A graph-based approach to Knowledge Representation

Marie-Laure Mugnier
LIRMM, France
mugnier@lirmm.fr
http://www.lirmm.fr/~mugnier/

February 26, 2003

Abstract

This paper presents an approach to knowledge representation and reasoning based on labeled graphs. The formalism is mainly based on the conceptual graphs introduced by Sowa. However, we emphasize the graph structure of conceptual graphs, while keeping logically founded reasoning. The basic notion to do reasoning is labeled graph homomorphism.

Contents

1 Introduction

1.1 Knowledge Representation and Reasoning

Knowledge Representation (KR) has long been recognized as a central issue in Artificial Intelligence (AI). The general problem tackled is to encode human knowledge in its various forms in such a way that this knowledge can be used by a computer to do reasoning.

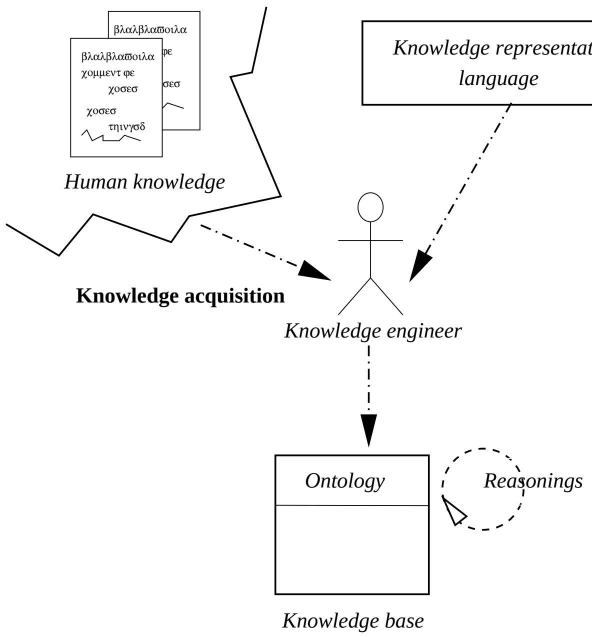


Figure 1: Building a knowledge representation

Figure 1 presents in a simplified way the classical process leading to a knowledge representation. Human knowledge about an application domain has to be acquired so that it can then be encoded using a KR language. The result is a knowledge representation under the form of a *knowledge base* (KB). The ontology provides the concepts and terms used to represent all knowledge about the application domain. It is the cornerstone of the knowledge representation. The capabilities of the knowledge formalism then determines the kinds of inferences that can be drawn from the knowledge base.

With respect to this schema, we are concerned with the study of *KR for-*

malisms that can be used as the theoretical foundations of a KR language[1].

Our very general definitions carefully avoid to specify what the notions of "human knowledge" and of "reasoning" exactly mean. Such a discussion would be far out of the scope of this paper. We shall rather indicate where our work takes place among existing KR formalisms or systems. For that, we shall adopt the requirements presented by Baader in [Baa99]. In short, we are interested in formalisms which:

- have a *declarative* semantics,
- are *logically* founded,
- allow for a *structured representation* of knowledge.

Declarative semantics. A KR formalism should allow to represent knowledge in a *declarative* way, that is to say the meaning of the knowledge represented should be definable independently of the programs processing the knowledge base. Namely, the knowledge engineer should not need to understand how reasoning procedures are implemented to build a knowledge representation; and one should be able to update the knowledge base content without modifying any program.

In this sense, Prolog for instance does not have a declarative semantics since the result of an inference computation may vary according to the form of the entries (order of the clauses or of the conjuncts). On the other hand, a language as Datalog fulfills this requirement. Ideally, the result of inferences should depend only on the semantics of the entries not on their syntax, i.e. semantically equivalent knowledge bases should lead to semantically equivalent results, whatever their syntactical forms are.

Logical foundations. By "reasoning" we mean "doing a logical inference". Generally speaking, doing an inference consists in producing a new expression from existing ones. By inference mechanism, we will more specifically mean a mechanism that is able, from explicitly encoded knowledge, to retrieve knowledge that is only implicitly represented. And the correctness of such a mechanism is defined relatively to logics, and more specifically to logical deduction.

[1] We distinguish between a KR formalism and a KR language, the latter including a specific syntax.

The expressions of the formalism should be translatable into a logic. This logic is not necessarily the first-order logic[2] (FOL). Such a mapping gives a logical semantics to the formalism. Let $\mathcal{K}$ be a KB and f be a mapping from $\mathcal{K}$ to a set of formulas of a logic. The inference mechanism should be *sound* w.r.t this semantics: if an expression i is retrieved from $\mathcal{K}$ then $f(i)$ has to be logically deducible from $f(\mathcal{K})$. It should also be *complete*: if, for an expression i, $f(i)$ is logically deducible from $f(\mathcal{K})$, then i should be retrieved from $\mathcal{K}$.

In other words, a procedure $\mathcal{P}$ (or algorithm, or system of rules) is sound with respect to a logical semantics if every inference made by $\mathcal{P}$ corresponds to a logical deduction. It is complete with respect to a logical semantics if every logical deduction in the logical language target of the formalism can be done by $\mathcal{P}$.

Soundness is usually ensured. But not all algorithms are complete. If an incomplete algorithm answers "yes" to the question "can i be retrieved from $\mathcal{K}$?", then the answer is correct. If it answers "no", then because of incompleteness, it does not mean anything. The incompleteness of an algorithm is generally motivated by efficiency reasons. Sometimes it is unavoidable. For instance deduction in FOL is only semi-decidable (it is not possible to build an algorithm checking that a formula can be deduced from another one, that stops in all cases when the answer is "no"), thus an algorithm that always gives a correct answer in a finite time is necessarily incomplete.

Knowledge structuration. A KR formalism should also provide a way of structuring knowledge, requirement that logics do not fulfill. One aspect of knowledge structuration is that semantically related pieces of information (for instance relative to an entity) should be gathered together. Another aspect is that different kinds of knowledge should be differenciated. For instance, ontological (or definitional) knowledge which is related to general categories (which may be called concepts or classes) and assertional knowledge, which makes assertions about a specific situation (such that "this specific entity belongs to a certain category, and has a certain relationship with another entity, ...").

This requirement is motivated by cognitive adequacy (i.e. its "conformity" to models of human cognition, cf. research in cognitive sciences) and also efficiency reasons (algorithmic efficiency of reasoning procedures, facil-

[2] In particular, much work has been done on knowledge representation with modal logics (allowing to represent time or beliefs for instance) and non-monotonic logics (allowing to deal with incomplete knowledge).

ity of managing the knowledge base).

These ideas were at the basis of the first KR formalisms, frames and semantic networks, that were far from logics. Frames have been introduced by Minsky [Min75] as record-like data structures for representing prototypical situations and objects. The key idea was to group together information relevant to a same situation/object.

Semantic networks have been developed for processing the semantics of natural languages (hence their name) [Qui68]. A semantic network is a diagram that represents connections (relationships) between concepts or objects (entities) in some specific domain of knowledge. Two basic links are the *is-a* link that relates two concepts or one object and the concept of which it is an instance, and the *property* link that assigns a property to an object or a concept. Inferences are done by following paths in the network. For instance, properties of an object are inherited following the is-a links. The system KL-ONE [BS85] and its descendants are the main representatives of this kind of semantic networks.

According to the classification presented in [Sow], such networks can be called *definitional*, since they emphasize the is-a relation between a concept and a newly defined concept. Other kinds of semantic networks are *assertional*, in the sense that they are designed to assert propositions. It is the case of the MIND system developed by Shapiro [Sha79], which later evolved into the Semantic Network Processing System (SNePS) [Sha00]. Cf. also [Leh92] for a synthesis about semantic networks.

The main criticism to semantic networks was their lack of a formal semantics. The same network could be interpreted in different ways, depending on the user intuitive understanding of its diagrammatical representation. Description logics (DLs), issued from the KL-ONE family, have been a successful attempt to combine well-defined logical semantics with efficient reasoning. They are now the most prominent family of KR languages. We will come back to DLs after the conceptual graphs introduction.

1.2 Conceptual Graphs

The conceptual graph (CG) model (introduced by Sowa in 1976 [Sow76], and developed in [Sow84]), is the synthesis of many works in AI, but can be seen as mainly finding its roots in the following areas: semantic networks, databases, logics, and specially the work on diagrammatical deduction of Peirce. One of the first motivation of Sowa was to represent the semantics of natural languages.

The brief presentation that follows comes essentially from Sowa's book

[Sow84] and our work [CM92] (for additional results).

The basic vocabulary is composed of two partially ordered sets of concept types and relation types. The partial order is interpreted as a subtype relation: $t_1 < t_2$ means that t_1 is a subtype of t_2, or equivalently, every entity of type t_1 is also of type t_2. There is also a set of names that can be used for individuals.

A simple conceptual graph (SG) is composed of two kinds of nodes, *concept* nodes representing entities and *relation* nodes representing relationships between these entities. Nodes are labeled by types, and one can indicate that a concept node refers to a special individual by adding to its label an individual name.

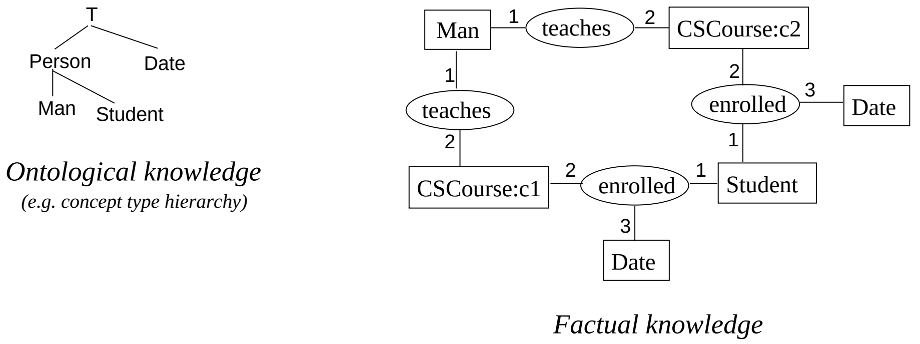


Figure 2: A simple conceptual graph

For instance, the SG of Fig. 2 asserts that "there is a man who teaches computer science courses c_1 and c_2 and there is a student who has enrolled in both courses and there are (unspecified) dates for these enrollments".

As presented here, CGs could be classified into the family of assertional semantic networks, but we will see later that its definitional component, here restricted to a minimum, can be more developed (see section 1.4).

Important differences with the CG formalism and its semantic networks ancestors are to be noticed: firstly, there is a clear distinction between ontological knowledge and other kinds of knowledge (such as factual knowledge); secondly, relations can be of any arity, whereas the edges of semantic networks represent binary relations only.

The model is provided with two (equivalent) operations, which directly manipulate the graphs. We present here the one we use in computations, projection. A *projection* from a SG G to a SG H is a global mapping from the nodes of G to the nodes of H, which preserves the relationships between entities of G, and may specialize entities and relationships. More formally, it is a graph homomorphism, that we will describe precisely later. For the moment, the only property to know is that this operation defines a *generalization/specialization* relation (or *subsumption* if we borrow the DL term) over SGs: G is more general than H (notation $G \geq H$ or $H \leq G$) if there exists a projection from G to H.

The basic problem is the following: given two SGs G and H, does $G \geq H$ hold? It is NP-complete, which is not too bad news in KR; furthermore we will see that it is equivalent to other problems in AI and databases, which have been very well studied from an algorithmic viewpoint.

An essential point is that CGs are provided with a logical semantics, called Φ. Roughly, the vocabulary is interpreted as follows. A predicate t is assigned to each type t (a unary predicate to a concept type and a predicate of same arity to a relation type) and a constant m to each individual m. Given n-ary types t_1 and t_2, $t_1 \leq t_2$ is interpreted by the formula $\forall X t_1(X) \rightarrow t_2(X)$, where X is a tuple of n variables (e.g. $\forall x Man(x) \rightarrow Person(x)$). To an SG is assigned an existentially closed formula, where terms (variables or constants) correspond to concept nodes.

E.g. the formula assigned to the graph of Fig. 2 is

$\exists x \exists y \exists t1 \exists t2 Man(x) \wedge CSCourse(c1) \wedge CSCourse(c2) \wedge Student(y) \wedge Date(t1) \wedge Date(t2) \wedge teaches(x, c1) \wedge teaches(x, c2) \wedge enrolled(y, c1, t1) \wedge enrolled(y, c2, t2)$

Projection is *sound* and *complete* with respect to the semantics Φ, i.e. given two SGs G and H, there exists a projection from G to H if and only if the formula $\Phi(G)$ can be deduced from the formula $\Phi(H)$ (using also the logical interpretation of the type hierarchies). We will show later that the simple graph model exactly corresponds to the existential, positive, conjunctive fragment of FOL (which we denote by FOL($\exists$, $\wedge$))[3].

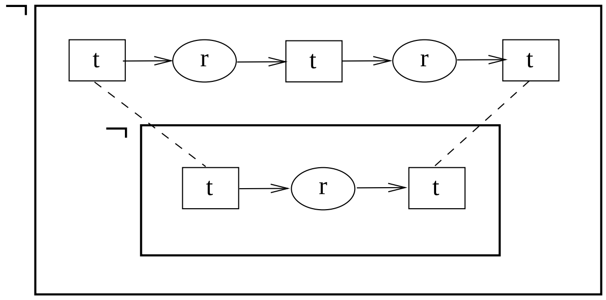
Simple graphs constitute the kernel of CGs. They can be used as such, to represent facts. They are also basic bricks for more complex constructs, corresponding to more expressive CGs.

In [Sow84], full CGs are introduced, inspired from Peirce's existential graphs, which form a diagrammatical system for logics (e.g. see their presentation in [Rob92]). The basic idea of the extension from SGs to full CGs is that every FOL formula can be written as a formula using solely the existential quantifier and the conjunction and negation operators. By adding to SGs *boxes* representing negation and *lines* (called co-reference links) indicating that two nodes represent the same entity, one obtains the same

[3] We will see that the universally quantified formulas associated with the support can be dropped.

expressive power as FOL (without functions). For instance, the graph of Fig. 3 says that the relation r is transitive on entities of type t, i.e. for all x, y, z of type t, if $r(x,\ y)$ and $r(y,\ z)$ then $r(x,\ z)$. Its logical translation is more precisely:

$$\Phi(G) = \neg(\exists x\ \exists y\ \exists z(t(x) \wedge t(y) \wedge t(z) \wedge r(x,y) \wedge r(y,z) \wedge \neg(r(x,z))))$$



G

For binary relation nodes, directions on edges may replace numbers 1 and 2.

Figure 3: A full CG

Peirce's existential graphs are provided with a sound and complete set of inference rules that can be adapted to CGs [Sow84] [Wer95a] (the only difference between full CGs and existential graphs is that in CGs nodes are typed). However, Peirce's rules heavily rely on human intuition[4], thus they cannot directly lead to an automatic procedure. A tableau calculus has been proposed in [Ker01].

1.3 A graph-based approach to KR

As explained in the first part of this introduction, our work belongs to a logical approach to KR but it is also *graph-based* in a sense we shall now explain. Our aim has been to develop and study a KR formalism, that is logically founded but autonomous from logics, based on graphs and graph-theoretic operations.

Motivations. Let us first outline our motivations for a graph-based approach to KR. They can be divided into two parts: qualities of graphs for knowledge modeling and qualities of graphs for computations.

[4]The rule of insertion namely allows one to insert *any* graph, which leads to an infinity of choices.

From a modeling viewpoint, we see two essential properties in the simple graph model. The *objects*, simple graphs, are easily understandable by an end-user (a knowledge engineer, or even an expert), at least if the graphs are reasonably small. And *reasoning* mechanisms are easily understandable too, for two reasons: projection is a global graph matching operation, thus it can be easily visualized on graphs; and the same language is used at interface and computing levels. In particular, no transformation has to be done on the components of the knowledge base before reasoning with it. Thus, reasoning can be explained on the user's representation itself.

Although there is a gap between the theoretical foundations studied here and a language usable in real knowledge-based applications, we would like to briefly mention some projects in which previous properties have been exhibited. The first project (GRAFIA) aimed at building a tool facilitating the task of the knowledge engineer in knowledge acquisition. The obtained tool, CGKAT, helps to index expertise documents by SGs, which can be retrieved using the specialization relation on SGs. CGKAT [Mar97] is built upon CoGITo, the CG workbench developed in our team, and a structured document editor. The tool was experimented in accidentology, where the knowledge engineer had to deal with expertise documents coming from experts of different domains (psychology, infrastructure engineering, vehicle engineering). The second project, MOGADOR, shared common features with the previous one. Indeed, simple conceptual graphs were used for indexing and querying documents. But the experiment was in document retrieval, which implies other requirements (e.g. dealing with large document bases, ordering answers to a query by decreasing relevance of corresponding documents with respect to the query, computing partial answers and not only exact answers ...) [Gen00]. For this application domain, the notion of a projection from a query graph Q to an indexing graph I is replaced by the more general notion of a partial projection from Q to a transformation of I. This operation was used to estimate the relevance of an answer relative to a query. Concept types were taken from the thesaurus of RAMEAU (about 400 000 types), a documentary language used in most French public and university libraries. The experiment proved the feasibility of the proposed system (w.r.t. computing time) and an improved relevance w.r.t. to the existing system based upon RAMEAU, mainly due to the use of semantic relations in addition to keywords. One side effect was also to prove the interest of SGs from a modeling viewpoint. Indeed, their graphical properties led to an indexing/querying tool that was considered as easy to use by the indexers. The indexers were master humanities students, not aware of conceptual graphs neither of RAMEAU; with the software and an indexing guide, they became extremely quickly able to build index graphs, that were considered of high quality by a senior university librarian.

A third project took place in knowledge engineering [BBV97]. Its purpose was the construction of tools for modeling and simulating human organizations, as emergency procedures for instance. One main difficulty in knowledge engineering is to validate a modeling, i.e. to check that the expert reasoning is correctly modeled. This validation is usually done when the design is achieved, here by simulating the constructed modeling of the organization. At this final stage, modifications are very costly. The key idea of the project was to overcome this difficulty by giving the expert the ability to use simulation *inside* the design cycle as a mean of enriching and building his modeling. This implies that the chosen modeling language enables the expert to follow reasoning step by step, directly on his own modeling. It was decided to build such a language upon conceptual graphs. General conceptual graphs equivalent to FOL were not considered as good candidates because they are indeed a diagrammatic system of logic that is not at the expert level. Instead, the language was grounded upon simple graphs and extensions (such as nested graphs) keeping their readability. Operations mixed simple graph deduction (i.e. projection) with non declarative procedures. First experiments were conclusive [BP98] and led to a prototype developed in the partner firm. Due to contingent reasons disconnected from scientific ones the project could not be continued.

From a computational viewpoint, labeled graph homomorphism firmly moors SGs to combinatorics. This notion (or its variant, relational structure homomorphism) has been recognized in the 90's as a central notion, unifying complexity results and algorithms obtained in several domains. On the other hand, considering graphs instead of logical formulas gives another view (for instance, some notions like path, cycle or neighborhood are natural on graphs) and provides other algorithmic ideas. Of course the algorithms obtained can be translated in logical terms, but they may have not be imagined in these terms. E.g. the *piece* notion on rules of Salvat's thesis [Sal97], the projection algorithm of Baget's thesis [Bag01] (also [BT01]) based on the *biconnected components* of a graph), or the polynomial projection algorithm in Kerdiles's thesis [Ker01] for a class of SGs having a *tree-like* structure. Sometimes, the advantages of graph and logical techniques can be combined as in the deduction mechanism for full CGs mixing the logical tableaux method and the SG projection, also presented in [Ker01].

Extensions of the SG model The full CGs *à la Peirce* are not graphs anymore: they are diagrams. The associated inference rules are not graph-based operations either. In our opinion, the qualities of the SG model (presented above) are lost: readability of objects as well as the inference mechanism, relationships with combinatorial problems.

Rather than jumping from SGs to full CGs, we prefer, depending on the kind of knowledge we would like to represent, to build extensions of the SG model, keeping its essential properties. These properties can be summarized as follows:

1. objects are *labeled graphs*

2. reasoning mechanisms are based on graph-theoretic operations, relying on a kind of graph homomorphism called *projection*;

3. it is logically founded, reasoning mechanisms being sound and complete w.r.t. FOL semantics, usually by way of the translation called Φ.

Let us give the example of an extension, SG rules. A rule represents an information of kind "if information H is present then information C can be added". H is called the hypothesis of the rule and C its conclusion. This notion of a rule has been widely used in AI to represent implicit knowledge, that can be made explicit by applying the rules.

Rules could be represented as full CGs, but so doing they would loose their specificity and could not be processed in a particular manner. We define a rule as a bicolored SG: one color (white in figures) defines the hypothesis, the other the conclusion (black in figures). For instance, the rule of Fig. 4 has the same semantics as a the full CG of Fig. 3: it says that the relation r is transitive on entities of type t. Rules are processed with graph operations in a forward or backward manner. The notion of a rule application is very simple. A rule R is said to be applicable to a SG G if its hypothesis H can be projected to G. Then, it can be applied to G: applying R consists in adding the conclusion of R to G in a way guided by the projection from H to G. Rules are provided with a logical semantics extending that of SGs, such that graph operations are sound and complete.

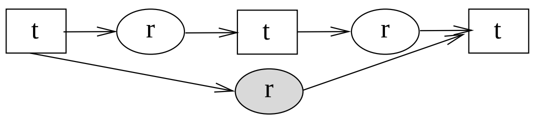


Figure 4: A rule

Other extensions presented in this paper are *nested graphs*, *constraints*, as well as models *combining* rules and constraints. These latter ones are not yet achieved extensions of SGs. Indeed, they have a simple operational semantics but are not provided with a logical semantics: it does not mean that they should not be provided with a logical semantics but only that the issue remains open.

Several approaches to CGs. Let us end this section by relating our approach to other works about CGs. Research about CGs can be roughly classified following three axes: CGs as a *diagrammatic interface* for other formalisms, CGs as a *diagrammatic system of logic*, and CGs as a *graph-based KR formalism*. Many works are mainly interested in the visual qualities of conceptual graphs. The expressivity and the readability of the obtained representations are the main criteria for evaluating a formalism. We shall not forget that a main motivation for conceptual graphs was the processing of natural language semantics: in this perpective, the similarity of a conceptual graph with the natural language sentence it represents is a dominating criterion. When reasoning mechanisms are associated with the representations, they are not part of this formalism: they are performed in a procedural way, or rely on another formalism into which the CGs are translated. In this approach, conceptual graphs are seen as a diagrammatical interface for other formalisms (e.g the Knowledge Interchange Format, KIF). Other works can be seen as the continuation of Peirce's work on a diagrammatical system of logics. Conceptual graphs are then diagrams and reasoning is based on diagrammatic operations. Finally, the graph-based approach emphasizes the following points which distinguishes it from the other two approaches:

- CGs are seen as a KR *and reasoning* formalism. Thus they are provided with their own operations to do reasoning.
- Reasoning mechanisms should be logically sound *and complete*
- Reasoning operations should be realizable in an *efficient way*. That is why decidability and complexity studies, as well as the design of efficient algorithms, are an important issue.
- Reasoning mechanisms should be based on graph-theoretic notions, as the structure underlying objects is graph.

Let us add that we started to be really interested in CGs when we proved that the graph operations on the SG fragment were logically complete. This established the graph homomorphism notion as a key operation for reasoning. The results we have obtained as well as recent results in other fields (see section 2.7) strengthen our belief that graph homomorphism is a key notion.

1.4 CGs and DLs

Since DLs and CGs are both rooted in semantic networks and logically founded, the question of their relationships has often been asked.

DLs are mainly concerned with reasoning about concept and relation definitions. The starting point is the declaration of atomic concepts and roles (relationships). Then, using a set of constructors, complex concepts and roles can be defined from existing concept/roles. The semantics of the concepts and roles is set-theoretic, which also gives a direct translation to FOL (let us call it f). The formula associated with a concept (binary role) term has one (two) free variable(s). The expressivity and complexity of the DL languages obtained depending on the constructors considered have been systematically studied (e.g. see [Baa99] for a synthetic presentation).

For instance, the concept definition below assigns the name D to a complex concept describing all men who teach at least one course and teach only computer science courses:

$D = Man \sqcap \exists teaches.Course \sqcap \forall teaches.CSCourse$

The FOL formula associated with the complex concept is:

$f_D(x) = Man(x) \wedge \exists y(Course(y) \wedge teaches(x,y)) \wedge \forall z(teaches(x,z) \rightarrow CSCourse(z))$

And the definition can be interpreted in FOL by the formula $\forall x D(x) \leftrightarrow f_D(x)$.

A knowledge base is split into its definitional and assertional components. It is composed of a terminology TBox $\mathcal{T}$, finite set of concept/roles definitions, and an ABox $\mathcal{A}$, finite set of assertions. There are two basic problems. The first one is the *subsumption* problem: Given a TBox $\mathcal{T}$ and two concepts C and D, is C a subconcept of D w.r.t. $\mathcal{T}$? In logical terms, if $f(\mathcal{T})$ denotes the logical interpretation of $\mathcal{T}$, $f_C(x)$ and $f_D(x)$ denote the logical formulas associated to the type definitions C and D, C is a subconcept of D w.r.t. $\mathcal{T}$ iff $f(\mathcal{T}) \vDash ((f_C(x) \rightarrow f_D(x))$. The other basic problem is the *instance* problem which asks whether a given individual is an instance of a given type, w.r.t. $\mathcal{A}$ and $\mathcal{T}$.

In CGs, similarly as in DLs, a concept (or a relation) can be defined in terms of existing ones. It is not defined by means of constructors, but by a SG, with a distinguished node (or k distinguished nodes for a k-ary relation), called a (k-ary) lambda-abstraction. E.g. consider the definition $T = \lambda x G$, where $\lambda x G$ is the graph of Fig. 5. This graph represents a concept type describing all men who teach a computer-science course and have a child who has enrolled to this course (the concept node head of the definition being marked $*x$). The formula associated to a lambda-abstraction has as many free variables as distinguished nodes. In previous example,

$\Phi(\lambda x G) = \exists y \exists z \exists t Man(x) \wedge CSCourse(y) \wedge Student(z) \wedge Date(t) \wedge teaches(x, y) \wedge hasChild(x, z) \wedge enrolled(z, y, t)$

Let us focus on concept types definitions (for relation types the principles are the same). The formula associated to the definition of a concept type T is $\forall x T(x) \leftrightarrow \Phi(\lambda x G)$.

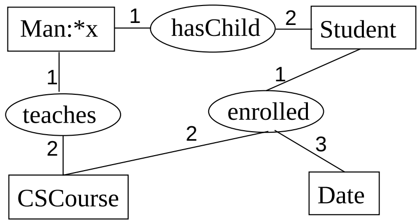


Figure 5: A CG concept type definition

Although type definitions were introduced as soon as 1984, their processing received little attention. However, Leclere extended the framework of simple graphs to concept and relation types definitions. First, two operations, type expansion and type contraction, are defined, allowing to "replace" in a SG a type by its definition, or, under certain conditions, a part of the graph with a defined type[5]. The specialization relation is extended by these two operations, and it is shown that soundness and completeness properties are kept, i.e. given two SGs G and H defined on a vocabulary $\mathcal{S}$, $H \leq G$ if and only if $\Phi(\mathcal{S}), \Phi(H) \vDash \Phi(G)$, where $\Phi(\mathcal{S})$ is the set of formulas associated with the vocabulary, which now includes formulas translating types definitions.

In this framework (direct or indirect) recursive definitions are forbidden. Then, to each SG (or lambda-abstraction) is associated a unique expanded form, obtained by performing type expansion until stability. The expanded form is a SG where all types are atomic. Thus comparing two expanded SGs can be done by projection. Let us add that at our best knowledge type definitions have never been studied beyond the SG model.

[5]These operations were introduced in [Sow84] but their definitions led to problems.

