Concept Types and Coreference in Simple Conceptual Graphs

Michel Chein and Marie-Laure Mugnier
chein@lirmm.fr, mugnier@lirmm.fr

LIRMM - Université Montpellier 2 and CNRS - 161, rue Ada - 34392 Montpellier cedex 5 - France

Abstract. This paper tackles the question of representing and reasoning with types and coreference in simple conceptual graphs (SGs). It presents a framework integrating a number of previous works. This proposal is guided by the usability of CGs in practice. In other words, notions should be easy to use in knowledge representation and operations for doing reasoning have to be efficiently realizable. We propose to use *conjunctive concept types*, which are conjunctions of primitive types. The conjunctive concept type set is defined by means of a primitive type set and a set of *banned* conjunctive types. For efficiency reasons our framework is based on projection. However it has been shown that projection is complete (w.r.t. logical deduction) only when SGs are in *normal form*. In some situations the original form of the SGs has to be kept; we thus define an extension of projection, called *coref-projection*, which is complete for SGs of any form. Coref-projection is in particular suitable for frameworks where it is not assumed that coreferent nodes are mergeable.

1 Introduction

This paper tackles the question of representing and reasoning with types and coreference in simple conceptual graphs (SGs). The analysis is guided by the usability of CGs in practice. More specifically the following points are emphasized:

- All notions should be *simple* to understand from a modeling viewpoint;
- Operations for doing reasoning should be realizable in an *efficient* way;
- Reasoning should be sound and *complete* w.r.t. to the formal semantics of SGs.

Let us first briefly introduce the problem. With efficiency in mind, the fundamental operation for doing reasoning with SGs is a graph homomorphism called *projection*. In SGs several concept nodes can refer to a same entity. They are said to be *coreferent*. When it has no coreferent concepts, a SG is said to be *normal*. On normal SGs projection is complete w.r.t. its usual formal semantics (its set semantic or its logical semantic Φ). But in general it is not. Several questions arise then. Can any pair of concepts be coreferent? This question is related to the interpretation of the concept type set. In which cases does a SG possess an equivalent normal SG? When a SG has no equivalent normal SG or when this latter is not computable in practice, is there an alternative to projection? Finally is the coreference notion really interesting for simple SGs? We shall study these questions and present answers found in the literature as well as a framework integrating a number of previous works.

Concept types. In [Sow84] the concept type set is a *lattice*. In later works and in the proposed standard for CGs [NCI98] it becomes a partially ordered set with a greatest element, thus a less constrained structure. Besides the structure an essential point is how the type hierarchy is interpreted. When a type is interpreted as a set of entities, the order over the hierarchy is interpreted as the inclusion. Then according to the terminology of [BHP+92] the greatest lower bound (glb) of two types, when it exists, can be *lattice-theoretically* or *order-theoretically* interpreted. In the order-theoretic interpretation there is nothing more than the interpretation of the subtype relation by set inclusion. With the lattice interpretation the set interpretation of the glb of two types is furthermore interpreted as the intersection of these sets. E.g. let Building and OldThing be two incomparable types and let HistoricLandmark be their glb (Figure 1 left, example from [CCW97]). With both interpretations, every entity of type HistoricLandmark is also of types Building and OldThing. With the lattice interpretation every entity of both types Building and OldThing is necessarily of type HistoricLandmark. With the order-theoretic interpretation the glb of two types is simply a subtype, thus a subset of their intersection; an entity of both types Building and OldThing is not necessarily a HistoricLandmark.

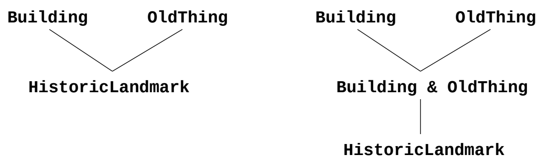

Fig. 1. Interpretation of types

The way the concept type set is interpreted has an effect on reasoning, as pointed out in [WL94] [CCW97]. For instance let Q be the query [HistoricLandmark:*] ("is there a HistoricLandmark?") and let G be an assertion containing the coreferent concept nodes [Building:a] and [OldThing:a] (thus G asserts that a is an entity which is a Building and an OldThing). With the lattice interpretation, G should answer Q while in the order-theoretic one it should not.

The lattice-theoretic interpretation requires to explicitely build all type intersections (for instance if a Building and OldThing is not necessarily a HistoricLandmark one has to build their common subtype Building&OldThing which is a supertype of HistoricLandmark, see Figure 1 right). Thus it leads to a number of new and artificial types which can be exponential in the number of useful types.

On the other hand, the drawback of the order-theoretic interpretation is that we loose the possibility of expressing conjunctions of types, e.g. HistoricLandmark is exactly the intersection of OldThing and Building. This property could be expressed by a rule, stating that every entity of type Building and of type OldThing is also of type HistoricLandmark; however coding the conjunction of types in the hierarchy itself is more efficient than using rules, exactly as coding the subtype relation in a type hierarchy is more efficient than considering a flat set of types and representing the hierarchy by rules (cf. the discussion in [AKN86]).

