# RDF to Conceptual Graphs Translations

## Calculating Shatterproof Transponds

Jean Francois Baget[1,2], Michel Chein[2], Madalina Croitoru[2], Jérôme Fortin[2], David Genest[3], Alain Gutierrez[2], Michel Leclere[2], Marie-Laure Mugnier[2], and Éric Salvat[4]

[1] INRIA Sophia Antipolis, 2004 Route des Lucioles 06902 Sophia Antipolis, France
jean-francois.baget@inria.fr
[2] LIRMM (CNRS & Université Montpellier II), 161 rue Ada, F–34392 Montpellier Cedex 5, France
{chein, croitoru, fortin, gutierre, leclere, mugnier}@lirmm.fr
[3] LERIA, Université d'Angers, 2, boulevard Lavoisier, 49045, Angers, France
genest@info.univ-angers.fr
[4] IMERIR, Avenue Pascot, 66004 Perpignan, France
salvat@imerir.com

**Abstract.** In this paper we will discuss two different translations between RDF (Resource Description Format) and Conceptual Graphs (CGs). These translations will allow tools like Cogui and Cogitant to be able to import and export RDF(S) documents. The first translation is sound and complete from a reasoning view point but is not visual nor a representation in the spirit of Conceptual Graphs (CGs). The second translation has the advantage of being natural and fully exploiting the CG features, but, on the other hand it does not apply to the whole RDF(S). We aim this paper as a preliminary report of ongoing work looking in detail at different pro and the cons of each approach.

## 1 Introduction and motivation

In this paper we will discuss the different translations between RDF (Resource Description Format) and Conceptual Graphs (CGs). We aim this paper as a preliminary report of the ongoing work of the authors looking in detail at different problems raised by the translation. We will give the overview of two possible translations and explain the pro and the cons of each one of them.

The objective of this work is to have a detailed translation from RDF(S) to COGXML and vice-versa. This translation will allow tools like Cogui [5] and Cogitant [6] to be able to import RDF(S) documents and to export RDF(S) documents. The translation between RDF and Conceptual Graphs is an important problem to be addressed for both Conceptual Graphs and Semantic Web communities:

– For the **Conceptual Graphs** people there is an obvious interoperability benefit (adhering to a well known standard). One consequence of this benefit is the fact that large RDF(S) benchmarks are available, hence ready for use in the context of testing CG algorithms.

[5] http://www.lirmm.fr/cogui/
[6] http://cogitant.sourceforge.net/

– For the **Semantic Web** people
  - We provide an editor for RDF that performs representation and reasoning at the same time (this is not possible yet with the other editors).
  - Another benefit for Semantic Web people will the translation of RDF into a data structure that is relying on a support hence potentially improving the performances of the algorithms manipulating RDF (subsumption).
  - And finally, the RDF community could benefit from the ideas behind the CG extensions.

A first translation has been provided by Tim Berners Lee [1]. While an important step towards the RDF - CG translation this paper remains at an intuitionist level. More in detail the translation has been addressed by [2] or [3] but the different technical difficulties encountered have been addressed within an implementation oriented setup. The authors of [4] also address the problem of converting CGs to RDF but certain practical cases are not addressed in their framework. A more theoretical approach has been taken by [5] and this translation will be discussed further on in the paper.

This paper discusses two different RDF(S) - CG translations. The first translation, following the work of [5] is sound and complete from a reasoning view point but is not a visual representation or a representation in the spirit of Conceptual Graphs (the support is flat).

The second translation has the advantage of being natural and fully exploiting the CG features (for instance, the RDFS constructs subClassOf and subPropertyOf are translated into partial orders, which are automatically taken into account by projection). On the other hand, it does not apply to the whole RDF(S), but only to RDF(S) documents conforming to a strict separation between concepts (or classes), relations (or properties) and individuals. In other words, all RDF(S) triples leading to meta reasoning are out of the scope of this translation. Note that for a given RDF(S) document, there may be several maximal subsets of triples satisfying the property. A simple way of choosing one maximal subset is to process the triples in the given order and to discard triples that contradict the separation according to the triples already processed. The separability condition is in accordance with usual assumptions in knowledge representation and reasoning, for instance in description logics (see namely OWL-DL). Moreover, it seems that most RDF(S) documents fulfill this condition in practice (which remains to be experimentally checked).