Let us now compare CGs and DLs. The results eventually obtained show that the relationships between both models are poor. Let us mention the work of Baader, Molitor and Tobies who have pointed out the equivalence of a fragment of SGs (connected graphs with a tree-like structure, binary relations only, with one distinguished concept node as in a unary lambda-abstraction) with a DL called $\mathcal{ELIRO}$ (provided with conjunction, existential restriction, inverse role and role conjunction) [BMT99]. They have also pointed out that they could not identify a natural fragment of CGs corresponding to an expressive known DL.

This result shows how one has to struggle to establish a correspondence between the two formalisms, amounting to poor models that are not interesting in themselves because each formalism looses its interesting features: cycles and n-ary relations for CGs, variety of constructors for DLs. For instance, the CG definition of Fig. 5 cannot be expressed in DL, whereas the DL definition D given above cannot be expressed by CG definitions[6]. On one hand we know that even the most expressive DLs cannot express the whole FOL($\wedge$, $\exists$) fragment [Bor96]. On the other hand, projection cannot handle negation, even restricted to atomic negation on primitive types (see section 5.3), thus one cannot add restricted forms of negation "for free" to the SG model.

Let us bring another argument. The reference DL is $\mathcal{ALC}$, a fairly general DL, in which subsumption remains decidable. It has been pointed out that $\mathcal{ALC}$ can be translated into the two variable fragment of FOL, L_2, which consists of all formulas without functions that can be constructed using two variables only, equality permitted [Bor96]. A result of Kolaitis and Vardi [KV00] [7] implies that acyclic SGs coincide in expressive power with the two variable fragment of FOL($\exists$, $\wedge$). Thus, the "intersection" of SGs and DLs is not likely to lead to better results than the one cited above.

One could consider more complex kinds of CGs, such as special kinds of rules or of full CGs, with the hope of finding correspondences with some DLs, but once again one would have first to restrict the SG components. For instance, the transitive closure of a relation cannot be expressed in $\mathcal{ALC}$, whereas it can be handled with a very specific fragment of CG rules (the so-called range-restricted rules) for which deduction is decidable.

[6]But it could be expressed by *rules*, cf. section 3.4.

[7]see "expressivity results" at the end of section 2.7.5.

1.5 Paper Organization

The first section is devoted to the foundations of our work: simple (conceptual) graphs. We give detailed definitions, discuss possible variations, and study the properties of the model - expressivity, structural properties, complexity of deduction. We also present the equivalence of SG deduction and other problems in AI and databases, via the notion of a labeled graph homomorphism, and draw complexity results from this equivalence.

We then present two extensions of simple graphs: rules and constraints. These extensions keep the fundamental properties of simple graphs, i.e. objects are built upon simple graphs, reasoning mechanisms are based on projection and they are logically founded, their logical semantics extending Φ. The decidability/complexity of the associated problems is established, for the general case as well as for some interesting specific cases.

Full conceptual graphs, equivalent to the first-order logic, are presented in section 5. We discuss ways of exploring the gap between simple graphs and full conceptual graphs, with negation in mind and graph-based reasoning as a guide.

Formalisms combining rules and constraints in different ways are then presented. The decidability/complexity of the associated problems is systematically studied. Reasoning mechanisms are non-monotonic and their logical interpretation remains an open issue.

Simple graphs, rules and constraints are the components defining a family of formalisms, we call the $\mathcal{SG}$ family. The generic deduction problem asks, given a knowledge base $\mathcal{K}$ and a SG Q, whether Q can be deduced from $\mathcal{K}$. Depending on what kind of objects compose the knowledge base one obtains the different members of the $\mathcal{SG}$ family. We end with an overview of this family.

About the paper form. We have chosen not to include any proof (we refer to the appropriate papers), except some straightforward ones.

In order to facilitate quick skimming through this document, some parts (about variations, related works, references) are marked as notes. In principle the document should be readable without them. A bullet ends each note. Recalls about orders and graphs are given in Appendix.

2 Basic Notions: the Simple Graphs Model

Simple conceptual graphs, or *simple graphs* (when no confusion can be made with classical graphs) were first introduced in [Sow84] (although they were not called like that). Then in [CM92] we corrected some flaws in definitions and results and made a first study of the model properties. With the years passing, definitions have varied, evolved and new results have been added, including connections with other domains. This section aims at providing essential definitions about the simple graphs model as well as synthesizing main results. I will also point out tight connections that have become obvious with other domains: graph homomorphism, problems around queries in databases, constraint satisfaction problems.

I will first present the formalism itself and its logical semantics, then the operations which define a specialization/generalization relation over simple graphs, and finally the results proving that simple graphs exactly correspond to the existential positive and conjunctive fragment of FOL, the graph operations being sound and complete with respect to logical deduction. Coreference links and nested graphs are then introduced. They are extensions to simple graphs, which are convenient modeling tools, but from an expressivity viewpoint they are actually equivalent to simple graphs. Last section is devoted to complexity of reasoning in this model. It gathers results obtained on simple graphs, as well as results obtained on equivalent problems in other domains which have been very active about complexity studies.

2.1 The descriptive level

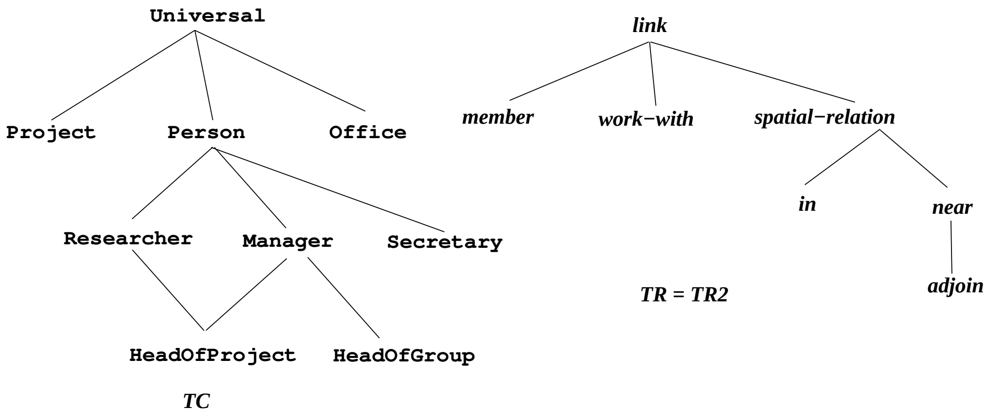
As explained in the introduction, the ontology is the cornerstone of a knowledge representation. However, since the simple graphs formalism is a basic model, it only allows to define a primitive ontology. We call it a *support* to enhance the fact that it contains very little knowledge about the world. More information is supposed to be encoded in other structures defined upon the support (e.g. types definitions, rules or constraints attached to types).

The support defines the vocabulary available to build simple graphs. It encodes a set of concept types and a set of relation types, both partially ordered by a specialization (or subtype, or a-kind-of) relation which has the following meaning: for two types t_1 and t_2, if $t_1 \leq t_2$, every instance of t_1 is also an instance of t_2 (in other words, the extension of t_1 is included in that of t_2). It also provides names (called individual markers) that can be used to mark specific entities.

Simple graphs (SGs) are defined with respect to a support. As such they represent factual knowledge, but are also basic bricks used in more complex constructs. A SG is a bipartite labeled graph (strictly speaking, it is a *multigraph*, since there can be several edges between two nodes). One class of nodes, concept nodes, represents entities, the other class, relation nodes, represents relationships between these entities. Nodes are labeled by elements of the support.

Definition 1 (Support) *A* support *is a 3-tuple* $\mathcal{S} = (T_C, T_R, \mathcal{I})$*.* T_C *and* T_R *are two partially ordered finite sets, respectively of* concept types *and* relation types*.* T_C *is required to have a greatest element, denoted by* $\top$*, and called the universal type, representing anything of the domain.* T_R *is partitioned into subsets* $T_R^1 \dots T_R^k$ *of relation types of arity* $1 \dots k$ *respectively* $(k \geq 1)$*. Two elements of distinct subsets are non comparable. Both orders on* T_C *and* T_R *are denoted by* $\leq$ *(*$x \leq y$ *means that* x *is a subtype of* y*).* $\mathcal{I}$ *is a countable set of* individual markers*.* T_C*,* T_R *and* $\mathcal{I}$ *are pairwise disjoint. All supports also possess the marker* $*$*, called the generic marker, which does not belong to* $\mathcal{I}$*. For convenience, the set* $\mathcal{I} \cup \{*\}$ *is partially ordered in the following way:* $*$ *is the greatest element and elements of* $\mathcal{I}$ *are pairwise incomparable.*

Individual markers are meant to refer to specific entities, whereas the generic marker is used to refer to an unspecified entity. Figure 6 illustrates the notion of a support, introducing types and markers we will use in subsequent examples.



$\mathcal{I} = \{J., K., \#124, ...\}$

Figure 6: A support

Note. Variations about the support. In [Sow84] the concept type set is a lattice. However, since the lowest greater (or greatest lower) bound of two concept types does not necessarily have a meaning, we prefer to relax this constraint and consider a looser structure, a partial order. Our interpretation of this structure is naturally order-theoretic (in the sense of [BHP+92]).

Relation types are often provided with a *signature*, which specifies the arity of the relation type but also the maximal concept type of each of its arguments. For instance, assume the relation $hasChild$ is a binary relation from one human being to another, interpreted as "the person represented by the first argument has a child who is the person represented by the second argument". Its signature could be $hasChild(Person, Person)$. Signatures are formally defined by a mapping σ, which to every relation type $r \in T_R^j$, $1 \leq j \leq k$ associates $\sigma(r) \in (T_C)^j$; moreover this mapping fulfills the following condition: $\forall r_1, r_2 \in T_R^j$, $r_1 < r_2 \Rightarrow \sigma(r_1) \leq \sigma(r_2)$, i.e. for all $1 \leq i \leq j$, the ith argument of $\sigma(r_1)$ is less or equal to the ith argument of $\sigma(r_2)$. For instance, assume the relation $hasDaughter$ is a subtype of $hasChild$; then it is necessarily a binary relation with arguments less or equal to $Person$, for instance $\sigma(hasDaughter) = (Person, FemalePerson)$, assuming $FemalePerson < Person$.

Individual markers may be typed, i.e. a concept type can be associated to each marker; this point will be discussed later.

We consider primitive types, in the sense that a type cannot be defined in terms of other types. The "meaning" of a type is only given by its position in the specialization hierarchy. [Sow84] introduces a type definition mechanism, which allows to define a new type by a simple graph. The specialization relation on types is no more *given* but needs to be *calculated* (as it is the case in description logics). We will not tackle this topic here (however more details can be found in sections 1.4 and 3.4). Although introduced in [Sow84], the processing of type definitions has been little studied. Leclere has proposed an extension of the simple graph model to defined types, keeping sound and complete graph reasoning [Lec95][Lec96][Lec97] [Lec98]. •

Definition 2 (Simple Graph) *A* simple graph G*, defined over a support* $\mathcal{S}$*, is a finite bipartite multigraph* $(V = (V_C, V_R), E, l)$*.* V_C *and* V_R *are the node sets, respectively of* concept nodes *and of* relation nodes*.* E *is the multiset of* edges*. Edges incident to a relation node are totally ordered; they are numbered from 1 to the degree of the node. An edge numbered* i *between a relation node* r *and a concept node* c *is denoted by* (r, i, c) *and identifies a unique element of* E*.* E *will also be seen as a set of such triples. Each node has a label given by the mapping* l*. A relation node* r *is labeled by* type(r)*, an element of* T_R*, called its* type*, and the degree of* r *must be equal to the arity of type(*r*). Thus, if* $r \in T_R^k$*,* $|\{(r,\ i,\ c) | (r,\ i,\ c) \in E\}| = k$

and $\{i|(r,\ i,\ c) \in E\} = \{1,\ \dots\ k\}$. *A concept node c is labeled by a pair* (type(c), marker(c)), *where* type(c) *is an element of* T_C, *called its* type, *and* marker(c) *is an element of* $\mathcal{I} \cup \{*\}$, *called its* marker.

A SG may be empty. It may contain concept nodes only. But as soon as it contains a relation node, it contains at least one concept node, since there are no 0-ary relation types. Also notice that a SG does not need to be a *connected* graph. We will often conflate a set of SGs, $\{G_1,\ \dots\ , G_k\}$, and the SG $\mathcal{G}$ obtained by performing the disjoint sum of $G_1,\ \dots\ , G_k$.

A concept node with a generic marker is called a *generic node*: it refers to an unspecified entity of a certain type; otherwise it is called an *individual node*: it refers to a precise individual. We will adopt the following classical conventions about SGs. In the drawing of a SG, concept nodes are represented by rectangles and relation nodes by ovals. In textual notation, rectangles are replaced by `[]` and ovals by `()`. Generic markers are omitted. Thus a generic concept label $(t, *)$ is simply noted t. An individual concept label (t, m) is noted $t : m$. When in our examples we use binary relations only, numbers on edges are replaced by directed edges: a relation node is then incident to exactly one incoming and one outgoing edge. Fig. 7 shows two (connected) simple graphs G and Q assumed to be defined over the support of Fig. 6.

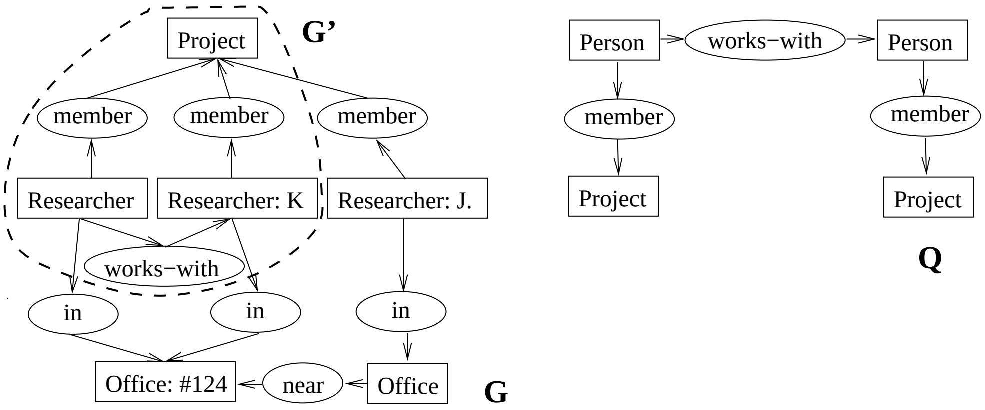


Figure 7: Simple graphs.

Simple graphs as hypergraphs A *directed hypergraph* $\mathcal{H} = (\mathcal{X},\ \mathcal{E})$ is composed of a set $\mathcal{X}$ of nodes and a set $\mathcal{E}$ of tuples over $\mathcal{X}$, called *hyperedges*. A simple conceptual graph can be seen as a directed hypergraph, labeled on nodes and hyperedges: the nodes of the hypergraph are the concept nodes, and there is one hyperedge $(c_1, ..., c_k)$ with label t_r for each relation node of type t_r and neighbors c_1, ..., c_k (in this order). If such a view of simple graphs may not be desirable at an interface level, Baget has shown in his thesis that considering relations as hyperedges instead of nodes could lead to more efficient algorithms [Bag01]. See also the hypertreewidth notion in section 2.7.5.

Conjunctive types Let us now consider the possibility of associating a set of types to a concept node instead of only one type. In object oriented-languages, one would call this mechanism *multi-instanciation*. The intended meaning of "this entity is of type $\{t_1 \ ... \ t_n\}$" is equivalent to "this entity is of all types t_1, ..., t_n", which can also be understood as "this entity is of type $t_1 \wedge ... \wedge t_n$", where the conjunction corresponds to the notion of the greatest lower bound (see below). Such a feature yields a more concise concept type set since several incomparable types always have a common subtype even if it is not encoded in the support.

Definition 3 (Conjunctive concept types) *Let T_C^* be the set of all non empty parts of the set T_C. T_C^* is provided with the following quasi-order*[8]*: given two sets $T = \{t_1 \ ... \ t_n\}$ and $S = \{s_1 \ ... \ s_p\}$, $T \leq S$ if for every $s_j \in S$, $1 \leq j \leq p$, there exists a $t_i \in T$, $1 \leq i \leq n$, such that $t_i \leq s_j$. Let us say that a set of T_C^* is redundant if it contains two types t_1 and t_2 such that one is greater than the other. In this case, the greater one can be removed since it does not add information. An irredundant set is obtained by removing all types greater than another type in the set. Let us call $T_C^\wedge$ the subset of T_C^* composed of the irredundant elements. An element of $T_C^\wedge$ is called a* conjunctive concept type.

Check that $T_C^\wedge$ is provided with a partial order (there are no more equivalent elements). Furthermore it is a lattice: the greatest lower bound of two conjunctive types is the irredundant type obtained from the union of these sets; and it possesses a greatest element $\{\top\}$; since $T_C^\wedge$ is a finite set, this two properties imply that it is a lattice.

Property 1 *$T_C^\wedge$ is a lattice*

In what follows, conjunctive types will prove to be useful for defining some basic mechanisms in a generic way, avoiding restrictions related to type compatibility.

[8] or preorder.

Bibliographic note. In his thesis [Bag01], Baget also defines *conjunctive relation types*, which allow him to obtain a more concise coding of simple graphs, called *compressed* graphs: in these graphs, there are no twin relation nodes, i.e. distinct relation nodes having the same neighbors in the same order. Any simple graph has a uniquely associated compressed graph, which is obtained by replacing every maximal set of twin relation nodes by one relation node whose type is the conjunctive type corresponding to the greatest lower bound of the original relation types. Both graphs are equivalent from an intuitive and logical viewpoint. Compressed graphs lead to more efficient algorithms[9].

Conjunctive concept types can be found in some CG formalisms, for particular purposes. [BMT99] introduces conjunctive types on unordered supports to establish the equivalence between a fragment of simple graphs and a specific description logic. In [CC98] a conjunction of fuzzy types is associated with a concept node. •

2.2 Logical Semantics

The semantics Φ maps SGs to the existential conjunctive and positive fragment of FOL. Given a support $\mathcal{S}$, a constant is assigned to each individual marker and an n-ary (resp. a unary) predicate is assigned to each n-ary relation (resp. concept) type. For simplicity, we consider that each constant or predicate has the same name as the associated element of the support. A set of formulas $\Phi(\mathcal{S})$ is assigned to any support $\mathcal{S}$, translating partial orders on types. More specifically, for all distinct types t_1 and t_2 such that $t_2 \leq t_1$, one has the formula $\forall x_1 ... x_p (t_2(x_1, \ ..., x_p) \rightarrow t_1(x_1, \ ..., x_p))$, where $p = 1$ for concept types, and p is otherwise the arity of the relation types[10].

Given any SG G, a formula $\Phi(G)$ is built as follows. A term is assigned to each concept node: a distinct variable for each generic node, and the constant corresponding to its marker otherwise. Then an atom is associated to each node. The atom $t(c)$ is assigned to a concept node x, where t is the type of x, and c is the term assigned to x. The atom $r(c_1, \ ..., \ c_k)$ is assigned to a relation node x of arity k, where r is its type and each c_i is the term assigned to the i-th neighbor of x. Let $\alpha(G)$ be the conjunction

[9] The basic property exploited in algorithms, is that, given two simple graphs G and H, there are less projections from the compressed form of G to the compressed form of H than from G to H.

[10] Notice that one could add the formula $\forall x \top(x)$ to the logical interpretation of the concept type set. For the results given in this paper, the presence or absence of this formula does not change anything.

of these atoms. $\Phi(G)$ is the existential closure of $\alpha(G)$. E.g. the formula assigned to the subgraph G' in Fig. 7 is $\exists x \exists y(Researcher(x) \wedge Project(y) \wedge Researcher(K) \wedge member(x,y) \wedge member(K,y) \wedge works\text{-}with(x,K))$.

The definition of Φ is naturally extended to conjunctive types. Let c be the term assigned to a concept node of type $\{t_1, \ ..., \ t_k\}$. Then the atom assigned to this node is $t_1(c) \wedge \ ... \ t_k(c)$. Similarly, in the case of conjunctive relation types, the atom assigned to a relation node of type $\{r_1, \ ..., \ r_p\}$ with arity k would be $r_1(c_1, \ ..., \ c_k) \wedge ... \wedge r_p(c_1, \ ..., \ c_k)$ [Bag01].

2.3 Reasoning

The *specialization/generalization* relation (denoted by $\leq$ or $\geq$) — or *subsumption* relation — is the fundamental notion for reasoning with simple graphs. We will see that it exactly corresponds to deduction for the fragment of FOL corresponding to SGs. This relation can be defined in terms of a sequence of elementary graph operations: given two simple graphs G and H, $H \leq G$ (G subsumes H) if H can be obtained from G by a sequence of such operations. It can also be computed by a global operation, the so-called *projection*, which essentially is a graph homomorphism.

In both cases, one first has to be able to compare node labels.

2.3.1 Comparing node labels

Relation node labels are naturally ordered by the partial order on T_R. For concept nodes, the orders on types and on markers have to be combined.

Definition 4 (Partial order on concept node labels) *Recall the following partial order is defined on the marker set $\mathcal{I} \cup \{*\}$: $*$ is the greatest element (for all $i \in \mathcal{I}$, $i \leq *$) and elements of $\mathcal{I}$ are pairwise non comparable. Then the* partial order on concept node labels *is the product of the partial orders on T_C and $I \cup \{*\}$, i.e. $(t,m) \leq (t',m')$ iff $t \leq t'$ and $m \leq m'$.*

In other words, a concept label (t,m) is more specific than a concept label (t',m') if t is a subtype of t' and, if m' is an individual marker, then $m = m'$ (otherwise m can be any marker).

For conjunctive concept types the order on $T_C^{\wedge}$ is considered. The following property holds true (but would not for non conjunctive types):

Property 2 *The set of concept conjunctive labels defined by a support, i.e. $T_C^{\wedge} \times (I \cup \{*\})$, is a sup-semi-lattice.*

It is not a lattice since two concept labels with different individual markers do not have a greatest lower bound (it would become one if we added a minimal element, the "absurd label").

Note. Unique Name Assumption. Let us open here a parenthesis. The fact that two concept labels with distinct individual markers are non-comparable is in accordance with the *unique name assumption* (UNA), often made in knowledge representation, which states that there is a unique way of naming a specific entity. In the simple graph formalism, this implies that two concept nodes with distinct individual markers refer to distinct entities, thus a concept node cannot be labeled by several individual markers. Translated in the logical model theory, it means that we do not consider interpretations of formulas that map two constants onto the same element of the interpretation domain.

Assuming UNA or not does not change any result about the correspondences between conceptual graphs and FOL presented in this paper, except for one in section 5 (theorem 19, establishing the descriptive equivalence between full CGs and FOL with equality [BMT99]; in order to be able to translate a logical atom $a = b$ in CGs, where a and b are constants, one has to relax UNA). •

Individual markers and labels Two concept nodes with the same individual marker represent the same entity. Then should these nodes have the same type? Previous definitions of a support and simple graphs do not impose such a constraint. However, problems arise when one wants to compare two nodes $[t_1 : B]$ and $[t_2 : B]$ referring to a same entity B, where t_1 and t_2 are not comparable. According to the label order, these labels are not comparable. In order to achieve logical completeness, one also has to merge nodes representing the same entity (see normal form, definition 9). Then, what should be the type of the obtained node? One radical way of solving this problem is to type individual markers in the support : a mapping τ associates a type of T_C to each marker of I. Then, if one requires that an individual marker m always appears with its given type $\tau(m)$ in a concept node label (for any concept node c, if marker(c) is an individual marker m, then type(c) $= \tau(m)$), comparison and merging of individual nodes can be safely done. We adopted this solution in most of our papers about theoretical aspects of conceptual graphs, but it could appear to be too rigid in applications. A certain freedom can be gained by relaxing the constraint type(c) $= \tau(m)$ into type(c) $\geq \tau(m)$. For instance, even if the marker Mary refers to an entity of type "woman-researcher-in-computer-science", it is allowed to say "woman Mary" or "researcher Mary". Then, two types are associated with a concept node c : its *surface* type, type(c), and its *deep* type $\tau(m)$, this latter being used in operations.

Another approach consists in using conjunctive types. In this case, merging two concept nodes with same individual marker is always possible, and the type of the resulting node is the greatest lower bound of the original types. Furthermore, one keeps an equivalent formula.

In what follows, we will assume that two concept nodes with the same individual marker can always be merged, giving a unique result. We will call the **individual typing assumption** the solution that types each individual marker, and the **conjunctive type assumption** the solution based on the use of conjunctive types.

2.3.2 Elementary rules

The set of rules proposed below is logically sound and complete. This set has been originally defined in [CM95], adapting the one given by Sowa in [Sow84]; here, some operation names have been changed in order to avoid confusion with the operations now proposed in the draft ANSI standard for conceptual graphs [NCI98] (see also the note below about this proposition).

There are four specialization operations and four inverse operations, called generalization operations. For all operations, the resulting graph is disjoint from the source graph(s). It goes without saying that in case that the support constraints graphs more than in our definition (e.g. individual markers are typed, or signatures are associated to relation types), operations have to be made more restrictive in order to produce only well-formed graphs with respect to the support.

Definition 5 (Specialization operations) *Let G be a simple graph; a simple graph H can be obtained from G by one of the following operations:*

- **Relation simplify**: *delete a duplicate relation node of G (i.e. a relation node r such that there is another relation node with the same type and exactly the same* i-*th neighbors, for all i).*
- **Restrict**: *decrease the label of a relation node (its type) or the label of a concept node (its type and/or its marker) in G.*
- **Join**: *merge two concept nodes of G with the same label.*

- **Disjoint sum**: *Let K be a simple graph disjoint from G; H is obtained by juxtaposing G and K.*

Definition 6 (Generalization operations) *Let G be a simple graph; a simple graph H can be obtained from G by one of the following operations:*

- **Relation duplicate**: *duplicate a relation node of G (i.e. add a relation node with the same type and exactly the same* i*-th neighbors, for all i, as an existing relation node).*
- **Unrestrict**: *increase the label of a relation or concept of G.*
- **Detach**: *split a concept node c into nodes c_1 and c_2, with the same label as c, such that the sets of edges incident to c_1 and c_2 are disjoint and their union is the edge set of c —one of the two sets may be empty.*
- **Substract**: *delete some (possibly all) connected components of G.*

H is a specialization of G ($H \leq G$) if H can be derived from G by specialization operations. Since disjoint sum is a binary operation, the derivation of H from G may have other sources than H. G is a generalization of H ($G \geq H$) if G can be derived from H by a sequence of generalization operations. Note that if $G \geq H$, then G can be obtained from H by considering H only (in terms of [MC93], the derivation from G to H is a path, and not a graph with several sources). It is straightforward to prove that $G \leq H$ *iff* $G \geq H$.

Note that the operation *relation simplify/duplicate* actually produces a graph equivalent to the original graph (where "equivalent" can be understood in the sense of the $\leq$ relation or in a logical sense).

Let us also point out that the specialization rules allow one to define the simple graph model as a **formal system**. A formal system is generally inductively defined by a basis, i.e. a set of primitive objects, and a set of rules generating new objects from existing ones. Let us take the following basis: for each relation type t_r of arity k in T_R, build a SG composed of one relation node of type t_r, and its k neighbors with label $(\top, *)$ (or in presence of relation signatures, take the types given the signature of t_r); let $\mathcal{B}$ be the set of these graphs; let $[\top : *]$ be the graph restricted to one generic concept node of universal type; then $\mathcal{B} \cup \{[\top : *]\}$ is our basis.

Property 3 ([MC93] or [Mug95] on connected graphs) *Given a support S, any simple graph on $\mathcal{S}$ can be generated using the specialization rules from the set $\mathcal{B} \cup \{[\top : *]\}$.*

See [MC93] for a study of specialization derivations.