A concept node represents an entity but an entity can be represented by several coreferent concept nodes. In order to ease the building and the understanding of a SG by a human being a way of gathering all or some concept nodes representing the same entity is desirable. A simple way to gather the different concept nodes representing the same entity is to merge all these nodes into one node. This operation has to keep the meaning of the original SG. Thus if an entity is represented by a node of type t and by a node of type t' there must be a type representing exactly the entities which are of both types t and t'. More generally, we need to express that a type is the conjunction of several types. In [Bag01] a mechanism for type conjunction in conceptual graphs is proposed (cf. also [CCW97] where type conjunction is used in the context of fuzzy types). In Baget's framework, the set of conjunctive types is defined in intension by a hierarchy of primitive concept types. Any subset of (incomparable) primitive types define a conjunctive type. The set of conjunctive set is partially ordered by an order extending that on primitive types.

However not all conjunctions of types have a meaning. Thus we need a way of expressing that the conjunction of two (or more) types is *banned*, or equivalently that they cannot have a common subtype. A classical way of doing is to add a special Absurd type below types with an empty intersection. But this technique is not always suitable. For instance, if t_1, t_2, t_3 are direct supertypes of Absurd then the intersection of any two of them is empty. We cannot say that there cannot be no entity of types t_1 and t_2 neither of types t_1 and t_3, but that there can be an entity of type t_2 and t_3 (e.g. t_1 = Animal, t_2 = Ship, t_3 = Robot, all being subtypes of MobileEntity).

Giving in extension all acceptable types is not conceivable in practice, therefore we define the set of acceptable conjunctive types by the primitive type set and the (maximal) banned conjunctive types. In previous example, the types $\{t_1, t_2\}$ and $\{t_1, t_3\}$ would be banned. Theoretically, the number of banned conjunctive types can be exponential in the number of primitive types, but, practically, considering banned types of cardinality two seem to be sufficient and their number is at most quadratic in the number of primitive types.

Coreference. Coreference of several concept nodes can be expressed by the fact that these nodes have the same individual marker or by coreference links [Sow84]. Let us say that these nodes are in the same coreference set.

More or less restrictive definitions of the coreference notion can be found in the literature. These definitions are related to the interpretation of types and markers. Let us point out that we study here coreference in SGs, which is not the same relation as coreference in general CGs (for instance, coreference is a transitive relation in SGs whereas it may be not in general CGs). An individual marker is generally considered as a surrogate or an identifier (in the programming meaning) of an entity. According to the Unique Name Assumption usually made in knowledge representation, it is thus assumed that two distinct individual markers represent distinct entities. Hence nodes with distinct individual markers cannot belong to the same coreference set (once again the situation is not the same in general CGs, where for instance coreferent nodes with distinct individual markers a and b can be used to express that "a is not b"). An exception is the framework of [Dau03] which allows coreference between different individual concepts[1].

Let us consider two concepts with incomparable types t_1 and t_2. Can these nodes be coreferent? In a lot of works the answer is positive. As already said, in our opinion important properties to ensure are, firstly, that *(a)* coreferent concepts can always be merged into a single node, secondly, that *(b)* the meaning of the obtained graph is the same as the one of the original one. With a lattice-interpretation of concept types these properties are ensured if the types of the coreferent nodes have a glb (distinct from the Absurd type), which becomes the type of the new node. With the order-theoretic assumption, and in absence of conjunctive types, the only way to ensure these properties is to impose that the set of types of the coreferent nodes possess a minimal element in the set itself. Indeed suppose for instance there are two coreferent concepts with types Building and OldThing respectively. Either these concepts cannot be merged and *(a)* is not satisfied, or they can and the type of the new node is a subtype of Building and OldThing and the obtained graph is (in general) strictly more specific than the original one. That is why in works where properties *(a)* and *(b)* are required it is imposed that coreferent nodes have the same type (e.g. [CMS98] [Ker01]). In the framework of conjunctive types proposed here *(a)* and *(b)* are obtained as soon as the conjunction of the types of the coreferent nodes is not a banned type. Then every SG can be easily transformed into an equivalent normal SG, which is called its *normal form*.

Whether a coreference set may contain both generic and individual nodes is a less important point. Indeed making a generic node and an individual node coreferent can always be done by restricting the marker of the generic concept to the individual marker of the other node without changing the meaning of the graph.

In general CGs or in positive nested CGs coreference is mandatory to express identity of nodes belonging to different contexts. Now, is coreference a useful notion for SGs? In frameworks where it is not always possible to merge coreferent nodes of a SG the answer is obviously positive. But as soon as every SG possesses a normal form one may ask whether coreference is needed at all. Firstly, there is no doubt that coreference links are interesting in *user interfaces*. Indeed one

[1] This case corresponds to the existence of aliases among the individual markers. It could be included in our framework with slight modifications.

interest of SGs is their readability, quality that is lost when the SG becomes too big (e.g. too big to be entirely drawn on a screen). A SG too big can be split into pieces each displayable on one screen. The split can also be performed to decompose a SG into semantic modules, or to enhance a particular substructure. During a knowledge acquisition step a user can build a SG piece by piece, then connect them with coreference links. The same situation occurs when a SG basis has been constructed by different people or at different times. Secondly, even if the normal form can be computed in a polynomial time, the variable of this polynomial is the size of the SG basis. Therefore, if the SG basis is very large (e.g. a significative part of the web or a document basis with millions of items) the cost of normalization can be prohibitive. Normalization can even be unthinkable if the basis is distributed among several servers.

