The Knowledge Engineering Review, Vol. 28:3, 249–271.
doi:10.1017/S0269888913000234

Visual reasoning with graph-based mechanisms: the good, the better and the best

MICHEL CHEIN, MARIE-LAURE MUGNIER and MADALINA CROITORU

LIRMM, 161 rue ADA, F34392 Montpellier, Cedex 5, France;
e-mail: chein@lirmm.fr, mugnier@lirmm.fr, croitoru@lirmm.fr

Abstract

This paper presents a graph-based knowledge representation and reasoning language. This language benefits from an important syntactic operation, which is called a graph homomorphism. This operation is sound and complete with respect to logical deduction. Hence, it is possible to do logical reasoning without using the language of logic but only graphical, thus visual, notions. This paper presents the main knowledge constructs of this language, elementary graph-based reasoning mechanisms, as well as the graph homomorphism, which encompasses all these elementary transformations in one global step. We put our work in context by presenting a concrete semantic annotation application example.

1 Introduction

Knowledge representation and reasoning. Knowledge representation and reasoning (KR) has long been recognized as a central issue in artificial intelligence (AI). Very generally speaking, the problem is how to encode human knowledge and reasoning by symbols that can be processed by a computer to obtain intelligent behavior. In general, AI is concerned with qualitative, rather than quantitative, problem solving. As a basic building block of AI applications, a knowledge representation formalism should reflect this idea by supporting reasoning instead of calculation. This is done by organizing the knowledge into an easily processable form, classifying information by its properties, preprocessing the information, etc. The subfield of AI precisely called KR especially studies computational models, languages and systems able to represent knowledge in an explicit way and to do inferences, or reasoning, on this knowledge.

Even if there have been heated debates about KR in the past, in particular with respect to the role of logic in KR, there is nowadays an agreement on some important properties of a KR language, namely: to be logically founded, to allow for a structured representation of knowledge, to have good computational properties, and to allow users to have a maximal understanding and control over each step of the knowledge base cycle.

The first point concerns the fact that the language is translatable into a logic: the expressions should be translatable into sentences of a given logic, and inferences in that language should correspond to deduction in this logic. In other words, the inference mechanisms should be logically *sound* (i.e. every piece of knowledge inferred is deducible in the target logical fragment) and *complete* (any piece of knowledge that cannot be inferred cannot be deduced either in the target logical fragment). This allows to give a precise semantics to expressions and inferences, and to compare different languages from an expressiveness viewpoint.

Knowledge structuring means that semantically related pieces of knowledge should be grouped together, and that different kinds of knowledge (such as facts, rules, constraints, goals, etc.) should be represented by different knowledge constructs. This can be motivated by model adequacy

(i.e. its conformity to the modeling of the application domain) and by computational efficiency concerns. A large part of KR research can be seen as the search of good tradeoffs between the expressivity of a representation formalism and the computational complexity of the associated reasoning.

Knowledge-based systems. Knowledge-based systems (KBS) are systems, built upon models, able to represent knowledge in an explicit way and do reasoning with. These systems include a *knowledge base* (KB), composed of different kinds of knowledge, and a reasoning engine. The reasoning engine processes knowledge in the KB to answer a question or to reach a certain goal (for instance to check the consistency of the KB or to build a sequence of actions in order to achieve a goal). The cornerstone of the KB is the *ontology*. From an epistemological viewpoint, an ontology answers the question 'what kinds of things exist in the application domain?' We consider here computational ontologies, which provide a symbolic representation of classes of objects, called *concepts*, as well as the possible relationships between objects, called *relations* or *roles*. All other pieces of knowledge in the KB are expressed by structures built with the ontology terms (concepts and relations).

Classically, in the building of a KBS, the first phase consists of knowledge *elicitation*: obtaining a system specification expressed in a language understandable by human beings, so that it can be checked by domain experts. This *mediating representation* does not need to be precise, consistent, or complete. Its role is to allow the experts to freely explicit knowledge and to communicate with others. The second phase consists of translating (the largest possible part of) this informal representation into a formal representation, expressed in a formal language provided with reasoning mechanisms, so that it can be checked whether this representation is consistent and complete with respect to the task to be solved. A major difficulty of this way of doing is ensuring the faithfulness of the formal representation with respect to the mediating representation. Checking faithfulness needs a validation and a revision cycle, which is usually long: while the expert understands the mediating representation, this representation cannot be checked, and he/she does not understand the formal representation provided with the reasoning tools. One of the solutions proposed in Shaw and Gaines (1995), Bos *et al.* (1997) is to create mediating representations that are formal: expressing the representation in a language that is both understandable by human beings and formal and in this way ensuring expert simulation at an early modeling stage. This is precisely the qualities claimed by our graph-based language detailed in Section 2.

However, for a KBS to be really used in practice, an essential point is that the user understands and controls the *whole process* whose main steps are not only building a KB and running the KBS (as previously discussed) but also obtaining results. By user, we either mean an end-user, who may be a knowledge engineer, who builds the KB, or an expert of the application domain, who uses the implemented representation to check its conformity with her own representation of the domain, or a user for whom the system has been built and who wants to solve real problems. It should be easy for this user not only to enter different pieces of knowledge and to understand their meaning but also to understand the results of the system and how the system computed these results. The last point, namely the ability of understanding why the system gives a certain answer, is especially important since the user computing expertise may vary. Furthermore, for any domain and level of expertise, explaining to the user each step that makes up the logical inference, generally remains a difficult process.

A graph-based language. In this paper, by 'graph' we understand the classical mathematical notion in *graph theory*, that is, a structure that consists of a set of nodes (also called vertices) and a set of edges that establish relationships (connections) between nodes. Please note that, regrettably, 'graph' in elementary mathematics also refers to a function graph, that is, a plot.

In the proposed graph-based approach, where all pieces of knowledge are represented by labeled graphs, *the same language* is used at all levels and for all KBS functionalities. The benefits of using graphs for representing knowledge at all levels of the KBS stem from the following:

- First, graphs are simple mathematical objects (they only use elementary naive set theory notions such as elements, sets and relations), which have graphical representations (a set of points and lines connecting some pairs of points) and thus can be visualized.

- Second, there is a rich collection of efficient algorithms for processing graphs, thus graphs can be used as effective computational objects (they are widely used, for instance, in Operational Research).
- Third, graphs can be equipped with a logical semantics: the graph-based mechanisms they are provided with are sound and complete with respect to deduction in the assigned logic.

Furthermore, graph-based mechanisms can be explained to the user because they can be easily visualized on the graphs themselves, either as a sequence of very simple operations or as a 'global' operation. This point will be further detailed in Section 3.

Semantic Networks. In general, KR languages rely on a purely textual representation with strict syntactic and semantic rules. Domain concepts, their properties, relations and restrictions, are all represented by words and sentences of the representation language. Textual communication is often supplemented with visual properties (such as character types, styles, structure, or layout), but a knowledge representation language can be regarded as visual only if it is based on a pictorial expression. The human short-term memory is limited, but visual organizations enable brain sensory information storage and ability to break down complex structures into more easily manageable chunks. Due to their visual qualities, semantic networks, which were originally developed as cognitive models, have been used for knowledge representation since the early days of AI, especially in natural language processing.

The term *semantic network* encompasses an entire family of graph-based visual representations. Since Nude (Richens, 1956) and the semantic network T (Masterman, 1962), which concern natural language processing, many semantic networks systems have been introduced (cf. Lehman, 1992) for a collection of papers concerning various families of network-based structures). They all share the basic idea of representing domain knowledge using a graph, but there are differences concerning notation, as well as rules or inferences supported by the language. In semantic networks, the *diagrammatical reasoning* is mainly based on path construction in the network. We can distinguish two major families of languages born in the 1980s. Let us start with the KL-ONE family. In addition to the large number of systems implementing KL-ONE variants (Woods & Schmolze, 1992), KL-ONE is considered as the ancestor of Description Logics (DLs) (Baader *et al.*, 2003), which are nowadays the most prominent KR languages dedicated to reasoning on ontologies. However, DLs have lost their graphical origins. Second, Conceptual Graphs. They were introduced by Sowa (cf. Sowa, 1976, 1984) as a diagrammatic system of logic with the purpose 'to express meaning in a form that is logically precise, humanly readable, and computationally tractable' (cf. Sowa, 1984). Throughout the remainder of 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 (cf. Chein & Mugnier 2009).

Paper organization. The sequel of this paper is structured as follows. Section 2 presents the main syntactic constructs of the language. Section 3 presents the elementary graph-based reasoning mechanisms and explains how labeled graph homomorphism encompasses all elementary operations in one global operation. Please note that, in this paper, we do not enter into precise definitions and notations but rather rely on visual intuition (see Chein & Mugnier, 2009 for more details and technical developments). An application scenario is presented in Section 4. We conclude the paper with Section 5 that presents related work in the domain and lays down future work directions. All figures depict graphs drawn using the conceptual graph editor Cogui[1]. Please note that Cogui is also fully integrated with the conceptual graph engine Cogitant[2] to perform reasoning on the above mentioned graphs.

[1] http://www.lirmm.fr/cogui/

[2] http://cogitant.sourceforge.net/

2 Using graphs for representation: the good

In our approach, all kinds of knowledge are encoded 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 (see Figures 1 and 2).
- 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 entity nodes to relation nodes. These nodes are labeled by elements in the vocabulary (see Figure 3).

These graphs have a semantics in first-order logic (FOL), that is, a KB can be translated into a set of FOL formulas. Reasoning tasks operate directly on the knowledge defined by the user and not on their translation into logical formulas. Stated in an other way, the logical semantics is only used to formally ground the graph model, that is, representation and reasoning mechanisms.