Note. Relationships with Sowa's proposals. In the original presentation of conceptual graphs [Sow84], simple graphs are *connected* graphs. Specialization operations (called canonical formation rules) are defined. These rules are logically sound and complete (completeness being proved in [CM92]) in the sense that $G \leq H$ iff $\Phi(H)$ can be deduced from $\Phi(G)$ (see section 2.4.1 for a precise definition). In the proposed standard for conceptual graphs [NCI98], simple graphs are no more necessarily connected graphs (and may include co-reference links, but we will take this aspect into account later). A new set of canonical formation rules is proposed: two equivalence rules (copy and its inverse, simplify), two specialization rules (restrict and join) and two generalization rules (unrestrict and detach), which are the inverse of specialization rules. These rules are said to be the foundation for all logical operations on CGs.

The set of equivalence and specialization rules (or the set of equivalence and generalization rules) is indeed logically *sound*, i.e. if H is obtained from G using these rules (say $H \leq G$) then $\Phi(H)$ implies $\Phi(G)$. But it is *not complete*, in the sense that it does not allow to compute all logical deductions [Mug00]. Another flaw of these sets of rules is that they are not minimal: some rules are not elementary operations and can be simulated with other rules [Mug00]. •

2.3.3 Projection

Projection [Sow84] is the fundamental operation on simple graphs since it allows the *effective* computation of the $\leq$ relation.

Definition 7 (Projection) *Let Q and G be two SGs defined on a support $\mathcal{S}$. A* projection *from Q to G is a mapping π from $V_C(Q)$ to $V_C(G)$ and from $V_R(Q)$ to $V_R(G)$ which preserves edges (it is a bipartite graph homomorphism) as well as their numbering, and may specialize concept and relation node labels:*

1. $\forall (r,i,c) \in E(Q), (\pi(r), i, \pi(c)) \in E(G)$;
2. $\forall x \in V(Q), l(\pi(x)) \leq l(x)$

In previous definition, Q is called the *source* graph and G is the *target* graph. Note that projection is generally not a one-to-one mapping (in other words, $\Pi(Q)$, the subgraph of G induced by the images of the nodes of Q, is not isomorphic to Q).

In Fig. 7, suppose *Researcher* $\leq$ *Person*, then there is one projection from Q into G. The image of Q by this projection is the subgraph G' of G. Typically, Q represents a query, G a fact, and projections from Q to G define answers to Q.

Theorem 1 (Equivalence Projection and Specialization) *Let G and H be two simple graphs; $H \leq G$* iff *there is a projection from G to H.*

Proof. The proofs of [Sow84] (*if* part) and [CM92] (*only if* part)— which apply to connected simple graphs — are immediately adapted.

We are now able to define the *deduction problem* on simple graphs. Notice that in the following definition the set $\mathcal{G}$ can also be seen as one simple graph, made up of the disjoint union of all its elements.

Definition 8 (Deduction problem on simple graphs) *The Deduction problem takes as input a set of simple graphs $\mathcal{G}$ and a simple graph Q and asks whether $Q \geq \mathcal{G}$.*

The following section will justify the use of the word "deduction". We will also call this problem Projection, when we want to emphasize that the $\geq$ relation is computed by checking the existence of a projection.

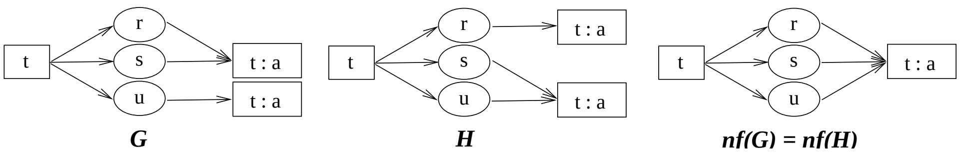
2.4 Equivalence with the FOL($\wedge$, $\exists$) fragment

2.4.1 Soundness and completeness of graph operations

Previous graph operations defining the specialization relation on simple graphs are sound and complete w.r.t. the semantics Φ, up to a normality condition for completeness.

Definition 9 (Normal form) *A SG is in* normal form *if it does not have two nodes with the same individual marker. The normal form of a graph (notation $nf(G)$ for a SG G) is obtained by merging all individual nodes with the same marker.*

Under individual typing assumption or conjunctive type assumption it always exists, is unique and computable in linear time with a naive algorithm. With the first assumption, the merging of two nodes is simply a join. With the second assumption, nodes may have different types, thus may need to be restricted first (their type being replaced by the greatest lower bound of the two types). Logical semantics of a graph and its normal form are obviously equivalent.



$\Phi(G) = \Phi(H) = \exists x t(x) \wedge t(a) \wedge t(a) \wedge r(x,\ a) \wedge s(x,\ a) \wedge u(x,\ a)$

G and H have same logical translation (thus same normal form) but they are not comparable by projection.

Figure 8: The need for normal forms

Figure 8 shows a counter-example to projection completeness when the target graph is not normal: G for instance does not project to H, even if both SGs have the same logical semantics, but it projects to $nf(H)$.

Following results have been first proved for connected simple graphs but proofs can be easily adapted to our simple graphs:

Theorem 2 (Soundness) *[Sow84] Given two simple graphs G and H defined on $\mathcal{S}$, if $G \geq H$ then $\Phi(\mathcal{S}), \Phi(H) \vDash \Phi(G)$.*

Theorem 3 (Completeness) *[CM92] [CM95] Given two simple graphs G and H defined on $\mathcal{S}$, if $\Phi(\mathcal{S}), \Phi(H) \vDash \Phi(G)$ then $G \geq nf(H)$.*

Bibliographic note. The first proof of completeness is in my thesis [Mug92] (also published in [CM92]) considering connected simple graphs. It is based on the resolution method (more precisely it is shown that if the empty clause can be obtained by the resolution method from the clausal form of $\{\Phi(\mathcal{S}),\ \Phi(H),\ \neg\Phi(G)\}$ then one can build a kind of substitution from $\Phi(G)$ to $\Phi(H)$, which corresponds to a projection from G to H). In the original proof, it was not clear that the graphs considered had to be in normal form. Preller pointed out this flaw with a counter-example to completeness (private communication, June 1994) which led us to define the notion of a normal form ([CM95] or [MC96] for the counter-example). This notion was independently defined by Ghosh and Wuwongse, who provided another proof of completeness [GW95].

Since then, several other proofs have been provided. Simonet reformulated the first proof in the context of the Ψ semantics [Sim98] or [CMS98] (see section 2.6.1, "alternative logical semantics").

In his thesis, Kerdiles gave a set-theoretic interpretation of a support and simple graphs by directly embedding them into classical relational structures, the notion of truth of a SG w.r.t. a structure corresponding to the existence of a model of $\Phi(G)$ using that structure as a domain, and proved completeness by showing that if for every structure in which H is true, G is also true, then a projection from G to H can be built.

Independently Baget [Bag01] also built his proof on a correspondence between the notion of a model and that of a projection. He defined a simple graph (called the interpretation graph) which encodes a domain and an interpretation of the types (predicates) of a support relative to this domain. Then it was shown that, for any simple graph G, a model of G corresponds to a projection from G to the interpretation graph associated to this model. Finally, it was shown that if $\Phi(\mathcal{S}), \Phi(G) \to \Phi(H)$ then, considering a certain model of $\Phi(G)$, the projections from G and H to the interpretation graph define a projection from H to G. •

2.4.2 Equivalence with FOL($\exists$, $\wedge$)

We claimed in the introduction that SGs are equivalent to the positive, conjunctive and existential fragment of FOL without functions (let us denote it by FOL($\wedge$, $\exists$)). Let us prove this claim. A SG language is composed of all simple graphs definable on a given support (the vocabulary). A FOL($\wedge$, $\exists$) language is composed of all positive conjunctive existentially closed formulas definable using a vocabulary composed of a set of predicates and a recursively enumerable set of constants (variables are external to the vocabulary).

Embedding a FOL($\wedge$, $\exists$) language into a SG language is immediate. For the other direction, the apparent hindrance is that formulas assigned to the support by Φ are universally quantified and are used in the deduction process. However, we can do without them, replacing the support by a flat one (where in particular partial orders on types are the identity), thus getting rid of universally quantified formulas.

Let us first define a flat support. A *flat support* $\mathcal{S} = (T_C, T_R, I)$ is such that T_c is restricted to one element $\top$ and T_R is provided with the identity partial order (distinct types are non comparable).

Let us define $f2G$ a transformation from FOL($\wedge$, $\exists$) to SGs. To a logical vocabulary is naturally associated a flat support (and reciprocally): there is one relation type for each predicate and one individual marker for each constant. To any formula f is assigned a SG G whose concept node set is in bijection with the set of terms occurring in f (one node $[\top : *]$ for each variable and one node $[\top : a]$ for each constant a) and relation node set is in bijection with the set of atoms of f (for each atom $p(e_1, ..., e_k)$ one node (p) whose i-ith neighbor is the node assign to e_i). Note that G is a normal SG. Also note that $f2G$ is injective, i.e. it maps different formulas (not identical up to variable renaming) to different SGs (non isomorphic). Moreover it is a bijection, if we restrict simple graphs to normal ones and supports to flat supports.

Property 4 *$f2G$ is a bijection from the set of FOL($\wedge$, $\exists$) formulas over a vocabulary V to the set of normal simple graphs on the flat support $\mathcal{S}$ naturally associated to V.*

Notice that, if we restrict ourselves to normal SGs on a flat support, the only difference between $f2G^{-1}$ and Φ is that $f2G^{-1}$ does not associate atoms to concept nodes. If we associate to a flat support $\mathcal{S}$ the logical interpretation $\Phi(\mathcal{S}) = \{\forall x\ \top(x)\}$ instead of the empty set, then for any graph G on this support, one has $\Phi(\mathcal{S}),\ f2G^{-1}(G) \vDash \Phi(G)$ (and of course $\Phi(G) \vDash f2G^{-1}(G)$). In [Wer95a] the universal type is processed in a particular way by Φ: if a concept node c is of type $\top$, the associated atom is not $\top(id_c)$ but $(id_c = id_c)$. Then, for any graph G, $f2G(G)$ is equivalent to $\Phi(G)$ (actually the formulas are equal up to the suppression of all atoms $(id_c = id_c)$ which are true for all interpretations). This way of processing the universal type enables to identify FOL($\wedge$, $\exists$) formulas and SGs on a flat support.

Let us now come back to general supports and define a transformation $G2f$ from a SG language to a FOL($\wedge$, $\exists$) language. $G2f$ is a variant of Φ, which maps a SG to an existential formula integrating the needed knowledge about the support. Let us call $\mathcal{S}_{flat}$ the flat support obtained from $\mathcal{S}$ in the following way: the new relation type set is the union of T_R and T_C, concept types becoming unary relation types; the new concept type set has only one new element $\top$; $\mathcal{I}$ is unchanged.

Given a SG G over $\mathcal{S}$, let us consider the SG $G_{\mathcal{S}}$ over $\mathcal{S}_{flat}$ obtained from G in the following way: for each concept node c of G of type t_c, the new type of c is $\top$, and for each concept type greater or equal to t_c (thus including t_c), one adds a relation node with type t_c and neighbor c; for each relation node r of type t_r in G, for each relation type $t_{r'}$ in $\mathcal{S}$ strictly greater than t_r, a relation node of type $t_{r'}$ with same neighbors as r is added. It is immediately checked that, given two SGs G and H over $\mathcal{S}$, there is a projection from G to H iff there is a projection from $G_{\mathcal{S}}$ to $H_{\mathcal{S}}$, both relative to $\mathcal{S}_{flat}$. Now let us define $G2f(G, \mathcal{S}) = \Phi(G_{\mathcal{S}})$. Finally, check that $G2f$ is an injective application from normal SGs to FOL($\wedge$, $\exists$) formulas.

Property 5 *$G2f$ is an injective application from the set of normal SGs defined on a support $\mathcal{S}$ to the set of FOL($\wedge$, $\exists$) formulas over a vocabulary V.*

Property 6 (Embedding FOL($\wedge$, $\exists$) into $\mathcal{SG}$) *Let g and h be two formulas of FOL($\wedge$, $\exists$) over a vocabulary V. The mapping $f2G$ associates to g and h (normal) SGs $G = f2G(g)$ and $H = f2G(h)$ over the flat support associated to V such that $h \vDash g$ iff there is a projection from G into H.*

Proof: Let g' = $\Phi(G)$ and h' = $\Phi(H)$. Check that $h \vDash g$ if and only if $h' \vDash g'$. We conclude using soundness and completeness of projection (theorems 2 and 3) and noticing that $\mathcal{S}$ is flat, thus $\Phi(\mathcal{S})$ is empty. □

Property 7 (Embedding $\mathcal{SG}$ into FOL($\wedge$, $\exists$)) *Let G and H be two normal SGs over a support $\mathcal{S}$. The mapping $G2f$ associates to G and H formulas of FOL($\wedge$, $\exists$) $g = G2f(G, \mathcal{S})$ and $h = G2f(H, \mathcal{S})$ over the vocabulary V naturally associated to a flat support obtained from $\mathcal{S}$, such that $h \vDash g$ iff there is a projection from G to H.*

Proof: One has $g = G2f(G, \mathcal{S}) = \Phi(G_{\mathcal{S}})$ and $h = G2f(H, \mathcal{S}) = \Phi(H_{\mathcal{S}})$. Using soundness and completeness of projection (theorems 2 and 3), $h \vDash g$ iff $G_{\mathcal{S}} \geq H_{\mathcal{S}}$, since $\Phi(\mathcal{S}_{flat}) = \emptyset$ and $G_{\mathcal{S}}$ and $H_{\mathcal{S}}$ are normal graphs. Check that $G_{\mathcal{S}} \geq H_{\mathcal{S}}$ iff $G \geq H$. □

2.5 Structure of the specialization relation

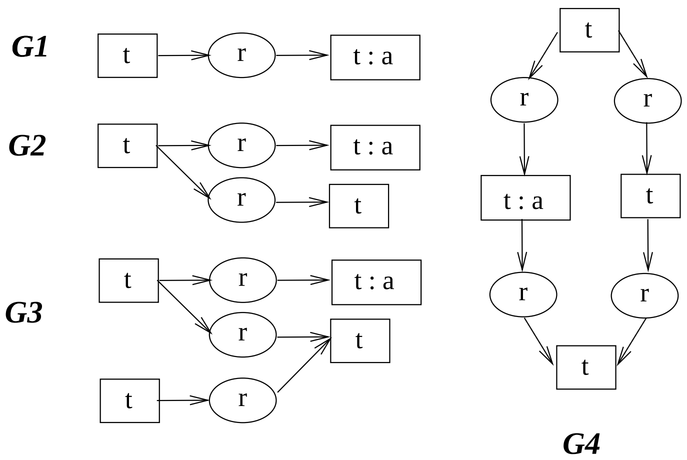


Figure 9: Equivalent graphs

Let us synthesize main results about the *structure* of the specialization relation:

1. $\leq$ is a quasi-order relation, and not an order: it is a reflexive and transitive but not anti-symmetrical relation (e.g. [Jac88]). E.g. Fig. 9 presents four non-isomorphic graphs which subsume each other.

2. Let us say that two simple graphs G and H are *equivalent* if they subsume each other; an *irredundant* graph is a graph that is not equivalent to one of its strict subgraphs. Then any equivalence class for $\leq$ contains a *unique* irredundant graph (up to isomorphism); moreover this graph is the *smallest* graph of the equivalence class. Cf. [CM92] (also [CG95]). E.g. in Fig. 9, G_1 is the irredundant form of the other (redundant) graphs.

3. The restriction of $\leq$ to irredundant graphs is a *lattice* [CM95]. Note that it is true even if the set of concept node labels is not a lattice. See Fig. 10 for instance: the lattice on the right represents all irredundant graphs constructible on the support given on the left (restricted to a concept type set, which is not a lattice).

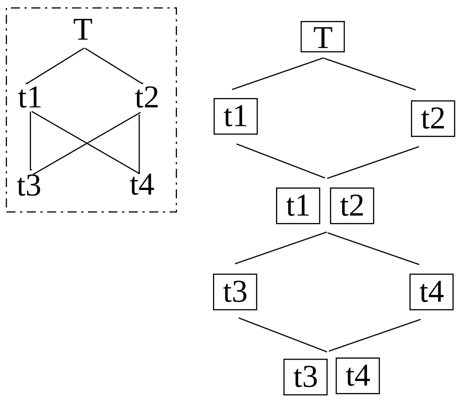


Figure 10: A lattice of irredundant graphs

2.6 Representational extensions

This section presents two extensions of simple graphs. First, the addition of co-reference links allowing to say that two distinct concept nodes represent the same entity. Second, nested graphs in which concept nodes may express complex information represented by nested graphs, thus allowing to structure knowledge hierarchically.

These extensions are not *strict* extensions of simple graphs, in the sense that they do not increase their expressiveness. One way of figuring it out is to see that their logical translation remains in the FOL($\wedge$, $\exists$) fragment. To emphasize the fact that they bring more comfort for representing knowledge but not more expressivity, we call them *representational* extensions.

2.6.1 Co-reference links

We have already seen that several concept nodes of a graph may refer to the same entity. It is the case when several individual concept nodes have the same individual marker (and thus the graph is not in normal form). To indicate that several *generic* nodes refer to the same entity, an additional structure called a *co-reference link* is used. In graph drawings, co-reference links are usually represented by dotted lines, or by a variable name following the generic marker.

More or less restrictive definitions of co-reference links can be found. The important properties to ensure are, firstly, that co-referent nodes can always be merged, secondly, that the Φ formula of the obtained graph is equivalent to that of the original one. For instance, it may be convenient to allow co-reference links between a generic node and an individual node. Then, under the *conjunctive type assumption*, the following condition has to be satisfied: a co-reference link should not, by transitivity, relate two individual nodes with different markers (unique name assumption).

For the sake of simplicity of definitions, we will assume that co-reference links relate *generic nodes* only. Furthermore, we will make the conjunctive type assumption.

Since both individual markers and co-reference links can be used to indicate that several nodes represent the same entity, we consider an equivalence relation, called *co-identity*, on the whole concept node set of a graph.

Definition 10 (Co-identity) *[CMS98] Two concept nodes are* co-identical *if they refer to the same entity. Co-identity, denoted by* co-ident*, is an equivalence relation over the set of concept nodes of a graph, fulfilling the following constraint on individual concept nodes: for all c_1 and c_2 in V_C, such that marker(c_1) $\in \mathcal{I}$ and marker(c_2) $\in \mathcal{I}$, co-ident(c_1,c_2) if and only if marker(c_1) = marker(c_2).*

The co-reference relation, denoted by *co-ref*, is the restriction of co-ident to generic concept nodes: for all distinct c_1 and c_2 in V_C, if co-ident(c_1,c_2) and c_1 and c_2 are not individual markers, then co-ref(c_1,c_2). In other words, co-ref is the part of the co-ident relation that needs to be stated explicitly. co-ref is also an equivalence relation.

Note that in drawings the co-reference relation is usually only *partially* represented by co-reference links (the dotted lines). co-ref is the reflexive and transitive closure of the relation defined by co-reference links. When it is totally represented in a graph drawing, the co-reference links form a "clique" (all possible links are drawn) on concept nodes belonging to the same co-reference class.

Previous simple graph framework can be naturally extended to integrate co-reference links. A SG with co-reference links (in short a SG^{ref}) is a graph $G = (V = (V_C,\ V_R),\ E,\ l, co\text{-}ref)$ where $G = (V,\ E,\ l)$ is a SG and $co\text{-}ref$ is the explicit part of a co-identity relation over V_C. A SG can be seen as a SG^{ref} where $co\text{-}ref$ is empty. The logical semantics Φ is extended in the following way. A term is associated to each co-identity class rather than to each concept node: if the class contains an individual marker, this term is the constant equal to this marker ; otherwise it is a new variable. In other words, the same term is assigned to two co-identical concept nodes.

For instance, the formula $\Phi(G)$ associated to the graph G of Fig. 11 is $\exists x\ t_1(x) \wedge t_2(x) \wedge t(a) \wedge t(a) \wedge r(x)$.

Now, projection has to respect co-identity, i.e. two co-referent nodes must have co-identical images (either the same image, or distinct images which are co-referent nodes or nodes with the same individual marker). A projection Π from a SG^{ref} $G = (V_G,\ E_G,\ l_G, co\text{-}ref_G)$ to a SG^{ref} $H = (V_H,\ E_H,\ l_H, co\text{-}ref_H)$ is a projection from the SG $(V_G,\ E_G,\ l_G)$ to the SG $(V_H,\ E_H,\ l_H)$ such that for all c_1, c_2 concept nodes of G, if $co\text{-}ref_G(c_1,\ c_2)$ then $co\text{-}ident_H(c_1,\ c_2)$.

The normal form of a simple graph with co-reference links is obtained by merging co-identical nodes (see Fig. 11: H is the normal form of G). In other words, a graph is in normal form if its co-identity relation is the identity. The logical formula associated with a graph is equivalent to that associated with its normal form.

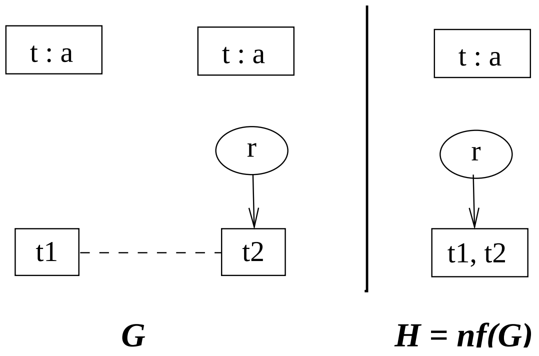


Figure 11: Co-reference links

To complete the extension of the simple graph model to co-reference links, it remains to revisit elementary rules (canonical formation rules). The set of specialization rules (Relation simplify, Restrict, Join, Disjoint Sum) is added with a new rule (Co-reference addition), and, since a part of the Join operation becomes redundant, Join is replaced with a rule merging co-identical nodes only (Co-identical join); from a logical viewpoint, it becomes an equivalence rule [Mug00].

- **Co-reference addition:** make the union the co-reference classes of two generic nodes (this may be visualized by the addition of one or several co-reference links)
- **Co-identical join:** merge two co-identical nodes (two generic co-referent nodes or two nodes with the same individual marker) with the same label.

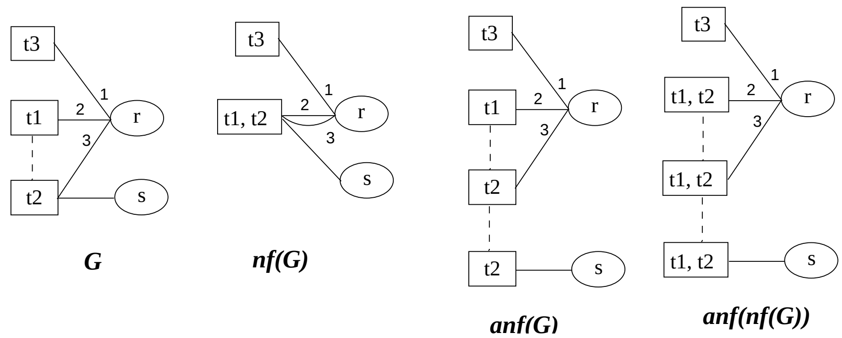
The set of generalization rules is modified in the same way. A new rule is added (Co-reference deletion) and Detach is replaced by an equivalence rule (Co-identical split), splitting a node into two co-identical nodes.

- **Co-reference deletion:** split a co-reference class into two co-reference classes (this may be visualized by the removal of one or several co-reference links)
- **Co-identical split:** split a node into two co-identical nodes with same label, i.e. split an individual node into two nodes with the same label, or split a generic node into two co-referent nodes with the same label The union of the sets of edges incident to the two nodes is the edge set of the original node —one of the two sets may be empty.

Property 3 is kept: all graphs with co-reference links can be generated with the specialization rules from the same basis. And the equivalence between the generalization relation and projection (theorem 1) is kept as well.

What are co-reference links useful for? A first aspect is that they can be exploited in *user interfaces*. Indeed one interest of simple graphs is their readability, quality that is lost when the graph becomes too big (e.g. too big to be entirely drawn on a screen) or too complex. Co-reference links enable the representation of a graph by several pieces, while keeping its global structure. A graph too big can be split into pieces displayable on one screen. The split can also be performed to decompose a graph into semantic modules, or to enhance a particular substructure. A user can build a graph piece by piece, then connect them with co-reference links. Graph decomposition is also a technique widely exploited in graph *algorithms* (see for instance the tree covering notion in section 2.7.3). Finally, if, in a simple graph, co-reference links can be easily suppressed while keeping an equivalent graph, it is not the case in nested graphs and in rules. And co-reference links are needed in full CGs to obtain equivalence with FOL.

Anti-normal form. In [GW95] the notion of the *anti-normal* form is introduced. In a graph in anti-normal form, each concept node is incident to at most one edge. More precisely, a graph is put into anti-normal form by splitting each concept node c with $k > 1$ incident edges, say $e_1, \dots, e_k$, into k co-identical concept nodes $c_1, \dots, c_k$, with same label as c, each edge $e_i = (r,\ j,\ c)$ becoming an edge $(r,\ j,\ c_i)$. In other words, when a graph is in anti-normal form, no co-identical split can be done anymore (except for "trivial" splits that would just create an isolated node, copy of the the original node). See Figure 12.



nf and anf stand for "normal form" and " anti-normal form" respectively.

Figure 12: Anti-normal form

Following properties are easily obtained:

- The anti-normal form and the normal form of a graph are unique (up to isomorphism).
- G projects to its normal form and the anti-normal form of G projects to G.
- All graphs with same anti-normal (or normal form) are logically equivalent by Φ.

The projection completeness theorem (theorem 3) can be rewritten by replacing the normal form condition on the target graph H with an anti-normal form condition on the source graph G:

Theorem 4 (Completeness with anti-normal form) *Given two SGs G and H defined on $\mathcal{S}$, $\Phi(\mathcal{S}), \Phi(H) \vDash \Phi(G)$ then $anf(G) \geq H$, where $anf(G)$ denotes the anti-normal form of G.*

Note about projection and logical completeness. Our approach gives greater importance to projection than to the elementary rules; the price to pay is that graphs have to be put in normal form (or anti-normal form) for ensuring logical completeness (w.r.t. Φ; see section 2.6.1 for an alternative semantic, called Ψ, for which the completeness results hold without restriction). Indeed, suppose one rather aims at providing a complete set of rules with respect to Φ, without any restriction on the form of the graphs. Such a result can be achieved with the rules we propose as described below. With the individual typing assumption, the following property holds.

Property 8 *[Mug00] Consider the set of specialization rules (co-reference addition, relation simplify, restrict, co-identical join, disjoint sum) added with co-identical split. Then, given two simple graphs (possibly with co-reference links) G and H, H can be derived from G if and only if $\Phi(\mathcal{S}), \Phi(H) \models \Phi(G)$.*

The basic idea of this property is that the co-identical split rule enables to derive H from $nf(H)$. A similar property is obtained with the set of generalization rules added with co-identical join.

To obtain the same results with the conjunctive type assumption, one has to replace the co-identical split with an operation that is the inverse of a merging: the type (say t) of the split node can be replaced by a greater one in the obtained nodes, provided that the greatest lower bound of both types is still t. If we consider generalization rules added with co-identical join, we also have to replace this latter operation by a merging operation, i.e. co-identical join can merge co-identical nodes which do not have the same label, the label of the obtained node being the greatest lower bound of the merged node labels.

To conclude this note, either one favors projection, and logical completeness of graphs operations (projection or an equivalent set of formation rules) is obtained with (minor) restrictions on the form of graphs (i.e. normal form); or one favors logical completeness of the set of elementary rules, and the correspondence with projection is lost. •

Alternative logical semantics Two alternative logical semantics have been proposed in [PMC98] and [CMS98] [Sim98]. [PMC98] defines a new logic, in which deduction is equivalent to projection. In [CMS98] [Sim98], a variant of Φ, called Ψ, is proposed, for which projection is sound and complete w.r.t. FOL deduction without any restriction. Both semantics rely on the same intuitive idea: if several nodes represent the same entity, they correspond to several different viewpoints on this entity; thus, the graph obtained by merging these nodes is not considered as equivalent to the original one.