Reasoning. Our framework is based on projection. However it has been shown that projection is complete only when SGs are in normal form. As explained above, in some situations one has to keep the original form of SGs; we thus propose an extension of projection, called *coref-projection*, which is complete without restrictions on the SG form. Coref-projection is in particular suitable for frameworks where it is not assumed that coreferent nodes are mergeable.

Paper organization. The sequel of the paper is organized as follows. Section 2 is devoted to the representation of the concept type set and the coreference relation. We provide basic definitions and properties and identify basic algorithmic problems. Section 3 deals with reasoning. It introduces the notion of coref-projection and elicits the relationships between projection and coref-projection.

2 Concept Types and Coreference

2.1 Concept types

Throughout this paper, notations of [DP02] are used for the notions about ordered sets. The first notion for dealing with concept types is the primitive concept type set.

Definition 1 (Primitive concept type set). *A* primitive concept type set *is a (finite) ordered set of primitive concept types. It is denoted by T. It possesses a greatest element, called the universal type, and denoted by $\top$.*

A conjunctive concept type is a set of primitive concept types. It can be *acceptable* or *banned*. A type is banned if there cannot be an entity of this type, conversely there can be entities of an acceptable type. If there are two comparable types, say $t_i \leq t_j$, in the acceptable conjunctive type $t = \{t_1, ..., t_n\}$, then the conjunction of t_i and t_j has the same meaning as t_i since every entity of type t_i is of type t_j. Therefore t_j is useless. In the same way, if the conjunctive type $t = \{t_1, ..., t_n\}$ is banned, and if $t_i \leq t_j$, then the smaller one, i.e. t_i is useless. In order theory a subset of non comparable elements of an ordered set is called an *antichain*. Thus, the set of conjunctive types is the set $T^{\sqcap}$ of antichains of the primitive type set T.

Definition 2 (Conjunctive concept type). *A* conjunctive concept type *is given by a (non empty) set of non comparable types i.e. an antichain of T.*

A primitive type can be seen as a conjunctive type with one type. Conjunctive types are provided with a natural partial order which extends that on primitive types: a conjunctive type t is a specialization of a conjunctive type s if every primitive type of s has a specialization (possibly equal to itself) in t.

Definition 3 (Conjunctive concept type set). *Let T be the set of primitive concept types. $T^{\sqcap}$ denotes the set of all conjunctive types over T. It is provided with the following partial order, which extends the partial order on T: given two types $t = \{t_1 \ldots t_n\}$ and $s = \{s_1 \ldots s_p\}$,*

$t \leq s$ if for every $s_j \in s$, $1 \leq j \leq p$, there is a $t_i \in t$, $1 \leq i \leq n$, such that $t_i \leq s_j$.

Property 1. $(T^{\sqcap}, \ \leq)$ is a lattice.

A direct proof of this property is straightforward: the set of minimal elements of the union of two antichains is the glb of these two antichains, therefore $(T^{\sqcap}, \ \leq)$ is a join-semi-lattice, and since it has a maximal element, it is a lattice[2].

The property of being a banned conjunctive type is hereditary: if a conjunctive type A is banned, all conjunctive types less than A are also banned. Therefore, the banned types is a down-set and one can assume that only the maximal banned conjunctive types are given.

Definition 4 (Banned type set). *Let $\mathcal{B}$ denote a subset of non comparable conjunctive types. An element of $T^{\sqcap}$ is said to be banned w.r.t. $\mathcal{B}$ if it is less or equal to an element of $\mathcal{B}$. $\downarrow \mathcal{B}$ denotes the set of all banned types i.e. $\downarrow \mathcal{B} = \{t \in T^{\sqcap} \mid \exists t' \in \mathcal{B}, t \leq t'\}$.*

The set of acceptable conjunctive types can be exponentially bigger than the primitive type set. That is why the concept type hierarchy is not defined in extension but by means of the primitive type set and a set of assertions stating which types are banned. The set of acceptable types is obtained from $T^{\sqcap}$ by removing the banned types $\downarrow \mathcal{B}$.

Definition 5 (Concept type hierarchy). *A concept type hierarchy T_C is given by a couple $(T, \ \mathcal{B})$ where:*

- *T is the set of primitive concept types*
- *$\mathcal{B}$ is the set of basic banned conjunctive types*
- *$\mathcal{B}$ complies with T, i.e. for all $b \in \mathcal{B}$, there is no type $t \in T$ with $t \leq b$ i.e. $\downarrow \mathcal{B} \cap T = \emptyset$.*

[2] It is well-known that the set of antichains of an ordered set is a lattice. But, usually, the order relation considered is the "complement-dual" of ours i.e. $t \leq s$ if for every $t_i \in t$ there is a $s_j \in s$ such that $t_i \leq s_j$ (with this order the lattice of antichains is isomorphic to the lattice of down-sets of T whereas, in our case, it is isomorphic to the lattice of up-sets of T).

T_C is defined as the set $T^{\sqcap} \setminus \downarrow \mathcal{B}$. $T^{\sqcap}$ is thus partitioned into the acceptable types T_C and the banned types $\downarrow \mathcal{B}$.

The lub (in $T^{\sqcap}$) of two acceptable conjunctive types is acceptable, therefore:

Property 2. The set of acceptable conjunctive types T_C is a meet-semi-lattice.