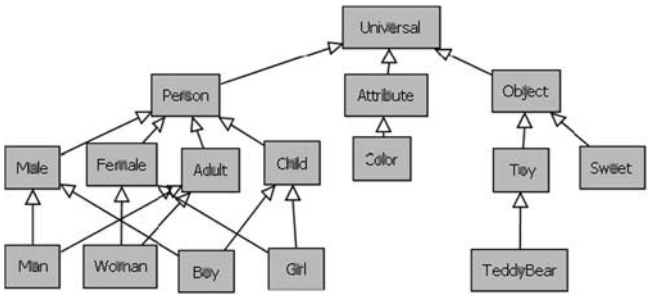

Figure 1 Basic ontology concepts for the childhood domain

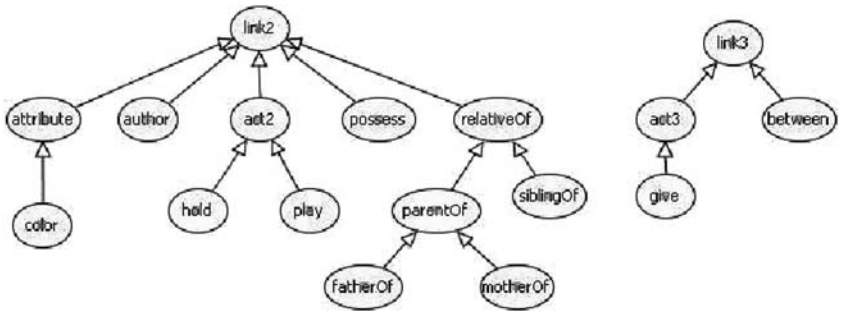

Figure 2 Basic ontology relations for the childhood domain

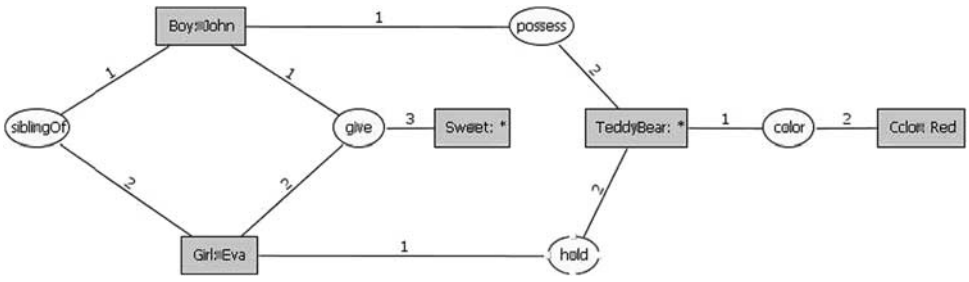

Figure 3 A situation described by a basic graph (*H*)

This makes it possible to explain reasoning to the end-user because it can be visualized in a natural way on the pieces of knowledge he/she is familiar with. Would a logical prover be used on the logical translation of these pieces of knowledge to compute reasoning, reasoning would become a black box for the user and could not be explained.

2.1 Conceptual 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'). Figures 1 and 2 show parts of these hierarchies visualized by their Hasse diagram ($t' \leq t$ if there is a path from t' up to t). For instance, the concept TeddyBear is a specialization of the concept Object, because of the path (TeddyBear, Toy, Object); the relation siblingOf is a specialization of the relation link_2 (which stands for any binary relation) because of the path (siblingOf, relativeOf, link_2). Note that a hierarchy is not necessarily a tree: for instance, there are two paths from Woman to Person, namely (Woman, Female, Person) and (Woman, Adult, Person).

Names of specific individuals can also be included in the vocabulary. The vocabulary can be further enriched by *signatures* for relations indicating the maximal concept that can be assigned to each of the relation arguments (e.g. the first argument of the relation motherOf is of maximal type Woman and its second argument is of maximal type Person). It can also contain statements of *disjointness* between concepts (e.g. the two types Object and Person are disjoint).

2.2 Basic graphs

A basic conceptual 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. For example, the BG in Figure 3 graphically represents the following situation: 'The boy John is a sibling of the girl Eva. John is giving a sweet to Eva, who is holding a red teddy bear belonging to John'.

A concept node is labeled by a couple $t:m$ where t is a concept (and more generally, a list of concepts) called the type of the node, 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. For example, in Figure 3 the node [`Sweet : *`] refers to 'a' sweet, while the node [`Boy : John`] refers to 'the' boy John. A relation node is labeled by a relation r called its type, and, if k is the arity of r, this node is incidental to k totally ordered edges. For example, in Figure 3 the relation node of ternary type *give* has three incidental edges and the order on these edges allows to distinguish between the *agent* of the gift act [`Boy : John`], its *recipient* [`Girl : Eva`] and its *object* [`Sweet : *`]. Classically, concept nodes are drawn as rectangles and relation nodes as ovals and the order on edges incidental to a *k-ary* relation node are numbered from 1 to k.

BGs are used to represent assertions called *facts*. They are also building blocks for more complex kinds of knowledge, as outlined in the next section.

2.3 More complex graphs

In the following we present two examples of BG extensions: *nested graphs*, which allow to structure facts by level of detail; and *inference rules*, which enrich the basic ontology with general knowledge about a domain.

Nested graphs. In a nested graph, a concept node may itself contain a (nested) graph, whose role is to further describe the entity represented by the node. This allows to distinguish between internal

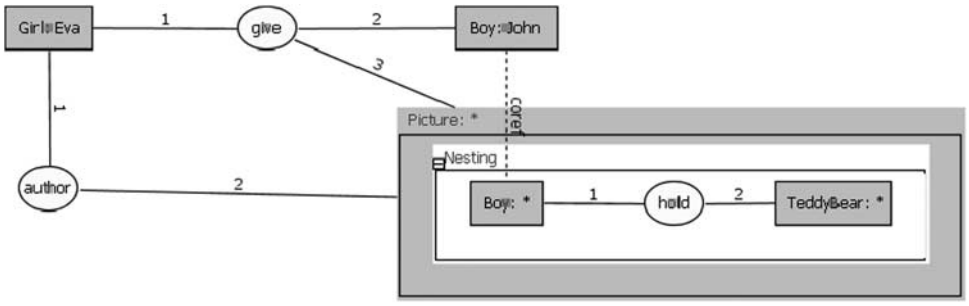

Figure 4 A nested graph

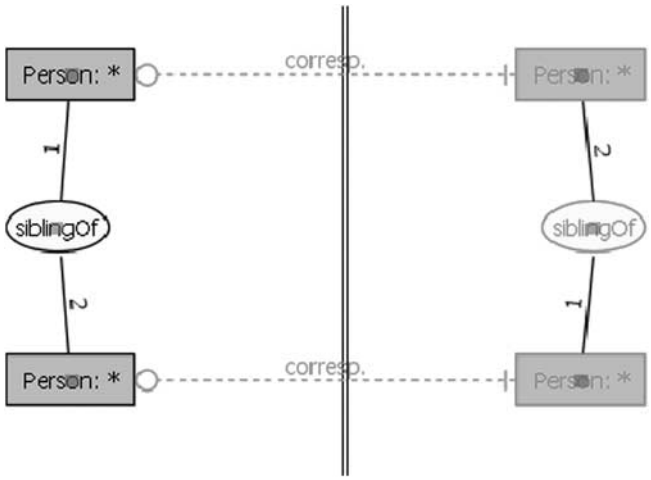

Figure 5 A rule (R_1)

and external information about an object, to represent zooming into an object or to contextualize the description of an object. For instance, let us consider the nested graph in Figure 4. At the outermost level, this graph says that 'Eva is giving a picture that she did to John' (note, however, that time is not represented here) and the graph nested in the node referring to the picture further describes this picture: 'this picture shows a boy holding a teddy bear'. The dotted line is a *coreference link*: it links two nodes that refer to the same entity (in this example the boy called John). The information that the picture has been done by Eva can be seen as an *external* information about the picture, while the detail of what is in the picture can be seen as an *internal* information, that can be obtained by zooming into the picture. It can also be said that the piece of information nested in a node is relevant within the context represented by this node (here, John is holding a teddy bear in the context of the picture, but it may not be true in the outermost context).

Rules. A rule expresses implicit knowledge of the form 'if *hypothesis* then *conclusion*', where hypothesis and conclusion are both basic graphs. This knowledge can be made explicit by applying the rule to a specific fact: intuitively, when the hypothesis graph is found in a fact, then the conclusion graph can be added to this fact (see Section 3.4 for more details). There is a one to one correspondence between some concept nodes in the hypothesis with concept nodes in the conclusion. Two nodes in correspondence refer to the same entity. These nodes are said to be *connection nodes*. For instance, Figures 5 and 6 present two rules, with the hypothesis on the left hand side and the conclusion on the right, separated by a vertical line. In Figure 5, rule R_1 says that 'if a person is a sibling of a person, then the inverse relation holds between these persons'. More formally: 'for all persons x and y, if x is a sibling of y then y is a sibling of x' (all concept nodes are connection nodes). In Figure 6, rule R_2 says that 'for all persons x and y, if x is a sibling

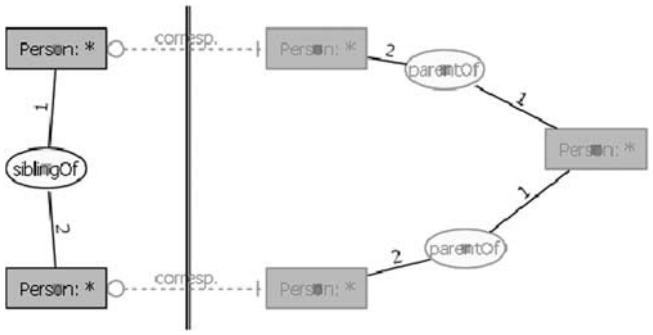

Figure 6 Another rule (R_2)

of y, then they have a common parent, that is, there is a person who is a parent of x and of y' (the node representing this person is not a connection node, since it is not in correspondence with a node in the hypothesis). Let us add that rules can also be defined as pairs of nested graphs instead of basic graphs.

All these graphical objects, that is, the vocabulary as well as basic graphs, nested graphs and rules, are provided with a semantics in FOL. This semantics specifies the meaning of knowledge constructs and allows to show the correctness of the associated graph mechanisms with respect to logical deduction (see Section 3.3).