I will just outline the main ideas of the Ψ semantics. This semantics translates the whole structure of the graph. The support is interpreted as for Φ, except that predicates associated with concept types become binary. Intuitively, $t(x,\ y)$ means "x is of type t and is represented by the node y". To translate a graph, first associate a term to each co-identity class: a constant a if the class contains a node with individual marker a, otherwise a new variable. Then, two terms are assigned to a concept node c: a new variable e_c that stands for the node, and the term i_c associated with its co-identity class. Assign the atom $t(i_c,\ e_c)$ to each concept node c of type t; assign the atom $t_r(e_{c_1},\ ...,\ e_{c_k})$ to each relation node r of type t_r with arity k, where c_i is the ith neighbor of r. $\Psi(G)$ is the existential closure of the conjunction of these atoms.

Let us for instance consider the graphs of Fig. 8. Let us recall these graphs are not equivalent for projection (G and H are not comparable, and have the same normal form, thus $G > nf(G)$ and $H > nf(G)$) but they have the same associated formula by Φ (up to the number of atoms $t(a)$):

$\Phi(G) = \Phi(H) = \Phi(nf(G)) = \exists x\ t(x) \wedge t(a) \wedge r(x,\ a) \wedge s(x,\ a) \wedge u(x,\ a)$.

Now, we shall see that it is not the case with Ψ anymore. Let x and a be the terms associated to the co-identity classes of the three graphs. In G and H, let y be the variable assigned to the generic node, z_1 and z_2 being assigned to the individual nodes.

$\Psi(G) = \exists x \exists y \exists z_1 \exists z_2 t(x,y) \wedge t(a,z_1) \wedge t(a,z_2) \wedge r(y,\ z_1) \wedge s(y,\ z_1) \wedge u(y,\ z_2)$.

$\Psi(H) = \exists x \exists y \exists z_1 \exists z_2 t(x,y) \wedge t(a,z_1) \wedge t(a,z_2) \wedge r(y,\ z_1) \wedge s(y,\ z_2) \wedge u(y,\ z_2)$.

These formulas are now incomparable and are both deducible from the following formula $\Psi(nf(G))$.

$\Psi(nf(G)) = \exists x \exists y \exists z\ t(x,y) \wedge t(a,z) \wedge r(y,\ z) \wedge s(y,\ z) \wedge u(y,\ z)$.

Theorem 5 (Soundness and completeness of Ψ) *[CMS98] Let Q and G be SGs defined on a support $\mathcal{S}$. Then $Q \geq G$ iff $\Psi(\mathcal{S}), \Psi(G) \vDash \Psi(Q)$.*

2.6.2 Nested Graphs

Context of our study. Nested conceptual graphs have been introduced by Sowa (see for instance [Sow94]), as a mean of structuring knowledge by levels. For instance, imagine the description of a scene (e.g. an anniversary party). At first level, a conceptual graph gives an overall description of the scene (people around a table on which is a cake; by zooming in on certain concept nodes (concept nodes referring to the cake or people) one sees more details of certain parts of the scene (candles and decorations on the cake), also described by conceptual graphs, and so on.

The main motivation for nested conceptual graphs is related to natural language processing. Figure 13 represents the sentence "Tom believes that Mary wants to marry a sailor" (example given on Sowa's web site).

Tom is the experiencer (expr) of the concept [Believe], which is linked by the theme relation (theme) to a proposition that Tom believes. The proposition is described by another graph, which says that Mary is the

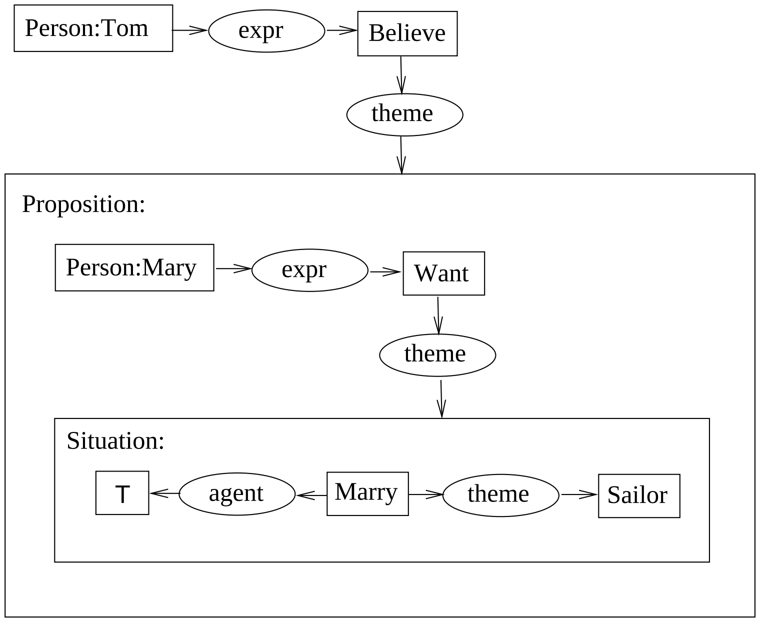


Figure 13: Nested graph

experiencer of [Want], which has as theme a situation that Mary hopes will come to pass. That situation is described by another graph, which says that Mary (represented by the concept [⊤]) marries a sailor. The co-reference link, shows that the concept [⊤] in the situation concept refers to the same individual as the concept [Person: Mary] in the proposition concept.

We experienced the need for nested graphs, in two applications we were involved in: one aimed at representing and simulating the required behavior of human agents in case of emergency situations (see [BBV97]), the other was devoted to document indexing and retrieval (see [Gen00])[11]. In both cases, knowledge can be naturally described by levels, and reasoning has to respect these levels.

In Sowa's work, nested conceptual graphs are defined at a *descriptive* level only, in the sense that no operations for reasoning with these graphs are defined. That is why in [CMS98] we defined a KR formalism based on nested graphs with co-reference links. We present below the main results of this paper.

[11] Cf. also the brief presentation of these applications in the introduction, section 1.3.

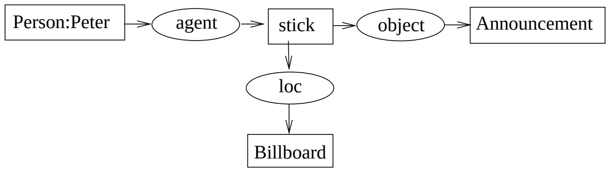


Figure 14: Simple graph

A logically founded nested graph formalism. Let us consider the simple graph of Figure 14, which says that "Peter is sticking an announcement on a billboard". Suppose one wants to add two pieces of information about the announcement. First, "Peter wrote this announcement", second "this announcement says that Peter is selling a bicycle". Let us assume that these pieces of information are considered to be of different levels, the first one being an *external* information about the announcement, and the second one an *internal* information. If one wants to keep this difference in the representation, then in the former the announcement node is a *black box* which is joined to the Peter node via a node representing a writing, and the latter information is put inside the announcement node: it can be obtained by zooming in on the announcement node, which in this case becomes a *glass box* (Figure 15).

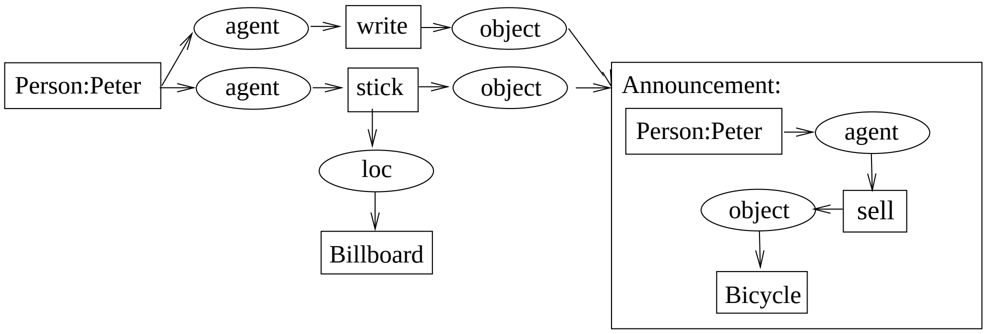


Figure 15: Adding external/internal information

It can also be said that the information nested in a node is pertinent within the context represented by this node. This information is called a *(partial) description* of the node. Thus, three kinds of information now label a concept node: a type, a marker and an internal description. Notice that, following [Sow94], most works in CGs consider only two elements: a type and a marker, which may be a nested graph. We prefer to give the possibility of having an individual node with a non empty description.

Let us first define nested graphs without co-reference links (as the one in Figure 14). **NGs** are defined by structural induction as follows.

1. A *basic NG* is obtained from a SG by adding to the label of each concept node c, a third field, denoted by $Desc(c)$, equal to $**$ ($**$ can be considered as the empty description, and a trivial bijection exists between basic NGs and SGs);

2. Let G be a basic NG, let $c_1,\ c_2,\ ...,\ c_k$ be concept nodes of G, and let $G_1,\ G_2,\ ...,\ G_k$ be NGs. The graph obtained by substituting G_i to the description $**$ of c_i for $i = 1,\ ...,\ k$ is a NG.

It is important to note that if a SG or a NG H is used several times in the construction of a NG G, we consider that several copies of H (and not several times the graph H itself) are used in the construction of G. Thus a NG can also be seen as a *rooted tree* of SGs, whose nodes are the SGs appearing in the construction of G and edges represent the nestings (see [CMS98] for a formal definition).

Let us denote it by $\mathbf{A(G)}$ this tree and by $R(G)$ its root (a simple graph). Let us also call a *complex node* a concept node of a NG with a non empty description.

We distinguish the set $V_c(G)$ of concept nodes of a NG G from the set $UV_c(G)$ of concept nodes *appearing in* G, i.e. the set of concept nodes of all SGs used in the construction of G. $UV_c(G)$ is the union of the set $V_c(G)$ and the sets $UV_{Desc(c_i)}(G)$ where c_i is a complex node of G (note that these sets are pairwise disjoint).

A projection from a NG G to a NG H is defined by induction on the depth of G (which is that of the associated tree, i.e. the maximal length of a path in $A(G)$)[12].

A **projection** from a NG G to a NG H is a family of mappings $\Pi = \Pi_0, \Pi_{c_1}\ ...\ \Pi_{c_k}$, where $c_1\ ...\ c_k$ are the complex nodes of G, which satisfies:

1. Π_0 is a projection from the SG $R(G)$ to the SG $R(H)$,

2. for all c_i complex node of G, $\Pi_0(c_i)$ is a complex node of H and Π_{c_i} is a projection from the NG description of c_i to the NG description of $\Pi_0(c_i)$.

[12] The depth of a NG is 0 if it is a basic NG, otherwise it is 1+max(depth(c_i)), for c_i complex node of G.

A projection Π from a NG G to a NG H naturally induces a mapping (also noted Π for simplicity) from $UV_c(G)$ to $UV_c(H)$ defined by induction on the depth of G. For any c in $UV_c(G)$, if c is in $V_c(G)$ then $\Pi(c)$ is equal to $\Pi_0(c)$ and otherwise $\Pi(c)$ is equal to $\Pi_{c_i}(c)$, where c_i is the complex node of G such that c is in $UV_{Desc(c_i)}(G)$.

As in SGs, several individual concept nodes with the same marker may appear in a NG, and, more generally, it may be useful to express that several concept nodes represent the same entity. Note that such nodes may be merged if they appear in the same SG used in the construction but not otherwise. Thus, the extension from SGs to NGs makes co-reference links necessary. A NG with co-reference links (in short a NG^{ref}) is provided with an equivalence relation *co-ident* on the set of concept nodes appearing in G (i.e. $UV_c(G)$). co-ident and co-ref are defined similarly as for SGs, except that the set of concept nodes considered is not V_c but UV_c. A **projection** Π from a NG^{ref} G to a NG^{ref} H must preserve co-identity: $\forall\ c, c' \in UV_c(G)$ if $co\text{-}ref_G(c, c')$ then $co\text{-}ident_H(\Pi(c), \Pi(c'))$.

The semantics Φ on simple graphs is extended to nested graphs. I will just give here the main ideas guiding the translation. Roughly said, one argument is added to each predicate translating a type. Thus concept type predicates become binary and predicates associated with k-ary relation types become (k+1)-ary. This new argument is called the context argument. For instance, in figure 15, if x (resp. y, z) is the variable assigned to the generic concept node of type *Announcement* (resp. *Sell*, *Bicycle*), then the interpretation of the atom $object(x, y, z)$ is "the bicycle z is the object of the sale y *in the context of* the announcement x".

A special constant a_0 is associated with the support. If c is a concept node of the basic simple graph root of the tree, then its associated atom is $t(a_0,\ e_c)$, which means that " e_c is of type t in the context of a_0". If c is a node of a simple graph description of a concept node c', then its associated atom is $t(e_{c'},\ e_c)$, where $e_{c'}$ is the term associated with c', which means that " e_c is of type t in the context of $e_{c'}$". Similarly, the first argument of the atom associated with a relation node is the context argument, which is the term associated with the surrounding concept node or the support.

E.g. the formula associated with the graph G of Figure 15 is
$\Phi(G) = \exists y_1...y_4 y_5 y_6 (Person(a_0, Peter) \wedge Stick(a_0, y_1) \wedge Announcement(a_0, y_2) \wedge Billboard(a_0, y_3) \wedge Write(a_0, y_4) \wedge agent(a_0, Peter, y_1) \wedge object(a_0, y_1, y_2) \wedge location(a_0, y_1, y_3) \wedge agent(a_0, Peter, y_4) \wedge object(a_0, y_4, y_2) \wedge \Phi'(y_2, Desc(c_2)))$,
where c_2 is the concept node of type *Announcement* and
$\Phi'(y_2, Desc(c_2)) = Person(y_2, Peter) \wedge Sell(y_2, y_5) \wedge Bicycle(y_2, y_6) \wedge agent(y_2, Peter, y_5) \wedge object(y_2, y_5, y_6)$.

The normality notion defined on simple graphs can be extended to nested graphs. The trouble is that not all graphs possess a normal form since in some cases an infinite graph would have to be created, which is not allowed by our definitions. Thus, the normality notion is weakened into that of k-normality, which limits the produced graph to a nested graph of depth k (we will not give the precise definition of k-normality here). Every nested graph can be put into k-normal form, provided that k is greater than its depth. The following soundness and completeness theorem is obtained:

Theorem 6 (Soundness and completeness for nested graphs) *[CMS98] Let G and H be two NG^{ref} and let k be an integer such that $k \geq depth(G)$. If $G \geq H$ then $\Phi(\mathcal{S}), \Phi(H) \models \Phi(G)$. If H is in k-normal form and $\Phi(\mathcal{S}), \Phi(H) \models \Phi(G)$ then $G \geq H$.*

The semantics Ψ is extended by adding a context argument in the same way as Φ. The context argument is the variable assigned to the surrounding node, and not the term assigned to its co-identity class. Then:

Theorem 7 (Soundness and completeness of Ψ for nested graphs) *[CMS98] Let G and H be two nested graphs.
$G \geq H$ iff $\Psi(\mathcal{S}), \Psi(H) \models \Psi(G)$.*

The complexity of NG^{ref} projection checking remains NP-complete : it belongs to NP since a set of mappings from each SG composing the nested graph G to a SG of H is a polynomial certificate, and it admits SG projection as a special case. Moreover, in the absence of co-reference links (thus for NGs and not NG^{ref}s), given an algorithm A for projection checking (or computing) on a class C of simple graphs, one can build an algorithm for projection checking (or computing) on the corresponding class of NGs, whose complexity is polynomially related to the complexity of A (using results on trees of graphs, cf. [Mug95], also section 2.7.3).

Let us also point out a result that follows from results in [CMS98] (in particular a reduction from NG^{ref}s to SGs, which keeps projections, is given in a proof).

Property 9 *The NG^{ref} fragment is not more expressive than the SG fragment.*

Note. Related works and extensions. As already explained, several kinds of CGs with complex nodes have been considered [CM97]. Most of them are used for natural language processing (e.g. for representing causality [Naz93], or linguistic contexts [Dic94]). And most of them follow the formalism described for instance in [Sow94]. [GW96] defines nested conceptual graphs where concept labels have three fields like ours. But none of these nested graphs includes non-trivial graph-theoretic operations (such as projection) for reasoning purposes.

In [CM97] we introduced a general framework, *graphs of graphs* provided with a homomorphism, in which different sorts of nested conceptual graphs can be expressed. We proved that the different kinds of nested conceptual graphs used in the CG literature are actually instanciations of this generic notion. Notice however that this framework did no take co-reference links into account. It was only concerned with the recursive structure of nested graphs.

Let us relate the logical semantics presented here with other logical semantics proposed for nested graphs. In [Sow94] a specific relation type called *description* and a special concept type called *Graph* are introduced. Concept nodes of type *Graph* may have a graph as referent. Adding an internal information D to a concept node c of a CG G is done by creating another concept node c' of type *Graph* and referent D, with c' and c related via a *description* relation. E.g. in Fig 13, the node [Proposition: G] can be seen as [Proposition:*]→(description)→[Graph: G]. The mapping Φ for SGs is extended by introducing a special binary predicate *descr* — associated with the *description* relation — whose first argument is a term and second argument is a formula. For each concept node c, $\Phi(G)$ contains the atom $descr(e,\ \Phi(D))$, where e is the term associated with c. This logical semantics is similar to the logical semantics of contexts of [McC93], [Guh91], with *descr* playing the same role as the *ist* predicate. Another logical semantics for nested graphs, but without co-reference links, is given in [PMC98]. A specific logical language is proposed, where formulas do not belong to classical FOL, and a Gentzen system is defined, which is sound and complete for projection.

In [CM97] we type nestings, which allows to specify the relationship between the surrounding node and one of its descriptions. In the NG and NG^{ref} models presented here, a nesting does not have a semantics more precise than "the graph inside the concept node represents a partial internal description of the entity represented by the node". It appeared in applications that a mean of specifying this semantics was desirable. We thus proposed to add a new type set to the support, called the nesting type set. Now the description of a complex node is composed of one or several couples $(n_i,\ G_i)$, where n_i is a nesting type, and G_i is a NG. Note that, since nestings are typed, one may separate the description field into several NGs, according to different nesting semantics. The tree associated to a typed NG is now labeled on edges by nesting types.

Such nested graphs, with typed nestings and co-reference links, are the objects implemented in the platform CoGITaNT[13].

In [Bag99] a generalization of nested graphs, boxed graphs, is defined in which relation nodes can link concept nodes belonging to different "boxes". The main motivation for this extension was to reify co-reference links. The idea is to replace co-reference links by relation nodes of different types. Each type is provided with SG rules which describe the properties of the type, thus leading to different kinds of co-reference links. A transformation from boxed graphs to simple graphs is given, which, as a side effect, gives another proof of equivalence between simple and nested graphs (see also [Bag01]).

In [Ker01] internal descriptions are named. However, this framework does not provide operations on nested graphs. A formal declarative semantics is given by an embedding into a nested structure, with an associated notion of truth, from which a notion of deduction on nested graphs is defined. A transformation from these nested graphs to simple graphs is exhibited, which preserves deduction. •

Note. Further work. An issue that has not been well explored is the kind of reasoning that can be done with nested graphs. Projection can be seen as a basic operation, allowing to compare two nested graphs. In order to be used in practice, nested graphs have to provide more "flexible" reasoning. Consider for instance a document retrieval system, where each document is indexed by a nested graph. One cannot assume that the user who formulates a query is aware of the nesting structures.

An immediate extension consists in allowing answers of form "yes, in the context of ...". For instance, let us consider the graph of Figure 13. The answer to a question `[Sailor]` (does there exist a sailor?) is "no", but could be extended to "no, but yes in the context of Tom's belief of Mary's desire". This can be easily done by authorizing a projection from the source NG to any NG appearing in the target NG (in other words, the source tree is not necessarily projected from the root of the target tree). On the other hand, one cannot generalize this procedure by "skipping" levels carelessly (e.g. in previous example, answering "yes" to the question "Does Tom believe that Mary marries a sailor?").

[13] Cf. "http://cogitant.sourceforge.net/library/index.html"

The question of how knowledge of different levels can be combined is obviously related to the semantics of nestings. Rules (see section 3) can be a mean of specifying these semantics, in particular when nestings are typed (see previous note). The idea is that nested graphs can then be enriched by rule applications and projection can be kept as the basic mechanism. About this point see also the ideas about rules and boxed graphs in [Bag99]. •

2.7 Complexity of reasoning

2.7.1 Complexity of basic problems

Let us recall the fundamental problem on simple graphs, **Deduction/Projection**: Given two SGs, G and H, determine whether $G \geq H$ (i.e. there is a projection from G to H). This problem is decidable, what is not surprising given that simple graphs correspond to a very limited fragment of FOL. However, it is NP-complete. The first proof of this result should be the one we gave in [CM92] based on a straightforward reduction from **Directed Graph Homomorphism** (Given two directed graphs G and H, is there a homomorphism from G to H?) or, very simple too, from **Clique** (Given a non directed graph G - without loops - and an integer k, does it contain a k-clique, i.e. a subgraph with k vertices and all edges between distinct vertices?). It appeared later that bringing together projection and graph homomorphism was a very fruitful approach, as we will show it throughout this section. For the moment, let us just see that Directed Graph Homomorphism is a subproblem of Projection. Let us recall that a *graph homomorphism* Π from a graph $G = (X_G,\ E_G)$ to a graph $H = (X_H,\ E_H)$, where X and E respectively denote the node set and the edge set of a graph, is a mapping from X_G to X_H which preserves node adjacency, i.e. if xy is an edge of E_G, then $\Pi(x)\Pi(y)$ is an edge of E_H. Let us consider a flat support with one concept type $\top$ and one binary relation type r.

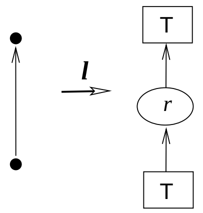


Figure 16: From a directed graph to a SG

A directed graph G can be seen as a simple graph $l(G)$ on this support

(Fig. 16): its concept nodes are the vertices of G and its relation nodes are the edges of G. More precisely, for each vertex x of G there is one concept node $l(x) = [\top : *]$; for each edge $(x,\ y)$ in G, there is a relation node labeled by r with first neighbor $l(x)$ and second neighbor $l(y)$. Check that each homomorphism from G to H defines a projection from $l(G)$ to $l(H)$ and reciprocally. To build a reduction from Clique to Projection, just see that an undirected graph can be seen as a directed graph replacing each undirected edge with two symmetrical directed edges, and that a graph possesses a k-clique if and only if there is a homomorphism from a k-clique to it.

Before we come back to the key notion of graph homomorphism, let us formulate the other basic problem for simple graphs, **Irredundancy**: given a SG G, is it irredundant? We proved in [CM92] that it was co-NP-complete, in other words its co-problem (given a SG G, is it redundant, i.e. does there exist a projection from G to one of its strict subgraphs?) is NP-complete. This complexity result shows that the fact that the specialization relation is a preorder and not an order is not a detail as claimed by some CG researchers by the past. In [Mug95] we give a simple algorithm for computing the irredundant form of a graph [14]. Its interest is to highlight the fact that it can be performed with a number of calls to a Projection oracle polynomial in the size of G. As a consequence, whenever we have a polynomial projection algorithm for a class of SGs, we also have a polynomial algorithm for computing the irredundant form of graphs from this class (provided that the property which makes projection polynomial is inherited by the subgraphs of the graph).

2.7.2 Projection as a graph homomorphism

A *labeled graph homomorphism* is a graph homomorphism which in addition preserves labels (on nodes, and edges if the graph is labeled on both). Projection is essentially a labeled graph homomorphism. It is a generalization of a labeled graph homomorphism in the sense that a partial order on node labels is added and the following additional condition has to be satisfied: for any node x, the label of x is greater or equal to the label of its image. When the partial order is trivial (all labels are incomparable), one comes back to a classical labeled graph homomorphism. Now, section 2.4.2 has shown that SGs defined on a support $\mathcal{S}$ can be translated into SGs defined

[14] Roughly, and excluding the specific case of isolated concept nodes, the idea is to repetitively remove a relation node from G, and if there is a projection from G to the subgraph obtained, take its image by this projection as the new graph G, otherwise try to remove another relation node, until an irredundant subgraph is obtained.

on a flat support $\mathcal{S}_{flat}$ such that projections are preserved. Thus, projection actually *is* a labeled graph homomorphism[15]. Structural properties and complexity of graph homomorphism has been widely studied in the graph theory community (see for instance recent syntheses in [HT97][HM01]).

The complexity of homomorphism checking has been particularly studied in the framework of the following problem, called general coloring problem, or **H-coloring**, where H is a fixed graph: Given a graph G, is there a homomorphism from G to H? Classical coloring problems can be seen as special cases of H-coloring. E.g. k-coloring (Given a graph G, is there a coloring of G with k colors, i.e. a mapping from the node set of G into $\{1 \dots k\}$ such that no adjacent nodes have the same color?) is a H-coloring problem where H is a k-clique. In turn, since H does not belong to the input of the problem, H-coloring can be seen as a special case of homomorphism where the size of H is bounded by a constant. The complexity study of H-coloring has focused on the structure of H, more precisely has tried to determine the exact conditions on the structure of H which make the problem polynomial. About non-directed graphs the answer is known: H-coloring is polynomial if H is bipartite (or if H has a loop or is restricted to one vertex) and NP-complete in all other cases [HN90]. About directed graphs, there are acyclic graphs H for which H-coloring is NP-complete. Therefore, Homomorphism, thus Projection, remains NP-complete even if H is a tree [HNZ94].

Corollary 1 (from [HNZ94]) *Projection remains NP-complete in the case where H is a tree.*

What about considering the structure of the source graph G rather than that of the target graph H? The question is more relevant for us, since G generally represents a goal or a query (or a part of a rule or constraint, see later), while H often encodes the knowledge base; G has then a small size and a less complex structure than H. The sequel of this part is devoted to this issue.

2.7.3 Tree-like SGs: first results

For simplicity reasons, we will often give complexity results for *connected* SGs. These results can be immediately extended to non connected SGs. Indeed, if G is a non connected SG, a projection from G to any SG H can be seen as a set of projections, from each connected component of G to H.

[15] Moreover, a labeled graph can be polynomially transformed into a non labeled graph, while preserving homomorphisms (P. Hell, private communication). Thus, if needed we could even get rid of labels.

Acyclic graphs (or, equivalently, trees when graphs are connected) often provide easier problems. And in the conceptual graph context, they are widely present. Let us say that a SG is a tree if it has no cycles except those of length 2, coming from multi-edges between a relation node and a concept node. It turns out that Projection is polynomial when G is a tree.

We gave in [MC92] (see also [Mug95] for a more detailed version) a polynomial algorithm for computing a projection from G to H when G is a tree. This algorithm also counts the number of projections from G to H. And for each node of G it memorizes possible images by projections from G to H. It can thus be used as a preprocessing step in an algorithm computing projections from G to H when G is a general SG (cf. [Mug95] for the principle, and [Sal93] for an implementation).

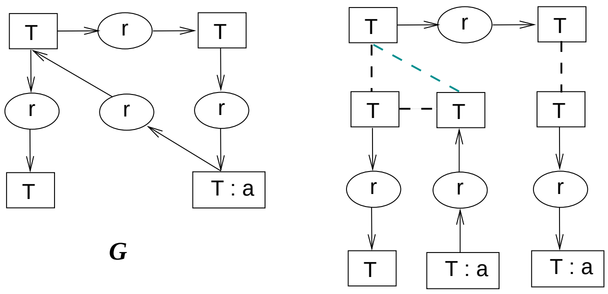
Property 10 *[MC92] Projection is polynomial when the source graph is a tree, as well as the associated counting problem.*

In [BMT99], the authors noticed that previous result can be extended to SGs that can be transformed into logically equivalent acyclic SGs by "cutting cycles". More precisely, if we split an individual concept node (as in the elementary operation Detach), we obtain a graph logically equivalent to the original one (but more general w.r.t. $\leq$). This would not be true with a generic concept node: the new graph would be logically more general than the original one. A SG G can be transformed into a logically equivalent acyclic SG by splitting some individual concept nodes if and only if every cycle of length greater than two contains an individual concept node. This property can be checked in polynomial time. Consequently, Projection can be solved in polynomial time when G is such a graph.