On the other hand, the first translation allows to process the whole RDF(S), but it does not allow to fully benefit from CG features (for instance, the RDFS constructs subClassOf and subPropertyOf are translated into CG relations, and rules are necessary to ex- press the transitivity of these relations (cf. rules 5 and 11 in Figure 2), which leads to a loss in algorithmic efficiency).

## 2 Preliminary notions

In this section, we will quickly recall the main basic structures of RDF(S) and Conceptual Graphs (CGs) in order to have a coherent notation throughout the paper. Since in this paper we will focus on the translation between RDF(S) and will first present RDF(S) and then CGs.

### 2.1 RDF(S)

**General notions** The Resource Description Framework (RDF) has been defined by the World Wide Web Consortium (W3C) as a metadata data model. It allows to make statements about resources in the form of subject–predicate–object expressions called RDF triples. The subject denotes a resource, the predicate gives the relationship between the subject and the object. Any RDF statement can be both stored in a XML format or in Notation 3 (or N3). A set of RDF statements can be dispayed by a graph.

**Syntax** The subject of an RDS statement can be a Uniform Resource Identifier (URI) or blank node. An URI is a a string of characters which identifies a resource which can be a locator (URL), a name (URN), or both. A blank node represents an anonymous resource which is not directly identifiable. The predicate is a URI which represents a relationship. The object can be an URI, a blank node or a Unicode string literal. Note that a given URI can be subject of one expression, predicate of another and object of a third one.

A predefined property, `rdf:type` is provided to classify entities into different categories. RDF permits to represent groups of entities. The first way to group things is the use of some containers. The three kinds of containers are `rdf:Bag`, `rdf:Seq` and `rdf:Alt`, depending if one wants to have a group of distinct things or not and if the order of the given elements is relevant or not. Their use permits to list entities that are part of the container, but there is no way to specify that no other item can be part of the group modelled by the container. Collections has been created for this purpose. A collection is a list structure based on the the predefined types `rdf:List`, the predefined properties `rdf:first` and `rdf:rest`, and the predefined resource `rdf:nil`. RDF Reification provides a built-in vocabulary used to describe RDF statements. This vocabulary consist in the type `rdf:Statement`, and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`.

The main field of a structured value is given by the `rdf:value` property.

`rdf:XMLLitteral` is used for the sake of simplicity when some litteral contains XML notation.

### 2.2 RDF Schema

**General notions** RDF Schema (RDFS) is a specification that explains how to describe RDF vocabularies. It provides a kind of type system for RDF.

**Syntax** Some classes of things can be defined in RDFS to model some categories. A class is defined as being a resource which has a `rdf:type` property for which the value is `rdfs:Class`. A resource can be an instance of sevral classes and every class is an instance of the class `rdfs:Resource`. The `rdfs:subClassOf` property induce a specialization relationship between classes. This relation is transitive. One can define some properties of classes: `rdf:Property` is the superclass of all properties. The properties `rdfs:domain`, `rdfs:range`, and `rdfs:subPropertyOf` are special

kinds of properties. `rdfs:domain` is used to express that a given property is designed to describe a particular class. `rdfs:range` ensures that the value of a property ranges in a given class. The `rdfs:subPropertyOf` property induces a specialization of two properties. The class `rdfs:Datatype` is used to identify an URIref as datatype. `rdfs:Literal` defines the class of all literal values such as strings and integers. Other built-in properties exit in RDFS. For example, `rdfs:comment` and `rdfs:label` to provide a human readable description or name of a resource.

