer:P,
     account:A), B > 1000000.
```

class `richPerson` collects (or re-classify) instances of `person`, which are inferred by using a logic rule asserting that a person `P` is rich if he holds a savings account `A` with a balance `B` of more than one million. Intensional classes are called *collection* classes in OntoDLP.

Importantly, the logic programs (set of rules) defining collection classes must be normal and stratified (see eg., [2, 7]).

Intensional relations are defined analogously. For example, the binary relation relative (modeling the common ancestry among persons) is defined as follows:

```
intensional relation
relative (sub:person, obj:person).
relative(sub:X,obj:Y) :-
X:person(father:Y).
relative(sub:X,obj:Y) :-
X:person(mother:Y).
relative(sub:X,obj:Y) :-
relative(sub:X,obj:Z),
relative(sub:Z,obj:Y).
```

The above statements can be read as follows: *X* is relative of *Y* if *X* is parent of *Y* (by the first two rules), and *X* is a relative of *Y* if exists a third relative *Z* of *X* and *Y* (last rule).

Taxonomies. Concepts in an ontology can be organized in taxonomies. For instance, employees are a special category of persons having extra attributes, like salary and company. OntoDLV taxonomies by means of the inheritance feature. For example the following statement

```
class employee isa {person}
(salary:integer, company:enterprise).
```

defines employee as a specialization (or *subclass*) of a more generic concept or *superclass*, namely person. Attributes defined in person (i.e., name, age, father, mother, and residence) are inherited by employee, and are implicitly present with salary and company. Each OntoDLP has a common built-in superclass called *object*.

Note that inheritance can be applied repeatedly, for example

```
class checkingAccount isa {account}
(overdraftAmount:integer).
class savingsAccount isa {account}
(interestRate:integer).
class goldAccount isa {checkingAccount}
(minimumBalance: integer).
class youngAccount
isa {savingsAccount,checkingAccount}().
```

models that bank accounts are divided in checking and savings accounts. Moreover, bank may offer two special types of checking account: *gold account* having a fixed minimum balance; and *young account*, which is reserved to customers aged up to 21 years, and is, at the same time, both a saving account and a checking account. Moreover, instances of employee are also instances of person. For example, the instance:

```
bob:employee(name:"Robert", age:25,
   father:jack, mother:betty,
   residence:rome, salary:2000,
   company:microsoft).
```

is automatically considered an instance of person.

Intensional relations and collection classes can also be organized in taxonomies.

Axioms and Consistency. *Axioms* are a consistency-control construct modeling sentences that are always true. If an axiom is violated, the ontology is inconsistent, i.e., it contains information which is contradictory or not compliant with the domain's intended perception.

Axioms can be used for constraining the information contained in the ontology and verifying its correctness. The following axiom enforces that the father cannot be younger than his son as follows:

```
::- X:person(age:AgeOfX,
father:person(age:AgeOfFatherOfX),
AgeOfFatherOfX < AgeOfX.
```

This can be read literally as: "it is not possible that there is a person `X` having a father whose age is smaller than the one of his children".

XBRL

XBRL stands for eXtensible Business Reporting Language. It is a XML-based language using tagging metadata to describe financial information. The language was introduced in 1998, and since then it has become a standard means of communicating business reporting information between

ISSN:2319-7900

businesses and government authorities. XBRL provides a language in which reporting terms can be authoritatively defined and referred uniquely in financial statements or other kinds of compliance, performance and business reports. XBRL documents are interchangeable between different information systems in entirely different organisations, and can cross the boundaries of different nations with different legislation.

The core of XBRL is the XBRL2.1 specification, which defines the language w.r.t. three different layers. The most basic layer is the metadata layer, where the metamodel for the data that can be described with XBRL is made explicit. The second layer is the definition of the concepts referred in financial reports. At a technical level, concepts correspond to element definitions of an XML Schema. Basically, a concept is a definition that provides the meaning for a piece of information contained in a report. An example of concept is “profit”. Related concept definitions are organized in hierarchies that are called taxonomies. Thus taxonomies capture the meaning contained in all of the reporting terms used in a business report, as well as the relationships between all of the terms. These typically correspond to particular reporting domains, and are usually produced by financial regulators agencies, accounting standards setters, government agencies and other groups with the goal of providing a clear and unambiguous definition of the data to be written in a business report. Taxonomies can be freely added and linked to existing ones, and there is no limit on the concepts that can be added while extending existing XBRL definitions. For example, international taxonomies may be extended by national regulators or large enterprises to meet specific reporting requirements or to model specific reporting needs. Concept definitions also define constraints (logical or mathematical business rules) on what can be reported, to ensure quality of reports. The semantics of concepts in a taxonomy, as well as the constraints to be satisfied are expressed by means of a linkbase. A linkbase is a collection of XML extended links based on the linking language XLINK. An XBRL Instance can refer to more than one taxonomy, and taxonomies can be interconnected, extended and modified in various ways. The set of related taxonomies is called a Discoverable Taxonomy Set (DTS). A DTS is a collection of Taxonomy Schemas and Linkbases.

A taxonomy defines reporting Concepts, but it does not contain the actual values of facts based on the defined concepts. The third layer of XBRL is in charge of representing the actual instances that are used to populate the schema defined in the second layer. The fact values are contained in XBRL Instances and are referred to as *facts*. Instance documents are collections of facts, which are statements of the form “profit for Acme Inc. in 2010 was $100m”. At a technical level, facts are represented by elements in an XML document. Besides the actual value of a fact, an instance provides contextual information necessary for interpreting it. XBRL provides a fixed set of built-in information to be reported in a fact. These include a reference to the concept this fact is an instance of, the entity related to the fact, the period to which the statement refers to, and the unit of measure used for specifying the reported valued.

A typical XBRL instance document, thus, reports a set of *facts*. Each item refer to a specific *context* (such as a company or an individual), and it defines the period of time to which the fact can be applied. Further contextual information about the facts can be provided as scenarios, defining the units for the metrics and references to XBRL taxonomies.

A number of different kinds of relationships can be applied to a given fact. To our ends here, the most interesting kind of relationships are those defines as calculation relationships, where the parent element can be defined as a function of the values known for its children.

In fact, calculation relationships can be used for a number of different reasoning tasks, for instance, they might allow to compare the calculated total to the total that is declared in the specification. More generally, calculation relationships are meant to enforce integrity constraints on the numerical data reported in the file. However, current validation systems for XBRL do not support sophisticated forms of reasoning over such relationships. In particular, they:

- detect inconsistencies only among values that are of the same period type (instant or duration);
- detect inconsistencies when facts are unreported, even though they are implicitly specified via suitable calculation rules;
- do not propagate values along the taxonomy.

Indeed, XBRL instances (as well as taxonomies and linkbases) must comply with the syntax requirements imposed by the XBRL specification, which are mostly expressed using XML Schemas. Compliance with the XBRL standard, as well as other checks that can be expressed by means of an XML Schema file can be performed using standard XML validation software. However, it might be the case that special validation requirements are required that cannot be expressed using XML Schemas, and these must be handled using other validation technologies.

Logic-based Consistency Checking of XRBL Instances

In this section we describe an approach for modeling XBRL instances and validation constraints in OntoDLP. Roughly the idea is to model the information present in an XBRL

ISSN:2319-7900

instance in OntoDLP, and to use logic programming to model validation constraints and/or other business rules.

In the following, we first present an OntoDLP that models XBRL instances, and then we show how constraints on the data of an XBRL instance can be modeled by means of OntoDLP.

Modeling XBRL Instances. We now present an OntoDLP that models the information present in an XBRL instance. XBRL instances are stored in XML files. To improve readability, we will mention the constituent tags of an XBRL instance and their corresponding concepts without explicitly presenting the verbose XML syntax. We first overview the main elements present in an XBRL instance, and then present their OntoDLP. The representation is limited to elements that can be used for validating the information contained in a business report, footnotes and other information needed for rendering graphically XBRL instances is purposely not considered in our model. We refer the reader to the XBRL specification available on the Internet [3] for the details.

Single facts or business measurement reported in an XBRL instance file are stored in *items*. Each *item* usually holds a value and always refers to a *context*. The *context* contains information about the *entity* being described, the reporting *period* and optionally a *scenario* that models the different reporting purposes (e.g., actual, pro forma, budgeted) of a business facts. The context basically associates a business fact captured as an XBRL item with its meaning and locates it w.r.t. time and other contextual information. The *entity* documents the business, government department, individual, etc. that fact describes, and may be associated with an optional *segment* to identify the business segment more completely in cases where the Entity identifier is insufficient. The period models the instant or interval of time for reference by an item, and it can be specified by reporting beginning and ending dates, a specific instant, or it can be set to forever (when a datum is not dependent of time).

XBRL items can be *numeric*, *non numeric*, or *tuples*. Tuples allow to aggregate facts that cannot be independently understood, e.g., because multiple occurrences of that fact are being reported. Tuples have complex content and can be made of items and other tuples. Numeric items are associated with a *unit* of measure (e.g., USD, EUR, number of shares) and may be reported either with a precision (number of digits counting from the left to be considered trustworthy) or with a number of decimal places to which the value of the fact represented may be considered accurate.

The elements of an XBRL instance can be modeled by the following OntoDLP :