Note the importance the notion of normality may have for complexity. To take a very simple example, restrict Projection to instances G and H whose all concept nodes are individual nodes. The problem remains NP-complete (classical graphs can be encoded into such graphs, taking the same label for all concept nodes). But if H is in normal form, it becomes trivially polynomial (since there is only one possible image for each concept node of G).

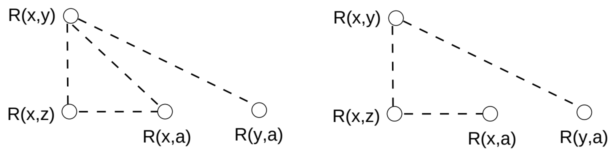
In [Ker01] Kerdiles defines the notion of a *tree covering* of the *crazed form* of a SG. The crazed form notion is similar to that of the anti-normal form. A SG is in crazed form if each concept node is adjacent to at most one relation node (and not "incident to at most one edge"). The crazed form of a SG is unique.

In the following it will be convenient to consider co-references links as edges. More precisely, let us replace in a crazed form the co-reference relation by edges, called co-reference edges (there is one co-reference edge between every pair of co-referent nodes). Let us call *component* of the crazed form of G one of its maximal connected subgraphs without co-reference edges (i.e. a subgraph composed of one relation node and its neighbors, or a concept node which is not adjacent to any relation in G). A *covering* of G is obtained from the crazed form of G by removing some co-reference edges in such a way that the co-reference relation is kept (if two nodes are co-referent in G, they are linked by a path of co-reference edges in the covering). A *tree covering* of G is a covering such that there is at most one path in the covering (thus taking into account both kinds of edges) between two concept nodes of distinct components. It is convenient to consider an abstraction of a covering: the multi-graph whose nodes are the components of the covering and such that, for each co-reference edge in the covering between a node in component C_i and a node in component C_j ($i \neq j$), there is an edge between C_i and C_j. A covering is a tree covering if its abstraction is an acyclic graph (i.e. it has no multi-edges and no cycles). It is shown that this notion characterizes the graphs of [BMT99]: a SG is transformable into a logically equivalent acyclic graph by splitting individual nodes iff it has a tree-covering. Fig. 17 represents a SG G as well as its crazed-form (on the right) and one of its tree coverings (co-reference edges in gray belong to the crazed form but not to the tree covering). Fig. 18 shows the abstractions of G crazed form (on the left) and previous tree covering (on the right).



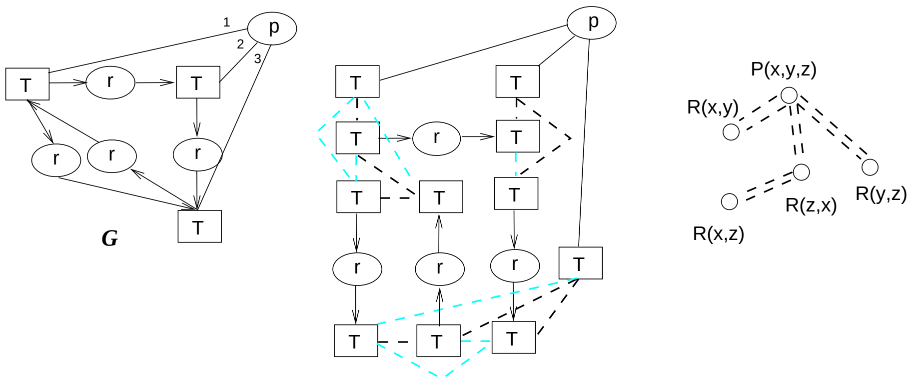
$$\Phi(G) = \exists x \exists y \exists z \top(x) \wedge \top(y) \wedge \top(z) \wedge \top(a) \wedge R(x,y) \wedge R(y,a) \wedge R(a,x) \wedge R(x,z)$$

Figure 17: Covering and tree covering



Abstractions of the crazed form and the tree covering in figure 17

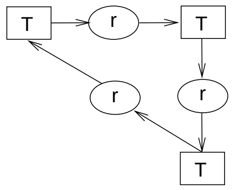
Figure 18: Covering abstractions



$$\Phi(G) = \exists x \exists y \exists z \top(x) \wedge \top(y) \wedge \top(z) \wedge R(x,y) \wedge R(y,z) \wedge R(z,x) \wedge R(x,z) \wedge P(x,y,z)$$

Figure 19: Guarded covering

Let us extend a bit more the class of tree-like SGs. Importing the notion of a guarded FOL formula [AvBN96], Kerdiles generalizes the notion of a tree covering to that of a *guarded covering*: the abstraction of a guarded covering is an acyclic multi-graph, i.e. cycles composed of multi-edges between two components are authorized. Fig. 19 gives an example of a guarded covering (from left to right: a SG G, its crazed-form and a guarded covering of it (ignore gray co-reference edges) and the abstraction of this guarded covering). Fig. 20 shows a SG that has no guarded covering.



$$\Phi(G) = \exists x \exists y \exists z \top(x) \wedge \top(y) \wedge \top(z) \wedge R(x,y) \wedge R(y,z) \wedge R(z,x)$$

Figure 20: Non guarded SG

A SG that has a guarded covering is called a *guarded SG*. A polynomial algorithm is provided that, given a guarded covering of a SG G and a SG H, determines whether there is a projection from G to H [16].

Property 11 *Projection is polynomial when the source graph G is a guarded SG and is given under the form of a guarded covering of it.*

Nothing is said about the complexity of determining whether a SG has a guarded covering, and if yes, computing such a covering. We will have to make a detour through works about hypergraphs to conclude that it is efficiently computable (see section 2.7.5, equivalence between a guarded covering and the joint tree of an acyclic hypergraph). For the time being we give only results immediately deducible from [Ker01].

As shown in [Ker01] guarded SGs exactly correspond to the guarded fragment of FOL($\exists$, $\wedge$). In this section, we will just give the definition of this fragment. More details can be found in the section 5.4.

Definition 11 (Guarded fragment of FOL($\exists$, $\wedge$)) *[Ker01] The guarded fragment of FOL($\exists$, $\wedge$), in short gFOL($\exists$, $\wedge$), is inductively defined as follows:*

1. *every atomic formula (i.e. every atom) belongs to gFOL($\exists$, $\wedge$)*
2. *gFOL($\exists$, $\wedge$) is closed under conjunction*
3. *if α is an atom, called a guard, and ϕ is a formula of gFOL($\exists$, $\wedge$), such that*

   (i) $free(\phi) \subseteq free(\alpha) = var(\alpha)$
   – where var (resp. $free$) denotes the set of variables (resp.free variables) of a formula –
   (ii) $X \subseteq free(\alpha)$

   then $\exists X(\alpha \wedge \phi)$ is in gFOL($\exists$, $\wedge$).

Any closed formula f of the guarded fragment of FOL($\exists$, $\wedge$) can be polynomially translated into a guarded SG, for instance by the mapping $f2G$ (see section 2.4.2). Reciprocally, any guarded SG together with a guarded covering can be polyniomally translated into a closed guarded formula: as such Φ does not produce a guarded formula, but a slight modification in

[16] This algorithm could be adapted to exhibit solutions.

its definition, arranging quantifiers and atoms in a certain order computed from the guarded covering, produces a trivially equivalent guarded formula.

Theorem 8 (Equivalence of guarded SGs and guarded FOL fragment) *[Ker01] Guarded SGs are equivalent to the (closed) guarded fragment of FOL($\exists$, $\wedge$).*

Different ways of measuring the tree-likeness of graphs, or their degree of acyclicity can be found in the literature. Before seeing what we can harvest from these studies, let us establish bridges between SG deduction and other fundamental and well-studied problems in computer science.

2.7.4 Equivalence with other well-known problems

Beside its interest in itself, identifying projection as a graph homomorphism allows to show its equivalence with important problems of computer science, whose complexity has been closely examined.

Conjunctive queries constitute a broad class of frequently used queries in databases because their expressive power is equivalent to that of the Select-Join-Project queries in relational databases [AHV95]. And **conjunctive query containment** is a fundamental problem in database query evaluation and optimization. A **constraint satisfaction problem** is a combinatorial problem, which has been widely investigated in artificial intelligence. What do these problems have in common, and what do they have in common with projection between SGs? Actually the three problems are essentially the same problem, even if they are formulated in very different ways.

The key notion explaining correspondences between these problems can be seen as that of a *labeled graph homomorphism.* Indeed the three problems can be formulated in terms of a labeled graph homomorphism problem. Such a result is not surprising in fact because the homomorphism notion is the fundamental notion between relational structures (which are the more general algebra structures).

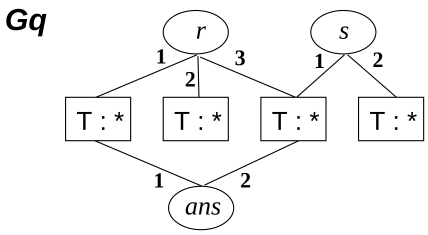
Let us state more precisely the links between a labeled graph homomorphism and a relational structure homomorphism. A vocabulary V is a finite nonempty set of relation symbols. A (finite relational) structure C over V consists of a finite universe U_c and for each k-ary relation $R \in V$ a set $R_c \subseteq U_c^k$ (the tuples of the relation). Given two structures C_1 and C_2 over the same vocabulary V, a homomorphism h from C_1 to C_2 is a mapping from the universe U_{c_1} of C_1 to the universe U_{c_2} of C_2 such that for all relation $R \in V$, if $(a_1, \ ..., \ a_k) \in R_{c_1}$ then$(h(a_1), \ ..., \ h(a_k)) \in R_{c_2}$. The fundamental algebraic problem is the following: given two finite relational structures A and B is there a homomorphism from A to B? This problem can be immediately recast as a labeled graph homomorphism problem since a finite relational structure can be formulated as a labeled graph of the same size. Actually, this labeled graph can be naturally seen as a SG. A vocabulary V is translated into a flat support $\mathcal{S}$, where the set of relation types is in bijection with the set of relations of V. A structure C over V is translated into a SG, with one generic concept node for each element of U_c and, for each $R \in V$ and each element $(a_1, \ldots, a_k)$ of R_c, one relation node of type R, with nodes corresponding to $a_1 \ldots a_k$ as neighbors. There is a homomorphism from a structure A to a structure B iff there is a projection from the SG associated to A to the SG associated to B.

Note. References. The translation from CQC to Projection was published in [CMS98]. Independently, in [KV98] it was shown that conjunctive query containment and CSP are essentially the same problem because they can be recast as a relational structure homomorphism problem.

Feder and Vardi stated that directed graph homomorphism and CSP are equivalent problems in [FV93]. Since the proof of this result was not given in their paper, we built our own transformations. A first version of the correspondences between CSP and Projection can be found in [CM95] (not published) or in [MC96] (in French). In these previous versions, correspondences were done between classical labeled graph homomorphism and binary CSP. In [Mug00] I completed and reformulated correspondences between the three problems CQC-CSP-Projection in a unified framework. •

The precise definitions of these correspondences will not be given here. For that we refer to [Mug00]. We will just define each problem, outline the ideas of the correspondences, and illustrate them on examples.

Query containment Let us first recall some basic database definitions about conjunctive queries. We follow here notations of [AHV95]. Let R denote a set of n-ary relation names ($n \geq 1$) and let *dom* denote a countably infinite set of constants. A (positive and non-recursive) conjunctive query q is usually written as a rule, $q = ans(u) \leftarrow r_1(u_1), \ldots r_n(u_n),\ n \geq 1$, where $r_1 \ldots r_n$ belong to R, *ans* is not in R (*ans* stands for "answer"), u and $u_1 \ldots u_n$ are tuples of terms (variables or constants of *dom*), and each variable of u occurs at least once in $u_1 \ldots u_n$. Notice that u may be empty. A database instance α on R maps each p-ary relation of R to



$q = ans(x,\ z) \leftarrow r(x,\ y,\ z),\ s(z,\ v)$

Figure 21: Transformation from a query to a SG

a finite subset of dom^p. Given a query $q = ans(u) \leftarrow r_1(u_1),\ \ldots\ r_n(u_n)$ and an instance α on R, $q(\alpha)$ denotes the set of answers to q in α; $q(\alpha)$ is the set of tuples $\mu(u)$ where μ is a substitution of the variables of q by dom constants such that for any j in $\{1, ..., n\}$, $\mu(u_j) \in \alpha(r_j)$. When the arity of ans is 0, $q(\alpha)$ is the set $\{()\}$ if there is a substitution μ such that for any j in $\{1, ..., n\}$, $\mu(u_j) \in \alpha(r_j)$ and otherwise $\emptyset$. A query q is said to contain a query q' ($q' \subseteq q$) if, for any instance α on R, $q'(\alpha) \subseteq q(\alpha)$. The **conjunctive query containment (CQC)** problem is the associated problem: given two queries q and q', does q contain q'? It is well-known in databases that this problem can be reformulated as a query homomorphism problem, where a homomorphism is defined as follows. A **homomorphism** from $q = ans(u) \leftarrow r_1(u_1),\ \ldots\ r_n(u_n)$ to $q' = ans'(u') \leftarrow r'_1(u'_1),\ \ldots\ r'_{n'}(u'_{n'})$ is a substitution θ of the variables of q by terms (variables or constants of dom) such that $\theta(u) = u'$ and for any j in $\{1, ..., n\}$, there is i in $\{1, ..., n'\}$ such that $\theta(r_j(u_j)) = r'_i(u'_i)$. The homomorphism theorem [AHV95] proves that, given two queries q and q', q contains q' iff there is a homomorphism from q to q'.

The transformations between the CQC problem and Projection, called Q2P and P2Q, are similar to the transformations $f2G$ and $G2f$ between a logical language of FOL($\exists$, $\wedge$) and a SG language (section 2.4.2). Cf. Figure 21.

Theorem 9 (Equivalence CQC and Projection) *[CMS98][Mug00]*
The problems Conjunctive query containment *and* Projection *are polynomially equivalent; furthermore by both transformations Q2P and P2Q, every solution to one problem directly gives a solution to the other one.*

Moreover, it can be proven that the query homomorphism theorem ([AHV95], 6.2.3) is equivalent to the soundness and completeness theorem for projection between simple graphs (in normal form) [CM92], the minimization theorem ([AHV95], 6.2.6) is equivalent to the theorem about irredundant simple graphs [CM92], and the complexity theorem for query decision problems ([AHV95], 6.2.10) is equivalent to the complexity theorem for simple graphs decision problems [CM92]. See [SCM98] for more detail about these equivalences.

CSP The input of a constraint satisfaction problem is a set of variables, a set of possible values for the variables and a set of constraints between the variables. The question is to determine whether there is an assignment of values to the variables that satisfies the constraints. More specifically, a *constraint network* $P = (X,\ C,\ D,\ R)$ consists of (see Figure 22):

- a set of variables, $X = \{x_1,\ ...\ ,x_n\}$
- a set of constraints, $C = \{C_1,\ ...\ ,C_p\} \subset \mathcal{P}(X)$,
  When constraints are binary, i.e. involve two variables, the pair $(X,\ C)$ is generally seen as a non directed graph, whose nodes are the variables, and edges are the constraints. In the general case, $(X,\ C)$ is considered as a hypergraph.
- a set of variable domains, $D = D_1 \cup ... \cup D_n$, where D_i is the domain of the variable x_i.
- a set of constraint definitions, $R = \{R_1,\ ...\ ,R_p\}$; for every constraint C_i, let $(x_{i_1}\ ...\ x_{i_q})$ be any ordering of C_i variables, then $R_i \subset D_{i_1} \times ... \times D_{i_q}$.

A *solution* to P is an assignment of values to the n variables such that all constraints are satisfied, more formally:

a mapping $\mathcal{I}\colon\ X \to D$
$x_i \mapsto a \in D_i$,
and, for every constraint $C_j = (x_{j_1}\ ...\ x_{j_q})$, $(\mathcal{I}(x_{j_1})\ ...\ \mathcal{I}(x_{j_q})) \in R_j$.

A **constraint satisfaction problem (CSP)** is the associated problem: given a constraint network P, is there a solution to P?

A CSP can be recast as a Projection problem by a transformation called $C2P$ (see Figure 23). Consider a constraint network $P = (X,\ C,\ D,\ R)$. P is transformed into two SGs G and H, with the following idea. G translates the *structure* of P, i.e. $(X,\ C)$: each concept node is generic and corresponds to a variable and each relation node corresponds to a constraint. H represents

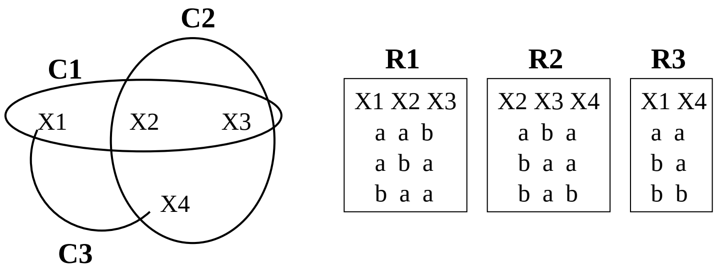
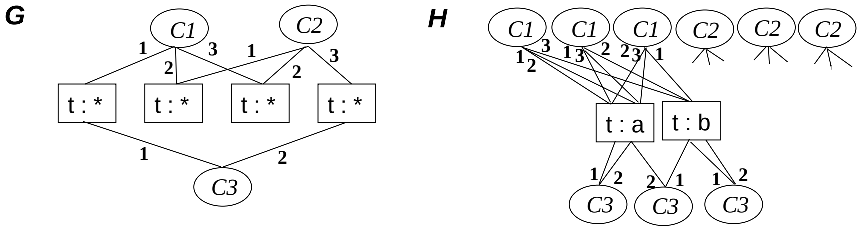


Figure 22: An instance of a CSP



G and H are obtained from P of Fig. 22 by the transformation C2P.

Figure 23: Transformation from CSP to Projection

the *constraint definitions*, i.e. $(D,\ R)$: there is one individual concept node for each value of a variable domain, and one relation node for each tuple of compatible values.

Roughly said, the question is whether there is a mapping from variables (concept nodes of G) to values (concept nodes of H) that satisfies the constraints (maps relation nodes of G to relation nodes of H), i.e. a projection from G to H. The transformation $P2C$ from Projection to CSP is based on the same ideas.

Theorem 10 (Equivalence CSP and Projection) *[CM95][Mug00]*
The problems Constraint Satisfaction Problem *and* Projection *are polynomially equivalent; furthermore by both transformations C2P and P2C, every solution to one problem directly gives a solution to the other one.*

Other equivalent problems In [GLS01a], other equivalent problems are mentioned: the *Boolean Conjunctive Query Problem* and *Query Output Problem* in databases, and the *Clause subsumption problem* in artificial intelligence.

2.7.5 Gathering complexity results

Now that correspondences between key problems have been tightly established let us gather tractability results. These results are all related to generalizations of the tree notion that keep the nice properties of trees.

Graphs and treewidth The notion of treewitdh has been introduced by Robertson and Seymour in the framework of their fundamental research on graph minors (see f.i. [RS86]). It has proved to be of prime importance in complexity theory. Indeed, many well-known combinatorial NP-hard problems become polynomially solvable, when restricted to graphs with treewidth bounded by a constant (see f.i. [Bod93]). This is the case for Homomorphism[17].

The treewidth can be seen as a measure of tree-likehood, or degree of acyclicity. Roughly a graph has small treewidth if it can be decomposed into a tree structure where each node represents a small subgraph.

More precisely, a tree decomposition of a graph G is a couple (T, χ), where $T = (X_T,\ E_T)$ is a tree and χ is a labeling function associating to each vertex $x \in X_T$ a set of vertices of G such that the following conditions are satisfied:

1. for each vertex v of G, there exists $x \in X_T$ such that $v \in \chi(x)$.
2. for each edge uv of G, there exists $x \in X_T$ such that $u \in \chi(x)$ and $v \in \chi(x)$.
3. for each vertex v of G, the set $\{x \in X_T | v \in \chi(x)\}$ induces a connected subgraph of T (equivalently: if $v \in \chi(x)$ and $v \in \chi(y)$, then, for every vertex n in the path between x and y, $v \in \chi(n)$).

A tree decomposition has width k if the largest subgraph occurring in it has $k+1$ nodes. The treewitdh of a graph is the minimum width over all its tree decompositions. In particular, a graph is acyclic iff it has treewidth 1.

The procedure to solve a problem on a graph with bounded treewidth k involves two steps: first, a tree decomposition of width k is computed; then, an algorithm based on the tree decomposition is applied.

The treewidth is NP-hard to compute. More precisely, given a graph G and an integer k, determining whether G has a treewidth less or equal to

[17]Reference? It is not difficult to see that if we have a tree decomposition of G of width k, then Homomorphism from G to H can be solved in a time exponential in k, and polynomial in the size of G and H. Indeed, there are at most n_H^{k+1} homomorphisms from a subgraph of G with $k+1$ vertices to H (n_H being the number of vertices of H).

k is an NP-complete problem [ACP87]. However, for fixed k (i.e. k does not belong to the data of the problem anymore, as it is the case for H in the H-coloring problem w.r.t. to Homomorphism), linear time algorithms have been developed for determining whether a given graph G has treewidth bounded by k and if so output a tree decomposition. See f.i. [Bod96].

Corollary 2 *The following problem "k-tree Projection" is polynomial: given a SG G with treewidth at most k, and a SG G, is there a projection from G to H?*

However, if there exist polynomial algorithms for computing "good" tree decompositions, their use in practice is very limited since the running time actually contains a large constant with k in the exponent. In other words, the best algorithms from a theoretical viewpoint are exponential in the size of the treewitdh. In [KBvH01] alternatives to these algorithms are studied; they are based on heuristics giving algorithms that are not exponential in the treewidth (but provide no theoretical guarantee about the quality of the result). The main conclusion of the paper is that further research is necessary to build algorithms usable in practice and giving good lower and upper bounds for the treewidth of a graph.

Hypergraphs and hypertreewidth We have already mentioned that from an algorithmic viewpoint it is worthwhile to see a SG as a *hypergraph* rather than a graph (cf. the projection algorithm developed by Baget in his thesis [Bag01]). Roughly, seeing a SG as a hypergraph allows to consider a relation node and their neighbors as a whole and thus take benefit of this information. Similarly, a conjunctive query or a constraint network can be seen as a hypergraph. More precisely, these objects can be seen as *directed* hypergraphs, labeled on nodes and hyperedges.

Let us take *(directed) hypergraph homomorphism* as our reference problem: given two directed hypergraphs $\mathcal{E}$ and $\mathcal{F}$ labeled on nodes and hyperedges, is there a homomorphism from $\mathcal{E}$ to $\mathcal{F}$?

Several definitions of acyclicity of a hypergraph have been defined, for which NP-complete problems on general hypergraphs become polynomial on acyclic instances. These definitions are based on the association of a graph to the hypergraph; then the notion of acyclicity of the hypergraph is defined by the notion of *acyclicity of the associated graph.* For instance, the notion of the *incidence graph* of an hypergraph corresponds to that of a SG: hyperedges are transformed into nodes and related to the nodes composing the hyperedge.

The most general notion of acyclicity among those proposed is the following: a hypergraph is acyclic if its *primal graph* is chordal (i.e. every cycle greater than three has a cord). The primal graph (or Gaifman graph) of $\mathcal{H}$ is the graph having the same set of nodes as $\mathcal{H}$ and an edge connecting two nodes if they belong to a same hyperedge of $\mathcal{H}$. The acyclicity of $\mathcal{H}$ can equivalently be defined by way of another graph, the *joint tree* JT of $\mathcal{H}$: it is a tree whose nodes are the hyperedges of $\mathcal{H}$, and such that whenever the same node x occur in two hyperedges E_1 and E_2, there is a (unique) path between them in JT and x occurs in each node on this path. A hypergraph is acyclic iff it has a joint tree. Acyclicity is efficiently recognizable, and a joint tree is computable in linear time [TY84].

Interesting to notice is the fact that the notion of a guarded covering of a SG defined by Kerdiles [Ker01] (see section 2.7.3) corresponds to that of a joint tree: more precisely, the only difference between a joint tree and the abstraction of the guarded covering, is that in this latter multi-edges are considered instead of simple edges. Thus, guarded SGs exactly correspond to acyclic SGs seen as hypergraphs.

A measure of hypergraph acyclicity can be obtained by importing the notion of treewidth and applying it to the graph associated with the hypergraph. So doing, one obtains classes of hypergraphs whose associated graphs have a treewidth bounded by a certain k, for which homomorphism is solvable in polynomial time. However, none of these notions generalize the notion of hypergraph acyclicity [FFG01]: there are families of acyclic hypergraphs whose associated graph (primal graph or incidence graph f.i.) have unbounded treewidth, and conversely there are families of hypergraphs whose associated graphs have bounded treewidth which are not acyclic.

Querywidth [CR97] has been an attempt to define a measure of acyclicity on the hypergraph directly, which would play the same role as treewidth for graphs. It generalizes hypergraph acyclicity (a hypergraph is acyclic iff its querywidth is 1). The negative point is that the problem of determining whether a hypergraph has a querywidth less or equal to k is NP-complete for $k = 4$, as it has been shown by Gottlob, Leone and Scarcello [GLS99][18]. Therefore, these authors introduce another notion, *hypertreewidth*.

Hypertreewidth is defined as follows. A *hypertree* for a hypergraph $\mathcal{H}$ is a triple $(T,\ \chi,\ \lambda)$ where $T = (X_T,\ E_T)$ is a rooted tree and χ and λ are labeling functions associating to each vertex $x \in X_T$ two sets: a set $\chi(x)$ of vertices of $\mathcal{H}$ and a set $\lambda(x)$ of hyperedges of $\mathcal{H}$. A *hypertree decomposition*

[18] However, as proved in [CR97], the querywidth of a hypergraph can be approximated by the treewidth of its incidence graph (in particular it is strictly smaller).

of $\mathcal{H}$ is a hypertree $\mathcal{HD} = (T,\ \chi,\ \lambda)$, such that the following conditions are satisfied:

1. for each hyperedge h of $\mathcal{H}$, there exists $x \in X_T$ such that the vertices of h belong to $\chi(x)$ (it is said that x covers h).

2. for each vertex v of $\mathcal{H}$, the set $\{x \in X_T | v \in \chi(x)\}$ induces a connected subgraph of T.

3. for each $x \in X_T$, $\chi(x)$ is included in the set of vertices appearing in $\lambda(x)$.

4. for each $x \in X_T$, let $\chi(T_x)$ be the union of the sets $\chi(y)$ for all y in the subtree T_x rooted in x; then, the intersection between the set of vertices appearing in $\lambda(x)$ and $\chi(T_x)$ is included in $\chi(x)$ (due to (3), the inclusion is in fact an equality).

The width of a hypertree decomposition is the maximum number of *edges* labeling a node (i.e. $max_{x \in T_x} |\lambda(x)|$). The hypertreewidth of a hypergraph is the minimum width over all its hypertree decompositions.

Hypertreewidth has the nice wanted properties:

- it generalizes hypergraph acyclicity and querywidth (if a hypergraph has querywidth $\leq k$, it also has hypertreewidth $\leq k$, but the converse does not hold)

- it is polynomially recognizable (given a hypergraph, determining whether it has a treewidth less than a constant k can be done in polynomial time)

- (directed) hypergraph homomorphism is polynomial on classes of hypergraphs with bounded hypertreewidth.