3 Using graphs for reasoning: the better

Different kinds of reasoning concerning BGs can be graphically defined, for example, applying inference rules or contextual reasoning. These reasonings are based on a *subsumption* relation between conceptual graphs. This section is devoted to the presentation of this fundamental reasoning notion in the simplest case, that is, BGs.

Let G and H be two BGs over the same vocabulary. Intuitively, G subsumes H if the fact—or the information—represented by H entails the fact represented by G, or in other words, if all information contained in G is also contained in H.

A query-answering mechanism using BGs and subsumption can be defined as follows. Let us consider a KB B composed of a set of BGs, representing some assertions about a modeled world, for example, the fact in Figure 3. Elements in B answering a query Q are intuitively defined as the elements that entail Q, or equivalently, elements that are specializations of Q, or also, elements that are subsumed by Q. Let us consider for instance the query in Figure 7. This query is easily visualized as a graph, but is more complex to express textually: it asks for a situation where a boy and a girl, who is one of the boy's relatives, are each in relation with a red toy. We will see hereafter why and how the fact in Figure 3 answers this query.

Relationships with logics is mentioned at the end of this section and we will see that the subsumption relation exactly corresponds to deduction in a fragment of FOL. The subsumption relation can be defined either by a sequence of elementary operations or by the classical homomorphism notion applied to BGs. Both are very easily visualizable and they are defined below by means of drawings.

3.1 Generalization and specialization operations

There are five elementary generalization operations and five inverse operations, called elementary specialization operations. The terms *generalization* and *specialization* are used here with the following intuitive meaning: let I be a piece of information; whenever a piece of information is added to I, the obtained information is a specialization of I (it contains more specific knowledge) and, conversely, deleting a piece of information from I yields a final piece of information that is more general than I (it contains less precise knowledge).

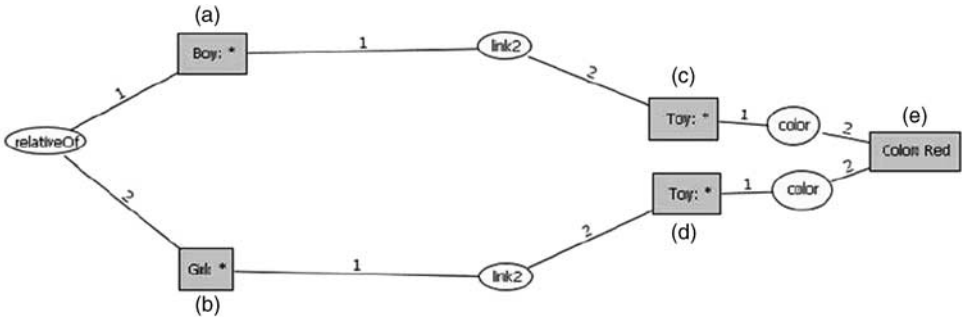

Figure 7 A query described by a basic graph (G)

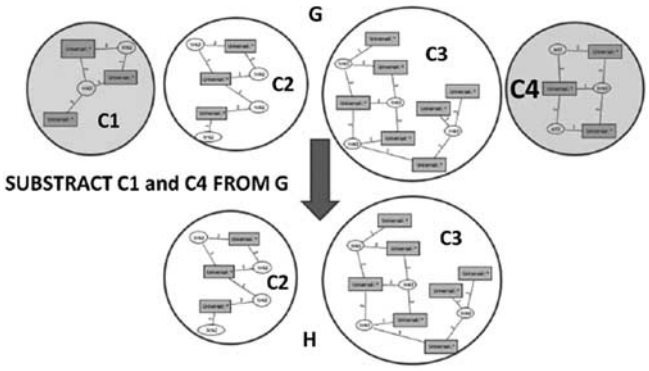

Figure 8 Substract

3.1.1 Elementary generalization operations for BGs

Any generalization operation is a 'unary' operation, that is, it transforms a BG into another BG. It can be pictured by a drawing representing the transition from the input BG to the output BG. Generalization has to be taken in a broad sense, that is, the BG obtained may contain a strictly more general information than the initial BG or the same information.

The elementary generalization operations can be graphically defined as follows:

- **Substract** (Figure 8). The *substract* operation consists of deleting some connected components of a BG. The BG obtained is clearly more general than (or equivalent to) the original BG, since a piece of information is deleted.
- **Detach** (Figure 9). The *Detach* operation consists of splitting a concept node c into two concept nodes c_1 and c_2, the edges incident to c being shared between c_1 and c_2. For instance, let us consider the detachment of the generic concept node c in Figure 9. In the initial BG it is said that 'there is a boy who possesses something and is giving something to a person', and in the resulting BG it is said that 'there is a boy who possesses something and there is a boy giving something to a person'. In the resulting BG, the two boys may be different, while in the initial BG it is necessarily the same boy. Therefore, the final situation is more general than the initial situation.
- **Increase** (Figure 10). The *Increase* operation consists of increasing the label of a concept or relation node. In the case of a concept node, it means that one can increase its type, for example, replacing 'the girl Eva' by 'the person Eva' and/or replace an individual marker by the generic marker, for example, transforming 'the person Eva' into 'a person'. Formally, the generic marker is considered as greater than all individual markers, which are pairwise non-comparable. Thus, increasing a concept node label consists of increasing its type and/or its marker. Clearly, 'a person' is more

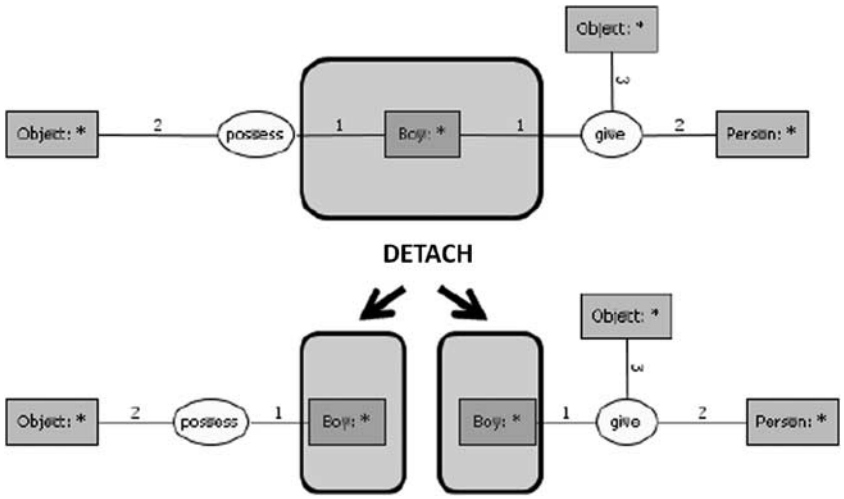

Figure 9 Detach

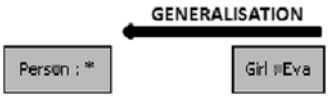

Figure 10 Increase

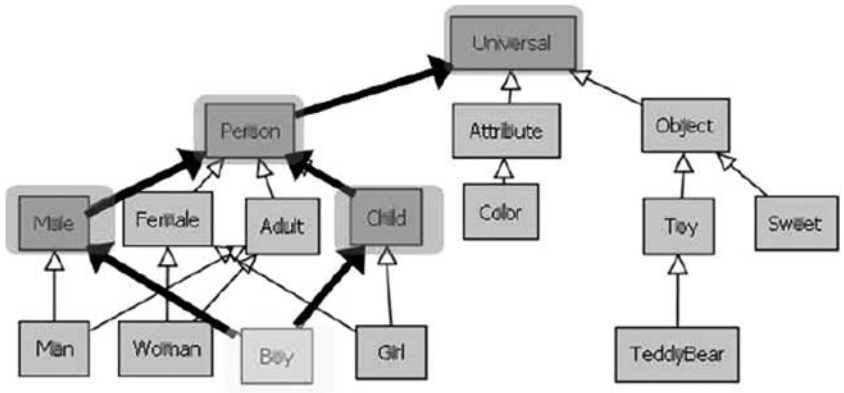

Figure 11 Types greater than or equal to the type Boy

general (in an intuitive sense) than 'the girl Eva'. In the same way, replacing a relation type by a greater type is also clearly a generalization operation, for example, 'John is a relative of Eva' is more general than 'John is a sibling of Eva'.

Let us remark that checking if a type t is greater than another type t' is the most frequent operation in hierarchies. It is a basic operation (t is $>t'$ if and only if there is a path from t' to t) for which efficient algorithms have been developed (cf. Chein & Mugnier, 2009). In Figure 11, the bold paths contain all types greater than the type Boy.

- **Relation duplicate** (Figure 12). The *Relation duplicate* operation consists of duplicating a relation node, that is, adding a new relation node r' having the same type and the same list of arguments as a relation node r. r and r' are said to be *twin* relation nodes.
- **Copy** (Figure 13). The *Copy* operation consists of duplicating a whole BG, that is, adding to it an isomorphic and disjoint copy of it.

