Towards Benchmarks for Conceptual Graph Tools

Jean-François Baget[2,1], Olivier Carloni[1,6], Michel Chein[1], David Genest[3], Alain Gutierrez[1], Michel Leclère[1], Marie-Laure Mugnier[1], Eric Salvat[4], and Rallou Thomopoulos[5,1]

[1] LIRMM (CNRS & Université Montpellier II)
{chein,gutierre,leclere,mugnier}@lirmm.fr
[2] INRIA Rhône-Alpes jean-francois.baget@inrialpes.fr
[3] LERIA genest@info.univ-angers.fr
[4] IMERIR salvat@imerir.com
[5] INRA rallou@ensam.inra.fr
[6] Mondeca olivier.carloni@mondeca.com

Abstract. This paper reports a collective reflection led in our team about conceptual graph benchmarks. We tackle four issues for which agreement should be obtained before benchmarks can be built: what are the fragments of CGs considered? How is information exchanged? What are the problems to be solved? What kinds of tool properties are evaluated by the benchmarks? We define a basic building block built upon simple conceptual graphs. Finally we propose to provide a first benchmark adapted from an industrial case study. This benchmark is composed on very simple structures and should allow to focus on interoperability issues.

1 Introduction

This paper reports a collective reflection led in our team [7], aiming at building CG benchmarks. We expect CGers to comment, amend and complete this contribution.

Tackling the issue of CG tools interoperability leads to several questions:

1. What are the *fragments of CGs* handled by the tools? Answering that question involves an agreement on a classification of specific forms of conceptual graphs.
2. How can tools *exchange* pieces of information ? Answering that question involves agreeing on an exchange format.
3. What are the *basic problems* to be solved by the tools? Answering that question involves defining common basic problems in a way precise enough to be able to define for a given instance of the problem the solution that should be found.

[7] http://www.lirmm.fr/~mugnier/RCR

4. What kinds of *benchmark data* should we provide? Answering that question involves defining what tool properties are to be evaluated by a benchmark.

In what follows we detail each above question, with identifying the difficulties that should be overcome. We make a first proposal, built upon the so-called simple conceptual graphs.

The tools we consider are those referenced on the wiki page maintained by Philippe Martin[8].

2 Fragments of conceptual graphs

Why is it essential to identify conceptual graph fragments? Conceptual graphs can be seen as a family of languages rather than one language. A reason is that, since John Sowa's seminal book, definitions, even of basic notions, have evolved along the years and each team has added its own development.

Another reason, which is essential in the context of tool interoperability, is that no tool implements conceptual graphs in their generality. General conceptual graphs are equivalent to first-order-logic (FOL) but no tool implements them. No tool does even implement any form of negation[9].

Our idea before writing this paper was to consider on one hand the CG standard (ISO/JTC1/SC32/WG2) that can be found on Sowa's website: [10] and on the other hand the kind of constructs handled in existing tools for proposing a classification of CG fragments, that could represent the most common constructs; a construct handled by a tool could then be defined by difference with elements of this classification. However the variations among tools are so big that we lowered our ambitions. As a first attempt, we restrict the proposal to a fragment built on simple conceptual graphs. This data model is indeed the simplest model shared by the CG community. In the perspectives, other candidate fragments are discussed.

Each CG fragment should be precisely defined and should possess a semantic in FOL. That is a minimal requirement. In our opinion, another requirement should be the definition of operators for dealing with this fragment. A subset of these operations should allow to compute deduction in a sound and complete way (see section 4).

Let us begin with the common unquestionable base: *simple conceptual graphs (SCGs)*. All tools implement them[11]. SCGs are furthermore fundamental for comparison with other languages/formalisms as they correspond to the conjunctive existential fragment of FOL. They are for instance equivalent to the conjunctive queries of databases. About the relationships with the semantic web, it

[8] `http://en.wikipedia.org/wiki/CG_tools`

[9] The CG editor CharGer allows to represent negative contexts but does not provide reasoning on them.

[10] `http://www.jfsowa.com/cg/cgstand.htm`

[11] Note however that Amine restricts CGs to binary relation types and the relations adjacent to a concept must be of different types.

is interesting to notice that SCGs subsume RDF (transformations from RDF to SCG preserving the semantics have been proposed in [CDH00] [Bag05]).

According to the standard, SCGs may include coreference links. Let us point out however that tools vary on the constraints that can be enforced on them. In Cogitant for instance coreferent concept nodes must have the same label (but this restriction should be relaxed in a next version). More generally, there are variations on the way of dealing with distinct nodes refering to the same entity (either by mean of coreference links or by identical individual markers). We thus propose to distinguish an even simpler subclass of CGs: SCGs in *normal form*, that is without two nodes refering to the same entity.