```
class Entity ( identifier:string,
identifierScheme:URI )
class Context ( entity:Entity,
period:Period )
class Period ( starting:Date,
ending:Date )
class Instant ( date:Date ) {
X:Instant( date:Date ) :-
X:Period( starting:Date,
ending:Date ).
}
forever:Period (
starting:"0000-00-00",
ending:"0000-00-00" ).
relation Segment (
context:Context,
element:SegmentElement )
class SegmentElement (
name:string, value: XSDType )
relation containsSegmentElement (
parent:SegmentElement,
child:SegmentElement )
relation Scenario (
context:Context,
element:ScenarioElement )
class ScenarioElement (
name:string,
value: XSDType )
relation containsScenarioElement (
parent:ScenarioElement,
child:ScenarioElement )
class Item (
name:string )
class NumericItem isa { Item } (
unit:Unit, value: XBRLNumeric )
relation Precision (
item:NumericItem, value:integer )
relation Decimals (
item:NumericItem, value:integer )
class NonNumericItem isa { Item }
(value:string )
class Tuple isa { Item }()
relation TupleContainsItem (
container:Tuple,
contains:Item )
```

Basically, almost all elements are modeled by an OntoDLP, and associations among elements are modeled by relations. Given an XBRL instance file, each element can be translated in the corresponding instance fact in OntoDLP. Object ids can be valued either by taking the id associated with the corresponding elements, or by using a proper generation strategy (e.g., based on a sequence generator).

More in detail, XBRL entities refer to the corresponding term in a taxonomy by means of an identifier and of a URI pointing to the namespace of the identification scheme. For example:

```
<identifier
     scheme="http://www.nasdaq.com">
     SAMP</identifier>
```

identifies the company with NASDAQ ticker symbol SAMP is modeled in OntoDLP :

ISSN:2319-7900

```
samp:Entity( identifier:"SAMP",
   identifierScheme:"http://www.nasdaq.com"
```

Items are instances of the *Item* class, which features a sub-class for each specific type of element.

For example the following two are items:

```
    <ci:capitalLeases contextRef="c1"
    unitRef="u1" precision="3">
    727432
    </ci:capitalLeases>
<ci:concentrationsNote contextRef="c1">
    Concentration of credit risk with regard
to short term investments is not  considered
to   be   significant due   to the Company’s
cash management policies. ...
    </ci:concentrationsNote>
```

The first is a numeric one means that Capital Leases in context c1 is 727000 accurate to 3 significant figures. The second reports a textual note concerning context c1. These are modeled in OntoDLP follows:

```
capitalLeases01: NumericItem(
name:"ci:capitalLeases", unit:u1,
value: 727432).
Precision(item:capitalLeases,
value:3 ).
concentrationsNote01: NonNumeriItem (
name:"ci:concentrationsNote",
value: "Concentration of
credit risk with regard to short
term investments is not
considered to be significant due
to the Company’s cash
management policies. ... ").
```

In turn, contexts associated using instances of the class concept, and in our example c1 could be as follows:

```
c1:Context( entity:samp,
    period:forever ).
```

A similar procedure can be applied for all the other enities, for instance the following scenario is associated to context c1

```
<scenario>
  <other:bestEstimate/>
  <fid:dwSlice>
    <fid:residence>MA</fid:residence>
    <fid:nonSmoker>true</fid:nonSmoker>
    <fid:minAge>34</fid:minAge>
    <fid:maxAge>49</fid:maxAge>
    </fid:dwSlice>
</scenario>
```

to indicate that the reported values relate to a "best estimate" scenario for non-smokers residing in Massachusetts of the specified age group. This is represented in OntoDLP follows:

```
Scenario( context:c1,
element: otherbest1 ).
Scenario( context:c1,
element: dwSlice1 ).
otherbest1:ScenarioElement (
name:"other:bestEstimate",
value:"" ).
dwSlice1:ScenarioElement (
name:"fid:dwSlice",
value:"" ).
dwResidence1:ScenarioElement (
name:"fid:residence",
value:"MA" ).
dwSmoker1:ScenarioElement (
name:"fid:nonSmoker",
value:true ).
dwminage1:ScenarioElement (
name:"fid:minAge",
value:34 ).
dwminage1:ScenarioElement (
name:"fid:maxAge",
value:49 ).
containsSegmentElement (
parent:dwSlice1,
child:dwResidence1).
containsSegmentElement (
parent:dwSlice1,
child:dwSmoker1).
containsSegmentElement (
parent:dwSlice1,
child:dwminage1).
containsSegmentElement (
parent:dwSlice1,
child:dwminage1).
```

Basically, an instance of this ontology can then be populated efficiently by analyzing an XBRL file and asserting properly the objects of the ontology.

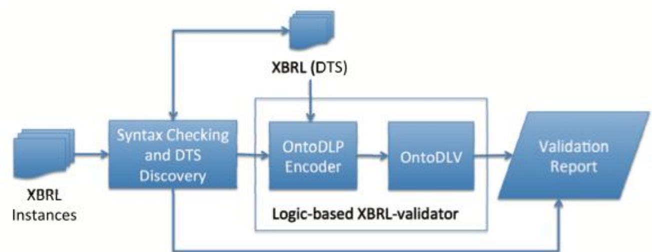

Figure 1: Validating XBRL Instances via OntoDLV .

Modeling Validation. XBRL instances must comply with

ISSN:2319-7900

the requirements imposed by the XBRL specification, as well as the must be coherent with the information modeled by a DTS. Clearly, validation requirements may go beyond the mere syntactical check that can be obtained by validating an XML file w.r.t. an XML Schema, and so the standard mandates that they must be handled using other validation technologies.

As previously pointed out we are mostly interested in calculation relationships, which may be defined by in a linkbase. These can be modeled by a combination of logic rules and OntoDLP.

First of all we point out that *calculation arcs* of a linkbase can be easily modeled by axioms. For example

```
<calculationArc xlink:type="arc"
  xlink:arcrole="http://www.xbrl.org/
  2003/arcrole/summation-item"
  xlink:from="totalPrepaidExpenses"
  xlink:to="prepaidExpenses"
  weight="1.0"/>
```

requiring to sum (with weight of one) prepaid expenses items into current assets can be expressed as:

```
::- not  #sum{V,I : X:NumericItem(
    name:"prepaidExpenses", value:V)}=CS,
    NumericItem(name:"totalPrepaidExpenses",
    value:CS).
```

This process can be generalized, and an axiom having in the body a #sum aggregate properly filled by elements to be accumulated can be produced automatically by translating the this part of the linkbase in OntoDLP.

```
 relation summationArc (
   from:Entity, to:Entity )
summationArc(
  from:totalPrepaidExpenses,
to:prepaidExpenses, weight:1).
::- not
  #sum{V,I : F:NumericItem(value:V)}=CSW,
  CSW=CS*W, T:NumericItem(value:CS),
  summationArc(from:F, to:T,
  weight:W).
```

where summation arcs are represented by a relation in OntoDLP. Analogously other linkbase arcs can be encoded by means of OntoDLP, and can be exploited for modeling constraints that are not checked by a pure syntactic validator, such as items and values that can be inferred through essence-alias relationships.

Additional constraints, that go beyond what can be expressed using a linkbase, can be specified directly using logic programming. For example there is the possibility of accessing data specified in XBRL taxonomies, and we can exploit it for encoding properties where the parent element can be defined as a function of the values known for its children.

The information present in the DTS can be used to enrich our ontology by populating the isA relation, that can be used to model the structure of a taxonomy. Then the isA relation can be exploited in constraints, for example the above constraint can be extended to sum over all subconcepts of a given concept as follows:

```
relation isA ( super:Entity,
sub:Entity )
isA(super: X, sub:Z) :-
isA(super: X, sub:Y ),
isA(super: X, sub:Z ).
isA( super: Expenses,
sub:prepaidExpenses ).
::- not
 #sum{V,I : F:NumericItem(value:V)}=CSW,
CSW=CS*W, T:NumericItem(value:CS),
summationArc(from:F, to:TS,
weight:W), isA(super: TS, sub:T).
```

Clearly, besides constraints asserted in linkbases, one may think to design more complex requirements by exploiting logic rules and axioms. These can be easily added to the ontology, which thus provides a mean of implementing (using a declarative language) involved consistency checks on XBRL instances.

System Architecture

We propose a logic-based approach to validation of XBRL instances. The architecture of a system implementing our approach is depicted in Figure 1. The validator takes in input a number of XBRL Instance documents. The first module applies standard XML processing technology to (*i*) discover the Discoverable Taxonomy Set (DTS) associated with the input documents; and (*ii*) applies the syntactic checks of conformance with the files in the discovered DTS. Recall that a DTS is a collection of taxonomy schemas and linkbases. The bounds of a DTS are such that the DTS includes all taxonomy schemas and linkbases that can be discovered starting from the instance files, and following links or references in the taxonomy schemas and linkbases included in the DTS. The DTS discovery process is fully described in the XBRL specification [3]. Concerning the syntactic checks, these include the basic validation of instance documents w.r.t. the XML schemas defining taxonomies, as well as a number of other syntactical coherence tests. The result of these preliminary test are immediately printed in the final validation report. In case this first validation succeeds, the discovered DTS together with the input instances are passed to the next modules which are the kernel part of out logic-based approach to validating XBRL instances. In particular, these are read and processed by the OntoDLP module. The

ISSN:2319-7900

encoder creates an instance of the XBRL ontology described in previous section, modeling to the input documents and their constraints. Basically suitable logic facts model XBRL facts, and a number of logic rules and axioms model the validation to be performed. This specification is fed as input to the OntoDLV which executes an ontology consistency checking task. The result of this checking step is used to produce he final report, which will include also suitable motivations in case of failure of some specific axiom.

Note that the architecture depicted in Figure 1 can be easily extended to incorporate additional checks, which may be specified either in OntoDLP by using other technologies, such as XSLT transformations. Indeed, the syntax checking and discovery module can be cascaded with other validation software, e.g., running XSLT specifications, that produce additional information to be added to the final report. Moreover, the OntoDLP can be configured to receive additional reasoning modules that performing complex checks directly-specified in in logic that, for instance, cannot not be modeled using current XBRL specifications.

Conclusion and Ongoing work

In this paper we have approached the problem of validating financial reports written in XBRL by using logic-based technologies. The idea is to represent by means of a logic-based language called OntoDLP data present in XBRL instance documents. The expressive power of OntoDLP then used to model both simple and complex numerical validations on XBRL instances. This paper paves the way for developing an advanced system for validating XBRL instances.

As far as future work is concerned, we plan to properly extend it to cope with all features of XBRL (e.g., supporting multi-dimensional instance sets), and we will design a library of logic programs containing general purpose validation constraints. Moreover, we plan to implement our proposal by exploiting the OntoDLV and to experiment with in on real-world XBRL instances.

Acknowledgments

The author is grateful to Nicola Leone for the advise of using logic programming as declarative language for checking XBRL instances. He is also grateful to Gianluigi Greco and Francesco Ricca for their valuable comments in an early version of this paper.

References

[1] Mario Alviano, Francesco Calimeri, Wolfgang Faber, Nicola Leone, and Simona Perri. Unfounded sets and well-founded semantics of answer set programs with aggregates. *J. Artif. Intell. Res. (JAIR)*, 42:487–527, 2011.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a Theory of Declarative Knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Inc., Washington DC, 1988.

[3] XBRL Consortium. Xbrl consortium standards web site. 2014. http://specifications.xbrl.org/specifications.html.

[4] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-Solving Using the DLV System. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[5] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[6] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006.

[7] Teodor C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers, Inc., 1988.

[8] Francesco Ricca, Lorenzo Gallucci, Roman Schindlauer, Tina Dell'Armi, Giovanni Grasso, and Nicola Leone. OntoDLV : an ASP-based system for enterprise ontologies. *Journal of Logic and Computation*, 2009.

[9] Francesco Ricca and Nicola Leone. Disjunctive Logic Programming with types and objects: The DLV System. *Journal of Applied Logics*, 5(3):545–573, 2007.

Biographies

G. D'ATRI is professor in Computer science at the University of Calabria. He studies aspects of computer science related to economy and money.
Gdatri <gdatri@yahoo.com>