**RDFS Rules** The RDFS rules, as detailed in `http://www.w3.org/TR/rdf-mt/` are available in Figure 1. The rules should be read: "if one finds this information (as detailed in column 2) then the following information (column 3) should be added".

| Rule Name | If E contains: | then add: |
|---|---|---|
| rdfs1 | uuu aaa lll.<br>where lll is a plain literal (with or without a language tag). | _:nnn rdf:type rdfs:Literal .<br>where _:nnn identifies a blank node allocated to lll by rule rule lg. |
| rdfs2 | aaa rdfs:domain xxx .<br>uuu aaa yyy . | uuu rdf:type xxx . |
| rdfs3 | aaa rdfs:range xxx .<br>uuu aaa vvv . | vvv rdf:type xxx . |
| rdfs4a | uuu aaa xxx . | uuu rdf:type rdfs:Resource . |
| rdfs4b | uuu aaa vvv. | vvv rdf:type rdfs:Resource . |
| rdfs5 | uuu rdfs:subPropertyOf vvv .<br>vvv rdfs:subPropertyOf xxx . | uuu rdfs:subPropertyOf xxx . |
| rdfs6 | uuu rdf:type rdf:Property . | uuu rdfs:subPropertyOf uuu . |
| rdfs7 | aaa rdfs:subPropertyOf bbb .<br>uuu aaa yyy . | uuu bbb yyy . |
| rdfs8 | uuu rdf:type rdfs:Class . | uuu rdfs:subClassOf rdfs:Resource . |
| rdfs9 | uuu rdfs:subClassOf xxx .<br>vvv rdf:type uuu . | vvv rdf:type xxx . |
| rdfs10 | uuu rdf:type rdfs:Class . | uuu rdfs:subClassOf uuu . |
| rdfs11 | uuu rdfs:subClassOf vvv .<br>vvv rdfs:subClassOf xxx . | uuu rdfs:subClassOf xxx . |
| rdfs12 | uuu rdf:type rdfs:ContainerMembershipProperty . | uuu rdfs:subPropertyOf rdfs:member . |
| rdfs13 | uuu rdf:type rdfs:Datatype . | uuu rdfs:subClassOf rdfs:Literal . |

**Fig. 1.** RDFS rules

### 2.3 Conceptual Graphs (CGs)

Conceptual Graphs were introduced by Sowa (cf. [6, 7]) as a diagrammatic system of logic with the purpose "to express meaning in a form that is logically precise, humanly readable, and computationally tractable". In this paper we use the term "Conceptual Graphs" to denote the *family of formalisms* rooted in Sowa's work and then enriched and further developed with a graph-based approach in [8].

Conceptual Graphs encoded knowledge as graphs and thus can be visualized in a natural way:

- The vocabulary, which can be seen as a basic ontology, is composed of hierarchies of concepts and relations. These hierarchies can be visualized by their Hasse diagram, the usual way of drawing a partial order.
- All other kinds of knowledge are based on the representation of entities and their relationships. This representation is encoded by a labeled graph, with two kinds of nodes, respectively corresponding to entities and relations. Edges link an entity node to a relation node. These nodes are labeled by elements of the vocabulary.

The **vocabulary** is composed of two partially ordered sets: a set of concepts and a set of relations of any arity (the arity is the number of arguments of the relation). The partial order represents a specialization relation: $t' \leq t$ is read as "$t'$ is a specialization of $t$". If $t$ and $t'$ are concepts, $t' \leq t$ means that "every instance of the concept $t'$ is also an instance of the concept $t$". If $t$ and $t'$ are relations, then these relations have the same arity, say $k$, and $t' \leq t$ means that "if $t'$ holds between $k$ entities, then $t$ also holds between these $k$ entities").

A **basic graph** (BG) is a bipartite graph: one class of nodes, called *concept* nodes, represents entities and the other, called *relation* nodes represents relationships between these entities or properties of them. A concept node is labeled by a couple $t : m$ where $t$ is a concept (and more generally, a list of concepts) and $m$ is called the marker of this node: this marker is either the generic marker, denoted by $*$, if the node refers to an unspecified entity, otherwise this marker is a specific individual name. BGs are used to represent assertions called *facts*. They are also building blocks for more complex kinds of knowledge (such as rules, or nested graphs). In this paper we only detail rules as they are of direct interest to the framework we are proposing.