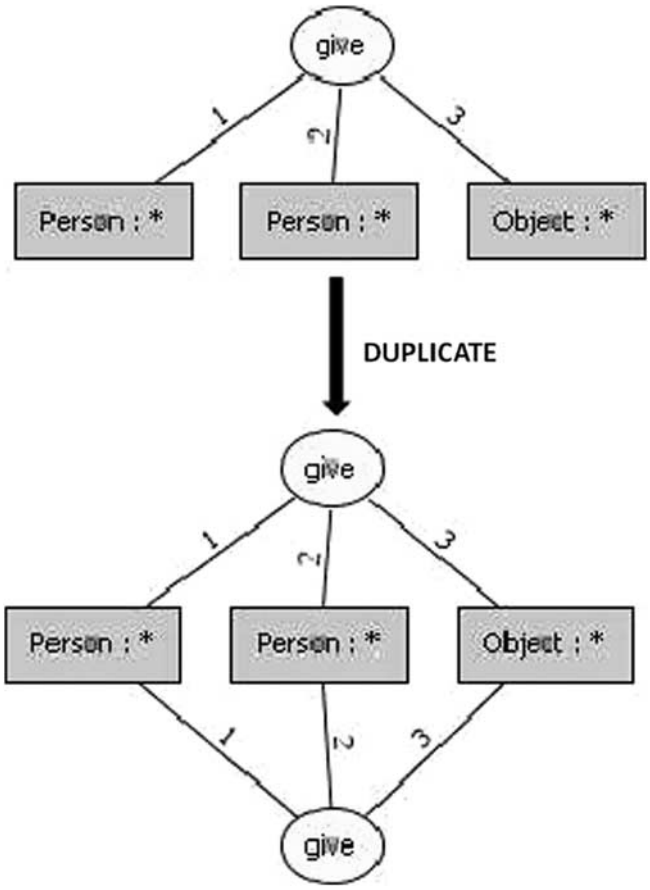

Figure 12 Duplicate

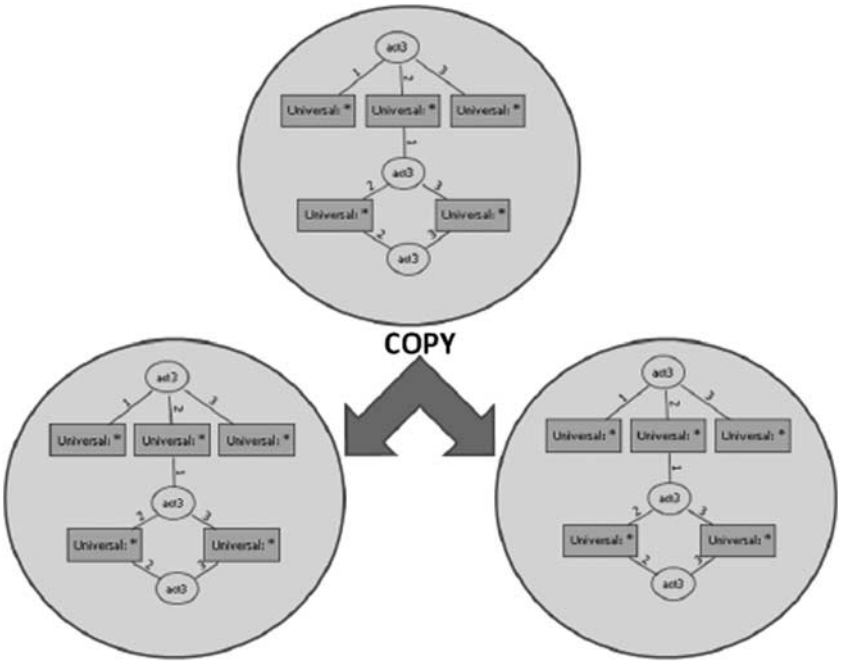

Figure 13 Copy

In the last two operations, the structure of the obtained BG contains the structure of the initial BG (one adds something), thus, at first glance, one can think that they are specialization operations. Nevertheless, as these operations duplicate *already existing information*, the BG obtained is semantically equivalent to the initial one, thus they are also generalization operations (in a broad sense).

We can now precisely define what '*G is a generalization of H*' means: G is a generalization of H if there is a sequence of elementary generalization operations leading from H to G, that is, there is a sequence of BGs $H_0 = (H), H_1, \ldots, H_n = (G)$, such that for all $i = 1, \ldots, n$, H_i is obtained from H_{i-1} by an elementary generalization operation.

Figures 14–18 present some of the graphs occurring in a generalization sequence from H, the BG in Figure 3 to G, the BG in Figure 7. H_1 is obtained from H by splitting both nodes [`Girl : Eva`] and [`Boy : John`] (two *Detach* operations). H_2 is obtained from H_1 by deleting the

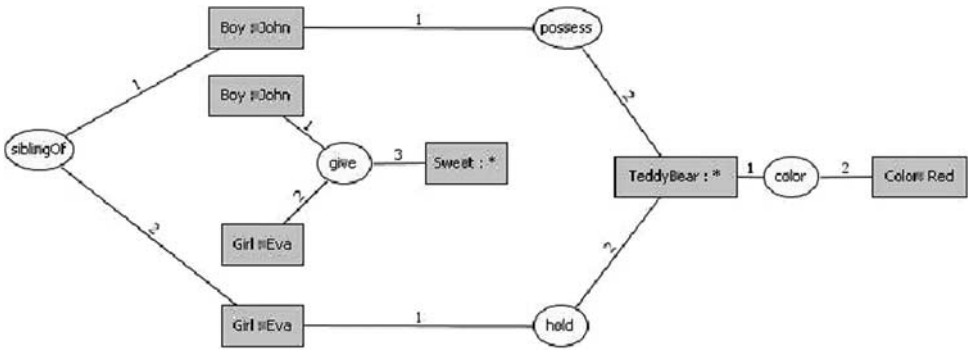

Figure 14 Generalization from H to G—step I (H_1)

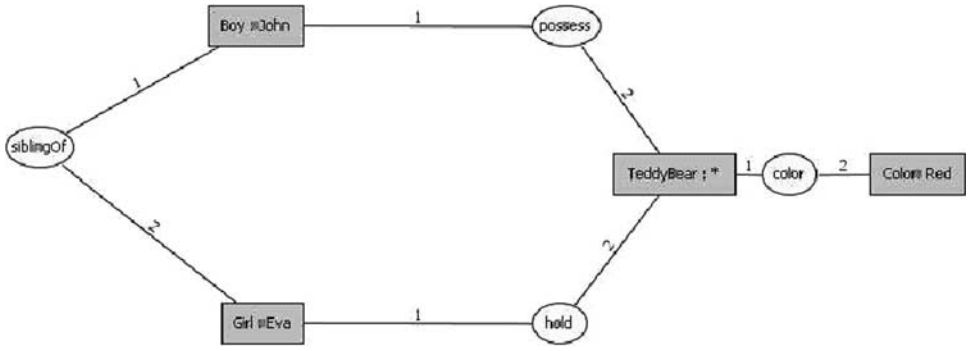

Figure 15 Generalization from H to G—step II (H_2)

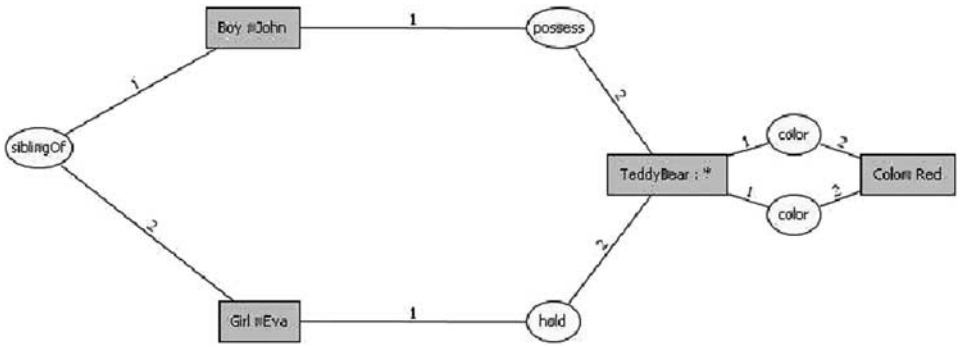

Figure 16 Generalization from H to G—step III (H_3)

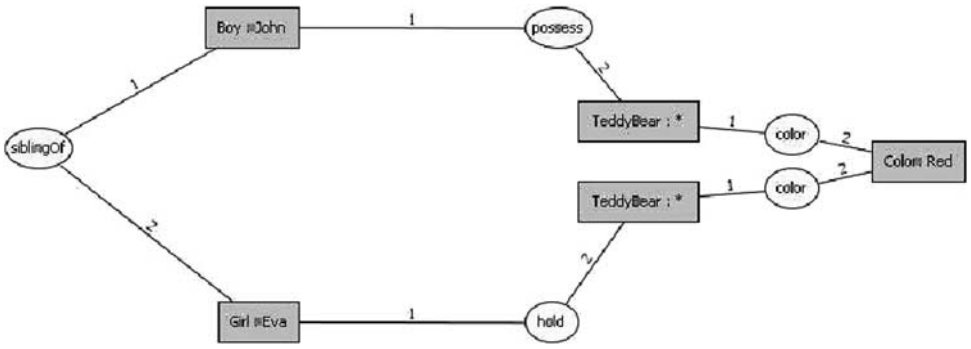

Figure 17 Generalization from H to G—step IV (H_4)

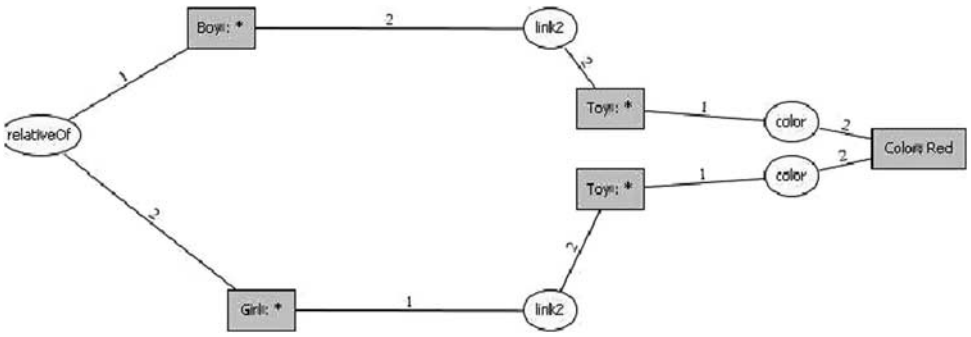

Figure 18 Generalization from H to G—step V (G)

connected component containing the *give* relation node (*Substract* operation). H_3 is obtained from H_2 by duplicating the relation node *color* (*Relation duplicate* operation). H_4 is obtained from H_3 by a *Detach* operation on the node [`TeddyBear:*`]. G is obtained from H_4 by a sequence of seven *Increase* operations: the relation *siblindOf* is replaced by *relativeOf*, the relations *possess* and *hold* are replaced by *link*2; the individual concept nodes [`Boy : John`] and [`Girl : Eva`] are made generic, and the teddy bears become toys.