SCGs come with two kinds of equivalent operations: elementary inference rules (see [Mug00] for a discussion on several sets of rules) and projection/coref-projection. Checking projection/coref-projection can be much more efficiently implemented than building a derivation with elementary rules, essentially because it allows efficient preprocessing (roughly said, restricting the possible images for nodes of the source SCG). As a matter of fact, projection is provided in all tools. It is complete w.r.t. deduction only if the target graph is in normal form. Coref-projection is a variant that is complete without this restriction [CM04]. It is not implemented in any tool at the moment.

In what follows, classical mathematical definitions of SCGs are given and are related, whenever it is possible, to the abstract syntax of the CG standard.

1. **SCG-module.** A SCG-module is a triple $\mathcal{M} = (V, I, B)$, where V is a SCG-vocabulary, I is a set of individual markers and B is a set of SCGs built over V and I. *(This is the same as the definition of a module in Sect. 6.10 of the standard where V is called type hierarchy, I catalog of individuals and B assertions. Furthermore, as in the SCG model there is no context, a module is simply a triple of sets and not a context composed of three contexts.)*
2. **SCG-vocabulary.** A SCG-vocabulary is composed of two (disjoint) sets T_C and T_R with T_C is a partially ordered set of concept types with a maximal element *(same as in Sect. 6.5, without neither defined types nor the absurd type)* and T_R is a partially ordered set of relation types. Each relation has an arity > 0 *(called valence)* and a signature *(same as in Sect. 6.6, without: 0-ary relation, defined types, and actors. The signature is defined in Sect. 6.3)*
3. **Individual markers.** I is a set of individual markers *(as in Sect. 6.7)*.
4. **Simple Conceptual Graph.** A SCG is a 5-tuple $(C, R, E, l, coref)$ where: C is a set of concept nodes *(as in Sect. 6.2 where a referent, cf. 6.7, is restricted to a designator that is either an individual marker or undetermined - also called generic)*. If c is a concept node its label $l(c)$ is the pair $(type(c), referent(c))$. R is a set of relation nodes *(as in Sect. 6.3 with the above restrictions for relation types)*. If r is a relation node its label $l(r)$ is the type of r. (C, R, E) is a bipartite multigraph *(as in Sect. 6.1 but with multiple edges explicitely allowed between two nodes)*. The labelling function l of the nodes has been given above; the labelling function of the edges is such that for any relation node r, r has exactly $k = arity(l(r))$ incident edges which are labelled $1, 2, \ldots, k$. The coreference relation $coref$ is a partition of

C, with each class being called a coreference set. *(We have gathered in this definition of a SCG: 6.1, 6.2, where the referent is restricted to the above designators, 6.3 with the above restrictions. The "blank" graph is also called the empty CG, and 6.9 where any concept must be in a coreference set and the dominant concept is not needed).*

5. **Normal Simple Conceptual Graph.** A normal SCG is a SCG where $coref$ is the identity. *There are no distinct nodes representing the same entity.*
6. **SCG-Query.** A SCG-query is a SCG. *We only define the most basic form of query; a straightforward extension is to add question marks on some generic concept nodes, that is to consider a lambda-SCG (see in the perspectives); we then obtain the existential conjunctive query of databases.*

As an example, let us consider a vocabulary where T_C contains the concept types *Top, Object, Cube, Ball, Attribute, Color, Shape, Ractangle, Square, Rhomb*, partially ordered as follows: *Object* $\leq$ *Top, Cube* $\leq$ *Object, Ball* $\leq$ *Object, Attribute* $\leq$ *Top, Color* $\leq$ *Attribute, Shape* $\leq$ *Attribute, Rectangle* $\leq$ *Shape, Rhomb* $\leq$ *Shape, Square* $\leq$ *Rectangle* and *Square* $\leq$ *Rhomb*. T_R contains the binary relations *prop, near*, and *onTop* and the ternary relation *between*. We have *onTop* $\leq$ *near*. Finally, this vocabulary contains the individual markers A and *blue*. We graphically represent the partial order T_C of this vocabulary in FIG. 1, as well as a simple conceptual graph built upon this vocabulary.