Type definitions. Let us come back to the example of the introduction. We can now express that an entity is of type Building and OldThing. We could associate a very simple type definition mechanism authorizing to *name* conjunctions, such as HistoricLandmark = {Building, OldThing}. Then concept types could be labelled by conjunctions of primitive types and named conjunctions. One can always associate to such a type a conjunction of primitive types by recursively decomposing named conjunctions. Thus this mechanism can be added on top of the techniques proposed here without difficulties.

2.2 Concept labels

CGs are defined with respect to a vocabulary, that we recall here to fix the notations.

Definition 6 (Vocabulary). *A* vocabulary *is a 3-tuple $\mathcal{V} = (T_C, T_R, \mathcal{I})$.*

- *T_C, T_R and $\mathcal{I}$ are pairwise disjoint (finite) sets*
- *T_C is the* concept type hierarchy
- *T_R, the* relation set*, is partially ordered by $\leq$, and is partitioned into subsets $T_R^1 \ldots T_R^k$ of relations of arity $1 \ldots k$ respectively ($k \geq 1$). Two relations of distinct subsets are non comparable.*
- *$\mathcal{I}$, is the (finite) set of individual markers. $*$ being the generic marker, the set $\mathcal{I} \cup \{*\}$ is partially ordered in the following way: $*$ is the greatest element and elements of $\mathcal{I}$ are pairwise incomparable.*

Definition 7 (Concept labels set). *The set of concept labels is the set of couples $(t,\ m)$ such that $t \in T_C$ and $m \in \mathcal{I} \cup \{*\}$. The* partial order on concept labels *is the product of the partial orders on T_C and $\mathcal{I} \cup \{*\}$, i.e. $(t, m) \leq (t', m')$ iff $t \leq t'$ and $m \leq m'$.*

Since the product of two meet-semi-lattices is a meet-semi-lattice, one has:

Property 3. The ordered set of concept labels is a meet-semi-lattice.

Thus a set of labels possesses a lub, furthermore if it has at least a common lower label then it possesses a glb. When an entity is represented by several concepts we impose that the labels of these concepts satisfy two conditions. The first condition corresponds to the unique name assumption. The second condition requires that the conjunction of the types of the concepts representing an entity is not a banned type. These two conditions are gathered in the following definition.

Definition 8 (Compatible concept labels). *The concept labels* $(t_1, m_1), \ldots, (t_k, m_k)$ *are* compatible *if*

- $|\mathcal{I} \cap \{m_1, \ldots, m_k\}| \leq 1$
- *the conjunctive type* t *defined as the set of minimal primitive types of* $\cup_{i=1}^{k} t_i$ *is acceptable i.e. is in* T_C

Property 4. The concept labels $(t_1, m_1), \ldots, (t_k, m_k)$ are compatible iff they possess a greatest lower bound.

The glb of concept labels $(t_1, m_1), \ldots, (t_k, m_k)$ is the concept label (t, m) with $t = min(\cup_{i=1}^{k} t_i)$ and $m = min(\cup_{i=1}^{k} \{m_i\})$ i.e. m is the generic marker if all m_i are the generic marker, otherwise m is the unique individual marker appearing in the m_i. By extension, concepts are said to be compatible if their labels are compatible.

Definition 9 (concept merging). *A set of concepts* $\{c_1, ..., c_n\}$ *is compatible if the set of their labels is compatible. Let* l *be the greatest lower bound of their labels. Merging* $c_1, ..., c_n$ *consists in replacing them by a unique node* c *with label* l.

2.3 Algorithmic problems related to types

Let us now list the algorithmic problems to be solved.

Pb0 Compare two elements of an ordered set T
This problem occurs when one has to compare two primitive concept types (or two relations). A lot of different coding schemes of ordered sets have been proposed for this basic problem and for other problems over ordered sets (cf. [Fal98] [FJN91] [Thi01]). In order to simplify the presentation of the following problems we assume that Pb0 can be solved in $O(1)$ (otherwise the given complexities must be multiplicated by the complexity of Pb0).

Pb1 Compute a conjunctive type
Data: an ordered set T, a subset $t \subseteq T$
Problem: compute the set of minimal elements of t i.e. $min(t) = \{t_i \in t | \forall t_j \in t, t_j \not< t_i\}$
Naïve algorithm: compare each element of t to the others; the complexity is $O(|t|^2)$.

Pb2 Compare two conjunctive types
Data: two antichains t and s of T
Question: $t \geq s$?
Naïve algorithm: compare all elements of t to the elements of s; the complexity is $O(|t| \times |s|)$.
It may be necessary to check if $\mathcal{B}$, as given by a user, is an antichain of $T^{\sqcap}$, and that can be done by an algorithm for Pb2. Furthermore, if $\mathcal{B}$ is not an antichain it can be replaced by the set of maximal elements of $\mathcal{B}$, and that can be done by a variant of Pb1 (considering $T^{\sqcap}$ instead of T and max instead of min).

Pb3 Check the acceptability of a conjunctive type

Data: $T, \mathcal{B}$ and t an antichain of T

Question: is t acceptable w.r.t. $\mathcal{B}$? i.e. $\forall s \in \mathcal{B}$ does $t \not\leq s$ hold ?

Naïve algorithm: compare, with an algorithm for Pb2, every element of $\mathcal{B}$ to t; the complexity is $O(size(\mathcal{B}) \times complexity(Pb2))$, where size($\mathcal{B}$) is equal to the sum of the cardinalities of the elements of $\mathcal{B}$.