3.1.2 Elementary specialization operations

We have seen in the previous section that the elementary generalization operations are indeed generalization operations in an intuitive manner, they are easy to draw and to understand, and they allow for a precise definition of the subsumption relation between BGs. Nevertheless, let us consider again the query-answering problem previously stated using generalization. Let Q be a BG query and B a set of BG facts. Answering Q consists of looking for subgraphs of BGs in B that are subsumed by Q, that is, such that Q generalizes them. It seems more intuitive and it is more efficient to start from the query Q and to look for specializations of Q in B. This can be done by defining elementary specialization operations, which are the inverse of the elementary generalization operations defined previously.

- **Disjoint sum** (Figure 19). Given two disjoint BGs, their *Disjoint sum* is the BG obtained by juxtaposing two copies of these BGs. This operation is the inverse of the *Substract* operation.
- **Join** (Figure 20). Given a BG, joining two concept nodes with the same label in this BG consists of merging them. This operation is the inverse of the *Detach* operation.
- **Restrict** (Figure 21). *Restrict* consists of decreasing the label of a concept or relation node. This operation is the inverse of the *Increase* operation.
- **Relation simplify**(Figure 22). This operation consists of deleting a twin relation node. This operation is the inverse of the Relation *duplicate* operation.
- **Copy.** This operation has already been defined as a generalization operation.

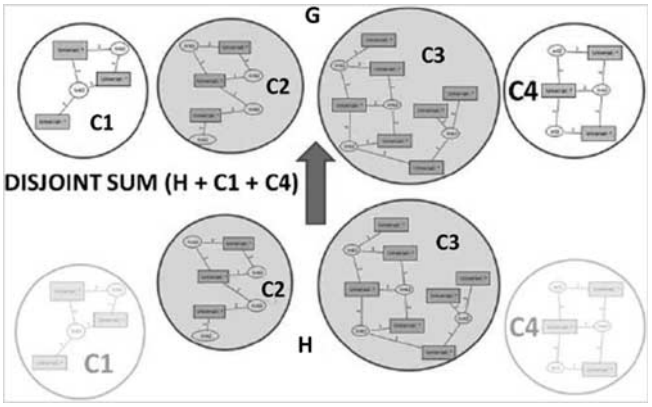

Figure 19 Disjoint sum

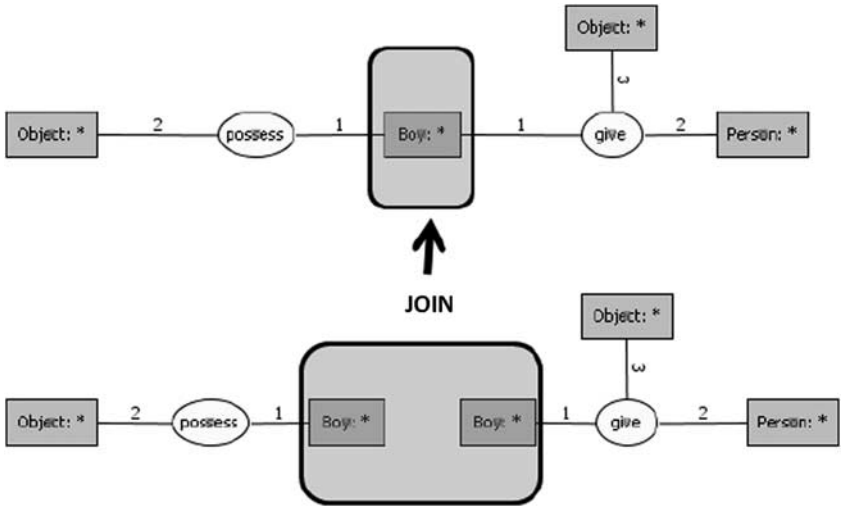

Figure 20 Join

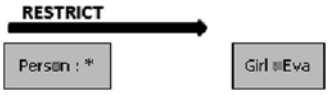

Figure 21 Restrict

H is a *specialization* of *G* if *H* can be obtained from *G* by a sequence of elementary specialization operations. In reading Figures 14–18 backwards, one obtains a specialization sequence from *G* to *H*. This sequence begins with a set of *Restrict* operations (leading from *G* to H_4). Then, H_3 is obtained from H_4 by a *Join* operation. H_2 is obtained from H_3 by *Relation Simplify*. H_1 is obtained from H_2 by a *Disjoint sum* operation, and *H* is obtained from H_1 by two *Join* operations.

The following property is straightforward to check:

G is a generalization of H if and only if H is a specialization of G.

3.2 Homomorphism

We have seen specialization operations, which are more convenient than generalization operations when considering the query-answering problem. Now, the problem is how to *find* a sequence of specialization operations from a BG to another BG.

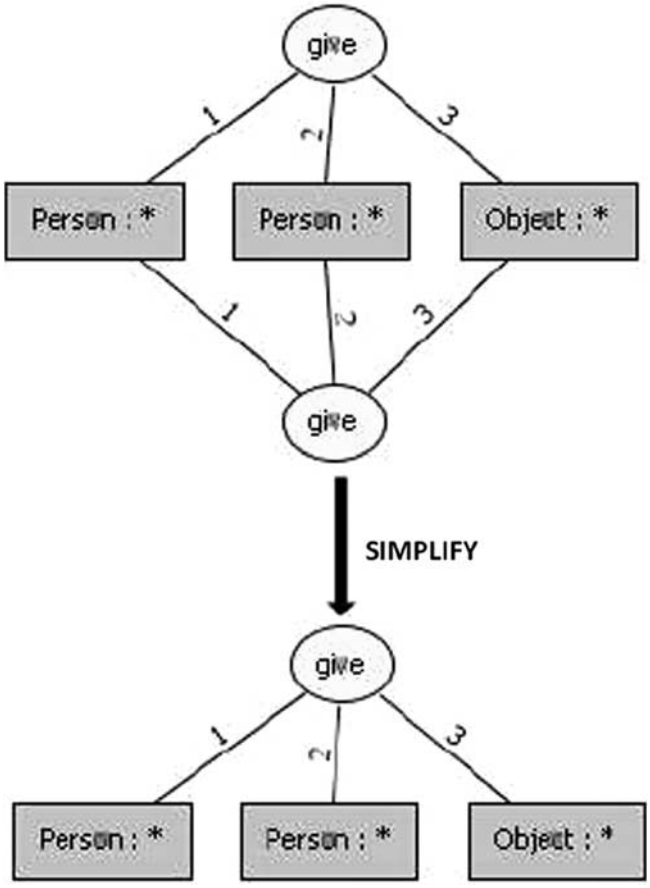

Figure 22 Simplify

In this section, we introduce the homomorphism notion between BGs. A homomorphism from G to H is a mapping from the node set of G to the node set of H, which preserves the adjacency between nodes of G and can decrease the node labels. If there is a homomorphism (say π) from G to H, we say that G *maps* to H (by π).

Let us consider again the query G in Figure 7 and the fact H in Figure 3. There is a homomorphism from G to H, which is pictured by the dashed lines in Figure 23. The concept node $a = [\texttt{Boy} : *]$ in G is mapped to [Boy : John] in H, the concept node $b = [\texttt{Girl} : *]$ in G is mapped to [Girl : Eva] in H, both concept nodes $c = [\texttt{Toy} : *]$ and $d = [\texttt{Toy} : *]$ in G are mapped to the same node [TeddyBear : *] in H, the concept node $e = [\texttt{Color} : \texttt{Red}]$ in G is mapped to [Color : Red] in H, the relation node (*relativeOf*) is mapped to the relation node (*siblingOf*), the relation node (*link*2) from a to c is mapped to the relation node (*possess*) and the other relation node (*link*2) from b to d is mapped to the relation node (*hold*); finally, both relation nodes (*color*) are mapped to the same node (*color*).

A BG homomorphism can be more precisely defined as follows. A *homomorphism* π from G to H is a mapping from the concept node set of G to the concept node set of H and from the relation node set of G to the relation node set of H, which preserves edges and may decrease concept and relation labels, that is:

- for any edge labeled i between nodes c and r in G, there is an edge labeled i between nodes $\pi(c)$ and $\pi(r)$ in H; in other words, if a relation r has neighbors $c_1 \ldots c_k$ (in this order) then its image $\pi(r)$ has neighbors $\pi(c_1) \ldots \pi(c_k)$ (in this order).
- for any (concept or relation) node x in G, the label of its image π (r) in H is less than or equal to the label of x.

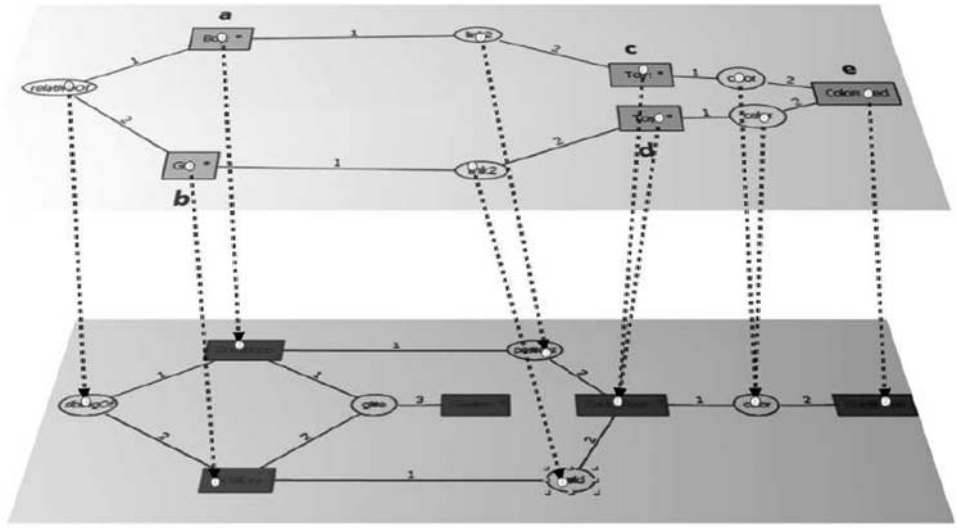

Figure 23 A homomorphism from G to H

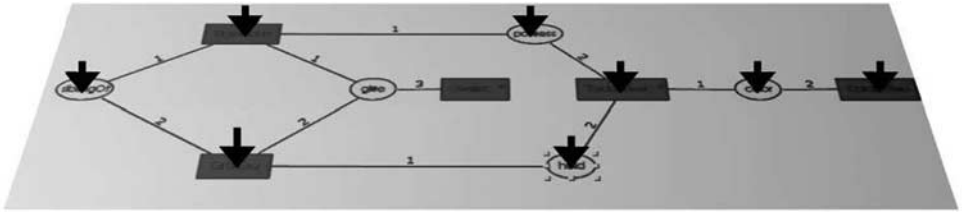

Figure 24 Image of G by the homomorphism in Figure 23

Let us consider again the homomorphism in Figure 23. Figure 24 highlights the subgraph of H induced by the nodes that are images of nodes in G. This subgraph is called the 'image graph' of G by this homomorphism.