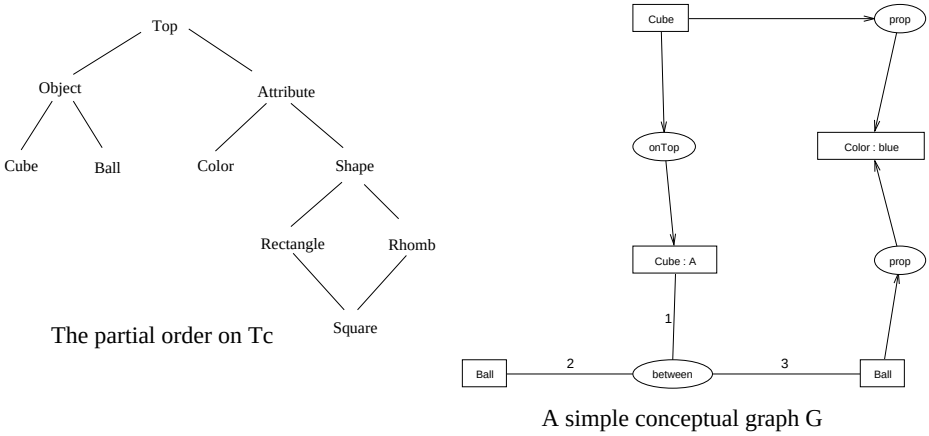

Fig. 1. A partial order of concept types T_C and a simple conceptual graph G.

The logical semantic Φ assigned to these constructs is well-known. A SCG is assigned to a conjunctive and existentially closed formula. If B is a set of SCGs, $\Phi(B)$ is the set of formulas assigned to the elements of B. Coreference is translated by equality. The set of formulas assigned to the vocabulary translates

the partial orders on types. Given a SCG module $\mathcal{M} = (V, I, B)$, $\Phi(\mathcal{M})$ is the set of formulas resulting from the union of $\Phi(V)$ and $\Phi(B)$.

3 Exchange format

Agreement on an exchange format is the first requirement for benchmarking. To decide on a particular format, we considered the three following criteria:

1. it must be able to express items in the SCG formalism (vocabulary, graph, ...);
2. it must be extensible enough to cope with more expressive CG fragments (e.g. rules) in forthcoming benchmarks;
3. to open the benchmark to a wide area of CG tools, this format must be already implemented in most tools.

A natural choice with respect to these criteria is CGIF. Since it is able to encode (at least) the whole FOL, it naturally satisfies the two first criteria. According to Philippe Martin's wiki page, it is implemented in all CG tools except our tool Cogitant, which is only able to write SCGs in CGIF (but not yet to read them).

Two versions of CGIF are available:

- the first is given in the "Conceptual graphs" ISO/JTC1/SC 32/WG2 N 000 proposed standard. We will refer to this version as CGIF 2001.
- the second version is given in the "Common Logic" ISO/IEC/JTC 1/SC32 1377 proposed standard. The basic version presented is called Core CGIF; a more expressive version is called Extended CGIF.

In the following, we recall the grammars of these three formats by considering their respective subgrammars that correspond to SCGs, and characterize the SCGs represented in these formats. Their semantics will be expressed via the translation Φ to FOL, and logical consequence between associated formulas is used to define the deduction problem of SCGs.

3.1 CGIF 2001

// Representation of a module
Module :=
`[Module:` *TypeHierarchy RelationHierarchy CatalogIndividuals Assertion* `]`

// Representation of the ordered set of concept types
TypeHierarchy :=
`[TypeHierarchy:` (*TypeLabelOrdering*)* `]`
TypeLabelOrdering :=
`(GT [TypeLabel "` *Identifier* `"] [TypeLabel "` *Identifier* `"])`

// Representation of an ordered set of relation types
RelationHierarchy :=
`[RelationHierarchy:` (*ValenceSpec* | *RelLabelOrdering* | *RelDefinition*)* `]`
ValenceSpec :=
`(Has [RelationLabel "` *Identifier* `"] [ Valence` *Integer* `])`
RelLabelOrdering :=
`(GT [RelationLabel "` *Identifier* `"] [RelationLabel "` *Identifier* `"])`
RelDefinition :=
`(Def [RelationLabel "` *Identifier* `"] [LambdaExpression "(lambda` *Signature* `)"])`
Signature :=
`()` | `(` (*Identifier* `,`)* *Identifier* `)`

// Representation of the catalog of Individuals
CatalogIndividuals :=
`[CatalogOfIndividuals:` (`['` *String* `']`)* `]`

// Representation of a CG
Assertion :=
`[Assertion:` (*Concept* | *Relation*)* `]`
Concept :=
`[` *Identifier* `:` ((`'` *String* `'`) | (`*` *Identifier*)) `]`
Relation :=
`(` *Identifier* (`['` *String* `']` | `[?` *Identifier* `]`)* `)`