Remark: $\mathcal{B}$ complies with T iff any primitive type is acceptable, that can be checked in $O(size(\mathcal{B}) \times |T|)$.

Pb4 Compare two concept labels

Data: two concept labels (t, m) and (t', m')

Question: $(t, m) \leq (t', m')$?

Algorithm: compare t and t' with an algorithm for Pb2 (or Pb0 if the types are primitive); therefore, the complexity is the same as the one of Pb2 because the comparison of two markers is trivial.

Pb5 Check the compatibility of concept labels and compute their glb

Data: a set of concept labels $\{(t_1, m_1), \ \ldots, \ (t_k, m_k)\}$

Problem: is that set compatible ? If it is the case compute the glb of this set.

Algorithm: compute $t = min(\cup_{i=1}^{k} t_i)$ i.e. make the union of the t_i then use an algorithm for Pb1, check if t is acceptable (cf. Pb3), and if $\{m_1, \ldots, m_k\}$ has a glb m. If (t, m) exists it is the glb of the set of labels.

Open problem As previously mentionned there are numerous coding schemes of ordered sets for efficiently solving Pb0. There are also coding schemes of lattices of antichains. But in order to solve very efficiently all preceding problems we need a scheme for coding *together* T, $\mathcal{B}$ and a dynamic set of antichains of T (e.g. the conjunctive types appearing in a SG).

2.4 Basic, Simple, and Normal CG

A *basic conceptual graph (BG)* is a SG without explicit coreference links. It is required that, if an individual marker appears more than once in the BG, the conjunctive type associated with these occurences is an acceptable type.

Definition 10 (Basic Conceptual Graph). *A* basic conceptual graph *(in short BG) defined over a vocabulary $\mathcal{V} = (T_C, T_R, \mathcal{I})$, is a 4-tuple $G = (C, R, E, l)$ satisfying the following conditions:*

1. $\boldsymbol{G} = (C, R, E)$ *is a finite, undirected and bipartite multigraph called the* underlying graph *of* G. C *and* R *are the node sets, respectively of* concept nodes *and of* relation nodes. E *is the family of* edges.
2. l *is a labelling function of the nodes and edges of* $\boldsymbol{G}$ *which satisfies:*
   (a) *A concept node* c *is labeled by a pair* (type(c), marker(c))*, where* type(c)$\in T_C$*, and* marker(c)$\in \mathcal{I} \cup \{*\}$.
   (b) *For any individual marker* i*, the set of types of the concepts with marker* i *is in* T_C *(i.e. is not banned).*
   (c) *A relation node* r *is labelled by an element* $l(r)$ *of* T_R.

(d) *The degree of a relation node r must be equal to the arity of $l(r)$.*

(e) *Edges incident to a relation node are totally ordered and they are labeled from 1 to $arity(l(r))$. If there is an edge labeled i between the relation node r and the concept node c, that edge is denoted by (r, i, c).*

If an individual marker i appears more than once in a BG the types associated with i in concept labels have a glb (condition $2(b)$ of Def. 10). Thus these concepts are compatible. Therefore any BG, say G, can be transformed into a normal BG by merging concepts having the same individual marker. The result is unique, we call it the *normal form* of G, that we note $norm(G)$. A BG has the same meaning as its normal form.

Having two concepts with the same individual marker is a particular case of coreference. In general coreferent nodes can be individual or generic. To indicate that a generic concept is coreferent with another concept a *coreference link* is used. It is represented in drawings by a dotted line between the two concepts. The coreference relation is the reflexive and transitive closure of the relation defined by coreference links and individual marker identity.

Note that the coreference relation is transitive on SGs, whereas it may be not in general CGs (depending on whether coreference is transitive or not one obtains general CGs equivalent to FOL with equality as in [BMT99] [Dau02] or without equality as in [Ker01]).

A SG is a BG provided with a coreference relation.

Definition 11 (Simple conceptual graph). *A* simple conceptual graph *(in short SG) over a vocabulary $\mathcal{V}$ is a 5-tuple $(C, R, E, l, coref)$ such that:*

- *(C, R, E, l) is a basic conceptual graph over $\mathcal{V}$*
- *$coref$ is an equivalence relation over C such that:*

  the concepts of any coref class are compatible

  if two individual concepts have the same marker then they are in the same coref class

A BG can be seen as a SG with a $coref$ relation completely determined by its individual concepts. A normal SG is such that $coref$ is the identity relation i.e. each concept is coreferent only with itself. A normal SG can thus be seen as a BG. In other words *normal SGs* and *normal BGs* are the same objects.

Every SG, say G, possesses a normal form, which is obtained by merging all concepts belonging to a same coreference class. We note it $G/coref$.

Definition 12 (G/coref). *Let $G = (C, R, E, l, coref)$ be a SG. $G/coref$ is the normal SG obtained as follows: for any coref class $X = \{c_1, \dots, c_k\}$ all the c_i are merged into one concept X. $G/coref$ is also called the* normal form *of G, and denoted by $norm(G)$.*

2.5 Algorithmic problems related to coreference

Algorithms for solving the following problems can be built by using classical list structures for graphs (for each node, either concept or relation, one has the list of its neighbors; each concept node contains its label and its coref class; for each coref class one has the list of its elements, etc.) and algorithms for the problems in section 2.3.