A **rule** expresses implicit knowledge of form "if *hypothesis* then *conclusion*", where hypothesis and conclusion are both basic graphs. Using such a rule consists of adding the conclusion graph (to some fact) when the hypothesis graph is present (in this fact). There is a one to one correspondence between some concept nodes of the hypothesis with concept nodes of the conclusion. Two nodes in correspondence refer to the same entity. These nodes are said to be *connection nodes*. The knowledge encoded in rules can be made explicit by applying the rules to specific facts.

These graphical objects are provided with a **semantics in first-order-logic**, defined by a mapping classically denoted by $\Phi$ in conceptual graphs [7]. First, a FOL language corresponding to the elements of a vocabulary $\mathcal{V}$ is defined: concepts are translated into unary predicates and relations of arity $k$ into predicates of arity $k$. Individual names become constants. Then, a set of formulas $\Phi(\mathcal{V})$ is assigned to the vocabulary. These formulas translate the partial orders on concepts and relations: if $t$ and $t'$ are concepts,

with $t' < t$, one has the formula $\forall x(t'(x) \rightarrow t(x))$; similarly, if $r$ and $r'$ are k-ary relations, with $r' < r$, one has the formula $\forall x_1 \ldots x_k(r'(x_1 \ldots x_k) \rightarrow r(x_1 \ldots x_k))$. A fact $G$ is naturally translated into a positive, conjunctive and existentially closed formula $\Phi(G)$, with each concept node being translated into a variable or a constant: a new variable if it is a generic node, and otherwise the constant assigned to its individual marker. The logical formula assigned to a rule $R$ is of form $\Phi(R) = \forall x_1 \ldots x_p\ ((hyp) \rightarrow \exists y_1 \ldots y_q\ (conc))$, where: $hyp$ et $conc$ are conjunctions of atoms respectively translating the hypothesis and the conclusion, with the same variable being assigned to corresponding connection nodes; $x_1 \ldots x_p$ are the variables assigned to the concept nodes of the hypothesis; $y_1 \ldots y_q$ are the variables assigned to the concept nodes of the conclusion except for the connection nodes.

More importantly, first order logic subsumption can also be translated in a graphical operation: homomorphism. A homomorphism from $G$ to $H$ is a mapping between the node sets of $G$ to the node sets of $H$, which preserves the adjacency between nodes of $G$ and can decrease the node labels. If there is a homomorphism (say $\pi$) from $G$ to $H$, we say that $G$ *maps* to $H$ (by $\pi$).

## 3 The sound and complete translation

This translation will simply translate each triplet RDF in a ternary relation where each of the concept nodes of the relation will represent the RDF triplet elements. See Figure 2 below where on top of the image the CG translation is shown for the RDF triplets shown at the bottom of the picture:

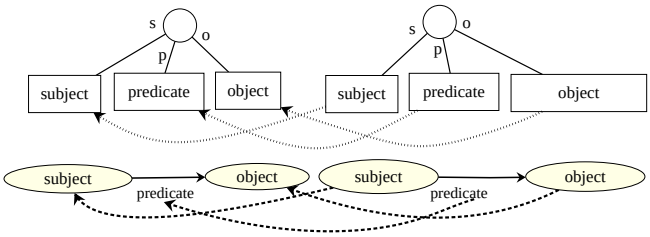


**Fig. 2.** RDF to CG translation example and the according homomorphism

This translation is straightforward and will ensure soundness and completeness of homomorphism (as opposed to the first translation where we are sound and complete but only with respect to a subset of RDF(S)). However, this translation is not visual or in the spirit of Conceptual Graphs as such (the support is totally flat).

### 3.1 Translating RDFS rules

The RDFS rules are translated in rules over graphs obtained from triplets as depicted in the following Figure 3. Darker nodes represent the conclusion of the rule.