To encode a SCG module in CGIF 2001, the following steps are performed:

- Let $<$ be the covering relation of the partial order on concept types in T_C. For each pair $(t1, t2)$ of concept types such that $t1 < t2$, the *TypeHierarchy* contains a *TypeLabelOrdering* `(GT [TypeLabel "t2"] [TypeLabel "t1"])`.
- For each relation arity k, let $<_k$ be the covering relation of the partial order on relation types of arity k. For each relation type r of arity k in T_R, of signature $(c_1, \ldots, c_k)$, *RelationHierarchy* contains a *ValenceSpec* `(Has [RelationLabel "r"] [Valence k])` and a *RelDefinition* `(Def [Relation Label "r"] [LambdaExpression "(lambda(c1, ..., ck))"])`. For each pair $(r1, r2)$ of relation types s.t. $r1 < r2$, the *RelationHierarchy* contains a *RelLabelOrdering* `(GT [RelationLabel "r2"] [RelationLabel "r1"])`.
- For each individual marker m in the vocabulary, *CatalogIndividuals* contains `['m']`.
- For each concept node c with marker m and type t, we note $f(c) = m$ if c is individual, otherwise $f(c)$ is a distinct identifier associated with each coreference class. Then *Assertion* contains the *Concept* `[t :'m']` is c is individual, or `[t : *`$f(c)$`]` if c is generic.
- For each relation node r with type t, arity k and whose ordered arguments are the concept nodes $(c_1, \ldots, c_k)$, *Assertion* contains the *Relation*

`(t` $f'(c_1) \ldots f'(c_k)$ `)` with $f'(c_i) =$ `['m']` if c_i is an individual concept node with marker m and $f'(c_i) =$ `[?`$f(c_i)$`]` otherwise.

The CGIF 2001 files provided for the proposed benchmarks will be obtained by applying this transformation to vocabularies and SCGs in normal form. By reading these files, the normal form of the encoded SCGs will be retrieved. The semantics of the objects encoded in those CGIF 2001 files will be the semantics Φ of the vocabularies and SCGs from which they were obtained.

As an example, here are the CGIF 2001 files associated with the vocabulary and graph of FIG. 1.
Encoding of the SCG vocabulary

```
[TypeHierarchy:
   (GT [TypeLabel:"Top"] [TypeLabel:"Object"])
   (GT [TypeLabel:"Top"] [TypeLabel:"Attribute"])
   (GT [TypeLabel:"Shape"] [TypeLabel:"Rhomb"])
   (GT [TypeLabel:"Shape"] [TypeLabel:"Rectangle"])
   (GT [TypeLabel:"Rectangle"] [TypeLabel:"Square"])
   (GT [TypeLabel:"Rhomb"] [TypeLabel:"Square"])
   (GT [TypeLabel:"Object"] [TypeLabel:"Cube"])
   (GT [TypeLabel:"Object"] [TypeLabel:"Ball"])
   (GT [TypeLabel:"Attribute"] [TypeLabel:"Shape"])
   (GT [TypeLabel:"Attribute"] [TypeLabel:"Color"])]
[RelationHierarchy:
   [RelationLabel:"prop"]
   [RelationLabel:"between"]
   (GT [RelationLabel:"near"] [RelationLabel:"onTop"])]
[CatalogOfIndividuals:
   [Cube:'A']
   [Color:'blue']]
```

Encoding of the SCG G

```
(prop [Ball *c2] [Color:'blue'])
(prop [Cube *c5] ['blue'])
(onTop ?c5 [Cube:'A'])
(between ['A'] [Ball] ?c2)
```

3.2 Core/Extended CGIF

In this version of CGIF, information encoded in SCGs will be written in Core CGIF, while information encoded in the support will be encoded in Extended CGIF.
// Representation of a CG
CG :=
   (*Concept* | *ConceptualRelation*)*
Concept :=

(*ExistentialConcept* | *CoreferenceConcept*)
ExistentialConcept :=
`[*`*Identifier*`]`
CoreferenceConcept :=
`[ :`*String*`]`
ConceptualRelation :=
`(` *Identifier* (`?`*Identifier* | *String*)*`)`

// Representation of the partial orders
PartialOrder :=
(`[If :` *CG* `[Then :` *CG* `]]`)*

To encode a SCG and its vocabulary in Core/Extended CGIF, we perform the following operations:

- Let $<_k$ be the covering relation for the partial order on relation types of arity k (note that concept types are considered as relation types of arity 1). For each pair of types $(t1, t2)$ such that $t1 <_k t2$, *PartialOrder* contains the rule `[If : [*arg1]` ... `[*argk] (t1 ?arg1` ... `?argk) [Then : (t2 ?arg1` ... `?argk)]]`.
- For each concept node c with marker m and type t, we note $f(c) = m$ if c is individual, otherwise $f(c)$ is a distinct identifier associated with each coreference class. For each individual concept node with marker m and type t in the SCG, *CG* contains `[ : m]` and `(t m)`. For each generic concept node c in the SCG, *CG* contains `[*`$f(c)$`]` and `(t ?`$f(c)$`)`.
- For each relation node with type t, arity k, and whose ordered arguments are the concept nodes $(c_1, \ldots, c_k)$, *CG* contains `(t` $f'(c_1) \ldots f'(c_k)$ `)` with $f'(c_i) =$ `m` if c_i is an individual concept node with marker m and $f'(c_i) =$ `?`$f(c_i)$ otherwise.

As for CGIF 2001, the SCGs retrieved by reading these files are the normal form of the encoded SCGs, and their semantics are defined by the translation Φ to FOL.

As an example, here are the Core/Extended CGIF files associated with the vocabulary and graph of FIG. 1.
Encoding of the SCG vocabulary

```
[If: [*x] (Object ?x) [Then: (Top ?x)]]
[If: [*x] (Attribute ?x) [Then: (Top ?x)]]
[If: [*x] (Rhomb ?x) [Then: (Shape ?x)]]
[If: [*x] (Rectangle ?x) [Then: (Shape ?x)]]
[If: [*x] (Square ?x) [Then: (Rectangle ?x)]]
[If: [*x] (Square ?x) [Then: (Rhomb ?x)]]
[If: [*x] (Cube ?x) [Then: (Object ?x)]]
[If: [*x] (Ball ?x) [Then: (Object ?x)]]
[If: [*x] (Shape ?x) [Then: (Attribute ?x)]]
[If: [*x] (Color ?x) [Then: (Attribute ?x)]]
```

```
[If: [x1] [x2] (onTop ?x1 ?x2) [Then: (near ?x1 ?x2)]]
```

Encoding of the SCG G

```
[:blue] [*c2] [:A] [*c4] [*c5]
(Ball ?c2) (Color blue)
(prop ?c2 blue)
(Cube ?c5)
(prop ?c5 blue)
(Cube A)
(onTop ?c5 A)
(Ball ?c4)
(between A ?c4 ?c2)
```

3.3 Discussion

As discussed above, a requisite for chosing a particular format (at least for the first installment of a benchmark) is that it should already be implemented in most CG platforms. Though CGIF is the best candidate for that purpose, we could not decide between the two versions (2001 and Core/Extended) for the following reasons:

1. On one hand, the adoption of the Core/Extended version of CGIF as an ISO standard should render obsolete the 2001 version;
2. On the other hand, it is doubtful that all CG platforms implementing CGIF have already migrated from the 2001 version to the Core/Extended version.

Should we want to decide between these two versions for further installments of a benchmark, we need to compare CGIF 2001 and Core/Extended CGIF:

- While CGIF 2001 exactly represents the objects (type hierarchies, SCGs) we manipulate, Core/Extended CGIF represents (with a CG notation) the formulas associated by Φ to these objects. Though the proximity of the Core/Extended version with logics could be useful to open CG tools to other KR formalisms, it also blurs the specificity of our graph-based approach.
- Core/Extended CGIF does not represent the signature (since the signature of a relation type is not a rule, but an integrity constraint in the database sense). Though the signature is not used to compute SCGs deduction (it is not taken into account by the semantics Φ), it is a useful guide when editing SCGs.
- In both versions of CGIF, nested graphs are used to represent SCGs and vocabularies. It can be a dangerous choice since the logical semantics of these nested graphs is not equivalent to the semantics of the SCGs and vocabularies they represent.

Ultimately, the choice of a format for our community's benchmarks should come from a collective decision. Moreover, this choice is not limited to the versions of CGIF used above, but we should consider XML-based formats (such that

CharGer XML format [Del05] or CoGiTaNT XML format (`http://cogitant.sourceforge.net/cogitant_html/cogxml.html`), to allow our tools to benefit from many efficient parsing tools.

Nevertheless, for a first installment of these benchmarks, we will provide the instances both in CGIF 2001, Core/Extended CGIF (as precised above), ... and CoGXML, that is the favored format for the tools (CoGITaNT, CoGUI) that will be used to generate these instances.