Pb6 Merge compatible concepts
Data: A SG G and a set of compatible concepts $\{c_1, ..., c_n\}$ of G.
Problem: Compute the SG obtained from G by merging $\{c_1, ..., c_n\}$.
Algorithm: first, compute l, the glb of their labels (cf. Pb5), then create a new concept node with label l, and with neighbors the concatenation of the neighbor lists of the c_i (this last operation can be computed with a time linear in the sum of the sizes of the lists).

Pb7 Compute the normal form of a BG or a SG
Data: a BG or a SG G
Problem: compute the quotient graph $G/coref_G$
Algorithm: merge each coreferent class (cf. Pb6)

3 Reasoning

3.1 Reasoning on BGs

The fundamental notion for reasoning with SGs is the *specialization/generalization* relation. This relation over SGs is originally defined by means of a set of elementary graph rules but for *effective* computation on SGs a global operation is used, called projection. This operation is equivalent to elementary rules (G is a generalization of H iff there is a projection from G to H, see [Mug00] for a synthesis on these notions).

Definition 13 (Projection). *Let G and H be two BGs. A* projection *from G to H is a mapping π from C_G to C_H and from R_G to R_H, which preserves edges and may specialize concept and relation labels*

1. $\forall (r, i, c) \in G, (\pi(r), i, \pi(c)) \in H$;
2. $\forall x \in C_G \cup R_G$, $l_H(\pi(x)) \leq l_G(x)$.

If there is a projection (say π) from G to H we say that G projects to H (by π). G is called the *source* graph and H is the *target* graph. The existence of a projection from G to H intuitively translates the fact that all information contained in G is also contained in H. But projection does not behave as expected on non normal graphs (as independently noticed in [GW95] and [CM95][MC96]). See Figure 2: G and H are intituitively/logically equivalent but they are not for projection; indeed there is a projection from H to G but not from G to H.

Fig. 2. Projection with a non normal target BG

Formally, projection is *sound* w.r.t. to the logical translation of BGs given by the mapping Φ of [Sow84] and *complete* when the target BG is in normal form.

Theorem 1 (Projection soundness and completeness). *Let G and H be two BGs defined on a vocabulary $\mathcal{V}$. Let $\Phi(\mathcal{V})$ be the logical translation of $\mathcal{V}$.*

- *If $G \geq H$ then $\Phi(\mathcal{V}), \Phi(H) \vDash \Phi(G)$ [Sow84].*
- *If $\Phi(\mathcal{S}), \Phi(H) \vDash \Phi(G)$ then $G \geq nf(H)$ [CM92] [CM95].*

3.2 Reasoning on SGs

Let us now consider SGs instead of BGs. A natural idea is to extend the BG projection in the following way: if two concepts are coreferent they must have coreferent images, that is to say either the same image (coref being a reflexive relation) or distinct coreferent images. We thus obtain the following definition.

Definition 14 (SG Projection). *Let $G = (C_G, R_G, E_G, l_G, coref_G)$ and $H = (C_H, R_H, E_H, l_H, coref_H)$ be two SGs. A* projection *π from G to H is a projection from the BG (C_G, R_G, E_G, l_G) to the BG (C_H, R_H, E_H, l_H) which preserves coreference i.e. $\forall x, y \in C_G, coref_G(x, y) \Rightarrow coref_H(\pi(x), \pi(y))$.*

As natural as it may seem previous definition does not solve the problem we have foreseen on BGs. Indeed, see that the SG projection behaves exactly as the BG projection in Figure 2. Let us study in more detail the relationships between SGs and their normal form.

Property 5. Every SG (thus every BG) projects to its normal form. The converse is false, as shown in Fig. 2 where G is the normal form of H.

Property 6. Given SGs G and H, there is a bijective mapping between the set of projections from G to $norm(H)$ and the set of projections from $norm(G)$ to $norm(H)$.

The normal form of a SG always exists and is easily computable. However in some cases we have to do reasoning on SGs exactly as they are. An alternative to projection is then needed. The idea is that projection does not deal correctly with coreference because a concept cannot be mapped onto several concepts. If we modify this operation so that it maps coreference classes onto coreference classes we obtain the desired operation.

3.3 coref-projection

Definition 15 (coref-projection). *Let $G = (C_G, R_G, E_G, l_G, coref_G)$ and $H = (C_H, R_H, E_H, l_H, coref_H)$ be two SGs. A* coref-projection *from G to H is a mapping π from $coref_G$ to $coref_H$ and from R_G to R_H such that:*

1. *$\forall (r, i, c) \in G$, let C be the coreference class of c, then there is a concept $c' \in \pi(C)$ such that $(\pi(r), i, c') \in H$;*
2. *$\forall C \in coref_G$, let $l_G(C)$ and $l_H(\pi(C))$ be the greatest lower bounds of the concept labels in C and $\pi(C)$ respectively; then $l_G(C) \geq l_H(\pi(C))$;*
3. *$\forall r \in R_G$, $l_G(r) \geq l_H(\pi(r))$.*

Each projection defines a coref-projection since by definition of a projection all coreferent nodes have their image in the same coreference class, but the converse is not true. The following property specifies the relationships between both notions.

Property 7. Given SGs G and H, there is a bijective mapping between the set of coref-projections from G to H and the set of projections from $norm(G)$ to $norm(H)$.

Since projection is (sound and) complete on normal SGs (theorem 1) we obtain the following corollary:

Corollary 1. *Coref-projection is sound and complete on SGs.*

Following property summarizes the "safe" ways of comparing two SGs.