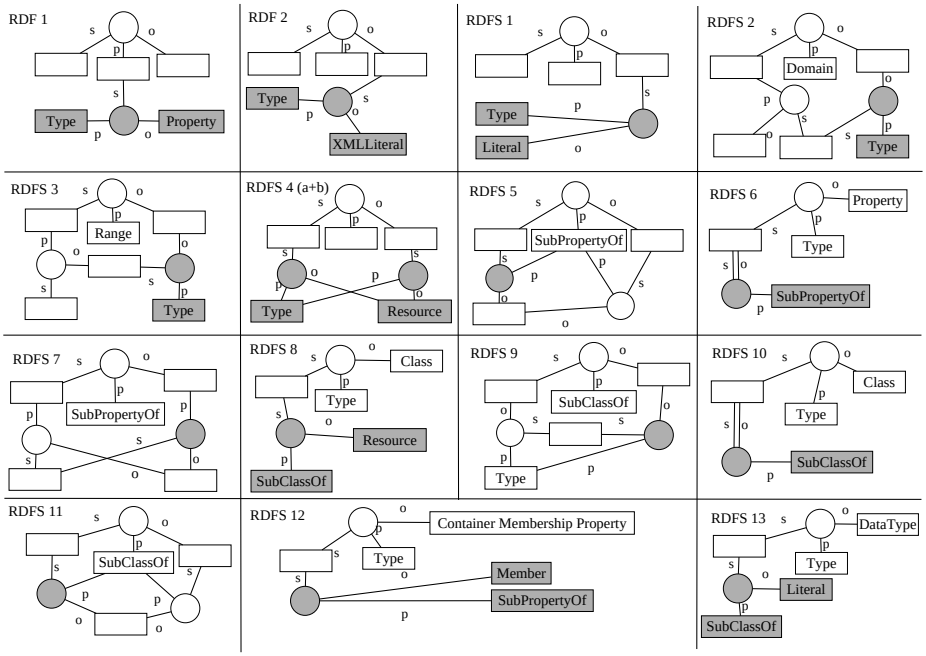


**Fig. 3.** Graph based RDFS rules

However, this translation is not in the spirit of Conceptual Graphs (as already disscussed in the introduction) given the (1) flatness of the support and (2) the visual properties of the representation. Indeed, for 2 nodes and one edge in the original graph, this representation will replace it with 4 nodes and 3 edges. In Figure 4 such a graph is depicted.

## 4 The CG spirit translation

In this section we present a more intuitive translation from RDF to CGs. The main idea behind it is to try to exploit as much as possible the separation between background knowledge and factual knowledge. However, RDF represents information at a different meta level than the one followed by the Conceptual Graphs optic. A direct consequence is that the same object can act as a individual marker, a relation or a concept type in RDF. For the work presented in this paper we will only focus on the RDF subset in which the three above mentioned sets are disjoint. Current and future work is looking at each case where the overlap might occur and how to address each case separately. These results are out of the scope of this paper. Also, the order in which we process the triplets will count. But for now we will only give the translation of each structural element in RDF versus CGs with a more in depth discussion of the problems that arise

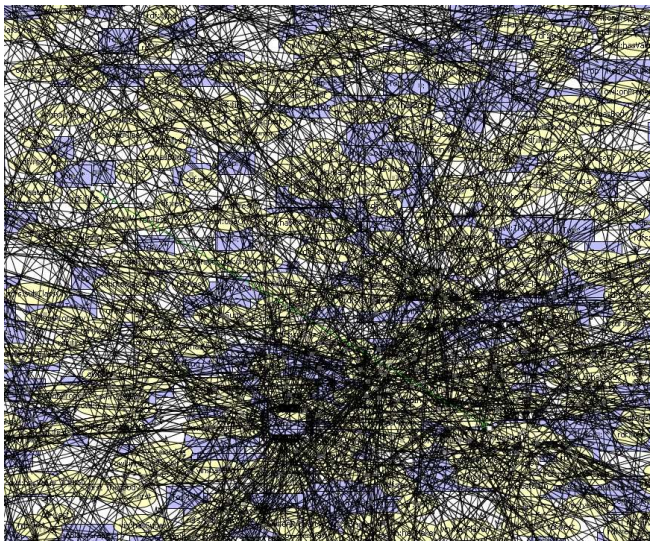

**Fig. 4.** Unintuitive visual representation of a RDF graph using CGs

as a consequence of the triplet ordering to be detailed in a different paper. A list of these constructs and their according translation in Conceptual Graphs is enumerated below.