Corollary 3 *The following problem "k-hypertree Projection" is polynomial: given a SG G with hypertreewidth at most k, and a SG H, does there exist a projection from G to H?*

However, recall that this nice theoretical result does not necessarily lead to practical algorithms (see the discussion about treewidth).

Let us also notice that all complexity results we have harvested have been obtained by restrictions on the *structure* of the graphs/hypergraphs, related to acyclicity. Now, seen as combinatorial objects, SGs are not only graphs, or hypergraphs, they are *labeled.* Many heuristics yielding practically efficient algorithms are based on labels. However, as far as I know, not much work has been done about how properties of the labeling function may influence the complexity of labeled graph/hypergraph homomorphism. For instance, as noticed in [LB94] (see also [LS98]), when the labeling function on H is locally injective (all neighbors of a node have different labels), the homomorphism (or projection on a flat support) problem becomes polynomial. Indeed, as soon as a potential image has been assigned to a node of G, there is at most one possibility for each other node of G.

Expressivity results Correspondences between the treewidth and hypertreewidth notions and fragments of FOL have been established, which give indications on the expressive power of the SG fragment obtained.

Kolaitis and Vardi [KV00] proved that the class of all conjunctive queries (thus SGs) having *treewidth* less than k (i.e. their incidence graph or equivalently Gaifman graph have treewidth less than k) are equivalent in expressive power to the *k-variable* fragment of FOL($\exists$, $\wedge$), i.e. the class of FOL($\exists$, $\wedge$) formulas using at most k variables.

The class of conjunctive queries (SGs) of *hypertreewidth* at most k coincides in expressive power with the *k-guarded* fragment of FOL($\exists$, $\wedge$) [GLS01b]. The k-guarded fragment is a generalization of the guarded fragment: k atoms can conjointly act as guards instead of only one. More precisely, the condition (3) of Definition 11 is replaced by:

if α_1, ... α_j are atoms, with $j \leq k$, and ϕ is a k-guarded formula of FOL($\exists$, $\wedge$), such that

(i) $free(\phi) \subseteq \bigcup_i free(\alpha_i)$
(ii) $X \subseteq \bigcup_{i:1 \dots j} free(\alpha_i)$

then $\exists X(\alpha_1 \wedge \ \dots \ \wedge \alpha_j \wedge \phi)$ is a k-guarded formula of FOL($\exists$, $\wedge$).

For $k = 1$, one has the guarded fragment of FOL($\exists$, $\wedge$), which is thus equivalent to acyclic hypergraphs.

3 Rules

We now introduce rules and define a model that is a strict generalization of the SG model. According to our previous work, and with the $\mathcal{SG}$ family in mind, we call this model $\mathcal{SR}$.

3.1 Rules as bicolored graphs

A rule expresses knowledge of form"If H is present then C can be added". H is called the hypothesis of the rule, and C its conclusion. Rules are used to enrich simple graphs, which typically represent facts: if the hypothesis of a rule can be projected into a simple graph, say G, then the rule is applicable to G, and its conclusion can be added to G following the projection.

A CG rule is traditionally seen as a couple of simple graphs, H and C, with co-reference links between some nodes of H and some nodes of C (e.g. [Sow84], [GW95], [SM96]). Following [Bag99], we will first define a rule as a bicolored simple graph.This simple notion provides a clear visualization of a rule. Then we will give a more general definition in terms of lambda-abstractions.

Definition 12 (Bicolored graph) *A bicolored simple graph is a simple graph provided with a coloring of its nodes with two colors, say 0 and 1. The subgraph induced by color 0 nodes must be a syntactically correct simple graph (i.e. if a relation node is colored by 0 then all its neighbors must have color 0 as well).*

Definition 13 (Rule as a bicolored SG) *A rule R is a (normal) bicolored simple graph. Nodes with color* 0 *are the hypothesis nodes and they induce a subgraph called the hypothesis of the rule. Concept nodes of color 0 with at least one neighbor outside the hypothesis part are the* frontier *nodes. Nodes with color 1 are the conclusion nodes and, together with the frontier nodes, they make up the* conclusion *of the rule,* R_1.

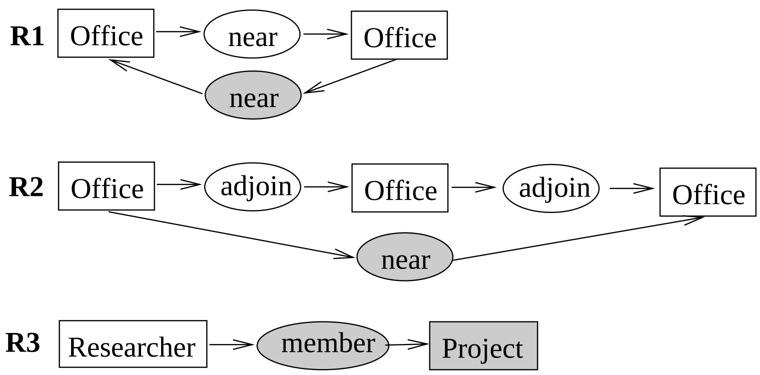


Figure 24: Rules

In drawings, hypothesis nodes are colored in white, and conclusion nodes in gray. Figure 24 shows three rules. R_1 and R_2 are about the *near* relation, supposed to be defined between offices only: (R_1) the relation *near* is symmetrical ("if an office x is *near* an office y, then y is *near* x"), (R_2) "if an office x *adjoins* an office y that *adjoins* an office z then x is *near* z". The rule R_3 says that every researcher is a member of a project ("if there is a researcher x, there is a project of which x is a member).

The logical semantics Φ is extended to rules in the following way: Let $\alpha(R_0)$ and $\alpha(R_1)$ be the conjunctions of atoms associated with the hypothesis R_0 and the conclusion R_1. Let $x_1 \dots x_n$ be the variables assigned to frontier nodes; let $x_{n+1} \dots x_p$ (resp. $y_1 \dots y_q$) be the variables assigned to the other generic concept nodes of R_0 (resp. R_1). Then, $\Phi(R) = \forall x_1 \dots \forall x_n\ (\exists x_{n+1} \dots \exists x_p\ \alpha(R_0) \rightarrow \exists y_1 \dots \exists y_q\ \alpha(R_1))$. Note that this formula is equivalent to the following one, where variables assigned to generic nodes of R_0 are all universally quantified: $\Phi(R) = \forall x_1 \dots \forall x_n\ \forall x_{n+1}\ \forall x_p\ (\alpha(R_0) \rightarrow \exists y_1 \dots \exists y_q\ \alpha(R_1))$. In both cases variables assigned to non frontier nodes of the conclusion part are existentially quantified thus the obtained formula is not a clause. For instance consider the rule R_3 of Fig 24, expressing that every researcher is a member of a project. $\Phi(R_3) = \forall x(Researcher(x) \rightarrow \exists y Project(y) \wedge member(x,y))$.

3.2 Forward chaining

A rule R is *applicable* to a SG G if there is a projection, say Π, from the hypothesis of R to G. In this case, the result of the *application of R to G following* Π is the SG G' obtained from G and the conclusion of R by restricting the label of each frontier node c in the conclusion to the label of its image $\Pi(c)$ in G, then joining c and $\Pi(c)$. We say that G' is *immediately derived* from (G,R). Notice that each projection from the hypothesis of a same rule to a SG defines a different way of applying this rule and is likely to add new information to the SG. Consider for instance the SG G of Fig. 25, which describes spatial information about offices, and rules of Figure 24. R_1 is applicable to G (since *adjoin* $\leq$ *near*), and so is R_2. Let us consider R_2. There are two ways of applying this rule, depending on whether its hypothesis is mapped to the path `[Office:#1]->(adjoin)->[Office:#2]->(adjoin)->[Office:#3]` or to the path `[Office:#2]->(adjoin)->[Office: #3]->(adjoin)->[Office:#4]`. In the first case for instance, a relation node `(near)` with predecessor `[Office:#1]` and successor `[Office:#3]` is added to the SG G, leading to G_1. In this example, applying all rules in all possible ways as long as they add "new"

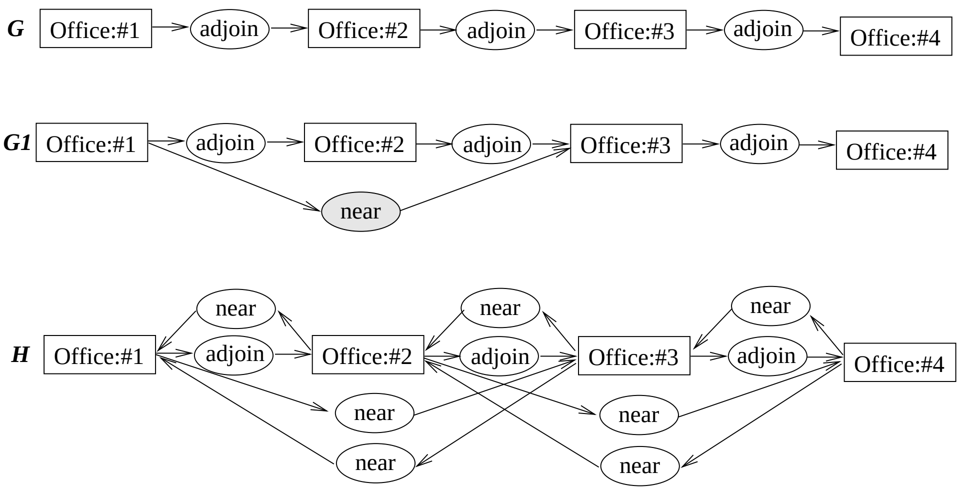


Figure 25: Graph derivation

information is a finite process (leading to the graph H of Figure 25) but it is not true in general (cf. section 3.5).

Let $\mathcal{R}$ be a set of rules. A SG G' is said to be *immediately derived* from $(G, \mathcal{R})$ if there exists a rule R in $\mathcal{R}$ such that G' is immediately derived from (G, R). A graph G' is said to be *derived* from $(G, \mathcal{R})$ if there exists a derivation $G = G_0, G_1, \ldots, G_k = G'$ $(k \geq 0)$ such that each G_{i+1} $(0 \geq i < k)$ is immediately derived from $(G_i, \mathcal{R})$.

When the KB is composed of facts and rules, the deduction problem asks whether there is a sequence of rule applications enriching the facts such that the goal can be reached, i.e. leading to a graph to which the goal can be projected. E.g. consider the fact G of Figure 25, and let Q be the SG `[Office:#4]->(near)->[Office]` ("is #4 near an office?"). Q does not project to $\mathcal{G}$, but applying the rules, one adds the information `[Office:#4]->(near)->[Office:#3]` (also `[Office:#4]->(near)->[Office:#2]`), thus answering Q.

Definition 14 (Deduction in $\mathcal{SR}$) *The Deduction Problem takes as input a KB $\mathcal{K} = (\mathcal{G},\ \mathcal{R})$ where $\mathcal{G}$ is a set of facts and $\mathcal{R}$ is a set of rules, and a goal Q, and asks whether there exists an $\mathcal{R}$-derivation from $\mathcal{G}$ to a SG H such that $Q \geq H$.*

Generally speaking, there are two classical ways of using rules. The *forward chaining* starts from the facts and applies rules to the facts in order to produce new facts; this cycle is repeated until either no rule can be applied (all knowledge deducible from the base has been made explicit), or a specified state is reached (for instance an answer to a question has been found). The *backward chaining* starts from a question, classically called a goal, and tries to build a derivation leading to an answer to this goal in a backward manner. The idea is that if a rule conclusion "matches" the goal, i.e. could answer part of the goal, then the corresponding rule application can be the end of a derivation answering the goal, provided that a new goal composed of the remaining part of the original goal and the hypothesis of the rule is answered; the process stops when the new goal is empty (i.e. the rule considered has an empty hypothesis, it is a fact, and the goal is completely matched by the rule conclusion) or all possibilities have been explored.

In our framework, forward chaining is naturally associated with the definition of a rule application. The following theorem shows that this mechanism is sound and complete w.r.t. FOL, i.e. given a KB $\mathcal{K} = (\mathcal{G}, \mathcal{R})$, a SG Q is deducible from $\mathcal{K}$ iff its logical translation is deducible from the logical translation of $\mathcal{K}$. As operations rely on projection, completeness is submitted to a normality condition. A KB is said to be in normal form if the set of facts considered as one graph ($\mathcal{G}$) is in normal form.

Theorem 11 (Soundness and completeness of forward chaining) *[SM96] Let $\mathcal{K} = (\mathcal{G}, \mathcal{R})$ be a KB, where $\mathcal{G}$ is a set of facts and $\mathcal{R}$ a set of rules, and let Q be a SG, all relative to a support $\mathcal{S}$.*

- *if Q is deducible from $\mathcal{K}$ (i.e. there is a SG H that can be derived from $\mathcal{K}$ such that Q projects to H) then $\Phi(\mathcal{S}),\ \Phi(\mathcal{R}),\ \Phi(\mathcal{G}) \vDash \Phi(Q)$*
- *provided that $\mathcal{K}$ (i.e. $\mathcal{G}$) is in normal form and the graph obtained at each derivation step is put into normal form: if $\Phi(\mathcal{S}),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(Q)$ then Q is deducible from $\mathcal{K}$.*

3.3 Backward chaining

In [SM96] (also [Sal98]) we defined backward chaining operations and proved associated soundness and completeness results. We will not detail these operations here because their definitions are rather technical. The general mechanism relies on the same ideas as a classical goal resolution (e.g. Prolog), except that it is based on graph operations. The basic operation is a matching, called a "unification", between a rule R (a simple graph being seen as a rule with empty hypothesis) and a goal Q, which builds a new goal Q'. Q' is such that if one applies R to Q' a specialization of Q is obtained. In other words, if Q' is proven to be deducible from the KB, so is Q. A sequence of unifications starting from Q is called a "piece resolution of Q". It ends successfully if the last produced goal is the empty graph.

Exploiting the graph structure of rules and goals, the mechanism tries to process subgraphs as large as possible without decomposing them into simpler structures. It relies on the notion of a *piece*, which is a subgraph of a rule conclusion defined as follows. A *cutpoint* of a rule conclusion is either a frontier node or an individual node. Then, two nodes of the conclusion belong two the same piece if there exists a path between them that does not go through a cutpoint. The unification between a goal and a rule is guided by the pieces of the rule conclusion, and a unification is eligible only if a part of the goal unifies with an entire piece of the rule conclusion. Thus the goal is erased by subgraphs corresponding to pieces.

The following result is obtained:

Theorem 12 (Soundness and completeness of backward chaining) *[SM96] Let $\mathcal{K} = (\mathcal{G}, \mathcal{R})$ be a KB, where $\mathcal{G}$ is a set of facts and $\mathcal{R}$ a set of rules, and let Q be a SG, all relative to a support S. There is a piece resolution of Q that ends successfully iff $\Phi(\mathcal{S}),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(Q)$.*

Let us also point out the originality of piece resolution. It is a really graph-based mechanism, in the sense that it processes goals and rules by subgraphs as large as possible, the pieces. Other resolution mechanisms known are more or less translated from logical resolution. Only one is (and has been proved) sound and complete [GW95]. The goal is put into anti-normal form, i.e. split into trivial subgraphs each composed of one relation node and its neighbors. Then unification is performed to each trivial subgraph. This mechanism is thus very similar to classical logical resolution mechanisms which erase the goal one atom after another.

A practical comparison between piece resolution and Prolog resolution was made by Coulondre and Salvat [CS98]. More precisely, the rule and piece notions were translated into the logical framework, and a straightforward implementation of the obtained piece resolution mechanism was compared to the SLD resolution of Prolog. It showed an important gain, sometimes huge, in the number of backtracks. One reason for this gain is obviously that a part of backtracks made by Prolog was hidden into the piece unification operation. But there are other reasons. Due to the piece notion, the graph resolution mechanism can perform unifications which are guided by the structure of the goal and rules. Then substitutions that are not piece unifications are rejected whereas Prolog uses them blindly and detects a failure later.

3.4 Rules as couples of lambda abstractions

In order to define a slightly more general kind of rules, we will use the notion of a *lambda-abstraction* introduced in [Sow84]. A (normal) lambda-abstraction $L = \lambda x_1 \ldots x_n\ G$, $n \geq 0$, consists of a (normal) SG G and n distinguished generic concept nodes of G, $x_1 \ldots x_n$. A SG can be seen as a lambda-abstraction with $n = 0$. The logical semantics of a lambda-abstraction $x_1 \ldots x_n\ G$ is obtained from that of G by removing the existential quantification of variables associated to $x_1 \ldots x_n$ (which are thus free variables).

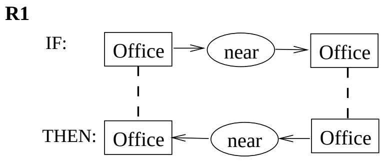


Figure 26: A couple of lambda-abstractions representing a rule

Definition 15 (Rule as a couple of λ-abstractions) *A lambda-rule $R : H \Rightarrow C$ is a couple of (normal) lambda-abstractions ($H = \lambda x_{1_1} \ldots x_{1_n}\ G_1$, $C = \lambda x_{2_1} \ldots x_{2_n}\ G_2$). H is called the hypothesis of the rule, and C is its conclusion. For simplicity, we denote by the same name x_i the nodes x_{1_i} and x_{2_i} when there is no ambiguity. These nodes are called the frontier nodes.*

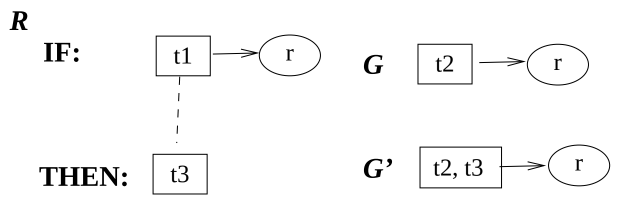
Fig. 26 presents a lambda-rule, saying that the relation *near* (between concepts of type Office) is symmetrical. This rule is equivalent to the rule R_1 of Fig. 24.

The logical semantics Φ of a rule is now defined in the following way: assign the same variables to frontier nodes x_{1_i} and x_{2_i} ; let $x_1 \ldots x_n$ be the variables thus assigned to frontier nodes; let $\Phi(H)$ and $\Phi(C)$ be the formulas associated with lambda-abstractions H and C. Then, $\Phi(R) = \forall x_1 \ldots \forall x_n\ (\Phi(H) \rightarrow \Phi(C))$[19].

When x_{1_i} and x_{2_i} have the same type for all i, a lambda-rule can be equivalently defined as a bicolored rule, and the rule application mechanism can be translated. But in general the application of a lambda-rule does not only add new concept and relation nodes, it may also specialize types of existing concept nodes. Does such an extension lead to difficulties? It depends of choices made about concept types. Let us consider the example

[19]Note that atoms associated to frontier nodes in $\Phi(C)$ could be suppressed to obtain a more concise formula, but we do not consider this kind of issue here.

of Fig. 27. There are three concept types t_1, t_2 and t_3, where t_1 is greater than t_2 and t_3, which are both incomparable. The rule R says that if there is an entity x of type t_1 such that $r(x)$ then x is of type t_3". R is applicable to G, the frontier node of type t_1 being mapped to the node of type t_2. The rule application mechanism should ensure that there is a way and a unique way of merging this node of G and the conclusion node of R. If conjunctive types are used, this condition is always ensured, leading on this example to a node of type $\{t_2,\ t_3\}$.



Assume t_1 greater than t_2 and t_3, both incomparable. R is applicable to G. Under the conjunctive type assumption, G' is obtained.

Figure 27: Lambda-rule application

Let us thus consider conjunctive types. Now, when applying a rule R to a SG G following a projection Π, one has to restrict the label of each frontier node c in the conclusion *and* the label of its image $\Pi(c)$ before joining c to $\Pi(c)$. The new label of the nodes is the greatest lower bound of the labels of c and $\Pi(c)$. This mechanism remains sound and complete with respect to logical deduction[20].

Note. Type definitions with rules. Let us open a parenthesis about type definitions [Sow84]. We briefly presented type definitions in the introduction (section 1.4) in order to compare CGs and description logics. A concept type T (or a n-ary relation type) can be defined as a specialization of a type t, called its genus (it is the supertype from which it is defined) and a unary (n-ary) lambda-abstraction, called its differentiae, specifying the properties that distinguish any t from a T. Let us consider concept types (for relation types definitions see [Lec97]). Let $T = (t,\ \lambda\, x\ G)$, where x has type t in G. This definition says an entity is of type T iff it is of type t

[20] This can be easily checked without even going back to the proof, by considering that SGs with conjunctive types can be transformed into SGs with only one concept type, original concept types becoming unary relation types. On these graphs, both rule definitions coincide. Note however that it would remain to check whether the *backward* chaining mechanism can still be used without changes.

and satisfies information encoded in G. The position of the new type T in the concept type set has to be calculated. In all cases, T will be a specialization of t, but may also be a specialization of other types (which are not generalizations of t). The logical formula assigned to such a definition is an equivalence $\forall x(T(x) \leftrightarrow \Phi(\lambda x G))$.

It can be represented by two rules, $(\lambda\ x\ [T],\ \lambda\ x\ G)$ and $(\lambda\ x\ G,\ \lambda\ x\ [T])$, where [T] is a generic concept node of type T.

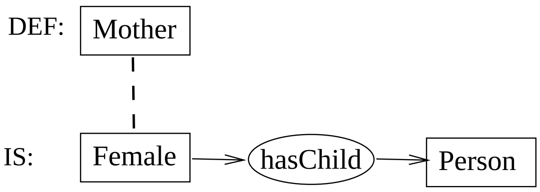


Figure 28: Concept type definition

For instance, the definition of Fig. 28 defines a “mother” as a “female who has a child”. It can be equivalently expressed with two rules, one (R_1) saying that “for all x, if x is a mother then x is a female who has a child” (assuming that in the support, Mother < Female, part of the knowledge contained in the rule conclusion is already known), the other saying that (R_2) “for all x, if x is a female who has a child then x is a mother”. Applying R_1 to the SG [Mother:Ann], one obtains [Mother:Ann]→(hasChild)→[Person], since the type $\{Mother, Female\}$ is equivalent to $\{Mother\}$. Applying R_2 to the SG [Female:Ann]→(hasChild)→[Female], one obtains
[Mother:Ann]→(hasChild)→[Female].

Notice that rules are actually more expressive than type definitions. For instance, the DL definition D given in the introduction (“all men who teach at least one course and teach only computer science courses”) cannot be expressed by a CG type definition, but can be with two rules.

3.5 Decidability and complexity

Semi-decidability of $\mathcal{SR}$ deduction. As first noticed by M.C. Rousset [21], the logical formula associated to a rule is similar to a tuple-generating dependency (TGD) in databases[22]. Indeed, the only differences between a TGD t and the formula $\Phi(R)$ associated with a rule R, are, on one hand that t may contain equality atoms, and on the other hand that $\Phi(R)$ goes

[21] Oral communication, 1997.

[22] We consider here the logical formulation of a TGD, cf. for instance [AHV95].

with a set of logical formulas translating the partial orders on types in the support. But t can be (polynomially) transformed into an equivalent set of formulas without equality (cf. [CS98] based on a result of [BV84]), and we can incorporate the support in the graphs as seen in section 2.4.2 (cf. the mapping $G2f$). The following problem, *implication problem for dependencies*, has been shown to be semi-decidable[23] on TGDs: given a set T of TGDs, and a TGD t, is t deducible from T? [Var84].

Coulondre and Salvat used this result to prove that $\mathcal{SR}$ deduction is semi-decidable [CS98]. We give here the reduction since it is very simple. Consider an instance of the implication problem for dependencies. Replace in the TGD t each universally quantified variable by a new constant not appearing in t and T. Let t' be the TGD obtained. Then t is deducible from the set T if and only if the conclusion of t' is deducible from T and the hypothesis of t'. Now, translating TGDs into rules (the conclusion and hypothesis of t' being actually translated into SGs), one obtains an instance of $\mathcal{SR}$ deduction.

In his thesis [Bag01], Baget gave another reduction from the *halting problem of a Turing machine*, showing that every Turing machine can be simulated in the $\mathcal{SR}$ model. Besides the non-decidability result, an important result follows from this reduction:

Theorem 13 (Rules computation model) *[Bag01] $\mathcal{SR}$-deduction is a computation model.*

In [BM02], another reduction is given from the *word problem in a semi-thue system* [Thu14]. This reduction is also interesting in itself since it proves that the problem remains semi-decidable when rules are of form "if path p_1 then path p_2" where p_1 and p_2 share origin and extremity.

Decidable cases. Let us now summarize results about decidable cases (for a detailed version see [BM02]).

First notice that a rule application may add redundant information to a graph. A trivial case is the following. Once a rule has been applied to a graph according to a given projection, it can be applied again to the resulting graph, according to the same projection, and this indefinitely. These further applications obviously produce redundant information. In [SM96], we characterized the cases when a rule application produces a graph equivalent to the original one: let G be an irredundant graph, let R be a rule. Let π_1

[23] i.e. an answer can be computed in finite time for all positive answers but not for all negative ones.

be the projection from the hypothesis of R to G. Then the graph obtained $R(G)$ is equivalent to G if and only if there is a projection from the conclusion of R to G, say Π_2, such that for each frontier node c, $\Pi_1(c) = \Pi_2(c)$. A stronger result would be the characterization (and efficient computation) of the part of the conclusion that adds new information to G, in order to partially apply a rule and obtain: if G is irredundant, so is $R(G)$.

More generally, let us say that an irredundant graph H is *full* w.r.t. a set of rules $\mathcal{R}$ if every graph that can be obtained by applying one of these rules to H is equivalent to H. In other words, no rule application can bring new information to the graph. Assuming that G is an irredundant graph and that graphs obtained by a rule application are put into irredundant form, if a full graph can be derived from G then it is unique.

Definition 16 (Finite expansion set) *A set of rules $\mathcal{R}$ is called a* finite expansion set *(f.e.s) if, for every SG G, there exists a (finite) $\mathcal{R}$-derivation $G \ldots G'$ such that irr(G') is full w.r.t. $\mathcal{R}$. We denote by $G^{\mathcal{R}}$ this full graph.*

If $\mathcal{R}$ is a f.e.s, deduction in $\mathcal{SR}$ becomes decidable (but it is not a *necessary* condition for decidability). Indeed, in order to determine whether a SG Q is deducible from a KB $(\mathcal{G}, \mathcal{R})$, it suffices to compute $\mathcal{G}^{\mathcal{R}}$, then to check the existence of a projection from Q to $\mathcal{G}^{\mathcal{R}}$.

Let us now focus on particular rules (which proved sufficient to solve the SISYPHUS-I problem for instance). A rule is said to be *range restricted* (r.r.) if its conclusion does not comprise any generic concept node. We use this expression by analogy with the so-called rules in Datalog, where all variables of the head must appear in the body [AHV95]. Such rules are also called *safe* in the literature. Consider for instance the rules of Fig 24: R_1 and R_2 are range-restricted, while R_3 is not.

Property 12 *A set of range restricted rules is a finite expansion set.*

Property 13 *Under the assumption that the maximum arity of a relation type is a constant, given a range restricted set of rules $\mathcal{R}$, the length of an $\mathcal{R}$-derivation from $\mathcal{G}$ is polynomially related to the size of $(\mathcal{G}, \mathcal{R})$.*

Property 14 (Complexity with range restricted rules) *Under the assumption that the maximum arity of a relation type is a constant, when $\mathcal{R}$ is a range restricted set, Deduction in $\mathcal{SR}$ is NP-complete.*

4 Constraints

Let us now introduce constraints, which are used to validate a set of facts. The model obtained is called $\mathcal{SGC}$. It will find its place in the $\mathcal{SG}$ family in section 6.5. We will see later more complex uses of constraints in relationship with rules.