Property 8. Let G and H be two SGs. There is a bijective mapping between:

- coref-projections from G to H
- projections from $norm(G) = G/coref$ to $norm(H) = H/coref$
- projections from G to $norm(H)$

Let us also cite [CH94] which deals with BGs and, to obtain projection completeness, splits the source SG instead of computing the normal form of the target SG. With the idea of adapting this technique to SGs we could imagine to split the source SG into its so-called anti-normal form ([GW95]) in which a concept is incident to at most one edge. This way of doing works well with a restricted definition of coreference (a coreference set does not mix generic and individual nodes and coreferent nodes have the same type) as presented in [Mug00]. Fig.3 shows that with the more general definition studied in this paper it does not work anymore: the antinormal form of G, denoted by $anf(G)$, does not project to H, whereas G does project to $norm(H)$ (actually $G = norm(H)$). Thus splitting the source SG into its anti-normal form is not an alternative to compacting the target SG into its normal form.

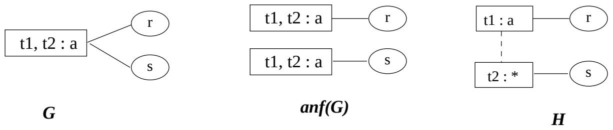

Fig. 3. Splitting the source SG is not an alternative

Finally, let us point out the interest of coref-projection in frameworks where it is not assumed that coreferent nodes can be merged into a single node (thus not all SGs have a normal form). See that the above definition of coref-projection can be reformulated replacing condition (2) by the following condition (2'): $\forall C \in coref_G$, $\forall c_i \in C$, $\exists c_j \in \Pi(c)$ with $l_G(c_i) \geq l_H(c_j)$. Equivalence between (2) and (2') follows from the definition of the partial order on conjunctive types (indeed, it is not true that for any lattice given two sets $E = \{a_1 \ldots a_p\}$ and $F = \{b_1 \ldots a_q\}$, $glb(E) \geq glb(F)$ implies for all a_i in E there is b_j in F with $a_i \geq b_j$). Rewritten in this way coref-projection can be used without any assumption on type compatibility.

4 Conclusion

In this paper we have proposed a framework integrating notions from previous works for dealing with the two related notions of conjunctive concept types and coreferent concept nodes in simple conceptual graphs. This framework keeps the qualities of CGs that we consider as essential. Namely, it is easy to use for knowledge representation because the notions are based on labelled graphs allowing intuitive semantics based on drawings; reasoning is grounded on graph operations which are easily understandable; graph operations can be efficiently implemented; and last but not least they are sound and complete w.r.t. the usual FOL semantics.

In the same way as it is proposed for concepts, including conjunctive relations in SGs could also be done (following [Bag01]). Designing and implementing efficient algorithms is an important issue. In particular coref-projection checking is an NP-complete problem (since projection checking is [CM92]). Coref-projection being very close to projection, algorithmic techniques developed for projection (as in [Bag03]) should be adaptable to it.

References

[AKN86] H. Aït-Kaci and R. Nasr. Login: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3(3):185–215, 1986.

[Bag01] J.-F. Baget. *Représenter des connaissances et raisonner avec des hypergrpahes: de la projection la dérivation sous contraintes.* PhD thesis, Université Montpellier II, Nov. 2001.

[Bag03] J.-F. Baget. Simple conceptual graphs revisited: Hypergraphs and conjunctive types for efficient projection algorithms. In *Proc. of ICCS'03*. Springer, 2003.

[BHP+92] C. Beierle, U. Hedtstück, U. Pletat, P. H. Schmitt, and J. H. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55(2):149–191, 1992.

[BMT99] F. Baader, R. Molitor, and S. Tobies. Tractable and Decidable Fragments of Conceptual Graphs. In *Proc. of ICCS'99, LNAI 1640*, pages 480–493. Springer, 1999.

[CCW97] T. H. Cao, P. N. Creasy, and V. Wuwongse. Fuzzy unification and resolution proof procedure for fuzzy conceptual graph programs. In *ICCS*, volume 1257 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 1997.

[CH94] B. Carbonneill and O. Haemmerlé. Rock : Un système question/réponse fondé sur le formalisme des graphes conceptuels. In *Actes du 9-ième congres RFIA*, pages 159–169, 1994.

[CM92] M. Chein and M.-L. Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, 1992. Available at http://www.lirmm.fr/˜mugnier/.

[CM95] M. Chein and M.-L. Mugnier. Conceptual Graphs are also Graphs. Research Report RR-LIRMM 95-003, LIRMM, 1995. Available at http://www.lirmm.fr/˜mugnier/.

[CMS98] M. Chein, M.-L. Mugnier, and G. Simonet. Nested Graphs: A Graph-based Knowledge Representation Model with FOL Semantics. In *Proc. of KR'98*, pages 524–534. Morgan Kaufmann, 1998. Revised version available at http://www.lirmm.fr/˜mugnier/.

[Dau02] F. Dau. *The Logic System of Concept Graphs with Negation And Its Relationship to Predicate Logic*. PhD thesis, Technische Universitat Darmstadt, 2002.

[Dau03] F. Dau. Concept graphs without negations: Standardmodels and standardgraphs. In *Proc. of ICCS'2003, LNAI 2746*, pages –. Springer, 2003.

[DP02] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambride University Press, Cambridge, UK, 2002.