- Support: Concept Hierarchy
  - **rdfs:Resource** - top of the concept type hierarchy
  - **rdfs:Class** - given the particularity of the relation between Class and Resource, Class will be implicitly represented. Every time we find "class" we will create a new concept in the concept type hierarchy (just underneath top). The classes will be linked up with **rdfs:subClassOf**.
  - **rdfs:Literal** will become a concept right under top. This concept will be further refined into a set of concepts referring to datatypes. **rdf:XMLLiteral** will be another one of these concepts.
    **rdfs:Datatype** will represent the union of all these concepts.
  - **rdf:Container** with the descendants **rdf:Bag**, **rdf:Seq** and **rdf:Alt** as a subclass of top
  - **rdf:List** as a subclass of top
  - **rdf:Statement** as a subclass of top
- Support: Relation Hierarchy
  - There will be two relation hierarchies: one for binary relations and one for ternary relations. The top of the binary relations hierarchy is T(Resource, Resource). The top of the ternary relation hierarchy is T(Resource, Resource, Resource).
  - **rdf:Property** - treated the same as **rdfs:Class**: when we encounter it we will replace it by a new binary relation. The signature will be also managed by the

RDFS rules. The binary relations hierarchy obtained in this manner will be given by **rdfs:subPropertyOf**
  - **rdfs:member** is a subrelation of T(Resource, Resource, Resource). Its signature is (rdfs:Container, rdfs:Resource, rdfs:Literal).
    **rdfs:ContainerMembershipProperty** is a subrelation of this one with the same signature.
  - **rdf:first**, **rdf:rest** and **rdf:value** are subrelations of T(Resource, Resource). The signature of rdf:first is (Resource, List), the signature of rdf:rest is (List, List) and the signature of rdf:value is (Resource, Resource).
- Other manipulations
  - **rdfs:label** - treated by the multi language facilities in COGXML
  - **rdfs:comment** - also treated by the multi language facilities in COGXML
  - **rdfs:seeAlso** - also treated by the multi language facilities in COGXML
  - **rdfs:range** and **rdfs:domain** will give the signature of the relation along with rdfs rules.
  - **rdfs:type** will create the individuals. The individuals are crated as we parse the document except for **rdf:nil** which is a predefined individual.
  - **rdf:statement** is treated as a concept type. For each statement we will do a different nesting, eventually with the concepts linked up by corefs. **rdf:subject**, **rdf:predicate** and **rdf:object** will give the graph inside the nesting.

### 4.1 The first translation and RDFS rules

In this section we will present how this translation deals with the semantics imposed by the RDFS rules presented in the preliminary notions Section. Please note that in the list below the item numbers of each rule correspond to the numbers of RDFS rules as found in the W3C document available at: `http://www.w3.org/TR/rdf-mt/`

- **SE1**: A blank node is treated as a generic concept node (as a consequence we will not have the renaming isses the semantic web community have with the new set of blank nodes.) For the rule SE1 given the fact that projection will take care of the matching individual / generic concept we do not have any extension to do.
- **SE2**: For the rule SE2 given the fact that projection will take care of the matching individual / generic concept we do not have any extension to do.
- **lg**: Just as above, the rule will provide the generalisation mechanism for literals (since literals cannot be a subject or a predicate of a statement).
- **gl**: This rule will only work for Datatypes (as in rule RDFS 13 and RDFS 1). The intuition here is since a literal can only be an object then one needs to create a blank node to act as a potential subject. The RDFS 13 rule does not apply in our case since the datatype in the concept hierarchy is the reunion of all datatypes. This means that if $x$ is of type literal then we put $x$ as a subclass of Literal. We will also write a constraint that certifies that Literal can only be a second neighbor.
- **RDFS 2**: This rule will mean that we need to update the signature of the relation type in the support with the according domain
- **RDFS 3**: This rule will mean that we need to update the signature of the relation type in the support with the according range

- **RDFS 4**: Is already expressed (Resource is Top) in the support
- **RDFS 5**: Already expressed in the support
- **RDFS 6**: Taken care by the subsumption relation of the support
- **RDFS 7**: Already in the support of relations
- **RDFS 8**: Already in the support
- **RDFS 9**: Already in the support of concepts
- **RDFS 10**: Taken care by the subsumption relation of the support
- **RDFS 11**: Already in the support
- **RDFS 12**: Taken care by the decision to put $member$ as a superclass of $containerMembership$
- **XMLClash**: We will add a $ILLTypedLiteral$ concept in the hierarchy and parse the nodes accordingly.
- **ext1** Already taken care by the projection mechanism
- **ext2** Already taken care by the projection mechanism
- **ext3** Will be done by adding conjunctive types for signatures
- **ext4** Will be done by adding conjunctive types for signatures
- **ext5 - ext 9**: We cannot do with CGs.