The following theorem holds. Given two BGs G and H, the three following propositions are equivalent:

1. G is a generalization of H
2. H is a specialization of G
3. There is a homomorphism from G to H

Even though it is easy to visualize a homomorphism and to check if a mapping between two BGs is a homomorphism (cf. Figure 23), this global graph matching can be replaced, if needed, for more detailed explanation purposes, by a sequence of elementary operations (cf. Figures 14–18). Assume for instance that the answer to the query G is visualized as in Figure 24 and the user wants to understand why the *image graph* of G is indeed a specialization of G. The homomorphism can be decomposed into a sequence of elementary specialization operations, starting from G, as follows:

- First, a sequence of Restrict showing how the label of each node of G is specialized (cf. the transformation from G to H_4);
- Second, a sequence of Join showing which concept nodes are merged into the same node (these are the nodes in G with the same image by the homomorphism, cf. the transformation from H_4 to H_3);
- Third, a sequence of Relation simplify removing relation nodes that have become redundant (cf. the transformation from H_3 to H_2).

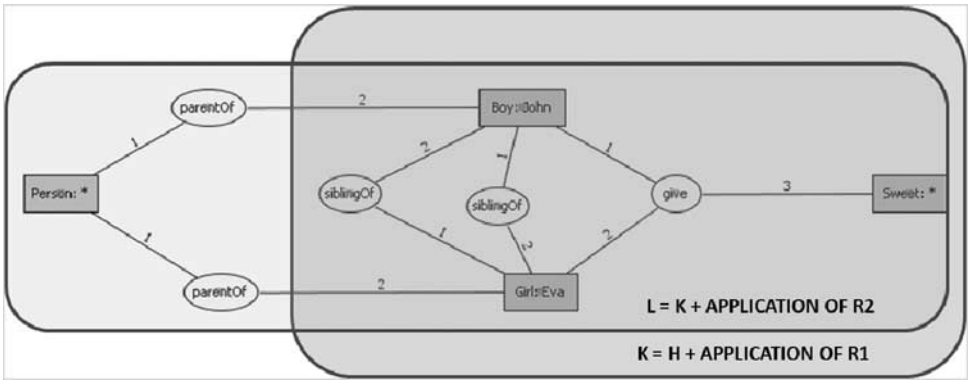

Figure 25 Applying rules

After these three steps, the image graph of G is obtained, which is sufficient to show that the fact contains a specialization of G. To build the fact graph itself from the image graph of G, one would need a disjoint sum (cf. the transformation from H_2 to H_1) and some joins (cf. the transformation from H_1 to H).

3.3 Logical correctness

Until now, we have introduced the reasoning operations (generalization, specialization and subsumption) using simple graphical operations. It remains to relate these notions to usual reasoning notions, that is, essentially, to relate BG subsumption to logical deduction. This can be done using the logical semantics of the knowledge constructs. This semantics is defined by a mapping from graphical objects to logical formulas. It is classically denoted by Φ in conceptual graphs (Sowa, 1984). The fundamental theorem states that given two BGs G and H built on a vocabulary $\mathcal{V}$, there is a homomorphism from G to H if and only if $\Phi(G)$ is a semantic consequence of $\Phi(H)$ and $\Phi(\mathcal{V})$ (this is a soundness and completeness theorem of BG homomorphism w.r.t. FOL entailment, cf. Chein and Mugnier (1992)). Let us point out that BGs are in fact equivalent to the positive, conjunctive and existential fragment of FOL.

Once again, logic is used to give a semantics to BGs, but not to reason with them. As detailed before, one motivation for preferring graph-based reasoning is its visual aspects permitting to understand (logically based) reasoning without doing logic. Another motivation is that BGs have good computational properties. Efficient graph-based deduction algorithms have been built, which are not the translations of logical procedures (see f.i. Chein & Mugnier, 2009).

3.4 Overview of reasoning on more complex pieces of knowledge

Previous generalization and specialization elementary operations, as well as the corresponding homomorphism notion, can be extended to nested graphs, while preserving soundness and completeness with respect to deduction in the associated fragment of FOL.

Let us now consider the graph rules presented in Section 2. A rule R can be applied to a BG H, if there is a homomorphism from its hypothesis to H. Applying R to H according to such a homomorphism π consists of 'attaching' to H the conclusion of R by merging each connection node in the conclusion with the image by π of the corresponding connection node in the hypothesis. See for instance the graph H in Figure 3 and the rules R_1 and R_2 in Figures 5 and 6. There is a homomorphism from the hypothesis of R_1 to H, thus R_1 can be applied to H, which yields the graph K (Figure 25). Similarly, R_2 can be applied to H, or to K. Let us apply R_2 to K: we obtain L (Figure 25). R_2 can be applied another time with a new homomorphism to L: however, this application would be redundant, since the part to be added is already present in L.

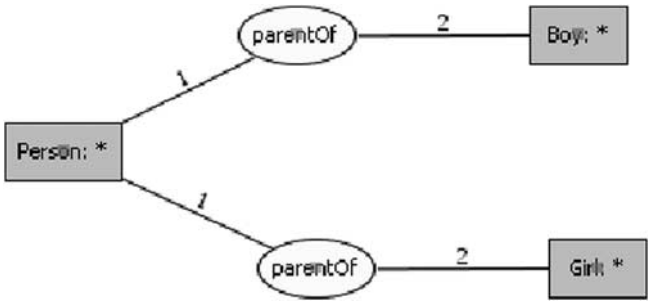

Figure 26 Another query example

When a KB contains a set of facts and a set of rules, the query mechanism has to take implicit knowledge coded in rules into account. The KB answers a query Q if a BG F' can be derived from the facts using the rules, such that Q maps to F'. Let us consider again the fact H in Figure 3 and the rules R_1 and R_2 in Figures 5 and 6 and let Q be the query in Figure 26 asking if there is someone who is parent of a boy and a girl: there is no homomorphism from Q to H; however, the KB containing H and the rules R_1 and R_2 answers Q: indeed Q maps to L (Figure 25), which is derived from H by applying the rules. This reasoning can be explained by visualizing a homomorphism from Q to L, as well as a sequence of rule applications allowing to add the knowledge involved in this homomorphism (for instance, the application of R_1 is not needed in our example).

Finally, let us point out that this rule application mechanism is sound and complete, that is, given a KB $\mathcal{K}$ and a BG Q, Q maps to a graph derived from $\mathcal{K}$ if and only if $\Phi(\mathcal{K}) \models \Phi(Q)$, where $\Phi(\mathcal{K})$ is the set of formulas assigned to the vocabulary, the set of facts and the set of rules composing $\mathcal{K}$.

4 Using graphs for applications: the best

In this section, we present a concrete AI application using the above described language for image annotation. The choice of the application is motivated twofold. First, this application clearly demonstrate the visual appeal of Conceptual Graphs from a representation depiction faithfulness viewpoint. Second, the visual reasoning capabilities allow for all levels of expertise when building, querying the KB and understanding why certain results will be returned.

Within the image annotation process, we will distinguish between *resources* (in this case electronic image files) and *metadata*. A metadata is a computational object always associated with a resource. Each resource is identified by an identifier, which is used in the metadata base for referencing the resource.

Metadata can be roughly categorized into two classes: *objective* metadata and *subjective* metadata. Examples of objective metadata include: resource address, authors name and the document size. Subjective metadata aims at representing information generally depending on the author of the metadata. Examples of subjective metadata include: the representation of the content of a resource (indexation of the resource), the representation of a comment, note, or remark, etc. In this case, an annotation is simply a piece of knowledge associated with a resource. In the following, we will present a Conceptual Graph approach for building and querying a KB aimed at annotating family photos. The KB used to illustrate notions throughout this section has been edited with the tool Cogui.