Here is the COGXML file associated with the graph G of FIG. 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<cogxml>
  <graph id="_g1" nature="fact" set="default_set" label="scene">
    <concept id="_c7" idType="_ct2" referent="individual"
      idMarker="_m1" gx="280" gy="140"/>
    <concept id="_c6" idType="_ct3" gx="310" gy="300"/>
    <concept id="_c5" idType="_ct1" referent="individual"
      idMarker="_m2" gx="150" gy="220"/>
    <concept id="_c4" idType="_ct3" gx="30" gy="300"/>
    <concept id="_c3" idType="_ct1" gx="150" gy="50"/>
    <relation id="_r7" idType="_rt0" gx="320" gy="220"/>
    <relation id="_r6" idType="_rt0" gx="320" gy="50"/>
    <relation id="_r5" idType="_rt1" gx="150" gy="140"/>
    <relation id="_r4" idType="_rt2" gx="150" gy="300"/>
    <edge label="1" rid="_r4" cid="_c5"/>
    <edge label="2" rid="_r4" cid="_c4"/>
    <edge label="3" rid="_r4" cid="_c6"/>
    <edge label="1" rid="_r5" cid="_c3"/>
    <edge label="2" rid="_r5" cid="_c5"/>
    <edge label="1" rid="_r6" cid="_c3"/>
    <edge label="2" rid="_r6" cid="_c7"/>
    <edge label="1" rid="_r7" cid="_c6"/>
    <edge label="2" rid="_r7" cid="_c7"/>
  </graph>
</cogxml>
```

4 Problems to solve

Existing tools are used in different contexts and propose different kinds of operations. In this paper we adopt the knowledge representation viewpoint, where the fundamental problem is deduction.

Definition 1 (SCG Deduction). *Given a SCG module $\mathcal{M} = (V, I, B)$ and a SCG Q defined on V and I, is Q deducible from $\mathcal{M}$ (*i.e. *is $\Phi(Q)$ deducible from $\Phi(\mathcal{M})$)?*

It is the basic problem from a knowledge representation perspective because deduction is the basic problem in logics. By considering SCG deduction as our

basic problem, we can compare our tools with FOL-based reasoners. Consistency (or satisfiability) is generally also a fundamental problem, but is not relevant here since every SCG is satisfiable. It is also the elementary operation to compute redundancy (a SCG is redundant if it is deducible from one of its strict subgraphs), to answer queries (see below), to check integrity of a SCG given a set of constraints [BM02], or to check applicability of a rule on a SCG [BM02], ...

Soundness and completeness are important (and even mandatory from our viewpoint) properties of tools implementing deduction. *Soundness* means that when the tool answers "yes", Q is indeed deducible from $\mathcal{M}$. *Completeness* means that when Q is deducible from $\mathcal{M}$, the tool actually answers "yes". In other words, when the tool answers "no" (and we assume that it always answers yes or no, as the problem is decidable), Q is not deducible from $\mathcal{M}$. If the completeness property is not fulfilled, a "no" answer does not mean anything, as it might be the case that Q is not deducible from $\mathcal{M}$ but it might also be the case that Q is deducible from $\mathcal{M}$ but the tool was not able to prove it.

As mentioned above, projection (or coref-projection) is sound and complete.

Another basic problem is "question answering".

Definition 2 (SCG Question answering). *Given a SCG module $\mathcal{M} = (V, I, B)$ and a SCG Q defined on V and I, give all answers to Q in $\mathcal{M}$.*

The associated counting problem (how many answers are there?) might be of interest too. The above problem assumes a specific definition of an answer. Roughly, an answer describes a way of instanciating the nodes of the query onto $\mathcal{M}$. More specifically, it is a mapping from the nodes of Q to the nodes of B such that if we replace each node of the query with its image we obtain a piece of knowledge deducible from the module. If B is in normal form (that is the set of SCGs considered as a single SCG is in normal form), we obtain a subgraph of B (the projection of Q in B).

If B is in normal form, an answer is a projection from Q to B. More generally, an answer is a coref-projection from Q to B, that is a mapping from each coref class of Q to a coref class of B that satisfies similar properties on labels and edges as projection. Each coref-projection from Q to B corresponds to a projection from the normal form of Q to the normal form of B if these normal forms exist (which depends on the constraints enforced on the coreference relation), and reciprocally.

5 Benchmarks

One can distinguish between several levels of benchmarks, depending on the tool properties that are to be evaluated. At first level, a benchmark can be used to check if tools accept a given format as input and as output. A second level is to test the ability of tools to solve a problem. As explained in previous section, we propose deduction and question answering as basic problems. When tools pass the two first levels, it is possible to consider a third level, which aims at

testing computational efficiency of tools. Each of these levels shoud be tested using different data, having different characteristics.

The first mandatory point consists in coming to an agrement on basic categories of conceptual graphs and one or several formats, with precise specifications for each CG fragment (as discussed in sections 2 and 3).