A visual depiction of such translation is given in Figures 5 and 6 where we show the facts, and respectively hierarchy of relations of a subset of the WINE ontology available at: `http://www.schemaweb.info/webservices/rest/GetRDFByID.aspx?id=62`.

## 5 Current and future work

This paper has presented the initial work carried out towards the analysis of the translation between RDF and CGs. While a lot of problems are still currently addressed (hence not detailed in this paper) the direction of work is clearly of benefit for both CG and Semantic Web community. An in depth analysis of the semantic tradeoffs of the translation at hand, where each particular case is separately addressed, and where soundness and completeness results are clearly provided, is of great benefit in an era where the development of effective techniques for knowledge representation and reasoning (KRR) is crucial for successful intelligent systems.

## References

1. Berners-Lee, T.: Conceptual graphs and the semantic web - reflections on web architecture (2001) `http://www.w3.org/DesignIssues/CG.html`, last accessed 25 June 2009.
2. Corby, O., Dieng, R., Hbert, C.: A conceptual graph model for w3c resource description framework. In: In In Proceedings of ICCS-2000, Springer (2000) 468–482
3. Martin, P., Eklund, P.W.: Knowledge retrieval and the world wide web. IEEE Intelligent Systems **15**(3) (2000) 18–25
4. Yao, H., Etzkorn, L.H.: Automated conversion between different knowledge representation formats. Knowl.-Based Syst. **19**(6) (2006) 404–412
5. Baget, J.F.: RDF entailment as a graph homomorphism. In: Proc. of the International Semantic Web Conference. (2005) 82–96

6. Sowa, J.F.: Conceptual graphs for a database interface. IBM Journal of Research and Development **20**(4) (1976) 336–357
7. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley (1984)
8. Chein, M., Mugnier, M.L.: Graph-based Knowledge Representation. Springer (2009)