4.1 Positive and negative constraints

A *constraint* can be a positive constraint or a negative constraint, expressing a knowledge of form "if A holds, so must B", or "if A holds, B must not". It has the same form as a rule. Thus let us first see it as a bicolored simple graph: the first color defines the condition part, and the second color the mandatory (or forbidden) part. A SG G satisfies a positive constraint C if *each* projection from the condition part of C to G can be extended to a projection of the whole C. And G satisfies a negative constraint if *no* projection of C to G can be extended to a projection of the whole C.

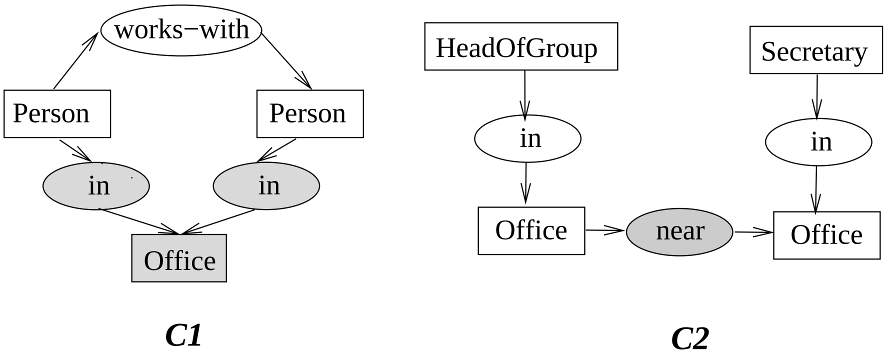


Figure 29: Constraints

Fig. 29 represents two constraints. The negative constraint C_1 expresses that "two persons working together should not share an office". The SG G of Fig 7 does not satisfy this constraint because "there is a researcher who works with researcher K" (projection of the condition part of $C1$) "and they share office #124" (extension of the projection to a projection of the whole $C1$). The positive constraint C_2 expresses that the office of a head of group must be near the offices of all secretaries.

As it was the case for rules, a more general definition of a constraint can also be given as a couple of lambda-abstractions. The definitions given above for bicolored graphs are easily adapted. Instead of extending the projection of the condition part, say Π_0, to the whole constraint, we compute a projection of the mandatory (or forbidden) part such that each frontier node has the same image as by Π_0. Let us give a very simple example that shows the interest of this generalization. Suppose one wants to define the signature of relation types by constraints (i.e., for each relation type, define the maximal type of each neighbor of a relation node of this type). The constraint to express is of form "if there is a relation node r of type t_r, then for all i ranging from 1 to the arity p of t_r, the *i-th* neighbor of r has to be of type less or equal to a certain type t_i". The condition part is made of one relation node (t_r) with p neighbors $[\top : *]$, each concept node belonging to the frontier; the obligation is restricted to the frontier $[t_1 : *]$ ... $[t_p : *]$. As they are defined bicolored graphs cannot express this constraint[24].

When the KB is composed of a set of facts $\mathcal{G}$ and a set of constraints $\mathcal{C}$, the role of constraints is to validate the fact base $\mathcal{G}$, i.e. to check its consistency. The base is said to be consistent if all constraints are satisfied. Deduction is defined only on a consistent base. Provided that the base is consistent, it is done as in $\mathcal{SG}$.

Even if they have the same form, constraints and rules are not to be confused. Consider for instance the bicolored graph R_3 of Figure 24: as a rule, it says that every researcher is a member of a project. Take the fact $\mathcal{G}$ = `[Researcher:K.]` and the query Q= `[Researcher:K.]->(member)->[Project]` ("is K. member of a project?"). If Q is asked on $\mathcal{K} = (\mathcal{G}, \mathcal{R} = \{R3\})$, the answer is "yes". Now, see R_3 as a positive constraint C. It says that every researcher *must* be a member of a project. $\mathcal{K} = (\mathcal{G}, \mathcal{C} = \{C\})$ is inconsistent, thus nothing can be deduced from it, including Q. The KB has to be repaired first.

Definition 17 (Constraint) *A* positive *(resp.* negative*)* constraint *C is a colored SG. $C_{(0)}$ is called the* trigger *(or* condition*) of the constraint, $C_{(1)}$ is called its* obligation *(resp.* interdiction*). A SG G* π-violates *a positive (resp. negative) constraint C if π is a projection of the trigger of C to the irredundant form of G (resp. to G) that cannot be extended (resp. that can be extended) to a projection of C as a whole. G* violates *C if it π-violates C for some projection π. Otherwise, G* satisfies *C.*

We have to point out the importance of the irredundancy condition on the graph to be validated by positive constraints: should we forget this

[24]They could express it if we removed the requirement that the condition part of a constraint has to be a SG; such an extension could be safely done, whereas it could not for rules since the application of a rule could then lead to a graph that is not a SG.

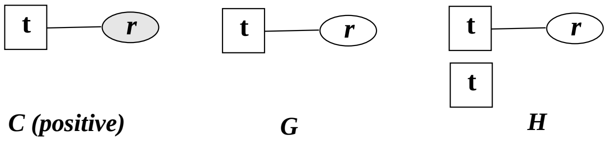


Figure 30: Redundancy and constraint violation

condition, there may be two equivalent SGs such that one satisfies a positive constraint and the other does not. Fig. 30 shows an example of such graphs. G satisfies C, but the equivalent (redundant) graph H, obtained by making the disjoint union of G and the trigger of C, does not. To avoid different consistency values for equivalent graphs, we have chosen to define positive constraint satisfaction w.r.t. the irredundant form of a SG. This problem does not occur with negative constraints. Indeed, let G_1 and G_2 be two equivalent graphs and suppose G_1 π-violates a negative constraint C; since there exists a projection from G_1 to G_2, say π_1, $\pi_1 \circ \pi$ is a projection from C to G_2, thus G_2 also violates C.

Two constraints C_1 and C_2 are said to be *equivalent* if any graph that violates C_1 also violates C_2 and conversely. Any negative constraint is equivalent to the negative constraint obtained by coloring all its nodes by 1. Furthermore, negative constraints are indeed a particular case of positive ones: consider the positive constraint C' obtained from a negative constraint C by coloring all nodes of C by 0, then adding a concept node colored by 1, with type `NotThere`, where `NotThere` is incomparable with all other types and does not appear in any graph of the KB, except in constraints. Then a simple graph G violates the constraint C if and only if it violates C'. Positive constraints strictly include negative constraints, in the sense that the associated consistency problems are not in the same complexity class (the proof follows from Th. 16).

Property 15 *Unless Π_2^P = co-NP, positive constraints are a strict generalization of negative ones.*

Since negative constraints are indeed a particular case of positive ones, we will now, unless indicated otherwise, denote by “a set of constraints” a set of positive constraints: some of them can be equivalent to negative ones.

Definition 18 (Consistency/Deduction in $\mathcal{SGC}$) *A KB $\mathcal{K} = (\mathcal{G}, \mathcal{C})$ is* consistent *if $\mathcal{G}$ satisfies all constraints of $\mathcal{C}$. A SG Q can be* deduced *from $\mathcal{K}$ if $\mathcal{K}$ is consistent and Q can be deduced from $\mathcal{G}$.*

Note that a SG Q that violates a constraint of $\mathcal{K}$ may still be deduced from $\mathcal{K}$. It does not matter since Q is a *partial* representation of knowledge deducible from $\mathcal{K}$.

4.2 Translation to FOL.

How can we translate the notion of consistency into FOL? Let us first point out that deduction in $\mathcal{SGC}$ is essentially non monotonic. Indeed, by adding information to a consistent base one may create a constraint violation. Since nothing can be deduced from the new base, previous deductions do not hold anymore. That is why in $\mathcal{SGC}$ and more general models (see section 6) it is impossible to obtain results of form "Q can be deduced from the KB $\mathcal{K}$ if and only if $\Phi(\mathcal{K}) \vDash \Phi(Q)$" as it was the case in $\mathcal{SG}$ and $\mathcal{SR}$. However, if deduction cannot be translated to FOL, consistency can.

For negative constraints, the correspondence is immediate, and relies on projection soundness and completeness w.r.t. the semantics Φ (theorems 2 and 3). Intuitively, a SG G violates a negative constraint C^- if and only if the information represented by C^- is deducible from the information represented by G.

Theorem 14 (Negative constraints and FOL) *[BM02] A SG G violates a negative constraint C iff $\Phi(\mathcal{S}), \Phi(G) \vDash \Phi(C')$, where C' is the SG underlying C (and $\Phi(C')$ is the logical formula associated to this SG).*

A bridge between FOL deduction and positive constraints consistency can be built using rules. Indeed, a graph G satisfies a positive constraint C if and only if, considering C as a rule, all applications of C to G produce a graph equivalent to G. Or, more specifically:

Property 16 *A SG G π-violates a positive constraint C iff, considering C as a rule, the application of C to G according to π produces a graph not equivalent to G.*

Property 17 *If a SG G satisfies a positive constraint C, then any graph in a $\{C\}$-derivation of G is equivalent to G.*

Using soundness and completeness of the $\mathcal{SR}$ deduction, and properties 16 and 17, one obtains the following relation with FOL deduction.

Theorem 15 (Positive constraints and FOL) *[BM02] A SG G violates a positive constraint C iff there exists a SG G' such that $\Phi(\mathcal{S}), \Phi(G), \Phi(C) \vDash \Phi(G')$ and not $\Phi(\mathcal{S}), \Phi(G) \vDash \Phi(G')$, where $\Phi(C)$ is the translation of C considered as a rule.*

This theorem can be reformulated in terms of abductive inference (see indirect abduction in [Kon96]). Indeed, consider a background theory $\Sigma = \Phi(\mathcal{S}), \Phi(G)$ and an observation $O = \neg\Phi(C)$, then G violates C iff there exists an abductive explanation for O of the form $\neg f$ where f is a formula of FOL($\wedge$, $\exists$).

4.3 Complexity.

The problem "does a given graph satisfy a given constraint?" is Π_2^P-complete for general constraints (Π_2^P is $(\text{co-NP})^{\text{NP}}$).

Theorem 16 (Complexity in $\mathcal{SGC}$) *[BM02] Consistency in $\mathcal{SGC}$ is Π_2^P-complete.*

Corollary 4 *Deduction in $\mathcal{SGC}$ is Π_2^P-complete.*

Note that it appeared later that Consistency in $\mathcal{SGC}$ is equivalent to a generalized constraint satisfaction problem, called MIXED-SAT, proved to be Π_2^P-complete ([FLS96], and [BM02] for a reduction from MIXED-SAT to $\mathcal{SGC}$-Consistency).

Let us consider particular cases of constraints, first of all negative constraints. The consistency check becomes co-NP-complete, since one has to check the absence of a projection.

Theorem 17 (Complexity with negative constraints) *[BM02] When $\mathcal{C}$ is composed of negative constraints only, $\mathcal{SGC}$-Consistency is co-NP-complete. $\mathcal{SGC}$-Deduction is DP-complete*[25].

Since there is a decrease in complexity when considering negative constraints only, it should be interesting to find classes of constraints more general than negative ones, with a "low" complexity. As we have done for rules, we can consider range-restricted constraints. Disconnected constraints, in which the trigger and the obligation are not connected, are also of interest (they for instance include the topological constraints defined in [MM97]).

Property 18 *Negative constraints are a particular case of range-restricted constraints and disconnected constraints.*

[25] A language is in DP if it is the intersection of two languages, one in NP, the other in co-NP. In other words, a decision problem P is in DP, if there are two problems, P_1 in NP and P_2 in co-NP, such that an instance of P has answer "yes" if and only if it has answer "yes" in P_1 *and* in P_2.

Indeed, as already noticed, a negative constraint is equivalent to a positive constraint whose obligation is composed of one concept node of type `NotThere`, where `NotThere` is incomparable with all other types and does not appear in any SG except $\mathcal{C}$ (it is thus a disconnected constraint). W.l.o.g. this node can be labeled by an individual marker (which, as `NotThere`, appears only in $\mathcal{C}$), thus leading to a constraint which is both disconnected and range-restricted.

Theorem 18 (Complexity with disconnected constraints) *[BM02] When $\mathcal{C}$ is composed of disconnected constraints only, $\mathcal{SGC}$-Consistency is co-DP-complete.*

Range-restricted constraints are trickier to study, we did not manage to define an exact complexity for this problem.

Note. Related works. Our study of constraints comes from the need of particular constraints to express "necessary conditions" on types (if a node c is of type t, then it must have a particular subgraph in its environment) as well as functional constraints (if c is of type t, then one must find at most one specialization of a certain subgraph in its environment). Note that functional constraints cannot be expressed by our constraints as they are defined.

Constraints presented here were first used for solving the Sisyphus-I case study, then we studied their theoretical properties.

Different forms of constraints can be found in the CG literature. To the best of my knowledge, the most general ones are the minimal/maximal constraints defined in [DHL98]. Our constraints (let us call them SG-constraints) are a particular case of these minimal descriptive constraints. A minimal descriptive constraint can be seen as a set of SG-constraints with the same trigger A; its intuitive semantics is "if A holds so must B_1 or B_2 or ... B_k". A SG satisfies a minimal descriptive constraint if it satisfies at least one element of the set. Note that the "disjunction" does not increase complexity w.r.t. to SG-constraints. The proof of theorem 16 (complexity of $\mathcal{SGC}$ -CONSISTENCY) can be used to show that consistency of minimal descriptive constraints is also Π_2^P-complete.

[DHL98] pointed out that minimal descriptive constraints generalize most constraints found in the CG literature. Actually, these latter constraints are also particular cases of SG-constraints. For instance, let us cite the topological constraints used by [MM97]. These authors propose to define non-validity intervals in the SG space (quasi)-ordered by specialization. A non-validity interval is a pair of SGs u and v, such that $v \leq u$. A graph G is said to fall into this interval, if $G \leq u$ and not $G \leq v$. Then it is said to be invalid. This can be expressed in terms of SG-constraints in the following way: G is invalid if it violates the disconnected constraint "if u holds, so must v".

Let us add that, in these CG works, constraints are used to check consistency of SGs solely (as in $\mathcal{SGC}$) and not of richer knowledge bases composed of rules (as in $\mathcal{SRC}$, see later) and they are not integrated into more complex reasoning (as in $\mathcal{SEC}$ or in $\mathcal{SREC}$, see later).

There should be other connections with works about verification of knowledge bases composed of logical rules (for instance Horn rules), namely with the work of [LR96], in which constraints are TGDs, thus have the same form than ours, but we did not find direct relationships between their framework and ours. •

5 Conceptual graphs and first-order logics

Full conceptual graphs are equivalent to first order logics (without functions). What does such a claim mean? If we stay at a *descriptive* level, it means that every FOL formula can be translated into a full CG, say by a mapping Ψ, and, reciprocally, every full CG can be translated to a FOL formula, say by Φ, such that the composition of the two translations keeps logical equivalence: for every FOL formula f, $\Phi(\Psi(f)) \equiv f$. This kind of correspondence allows one to consider full CGs as a diagrammatical representation of FOL formulas. For instance full CGs could be used at an interface level: the user manipulates CGs, which are translated into FOL formulas, on which logical provers can be used to do reasoning, and results are translated back into CGs. Notice however that for a formula f we only have equivalence between $\Phi(\Psi(f))$ and f and not equality (or isomorphism). The question of whether using full CGs at an interface level only is interesting at all is another issue we will not discuss.

We are interested in a stronger equivalence, at a *reasoning* level. Firstly, the full CG formalism has to be provided with its own operations for reasoning. These operations should compute sound and complete inferences, which means that there should be a translation from full CGs to FOL formulas, the above Φ, such that a CG G can be deduced from a CG H if and only if $\Phi(G)$ can be deduced from $\Phi(H)$. It is only after this first step that the FOL expressiveness of the formalism can be proved, by exhibiting a mapping Ψ as above. Notice that in this case the notion of the equivalence between two CGs has a meaning, and $\Psi(\Phi(G)) \equiv G$ for any G holds if and only if $\Phi(\Psi(f)) \equiv f$ for any FOL formula f holds.

At the end of the 19th century, the logician Peirce introduced a diagrammatical system for logics, *existential graphs*, composed of two parts, the alpha system corresponding to propositional logics and the beta system corresponding to FOL (there was also an unfinished third part, gamma graphs, aiming at dealing with various kinds of modalities)[26].

Negation is represented as an oval enclosing the part of the diagram it negates. Multiple occurrences of the same variable are represented by lines. The only –implicit– quantifier is the existential quantifier, and the universal quantifier is obtained as the negation of the existential one. Alpha and beta rules define diagrammatic transformations.

Adapting Peirce's existential graphs, Sowa defined full CGs from SGs by adding boxes representing negation, and co-reference links representing equality of terms, as well as diagrammatical inference rules [Sow84].

We give below two slightly different definitions of full CGs and their logical translation. Both versions extends SGs, but one corresponds to FOL with equality (in short $\text{FOL}_=$), the other to FOL without equality. They differ by their definition of co-reference links. We then skip inference rules *à la Peirce* and focus on a tableau calculus which has interesting features.

By adding negation to $\text{FOL}(\exists, \wedge)$, one obtains the whole FOL. Thus deduction become semi-decidable. We will see two ways of conciliating negation and decidability. The first one starts from simple graphs and tries to add a limited kind of negation with the hope that projection can process it in a complete way. We will see the limits of this approach. The second one restricts full CGs to CGs corresponding to a fragment of FOL in which deduction is decidable. This approach is only at its beginning.

Compilation of existing works. Full conceptual graphs and their logical semantics, as well as associated diagrammatical rules of inference, were introduced in [Sow84]. It was claimed that the full FOL was obtained, with arguments relying on the fact that Peirce's calculus was sound and complete. Indeed, the main difference between the two models is type processing. A proof of the completeness of the calculus was sketched.

In [Wer95a] minor mistakes in the original definitions were corrected. A proof of completeness was given in [Wer95b] but it is not available in

[26] E.g. see [Ohr97]. Peirce, who had earlier been working intensively on an algebraic approach to logic, came to consider that diagrammatical reasoning was "the only fertile reasoning". He considered his existential graphs as its most important contribution to logics.

English. However, Wermelinger's definitions took into account higher order concept and relation types we do not want to consider. [BMT99] stripped off higher order features and proved equivalence of full CGs with $\mathrm{FOL}^=$. Since they did not consider any calculus over full CGs, equivalence is shown at a descriptive level (two mappings are given: Φ from full CG to $\mathrm{FOL}^=$, and Ψ from $\mathrm{FOL}^=$ to full CGs, such that for any formula f of $\mathrm{FOL}^=$, $\Phi(\Psi(f)) \equiv f$).

[Ker01] defined full CGs with a slightly different definition of co-reference links (extending the one we gave for SGs), and another extension of Φ, which maps full CGs to FOL without equality. The really interesting point is that he provides full CGs with a tableau calculus, thus a potentially efficient procedure, which is sound and complete. Moreover SGs are considered as basic constructs, thus are never decomposed into more elementary elements.

5.1 Full CGs

The definition of full CGs given here rely on [BMT99].

Definition 19 (Graph proposition) *A* graph proposition *over a support $\mathcal{S}$ is:*

- *either a box $[g\ p_1\ ...\ p_n]$, which contains a SG g (possibly empty) and finitely many graph propositions $p_1\ ...\ p_n$.*
- *or a negated graph proposition $\neg p$.*

All SGs composing a graph proposition are disjoint. In a linear notation, graph propositions can be seen as the expressions generated by the grammar $p ::= [gp*]|\neg p$, where g stand for a SG.

Definition 20 (Context) *The boxes composing a graph proposition are also called the* contexts *of the proposition. A* negated context *is a box preceded by a negation. A context p* dominates *a context q if q is contained in p (p and q may be the same context). For each SG G, there is exactly one context which contains G at first level. It is called* the context *of G. By extension, we say that a node of a SG G_1* dominates *a node of a SG G_2 if the context of G_1 dominates the context of G_2.*

Definition 21 (full CG) *A full conceptual graph (CG) over a support $\mathcal{S}$ is a pair $(p, co\text{-}ref)$, where:*

1. *p is a graph proposition over $\mathcal{S}$.*

2. *Let $C(p)$, called the set of concept nodes of p, be the union of the concept node set of all simple graphs composing p. $co\text{-}ref$ is a binary symmetric relation over $C(p)$, such that $(c_1,\ c_2) \in co\text{-}ref$ implies that c_1 dominates c_2 or c_2 dominates c_1. $(c_1,\ c_2)$ is called a co-reference link.*

This definition calls for a few comments. Since full CGs are meant to represent any formula of $\mathrm{FOL}^{=}$, co-reference links are allowed between concept nodes of any label: co-referent nodes do not necessarily have the same type; one node can be generic, while the other is individual; or they can be two individuals with different markers. Indeed, in classical FOL, if a and b are two constants, they may be interpreted by the same element of a domain, thus $(a = b)$ is a well-formed formula; in other words, the unique name assumption usually made in KR is not made here. Co-reference links can link two nodes of the same context (since a context dominates itself), or of nested contexts.

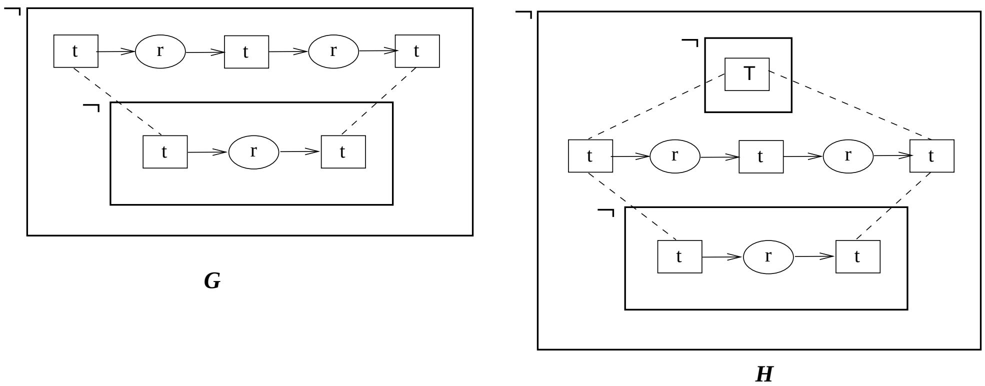


Figure 31: Full conceptual graphs

In drawings, negation is represented by a rectangular box preceded by a $\neg$ symbol. Fig. 31 shows two CGs. G says that the relation r is transitive on entities of type t, i.e., informally, for all x, y, z of type t, if $r(x,\ y)$ and $r(y,\ z)$ then $r(x,\ z)$); G is equivalent to the bicolored rule of Fig. 4. H adds to the hypothesis of the rule the condition $(x \neq z)$.

We will now give a precise definition of the meaning of G and H by way of their logical semantics.

Wermelinger [Wer95a] pointed out that the universal type $\top$ has to be translated in such a way that the graphs $[\top : *]$ or $[\top : a]$ are valid (i.e. the associated formula is true in every interpretation). He proposed to translate it by a trivial equality ($x = x$ or $a = a$).

We refer to [BMT99] for a formal definition of Φ corresponding to previous definition of a full CG. Informally, the simple graphs occurring in the CG, say G, are first translated. Distinct variables are associated to distinct generic concept nodes. Each simple graph g is interpreted as a conjunction of atoms $\alpha(g)$ (with the special processing of $\top$), without quantification for the moment. Co-reference links are processed in the dominated context: let $(c_1,\ c_2)$ be a co-reference link, with c_1 dominates c_2. In the simple graph of c_2 the atom $(id(c_1) = id(c_2))$ is added, where id is the function which associates terms to concept nodes. Now the formula associated to the CG G is defined following the inductive definition of the graph proposition underlying a CG. If the proposition of G is a box $p = [g\ p_1\ ...\ p_n]$ then $\Phi(G) = \Phi(p)$ is the existential closure of the conjunction of $\alpha(g)$ (free formula) and the $\Phi(p_i)$. If the proposition of G is a negated proposition $[\neg p]$, $\Phi(G)$ is the negation of $\Phi(p)$.

Let us consider the CG G of Fig. 31. Let us call G_1 and G_2 the simple graphs composing G, G_1 being the graph in the outermost context. We first assign distinct variables to generic nodes and free formulas to G_1 and G_2.

$\alpha(G_1) = t(x_1) \wedge t(y) \wedge t(z_1) \wedge r(x_1, y) \wedge r(y, z_1)$
$\alpha(G_2) = t(x_2) \wedge t(z_2) \wedge r(x_2, z_2) \wedge (x_1 = x_2) \wedge (z_1 = z_2)$

The graph proposition underlying G has the following structure: $\neg[G_1\neg[G_2]]$. The closed formula $\Phi([G_2])$ is $\exists x_2 \exists z_2 \alpha(G_2)$. The closed formula $\Phi([G_1\neg[G_2]])$ is $\exists x_1\ \exists y\ \exists z_1(\alpha(G_1) \wedge \neg(\exists x_2 \exists z_2 \alpha(G_2)))$. Finally, one obtains

$\Phi(G) = \neg(\exists x_1\ \exists y\ \exists z_1(t(x_1) \wedge t(y) \wedge t(z_1) \wedge r(x_1, y) \wedge r(y, z_1) \wedge \neg(\exists x_2 \exists z_2 t(x_2) \wedge t(z_2) \wedge r(x_2, z_2) \wedge (x_1 = x_2) \wedge (z_1 = z_2))))$.

This formula can be rewritten in a more classical way as:

$\Phi(G) = \forall x_1\ \forall y\ \forall z_1(t(x_1) \wedge t(y) \wedge t(z_1) \wedge r(x_1, y) \wedge r(y, z_1) \rightarrow \exists x_2 \exists z_2(t(x_2) \wedge t(z_2) \wedge r(x_2, z_2) \wedge (x_1 = x_2) \wedge (z_1 = z_2)))$.

It is equivalent to the following formula in FOL without equality (which is obtained by the alternative mapping Φ_2 discussed later):

$\Phi_2(G) = \forall x\ \forall y\ \forall z(t(x) \wedge t(y) \wedge t(z) \wedge r(x, y) \wedge r(y, z) \rightarrow r(x, z)$.

The graph H adds a negated box which is interpreted by the formula $\neg(\exists v(v = v) \wedge (x_1 = v) \wedge (z_1 = v))$, where the atom $(v = v)$ can be suppressed. An equivalent formula is $\neg(x_1 = z_1)$. In other words, H adds the condition that the concept nodes associated with x_1 and z_1 represent distinct entities.

Let us now state the equivalence between full CGs and FOL= at a descriptive level:

Theorem 19 (Equivalence full CGs and FOL$_=$) *[BMT99] Let $\mathcal{S}$ be an* *unordered support and $\Sigma_{\mathcal{S}}$ be the corresponding first-order signature (including equality). The mapping Ψ (defined in [BMT99]) associates to each FOL$^=$ formula f on $\Sigma_{\mathcal{S}}$ a full CG such that $\Phi(\Psi(f)) \equiv f$.*

This result shows that CGs can be used as a graph representation for FOL$^=$ (the partial orders on types being unused).

Although they are not defined in the same way, the full conceptual graphs defined in [Ker01] can be seen as a special case of the previous ones, obtained by the following restrictions:

1. Co-identical nodes (concept nodes with the same individual marker or related by a co-reference link) have the same type.

2. A co-reference link relates two generic nodes.

3. The co-reference relation is not only (reflexive and) symmetric, it is also transitive. Thus *co-ref* is a equivalence relation over the generic nodes occurring in the graph proposition. Moreover, in each co-reference class, there is at least one node that dominates all the others (there are possibly several such nodes). In other words, among the contexts of these nodes, there is a dominating context.

Conditions 1 and 2 are not important restrictions. Indeed, since the full CG calculus defined here is built upon projection, their role is to ensure that the normal form of a graph can be safely computed. Condition 3 prevents certain configurations of co-reference links as the one in the CG H of figure 31 for instance. As a consequence, these full graphs can be mapped to FOL without equality[27].

The translation from CG to FOL reflects these differences. The same term is associated with the nodes of a same co-reference set. Since the same variable is associated with generic co-referent nodes, this variable is quantified only once, at first level, when the nodes of the dominating context are translated.