Such annotation is built from an ontology fundamentally composed of a hierarchy of concepts (or terms) and a hierarchy of relations between concepts. The ontology can also contain representations of other knowledge. Relation signatures indicate the types of relation arguments. Rules represent knowledge of the form 'if a piece of information is present (within an annotation) then further information can be added (to this annotation)'. Thus, rules can be used to automatically

Figure 27 Example of photo that needs to be semantically annotated

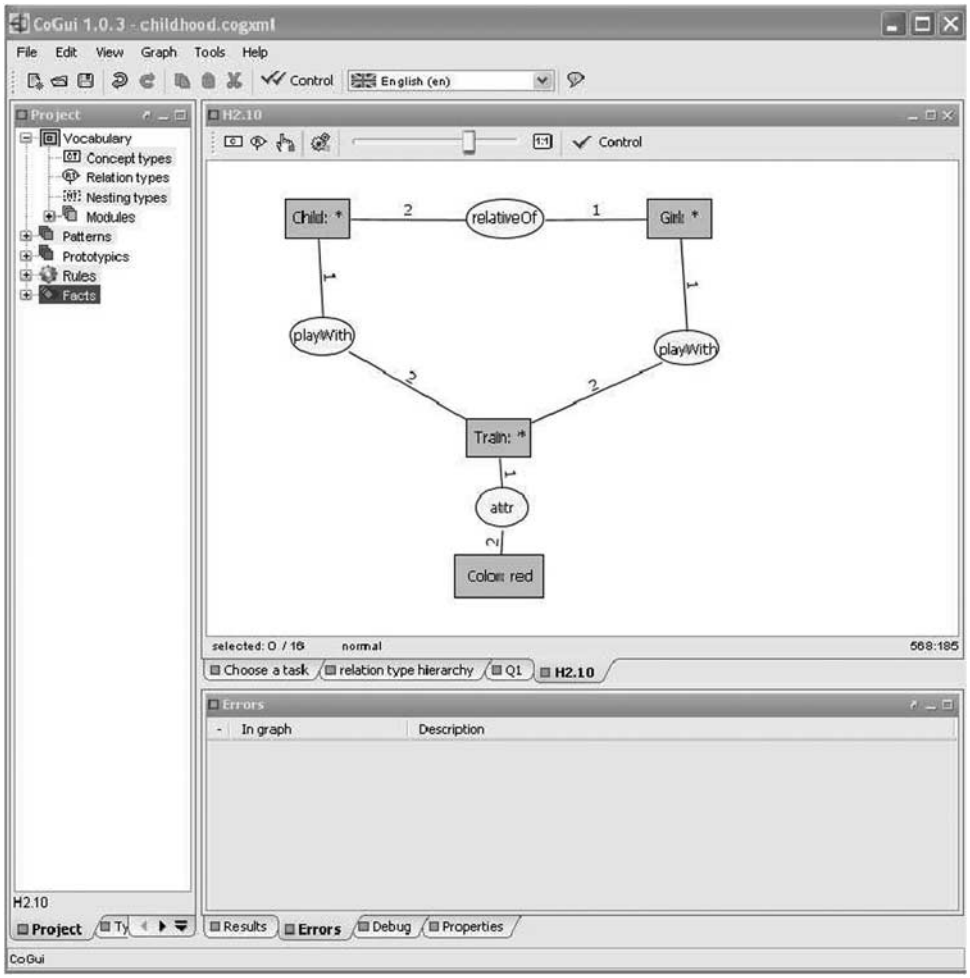

Figure 28 Example of a semantic annotation

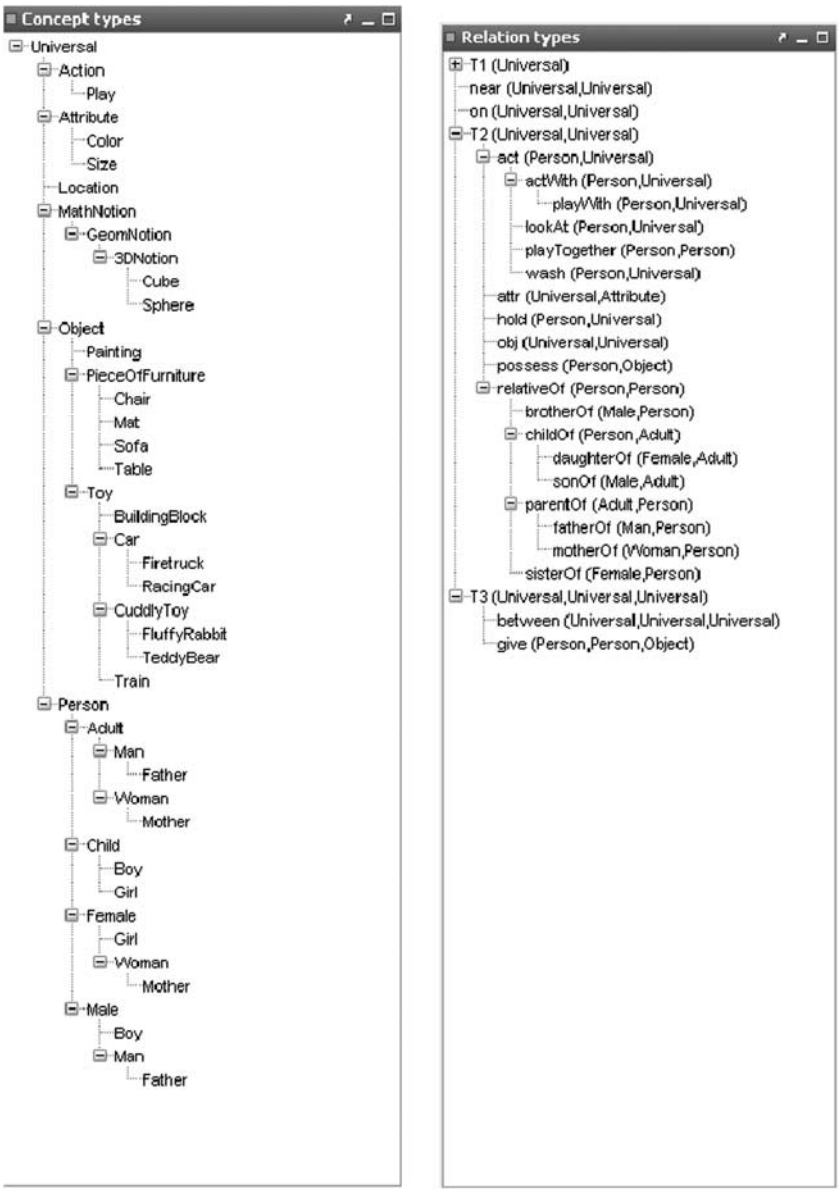

Figure 29 Concept/relation hierarchy for photo annotation

complete an annotation by implicit and general knowledge. Another kind of knowledge with the same syntactic form as rules but a different semantics are *constraints*. Constraints represent knowledge of the form 'if a piece of information is present (within an annotation) then other information cannot be present (in the annotation)'. Signatures and constraints are used to avoid the construction of absurd annotations. All these kinds of knowledge are represented by labeled graphs.

For clarity purposes, the example given in this paper is very simple. This framework have been successfully employed for annotation in the large-scale context of the LOGOS Framework 6 European Project (cf. for instance, Lalande *et al.*, 2009), as well as in other French projects (Genest & Chein 2005; Moreau *et al.*, 2007).

Let us consider the photograph in Figure 27. A semantic annotation of this image is depicted in Figure 28 where a fact represents a girl, the relative of a child, playing with the same red train that the child is playing with. As explained above, all of the concepts and the relations used in the facts need to be described and organized in the vocabulary. Figure 29 shows the concept and relation hierarchies purposely built for annotating family images.