Then we propose to collect several benchmark data, with for each of them:

1. the CG fragment considered;
2. a precise definition of the problem to solve;
3. possibly the kind of difficulties represented by the benchmark. E.g the size of the knowledge base, the density of the graphs, etc...

As a first experiment, we propose to provide a benchmark issued from an industrial knowledge management tool[12]. This benchmark is limited to the SCG fragment, and furthermore to a particular kind of SCGs: in particular, relation types are all binary and all concept nodes have individual markers. However we think that this simplicity can be an advantage for a first step, as it should not yield computational difficulties and thus allows to focus on interoperability aspects: the exchange format and the problems to be solved. In other words, we focus here on the two first levels of a benchmark.

The industrial tool at the source of the benchmark allows to manage different kinds of knowledge structures: ontologies, thesaurus, annotation bases... All this knowledge is stored in a repository based on Topic Maps.

Our team is developping a reasoning service for this tool. In this goal, we have defined a transformation from the network of topics contained in the repository to the conceptual graph formalism which maps ontologies into the type hierarchies of conceptual graph model and the thesaurus as well as the annotation base into SCGs.

The first knowledge base on which we have applied this transformation is the demonstration base of Mondeca which describes acquisitions of compagnies in the sector of pharmacology. These descriptions are issued from an automatical process of annotation based on NLP tools.

The obtained benchmark is composed of :

- a concept type hierarchy (tree-structured) of 89 types;
- a relation type hierarchy (flat) of 92 relations. All these relations are binary and are provided with a signature (2 concept types from preceding hierarchy);
- a set of 6505 individual markers;
- a SCG composed of 222 connected components and a big amount of isolated concept vertices. Each concept node is an individual one (i.e. there is no generic marker) and the entire graph is in normal form. It contains 6505 concept nodes corresponding to different markers and 5495 relation vertices. Most of connected components are trees of small size (approximately 10 nodes). Two components have more than 4000 nodes and contain cycles.

[12] ITM (Intelligent Topic Manager) is a set of knowledge management tools developed by Mondeca (cf. `http://www.mondeca.com`)

We also propose a list of query graphs with associated results. For each graph, we give the boolean result for deduction problem, the number of different projections from it into the base and the list of image graphs induced by these different projections.

Let us end this section with ideas related to the third benchmark level, that is testing tools efficiency.

Though a real-case, large scale knowledge base is a precious benchmark to evaluate the performance of our tools on practical instances, a more precise evaluation could use the notion of *phase transition* [MSL92,SG95]. Though this notion was introduced for constraint networks, it can be directly translated intp the SCG formalism: Let Ω be a random SCG generator. A parameter of this generator is its hardness. When generating an instance of the SCG-deduction problem with hardness close to 0, every star graph in the query Q has lots of images in the knowledge base. With hardness close to 1, each star graph in the query has very few images in the KB. Probabilistically, there are lots of projections of the query in the first case, and none in the second. By varying this parameter, we can find a value for which the expected number of projections is 1. This is where the phase transition stands. For this hardness value, all sound and complete SCG deduction algorithms will suffer from a tremendous peak of inefficiency. This peak exists for all NP-complete problems, and is preserved by polynomial reductions between these problems.

To evaluate CG tools deduction efficiency for various hardness value, we could use a constraint network random generator, and translate the networks obtained into SCGs with the polynomial reduction G2C presented in [Mug00]. Such a range of SCG-deduction instances allows a more precise comparison of efficiency: some tools can be better for some hardness values, and worse for others.

6 Perspectives

In this paper, we have defined a basic building block for which we propose to provide benchmarks if there is an agreement on it among CG tool developers. This building block is built upon normal SCGs, which have a well defined logical semantic and are equipped with a commonly implemented operation, namely projection. We have proposed a restriction of the CGIF formats corresponding exactly to this fragment. The basic problems are deduction and query-answering.