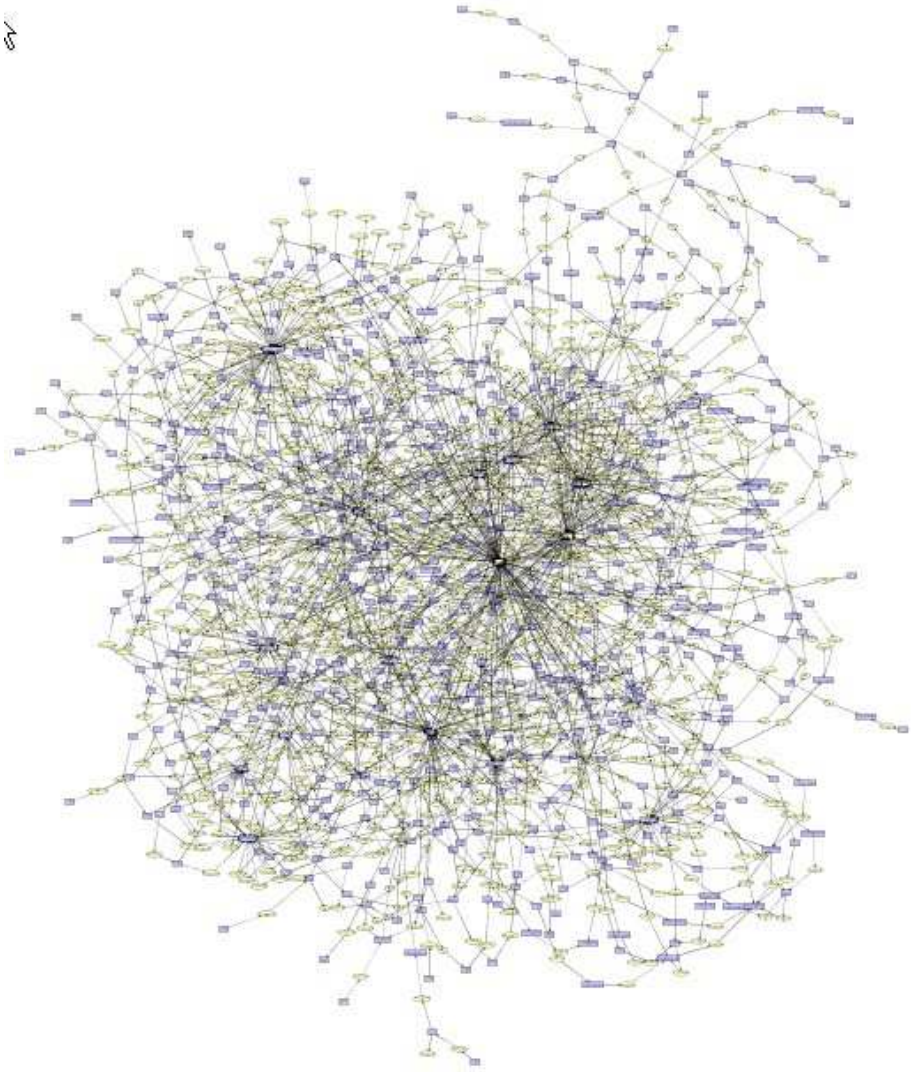

**Fig. 5.** RDF to CG translation for the WINE ontology (facts)

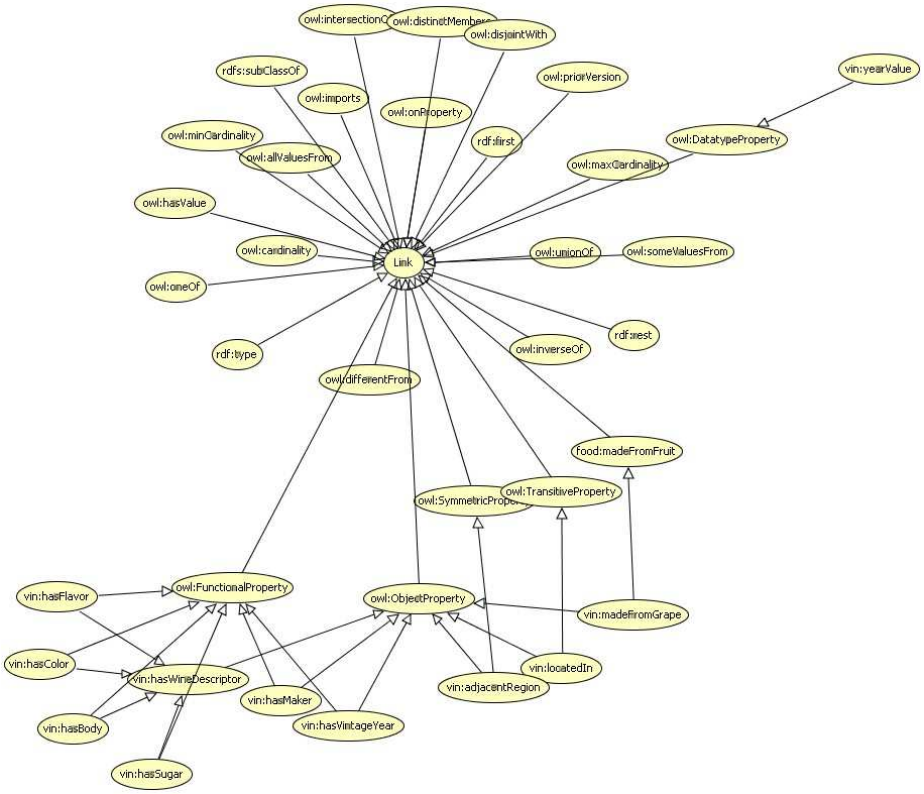


**Fig. 6.** RDF to CG translation for the WINE ontology (relation types)