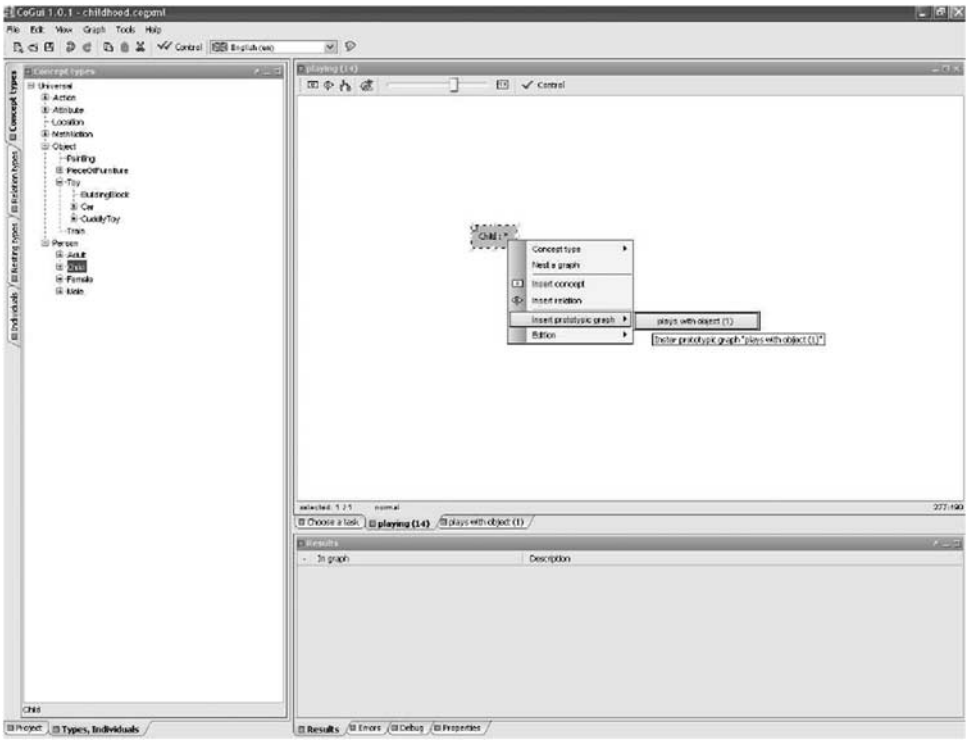

Figure 30 Prototypical graph insert for photo annotation

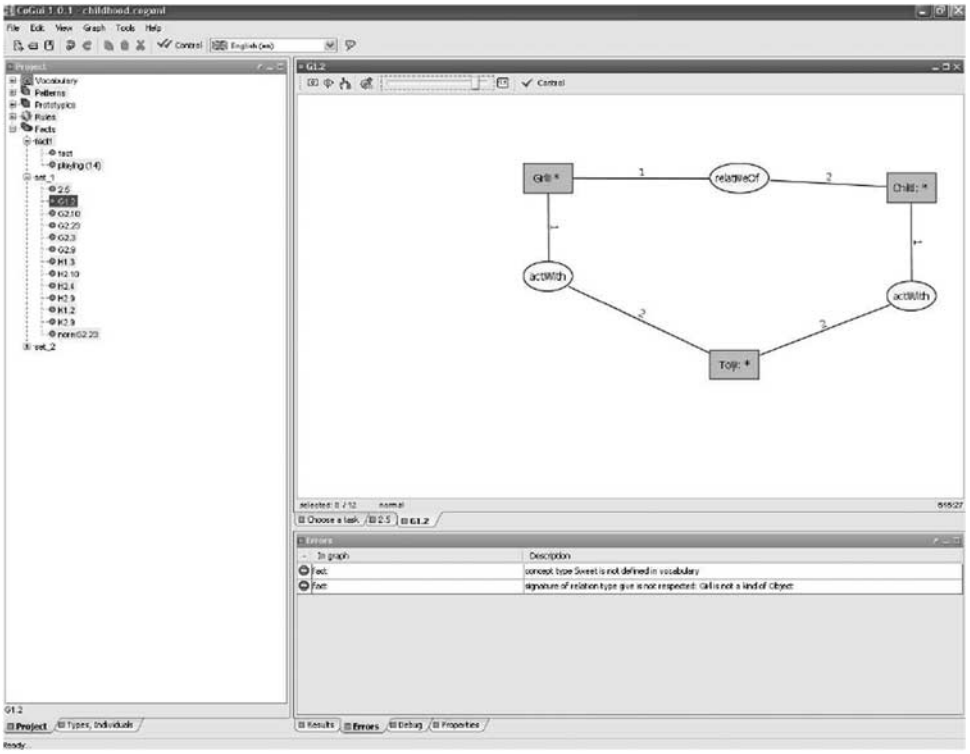

Figure 31 Query graph for photo annotation

Please note that the constructs introduced before (such as rules and nested graphs) are also used for annotation. Since certain chunks of knowledge appear often, other kinds of constructs have been introduced specifically for speeding up the annotation process, such as *prototype graphs*. For example, for a child we could always like to have annotated the fact that it is playing with a certain object. Notice this is not a rule, for example, it may apply in certain situation and not in others. Figure 30 represents how to insert the prototypical graph for a given concept.

As shown before, the user can then look for certain photos that contain a child and its relative, a girl, both acting with a toy. Such query is visually represented in Figure 31. Based on homomorphism (as previously explained) the query can be 'mapped' into the fact represented in Figure 28 and this will return the photo attached to it.

Note that we have only presented here the kernel of an information retrieval system, in which the search process is based on graph homomorphism. In order to take the intrinsic vagueness of information retrieval into account, that is, to search documents imprecisely and incompletely represented in order to answer a vague query, the exact search realized by graph homomorphism is not sufficient. Additional techniques based on graph transformations for doing approximate search and for ranking the answers to a query have been developed (cf. for instance Genest & Chein, 2005).

5 Related work and conclusion

In this section, we place our language in the landscape of graphical knowledge representation languages in order to enhance its original features. We will only detail each language in the light of its respective distinction with Conceptual Graphs. This choice is motivated by our aim of demonstrating the originality of our proposal in the context of graph-based languages for both knowledge representation and reasoning as opposed of doing a general synthesis of existing languages for KR.

An important criterium distinguishing Conceptual Graphs from other graph-based languages for knowledge modeling is the *reasoning* aspect. Indeed, numerous graphical languages have been proposed for data and knowledge modeling. Prominently among them, *UML* (Unified Modeling Language) unifies several previous languages and allows to model a system or a program by diagrams. However, this language does not have a denotational semantics and is not provided with inference mechanisms.

Furthermore, let us focus solely on graphical languages fundamentally dedicated to representing knowledge in the form of relationships between concepts and/or concept instances. The name 'map' is often used to describe mediating, thus informal, representations. A *concept map* is a diagram representing relationships between concepts, or more vaguely, ideas, images, words, etc. the aim being to organize informal knowledge (Novak & Canas, 2006). A *cognitive map* is a graphical representation of an influence network between notions. A *topic map* is a diagram representing associations between topics, and is more specifically dedicated to the description of resources for the Semantic Web. If all these languages provide graphical views of knowledge, none of them possesses a formally based reasoning mechanism.

Closer to our proposal, let us cite *RDF* (Resource Description Framework)[3], which is the basic Semantic Web language. RDF has a graph-based representation, it is provided with a formal semantics and an associated entailment notion (Hayes, 2004; Horst, 2004), but it does not come with an effective reasoning mechanism, even less with a graph-based mechanism that would operate on the graph representation.

Let us point out that for several of these languages, some form of formally founded visual reasoning based on graph homomorphism has been proposed: (Aissaoui *et al.*, 2003; Baget, 2005; Raimbault *et al.*, 2005; Carloni *et al.*, 2006; Chauvin *et al.*, 2008).

[3] `http://www.w3.org/TR/REC-rdf-syntax/`

We have already mentioned in Section 1 the semantic network family, and its successors, DLs, which are logically founded knowledge representation and reasoning languages, which have lost their diagrammatical aspects. In contrast, the Semantic Network Processing System (SNePS; see Shapiro, 1979, 2000), specially dedicated to the implementation of cognitive agents, remains graph based. It is provided with three kinds of inference mechanisms: a sound (but not complete) logic-based inference, as well as a path-based inference and a frame-based inference. All three kinds of inferences can be integrated, but there is no formal semantics for this combination.

Conceptual Graphs finally appears to be the only knowledge representation language both logic based and graph based, with logically sound and complete graph inference mechanisms (at least for the fragment developed here), thus allowing for visual reasoning. Another important feature of conceptual graphs is that relations can be of any arity (i.e. they can have any number of arguments), which allows for a more natural representation in many cases (for instance, when relations are extracted from data tables, their arity is the number of columns in the table). This latter feature is shared with topic maps, but none of the other languages mentioned above.

The sound and complete graph-based mechanisms for reasoning presented in this paper have been fully implemented and are available as part of the Cogui Editor and the CG reasoning engine Cogitant. While homomorphism proves to be an intuitive mechanism for query answering, more advanced graph-based tools could be envisaged to make this notion even more intuitive (see, e.g., the 3D manipulation of the images in Figure 23). This is not the only possible envisaged manipulation: different layouts, smooth zooming capabilities or colors could also be employed to increase the presentive qualities of our language.

References

Aissaoui, G., Genest, D. & Loiseau, S. 2003. Cognitive map of conceptual graphs: a graphical model to help for decision. In *ICCS*, de Moor, A., Lex, W. & Ganter, B. (eds), Lecture Notes in Computer Science **2746**, 337–350 Springer, ISBN 3-540-40576-3.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F. (eds) 2003. *The Description Logic Handbook*. Cambridge University Press.

Baget, J. F. 2005. RDF entailment as a graph homomorphism. In *Proceedings of ISWC'05*. Galway, Ireland.

Bos, C., Botella, B. & Vanheeghe, P. 1997. Modeling and simulating human behaviors with conceptual graphs. In *Proceedings of ICCS'97*, Lecture Notes in Artificial Intelligence **1257**, 275–289. Springer.

Carloni, O., Leclère, M. & Mugnier, M.-L. 2006. Introducing graph-based reasoning into a knowledge management tool: an industrial case study. In *Proceedings of IEA/AIE*. 590–599. Annecy, France.

Chauvin, L., Genest, D. & Loiseau, S. 2008. Contextual cognitive map. In *ICCS*, Eklund, P. W. & Haemmerlé, O. (eds), Lecture Notes in Computer Science **5113**, 231–241, Springer. ISBN 978-3-540-70595-6.

Chein, M. & Mugnier, M.-L. 1992. Conceptual graphs: fundamental notions. *Revue d'Intelligence Artificielle* **60**(4), 365–406.

Chein, M. & Mugnier, M. 2009. *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer.

Genest, D. & Chein, M. 2005. A Content-search information retrieval process based on conceptual graphs. *Knowledge and Information Systems (KAIS)* **8**, 292–309.

Hayes, P. 2004. *RDF Semantics. W3C Recommendation*. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.

Horst, H. 2004. Extending the RDFS entailment lemma. In *Proceedings of the 3rd International Semantic Web Conference ISWC'04*, Sheila A. McIlraith, Dimitris Plexousakis & Frank van Harmelen (eds). Lecture Notes in Computer Science **3298**, 77–91. Springer.

Lalande, S., Staykova, K., Chein, M., Gutierrez, A., Saraydarova, V. & Dochev, D. 2009. Using domain knowledge to speed up the annotation of digital content with conceptual graphs. *Cybernetics and Information Technology* **90**(3), 22–38.

Lehman, F. 1992. *Semantics Networks in Artificial Intelligence*. Pergamon Press.

Masterman, M. 1962. Semantic message detection for machine translation, using an interlingua. In *International Conference on Machine Translation of Languages and Applied Language Analysis*, 438–475.

Moreau, N., Leclère, M., Chein, M. & Gutierrez, A. 2007. Formal and graphical annotations for digital objects. In *Proceedings of the International Workshop on Semantically Aware Document Processing and Indexing (SADPI'07)*, ACM *Digital Library*, *ACM International Conference Proceeding Series* **259**, 69–78. ACM.

Novak, J. & Canas, A. 2006. The origins of the concept mapping tool and the continuing evolution of the tool. *Information Visualization Journal* **5**, 175–184.

Raimbault, T., Genest, D. & Loiseau, S. 2005. A new method to interrogate and check uml class diagrams. In *ICCS*, Dau, F., Mugnier, M.-L. & Stumme, G. (eds), Lecture Notes in Computer Science **3596**, 353–366. Springer, ISBN 3-540-27783-8.

Richens, R. 1956. Programming for mechanical translation. *Mechanical Translation* **3**.

Shapiro, S. 1979. The sneps semantic network processing system. In *The Representation and Use of Knowledge by Computers*, Findler, N.V. (ed.). Academic Press, New York, 179–203.

Shapiro, S. 2000. Sneps: A logic for natural language understanding and commonsense reasoning. In *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, Lucja Iwanska & Stuart C. Shapiro (eds), 175–195.

Shaw, M. & Gaines, B. 1995. Knowledge and requirements engineering. In *Proceedings of the 9th Banff Knowledge Accquisition For Knowledge-Based Systems Workshop*.

Sowa, J. F. 1976. Conceptual graphs. *IBM Journal of Research and Development* **20**(4), 336–375.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Woods, W. & Schmolze, J. 1992. The kl-one family. *Computers Mathematical Applications* **23**, 133–177.