[Fal98] A. Fall. The foundations of taxonomic encodings. *Computational Intelligence*, 14:598–642, 1998.

[FJN91] R. Freese, J. Ježek, and J.B. Nation. *Free Lattices*. American Mathematical Society, Providence, Rhode Island, 1991.

[GW95] B. C. Ghosh and V. Wuwongse. A Direct Proof Procedure for Definite Conceptual Graphs Programs. In *Proc. of ICCS'95, LNAI 954*, pages 158–172. Springer, 1995.

[Ker01] G. Kerdiles. *Saying it with Pictures: a logical landscape of conceptual graphs*. PhD thesis, Université Montpellier II and University of Amsterdam, Nov. 2001. Available at http://turing.wins.uva.nl/˜kerdiles/.

[MC96] M.-L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1):7–56, 1996. Available at http://www.lirmm.fr/˜mugnier/.

[Mug00] M.-L. Mugnier. Knowledge Representation and Reasoning based on graph Homomorphism. In *Proc. of ICCS'00*, pages 172–192. Springer, 2000. Eevised version available at http://www.lirmm.fr/˜mugnier/.

[NCI98] NCITS.T2. Conceptual graphs: draft proposed American National Standard (dpANS). NCITS.T2/98-003; also Sowa's web site "http://www.jfsowa.com", 1998.

[Sow84] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[Thi01] E. Thierry. *Sur quelques interactions entre structures de données et algorithmes efficaces*. PhD thesis, Université Montpellier II, Oct. 2001.

[WL94] M. Wermelinger and J.G. Lopes. Basic conceptual structures theory. In *Proceedings of the 2nd International Conference on Conceptual Structures, ICCS'94*. Springer Verlag, 1994.

5 Appendice (for the reviewer)

We give here the proofs of properties 5, 6 and 7 (part 3).

Proof of property 5 . Let π be the mapping from G to $norm(G)$ which maps each node to itself except for nodes merged by the normalization, which map to the new node. Check that π is a projection. We call it the canonical projection from G to $norm(G)$. $\square$

Proof of property 5 . We build a mapping P as follows. Let π_G be the (canonical) projection from G to $norm(G)$. To every projection π from $norm(G)$ to $norm(H)$ P associates the projection $\pi' = \pi \circ \pi_G$ from G to $norm(H)$. P is injective because, given two distinct projections π_1 and π_2 from $norm(G)$ to $norm(H)$ and any node x of $norm(G)$, if $\pi_1(x) \neq \pi_2(x)$ then for any node y with $\pi_G(y) = x$, $\pi_G(\pi_1(y)) \neq \pi_G(\pi_1(y))$. P is surjective too, since every projection π' from G to $norm(H)$ can be decomposed into a projection $\pi \circ \pi_G$. Indeed π is built as follows: for every node x from $norm(G)$, if x is a node of G then $\pi(x) = \pi'(x)$, otherwise let $c_1 ... c_n$ be the nodes of G merged into x; by definition of π_G, for all these nodes c_i, $\pi_G(c_i) = x$, and since π' is a projection and H is normal, all c_i have the same image by π', then, given any c_i, $\pi(x) = \pi'(c_i)$. By definition of coreference and projection, $label(\pi(x)) \geq label(\pi'(c_i))$. π is thus a projection. Check that $\pi \circ \pi_G = \pi'$. P is thus bijective. $\square$

Proof of property 6 . There is a bijection, say b, between concept nodes of $norm(G)$ (resp. $norm(H)$) and coreference classes in G (resp. H). Let us show that b defines the desired mapping between the set of coref-projections from G to H and the set of projections from $norm(G)$ to $norm(H)$. Let π be a coref-projection from G to H. Let π' be the induced mapping from nodes for $norm(G)$ to nodes $norm(H)$ (for each concept c of $norm(G)$,$\pi'(c) = b^{-1}(\pi(b(c)))$ and for each relation r of $norm(G)$, $\pi'(r) = \pi(r)$). We check that π' is a projection from $norm(G)$ to $norm(H)$: for every edge (r, i, c) in $norm(G)$, there is an edge (r, i, d) in G with d is in the coreference class $b(c)$, thus, due to condition 1 of coref-projection, there is an edge $(\pi'(r), i, d')$ in H with d' in the coreference class $\Pi(b(c))$, thus an edge $(\pi'(r), i, \pi'(c))$ since $\pi'(c) = b^{-1}(\pi(b(c)))$. For every concept c in $norm(G)$, condition 2 of coref-projection ensures that for all $c_i \in b(c)$, $l_G(c_i) \geq l_H(\pi'(c))$, thus by definition of a glb, $l_G(c) \geq l_H(\pi'(c))$. In the same way we check that any projection π' from $norm(G)$ to $norm(H)$ yields a coref-projection π from G to H using the bijection b. For every edge (r, i, c_j) in G, let C be the corefence class of c_j, there is an edge (r, i, c) in $norm(G)$ where $C = b(c)$, thus an edge $(r, i, \pi'(c))$ in $norm(H)$, thus an edge (r, i, c'_j) in H such that c'_j belongs to the coreference class $b(\pi'(c))$, which is exactly $\pi(C)$. For every coreference class C in G, for every c in C, $l_G(c) \geq l_H(\pi'(b^{-1}(C)))$, which is exactly $l_H(\pi(C))$.

$\square$