For the first installment of these CG benchmarks, we will be able to evaluate if CG tools developpers really agree on the syntax and semantics of simple CGs, that are always presented as the common model for our community. Further benchmarks should be proposed each year, to compare and improve our tools on this common formalism, as well as to agree on the extensions we should cope with for the next benchmarks. This would ultimately lead to an increased expressivity of the formalisms handled by our tools, and to an enlarged common ground, necessary to publicize our results and tools outside the CG community.

In our opinion, the next data model to consider after SCGs would be *rules*, expressing knowledge of form "if Hypothesis then Conclusion" as they are essential in all knowledge-based systems.

Let us make a try. A SCG-rule is classically defined as a pair of lambda SCGs. A lambda SCG is a pair composed of a SCG and a list of distinguished concepts of the graph (formal parameters), which are generic concepts. A SCG rule is a pair of lambda SCGs with a bijection between the two lists of distinguished concepts. The first SCG is the hypothesis of the rule, the second is its conclusion. Now, the base B of a module is composed of a set of SCGs and a set of rules.

The logical semantic does not lead to any problem. In lambda-SCGs variables assigned to distinguished nodes are kept free. The formula assigned to a rule is the universal closure of the formula $\Phi(hypothesis) \rightarrow \Phi(conclusion)$.

The difficulty arises for defining rule application. The reason is that the way a rule is applied depends on the restrictions put on the form of the rule. Let us consider the three tools currently processing rules: Amine (Prolog+CG), Cogitant and Corese. In Cogitant and Corese, corresponding distinguished concept nodes must have the same type in the hypothesis and in the conclusion, and coreference links are not allowed inside the conclusion. These restrictions lead to a simple definition of a rule application. Rules are processed by a forward chaining mechanism, that was proven sound and complete. In Amine there is no restriction on the form of the rules (except that relation types are binary) and rules are processed by a Prolog-like backward chaining.

When rules are involved the deduction problem becomes the following: Given a module $\mathcal{M}(V, I, B)$ where B is composed of SCGs and rules, and a SCG Q defined on the same V and I, is Q deducible from B (that is $\Phi(Q)$ deducible from $\Phi(\mathcal{M})$)?

A SCG is said to be deducible from B if it can be obtained from the SCGs of B by applying (a finite number of times) rules of B. An answer is now a mapping (projection or coref-projection) from Q to a SCG deducible from B.

Deduction on rules is not decidable (we indeed obtain a computability model). One can distinguish a specific case of rules, which is decidable and exactly corresponds to rules used in Datalog, called range-restricted or safe. In these rules no new variable appears in conclusion; expressed in CGs, the conclusion part (excluding connection nodes) has no generic node. Examples of such rules are rules expressing properties of relations, as symmetry or transitivity.

Another data fragment that should be considered is that of *(positive) nested conceptual graphs*, which are widely present in tools but unsurprisingly with variations. The nested description is either a kind of referent or a third field of the concept label. In Cogitant there is in addition types of nestings. A trouble concerning nested conceptual graphs is their logical semantic. The semantic proposed by Sowa is not in FOL (as the special predicate representing description has an argument which is a formula - the formula representing the nested subgraph). A FOL semantic has been proposed in [CMS98]. As there is no general agreement on the logical semantic of nested CGs, defining the deduction and query answering problems is not easy.

Acknowledgements

We thank Mondeca for allowing CG community to freely use their topic map demonstration base.

References

[Bag05] J.-F. Baget. RDF Entailment as a Graph Homomorphism. In *The Semantic Web - ISWC 2005 (Proc. of 4^{th} International Semantic Web Conference)*, volume 3729 of LNCS. Springer, 2005.

[BM02] J.-F. Baget and M.-L. Mugnier. The Complexity of Rules and Constraints. *JAIR*, 16:425–465, 2002.

[CDH00] O. Corby, R. Dieng, and C. Herbert. A Conceptual Graph Model for W3C Resource Description Framework. In *Proc. of ICCS'00*, volume 1867 of LNAI, pages 468–482. Springer, 2000.

[CM04] M. Chein and M.-L. Mugnier. Types and Coreference in Simple Conceptual Graphs. In *Proc. ICCS'04*, volume 3127 of *LNAI*, pages 303–318. Springer, 2004.

[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/.

[Del05] Harry Delugach. *Charger User's Guide v3.5*, 2005. `http://charger.sourceforge.net/CharGer-Manual.pdf`.

[MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the Tenth National Conference on Artificial Intelligence*, pages 459–465, 1992.

[Mug00] M.-L. Mugnier. Knowledge Representation and Reasoning based on Graph Homomorphism. In *Proc. ICCS'00*, volume 1867 of *LNAI*, pages 172–192. Springer, 2000.

[SG95] B. M. Smith and S. A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 646–654, 1995.