Roughly said, the translation is done in three steps. First associate a term to each co-reference class. The term assigned to a generic concept node is the one assigned to its co-reference class. Then, for each simple graph g composing the CG, let x_1 ... x_q be the variables associated to its concept nodes co-referent with strictly dominating nodes, and let $x_{q+1}...x_s$ be the variables associated with the other generic nodes. Let $\alpha(g)$ be the

[27] The exact relationships between the two definitions of full CGs would need to be studied more precisely.

conjunction of atoms associated to g. Construct the whole formula in the following inductive way: if the graph proposition of G is a box $p = [g\ p_1\ ...\ p_n]$ then $\Phi(G) = \exists x_{q+1} ... x_s(\alpha(g) \wedge \Phi(p_1)\ ...\ \wedge \Phi(p_n))$ (notice that the variables, if any, corresponding to nodes strictly dominated are not quantified yet). If the graph proposition of G is a negated proposition $[\neg p]$, $\Phi(G)$ is the negation of $\Phi(p)$.

E.g. the formula associated to the CG G of Fig. 31 is constructed as follows. First let x, y and z be the variables respectively associated to each co-reference class.

$\alpha(G_1) = t(x) \wedge t(y) \wedge t(z) \wedge r(x, y) \wedge r(y, z)$

$\alpha(G_2) = t(x) \wedge t(z) \wedge r(x, z)$

$\Phi([G_2]) = \alpha(G_2)$ since all nodes of G_2 are co-referent with dominating nodes.

$\Phi([G_1 \neg [G_2]]) = \exists x \exists y \exists z(\alpha(G_1) \wedge \neg\alpha(G_2))$.

$\Phi(G) = \forall x \forall y \forall z(\alpha(G_1) \rightarrow \alpha(G_2))$.

5.2 A sound and complete procedure for full CGs

In [Sow84] a calculus for full CGs is proposed, which is inspired from the α and β systems of Peirce. a he trouble is that Peirce's rules are not suited to an automatic proof procedure. The *rule of insertion* especially leads to an infinity of choices, by allowing to insert any graph in a negative context (i.e. at a place surrounded by an odd number of negated contexts; it corresponds to deduce $\neg(A \wedge B)$ from $\neg(A)$). The proof system is not analytical, in the sense it does not allow to built proofs by analyzing/ decomposing the input into its successive components. Another point, already outlined, is that there is no set of rules for CGs precisely defined and proved complete (at least no reference available).

Kerdiles proposes an analytical method, which takes advantage of the tableau method and the projection operation. As far as we know, it is the only reasoning procedure for full CGs with available proofs of soundness and completeness. We give the main ideas here and refer to the thesis for more details.

Let us first outline the principles of the tableau method [Smu68]. The tableau method consists of a set of decomposition rules which allows to build *tableaux* associated with a formula. A tableau for a given formula transforms this formula into a disjunction of conjunctions while preserving its satisfiability. It can be seen as a tree whose nodes are labeled by formulas. Each successor of a node is a consequence of this node. When a decomposition rule leads to several possibilities, the tableau branches: the current node has several successors, one for each possibility. A branch is interpreted as the conjunction of its nodes. The tree is interpreted as the disjunction of the conjunctions associated to its branches. A branch is said to be *closed* if it contains two formulas such that one is the negation of this other. In other words, it represents an unsatisfiable formula. If all branches are closed, the tableau is *closed*, and it is not possible to find a model for the disjunction of conjunctions. Since the satisfiability of the input is preserved by the tableau construction, then the input is not satisfiable.

The following result shows that the tableau method is sound and complete for FOL: for any FOL formula X, X is valid iff there exists a closed tableau for $\neg X$ [Smu68].

Kerdiles adapted this method to full CGs. Simple graphs are considered as the basic components, thus are never decomposed. A branch is said to be closed if it contains a node $\neg[G]$ where G is is simple graph, and there exists a simple graph H possibly obtained by summing several simple graphs of the branch such that G projects into the normal form of H. Thus a validity proof of a CG is a closed tableau of its negation.

Theorem 20 (CG tableau soundness and completeness) *[Ker01] There is a closed tableau started with* $\neg[G]$ *if and only if* G *is valid.*

Since the definitions of the rules involve a lot of preliminary technical definitions, we only give an example.

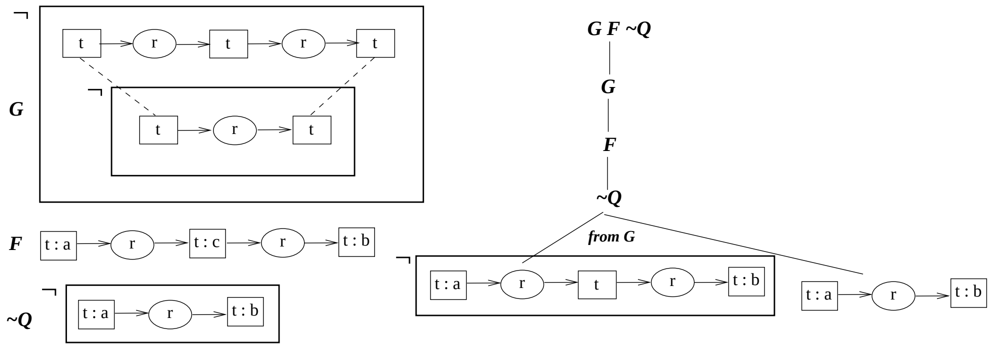


Figure 32: A tableau

Let G be the CG of Fig. 31. Let F be the SG $[t : a] \to (r) \to [t : c] \to (r) \to [t : b]$. And Q is the SG $[t : a] \to (r) \to [t : b]$. One wants to know whether Q is deducible from G (a rule) and F (a fact), i.e. prove that $(G \wedge F \to Q)$ is valid. Fig 32 shows a closed tableau for its negation,

$(G \wedge F \wedge \neg Q)$. First, it is decomposed into G, F and $\neg Q$, on a same branch. Then, from G, which is of form $\neg[g_1 g_2]$, one deduces $\neg g_1$ OR $\neg g_2$, thus splits the branch. Co-referent nodes are instanciated by the same constant, which in this case may be any constant (new or occurring in the tableau). The left branch is closed because the leaf graph is the negation of a simple graph that projects to F. The right branch is also closed because the graph $\neg Q$ is the negation of a simple graph that projects to the leaf graph (actually $\neg Q$ is the negation of the leaf graph). Thus the tableau is closed.

5.3 Atomic negation

Simple graphs allow one to describe the world in terms of conjunction of positive assertions. Let us add a very restricted form of negation, *atomic negation*, giving the possibility of saying that a certain relation does not hold between certain entities. Beside positive relation nodes, we now have negative relation nodes. A negative relation node $(-r)$, where r is its type, with neighbors $(c_1 \ldots c_k)$ says that "there is not the relation r between $c_1 \ldots c_k$"; it is logically interpreted by the atom $\neg r(x_1 \ldots x_k)$, where x_i is the term associated to c_i. Then, the order on relation labels is extended as follows: given two relation labels l_1 and l_2, $l_1 \leq l_2$ if, either l_1 and l_2 are both positive labels and $l_1 = r_1$, $l_2 = r_2$, and $r_1 \leq r_2$, or l_1 and l_2 are both negative labels and $l_1 = -r_1$, $l_2 = -r_2$, and $r_1 \geq r_2$. In other words, the opposite type order is considered for negative nodes: a negative relation node is mapped to a node with a more general type. Given this extended label ordering, projection can be used without changes in its definition. The bad news is that it is sound but not complete anymore, as shown by G. Simonet (unpublished note, 1998). Negation introduces a disguised disjunctive information that cannot be taken into account by projection.

Atomic negation was recently reconsidered by Kerdiles in his thesis [Ker01]. He gave simpler counter-examples which show very clearly how the hidden disjunctive information prevents projection completeness. These examples are given in Fig. 33 and 34.

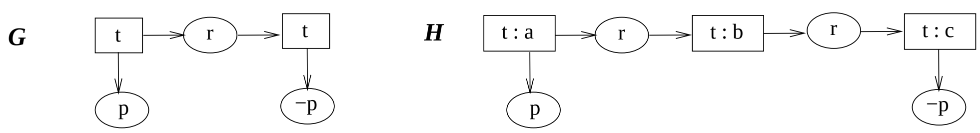


Figure 33: Atomic negation and projection (1)

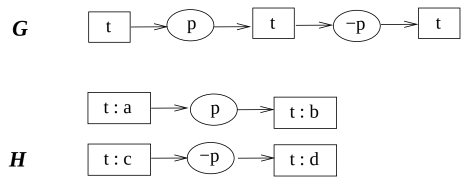


Figure 34: Atomic negation and projection (2)

In Fig. 33 the formulas assigned to G and H by Φ (we ignore here the atoms associated with concept nodes) are respectively $\Phi(G) = \exists x \exists y P(x) \wedge \neg P(y) \wedge R(x,\ y)$ and $\Phi(H) = P(a) \wedge R(a, b) \wedge R(b, c) \wedge \neg P(c)$. $\Phi(G)$ can be deduced from $\Phi(H)$ using the tautology $P(b) \vee \neg P(b)$ (indeed, every model of $\Phi(H)$ satisfies either $P(b)$ or $\neg P(b)$; if $P(b)$, then x and y are interpreted as b and c ; in the opposite case, x and y are interpreted as a and b; thus every model of $\Phi(H)$ is a model of $\Phi(G)$). In the second example, illustrated by Fig 34, the target graph is composed of two connected components, one containing positive information only, the other negative information only. There is not projection from G to H, even if $\Phi(G)$ can be deduced from $\Phi(H)$, using the tautology $(P(b, c) \vee \neg P(b, c))$.

Another example of a limited kind of negation resulting in the same negative conclusion was given in [Mug00]. The idea was to deal with difference links, expressing that two entities represent distinct entities (the contrary of a co-reference link). A difference link is logically interpreted by the atom $\neg(x = y)$, where x and y are the terms assigned to concept nodes extremities of the difference link.

Let us for the sequel of this section adopt the notations of [Ker01], who defines a *polarized* simple graph as a simple graph provided with a function from its relation nodes to $\{+, -\}$.

Despite previous negative result, Kerdiles identifies a restricted form of polarized graphs for which projection is complete. Starting from the observation that projection is complete if the source graph contains positive relation nodes only, or negative relation nodes only, he defines a *discriminated* graph as a polarized graph in which no relation node share a neighbor with a relation node of opposite sign. In other words, in a discriminated graph, positive and negative relation nodes are distributed into distinct connected components. The associated formula can be seen as a conjunction of two existentially closed formulas, one with positive relation atoms only, the other one with negative relation atoms only. Since an individual node can be split without changing the logical semantics of the graph, a polarized graph can be transformed into a discriminated graph iff all paths between two relation nodes of opposite sign contain an individual node.

In order to state the completeness result, it remains to deal with contradictory information. First see that the formula associated with a polarized graph is absurd (inconsistent) if and only if the graph under normal form contains two relation nodes with opposite sign, say $r_1 = (+r)$ and $r_2 = (-s)$, with exactly the same i-th neighbors, and $r_1 \leq r_2$. In this case, the graph is said to be absurd. Kerdiles defines a special graph, called *the* absurd graph, into which all graphs project, and the normal form of an absurd graph is the absurd graph.

Theorem 21 (Projection completeness for discriminated SGs) *[Ker01] For a discriminated graph G and a polarized graph in normal form H, $\Phi(\mathcal{S})$, $\Phi(H) \vDash \Phi(G)$ if and only if there is a projection from G to H.*

Finally, let us note that [Ker01] extends the complexity result obtained on guarded simple graphs (for which deduction is polynomial, see section 2.7.3) to guarded discriminated polarized graphs: given two polarized graphs G and H, one can check in polynomial time whether G is deducible from H if G is guarded and discriminated.

5.4 The loosely guarded fragment

Let us conclude this tour of relationships between conceptual graphs and FOL, with a fragment of FOL introduced in [vB97], the *(loosely) guarded fragment*. Deduction in this fragment is decidable. This fragment turns out to have interesting properties (such as the finite model property and others [AvBN96]). It contains the FOL translation of many modal logics and description logics.

The idea of the guarded fragment is to restrict the use of quantifiers: a quantifier can range over a tuple of variables but it is "guarded" by an atom α that contains all free variables of the subformula it quantifies. The loosely guarded fragment generalizes it by authorizing conjunction of atoms of a certain form to act conjointly as a guard.

The guarded fragment of FOL, in short gFOL, is inductively defined as follows:

1. every atomic formula belongs to gFOL (equality included)

2. gFOL is closed under the boolean connectives

3. if α is an atom, called a guard, and ϕ is a formula of gFOL, such that

(i) $free(\phi) \subseteq free(\alpha) = var(\alpha)$
– where var (resp. $free$) denotes the set of variables (resp.free variables) of a formula –
(ii) $X \subseteq free(\alpha)$

then:
$\exists X(\alpha \wedge \phi)$
$\forall X(\alpha \rightarrow \phi)$
are in gFOL

In the loosely guarded fragment of FOL (in short lgFOL), the condition (3) of previous definition is replaced by:

if α_1, ... α_m are atoms and ϕ is a loosely guarded formula of FOL, such that

(i) $free(\phi) \subseteq \bigcup_{i:1...m} free(\alpha_i)$
(ii) $X \subseteq \bigcup_{i:1...m} free(\alpha_i)$
(iii) for every variable x_i of X and every other variable z of $\bigcup_i free(\alpha_i)$ there is at least one atom α_j that contains both x_i and z

then:
$\exists X(\alpha_1 \wedge \ ... \ \wedge \alpha_m \wedge \phi)$
$\forall X(\alpha_1 \wedge \ ... \ \wedge \alpha_m \rightarrow \phi)$
are loosely guarded formulas of FOL.

In [BMT99] the authors identify a restricted form of full conceptual graphs corresponding to lgFOL: more precisely, for every such graph G (say l.g. graph), $\Phi(G)$ is equivalent to a closed lgFOL formula (but is not necessarily an lgFOL formula); and for every closed lgFOL formula f, there is an l.g. graph G such that $\Phi(G)$ is equivalent to f. One can polynomially check if a conceptual graph is loosely guarded (the graph has to satisfy some syntactic restrictions), and compute the lgFOL formula equivalent to G and reciprocally.

The next step would be to build a sound and complete proof procedure for loosely guarded graphs, while trying to exploit graph features of loosely guarded CGs.

Notice that deduction in the loosely guarded fragment does not include deduction in the $\mathcal{SR}$ model, even if rules are restricted to the range restricted rules. For instance the following formula that asserts the transitivity of a binary relation R is not a (loosely) guarded formula and has no equivalent formula in the loosely guarded fragment [Grä99]: $\forall x \forall y \forall z R(x,\ y) \wedge R(y,\ z) \rightarrow R(x,\ z)$[28].

6 Ongoing work: combining rules and constraints

In [BM02] several ways of combining rules and constraints are introduced. The first one, yielding the model $\mathcal{SRC}$, consists in using constraints for evaluating the validity of a base composed of facts *and* rules. The second one, yielding the model $\mathcal{SEC}$, gives another semantics to rules: rules represent possible transitions from a world to other worlds.

6.1 The $\mathcal{SRC}$ model

Inference rules (as in Figure 24) represent implicit knowledge that is made explicit by rule applications. Facts and inference rules can be seen as describing a world, and a rule application modifies the explicit description of the world (the facts actually). Now, if we consider a KB composed of a set of facts $\mathcal{G}$, a set of inference rules $\mathcal{R}$, and a set of constraints $\mathcal{C}$, the notion of consistency has to take rules into account. For instance, add to the SGs G and H of Figure 25 the following information about office assignments: `[HeadOfgroup:L.]->(in)->[Office:#1]`, `[Secretary:H.]->(in)->[Office:#2]` and `[Secretary:P.]->(in)->[Office:#3]`. Let G' and H' be the (normal) SGs obtained. Consider the world composed of the SG G', rules $\{R_1, R_2\}$ of Figure 24, and the positive constraint C_2 of Figure 29. The SG G' alone does not satisfy the constraint $C2$ (because "the head of group K. is in office #1, and the secretary P. is in office #3", but it does not hold that " #1 is near #3"). But after a certain number of rule applications, it does. Thus the KB is said to be consistent. In this case it is easy to define and check consistency because the world description can be made completely explicit by a finite SG (the graph H', said to be *full* w.r.t. $\mathcal{R}$), thus it suffices to check whether this graph is consistent. In the general case, consistency relies on whether each "constraint violation" can be repaired by rule applications, as formally defined below. As in sim-

[28]The loosely guarded-fragment does not seem to include FOL($\exists$, $\wedge$) either, i.e. the logical fragment corresponding to SGs. E.g. the formula $\exists x \exists y \exists z \exists u R(x,\ y) \wedge R(y,\ z) \wedge R(z,\ u) \wedge R(u,\ x)$ is not loosely guarded and it does not seem to be equivalent to a loosely guarded formula. Notice however that there is a (polynomial) reduction from "Projection" to "Projection with loosely guarded simple graphs", thus, even if the SG fragment seems to be more expressive than the loosely guarded SG fragment, the deduction problems in both fragments are actually equivalent.

pler worlds described by facts only, deduction is not possible on inconsistent knowledge bases.

Let us now give precise definitions of the $\mathcal{SRC}$ model. Applying a rule to $\mathcal{G}$ can create inconsistency, but a further application of a rule may restore consistency. Let us formalize this notion of *consistency restoration*. Suppose there is a π-violation of a positive constraint C in $\mathcal{G}$; this violation (C, π) is said to be $\mathcal{R}$*-restorable* if there exists an $\mathcal{R}$-derivation from $\mathcal{G}$ to a SG G' and a projection π' from G' to $\mathrm{irr}(G')$ such that the projection $\pi' \circ \pi$ of the trigger of C to $\mathrm{irr}(G')$ can be extended to a projection of C as a whole. The violation of a negative constraint can never be restored. Note that the $\mathcal{R}$-restoration can create new violations, that must themselves be proven $\mathcal{R}$-restorable.

Definition 22 (Consistency/Deduction in $\mathcal{SRC}$) *A KB $\mathcal{K} = (\mathcal{G}, \mathcal{R}, \mathcal{C})$ is* consistent *if, for any SG G' that can be $\mathcal{R}$-derived from $\mathcal{G}$, for every constraint $C \in \mathcal{C}$, for every π-violation of C in G', (C, π) is $\mathcal{R}$-restorable. A SG Q can be deduced from $\mathcal{K}$ if $\mathcal{K}$ is consistent and Q can be deduced from $(\mathcal{G}, \mathcal{R})$.*

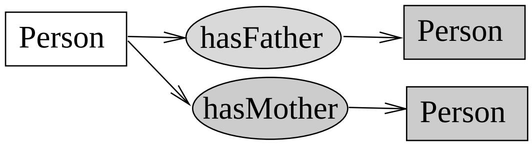
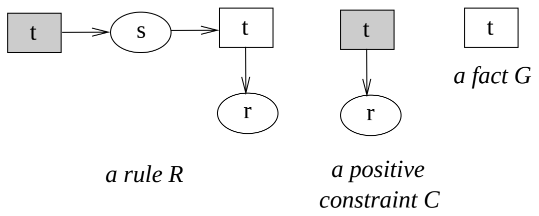


Figure 35: A rule/constraint

Let give another example (similar the the one given in [BM02] about integers). Consider a KB composed of one fact G= [Person:Mary], one rule and one constraint, both represented by the same bicolored graph shown in Figure 35. The rule says that "every person has a father and a mother", and the constraint says that it must be the case. G violates the constraint but a rule application can repair this violation, while creating another violation. Finally, every constraint violation can be repaired by a rule application, thus the base is consistent.

In order to translate the notion of $\mathcal{SRC}$ consistency into FOL, one may try to extend the result obtained for $\mathcal{SGC}$-consistency, i.e. a KB composed of facts is inconsistent if and only if there exists a SG (a FOL($\exists$, $\wedge$) formula) that can be deduced from the facts and the constraints seen as rules, but cannot be deduced from the facts alone. The next theorem shows that one direction of the above equivalence holds for a KB composed of facts *and rules*. And a counter-example then shows that the other direction does not hold.

Theorem 22 (FOL translation of $\mathcal{SRC}$ consistency) *[BM02] Let $\mathcal{K} = (\mathcal{G}, \mathcal{R}, \mathcal{C})$ be a KB. If there exists a SG G' such that $\Phi(\mathcal{S}), \Phi(\mathcal{G}), \Phi(\mathcal{R}), \Phi(\mathcal{C}) \vDash \Phi(G')$ and not $\Phi(\mathcal{S}), \Phi(\mathcal{G})\Phi(\mathcal{R}), \vDash \Phi(G')$, where $\Phi(\mathcal{C})$ is the translation of the constraints of $\mathcal{C}$ considered as a rules, then $\mathcal{K}$ is inconsistent.*



Counter-example to the converse of theorem 22

Figure 36: Counter-example

Property 19 *The converse of theorem 22 is false, as shown in Figure 36.*

In Fig. 36, check that every graph that can be $\{R,\ C\}$-deduced from G can also be $\{R\}$-deduced from G. However the base is inconsistent: G violates the constraint C and this violation cannot be restored.

Notice that when the set of rules is restricted to a finite expansion set (see definition 16), $\mathcal{SRC}$ consistency can be translated into FOL in the same way as $\mathcal{SGC}$ consistency, i.e. in this particular case the converse of theorem 22 holds.

Theorem 23 (FOL translation of $\mathcal{SRC}$ consistency with f.e.s. of rules) *[BM02] Let $\mathcal{K} = (\mathcal{G}, \mathcal{R}, \mathcal{C})$ be a KB where $\mathcal{R}$ is a finite expansion set. $\mathcal{K}$ is inconsistent if and only if there exists a SG G' such that $\Phi(\mathcal{S}), \Phi(\mathcal{G}), \Phi(\mathcal{R}), \Phi(\mathcal{C}) \vDash \Phi(G')$ and not $\Phi(\mathcal{S}), \Phi(\mathcal{G})\Phi(\mathcal{R}), \vDash \Phi(G')$, where $\Phi(\mathcal{C})$ is the translation of the constraints of $\mathcal{C}$ considered as rules.*

6.2 The $\mathcal{SEC}$ model

Let now define another use of rules. The objects and the application mechanisms are the same, but rules are used with another intention: instead of representing implicit knowledge about a world, they represent possible

Figure 37: An evolution rule

actions leading from one world to another one. We call them evolution rules.

E.g. consider the colored graph of Figure 37. Taken as an inference rule, it would say that all persons are in all offices. Taken as an evolution rule, it says that "when there are a person and an office, (try to) assign this office to that person". Consider a KB composed of a set of facts $\mathcal{G}$, a set of evolution rules $\mathcal{E}$, and a set of constraints $\mathcal{C}$. Facts describe an initial world; evolution rules represent possible transitions from one world to other worlds; constraints define consistency of each world; a successor of a consistent world is obtained by an evolution rule application; given a SG Q, the deduction problem asks whether there is a path of consistent worlds evolving from the initial one to a world satisfying Q.

Definition 23 (Deduction in $\mathcal{SEC}$) *Let $\mathcal{K} = (\mathcal{G}, \mathcal{E}, \mathcal{C})$ be a KB, and let Q be a SG. Q can be deduced from $\mathcal{K}$ if there is an $\mathcal{E}$-derivation $\mathcal{G} = G_0, \ldots, G_k$ such that, for $0 \leq i \leq k$, $(G_i, \mathcal{C})$ is consistent and Q can be deduced from G_k.*

In $\mathcal{SEC}$, $\mathcal{G}$ is seen as the initial world, root of a potentially infinite tree of possible worlds, and $\mathcal{E}$ describes the possible evolutions from one world to others. In $\mathcal{SRC}$, $\mathcal{G}$ provided with $\mathcal{R}$ is a finite description of a potentially infinite world, that has to be consistent.

6.3 The most general model: $\mathcal{SREC}$

The most general model of the family considers both kinds of rules, i.e. a set $\mathcal{R}$ of inference rules and a set $\mathcal{E}$ of evolution rules. In the particular case of the SISYPHUS-I modeling, $\mathcal{G}$ and $\mathcal{R}$ describe the initial information about office locations, persons and the group organization. $\mathcal{R}$ also encodes general knowledge (such as properties of the *dif* relation put between two concept nodes representing distinct entities). $\mathcal{C}$ represents obligations and interdictions defining what acceptable assignments are (including cardinality constraints such as "a person cannot be *in* several offices" or "a small office cannot contain more than one person", using the *dif* relation). $\mathcal{E}$ consists of one evolution rule trying to place a person into an office (it could also be composed of several rules considering specific preconditions before trying an assignment). The goal represents a situation where each person of the group has an office. A solution to the problem is a world obtained from the initial one by a sequence of office assignments, where each person has an office, while satisfying the allocation constraints.

Let us give a precise definition of deduction in $\mathcal{SREC}$.

Definition 24 (Deduction in $\mathcal{SREC}$) *A SG G' is an* immediate $\mathcal{RE}$-evolution *from a SG G if there exists an $\mathcal{R}$-derivation from G to a SG G'' and an* immediate $\mathcal{E}$*-derivation from G'' to a SG G'. An* $\mathcal{RE}$-evolution *from a SG G to a SG G' is a sequence of SGs $G = G_0, \ldots, G_k = G'$ such that, for $0 \leq i \leq k$, $(G_i, \mathcal{R}, \mathcal{C})$ is consistent and, for $1 \leq i \leq k$, G_i is an immediate $\mathcal{RE}$-evolution from G_{i-1}. Given a KB $\mathcal{K} = (\mathcal{G}, \mathcal{R}, \mathcal{E}, \mathcal{C})$, a SG Q can be* deduced *from $\mathcal{K}$ if there is an $\mathcal{RE}$-evolution $\mathcal{G} = G_0, \ldots, G_k$ such that Q can be deduced from $(G_k, \mathcal{R})$.*

6.4 Decidability/complexity of models combining rules and constraints

In [BM02] we did a systematic study of decidability and complexity of models of the $\mathcal{SG}$ family, including some particular cases of rules and constraints. I recalled the main results about $\mathcal{SR}$ and $\mathcal{SGC}$ in the sections devoted to these models. I will summarize below the main results about models combining rules and constraints:

Theorem 24 (Complexity of combining rules and constraints) *[BM02]*

- *$\mathcal{SRC}$-Consistency is truly undecidable (thus $\mathcal{SRC}$-Deduction also is. And since $\mathcal{SREC}$ generalizes $\mathcal{SRC}$, problems in $\mathcal{SREC}$ are truly undecidable as well).*
- *$\mathcal{SEC}$-Deduction is semi-decidable.*

We exhibited some cases in which these problems become decidable, related to finite expansion sets (see definition 16):

Property 20 (Complexity with finite expansion rule sets) *[BM02]*

- *When $\mathcal{R}$ is a f.e.s, deduction in $\mathcal{SR}$ is decidable, consistency and deduction in $\mathcal{SRC}$ are decidable, deduction in $\mathcal{SREC}$ is semi-decidable.*
- *When $\mathcal{E}$ is a f.e.s, deduction in $\mathcal{SEC}$ is decidable, but remains truly undecidable in $\mathcal{SREC}$.*

- *When $\mathcal{R} \cup \mathcal{E}$ is a f.e.s, deduction in $\mathcal{SREC}$ is decidable (but it is not if $\mathcal{R}$ and $\mathcal{E}$ are both f.e.s. but not their union)*

It is to be noticed that when rules are further restricted to range-restricted rules problems fall in the polynomial hierarchy (see summary in table 1).

6.5 The $\mathcal{SG}$ family

We can now give a general view of the so-called $\mathcal{SG}$-family. The *Deduction* problem asks, given a KB and a SG Q, whether Q can be deduced from the KB. Depending on what kinds of objects compose the KB (facts, inference rules, evolution rules, constraints), one obtains different models. Note that, in the absence of constraints, inference and evolution rules behave in the same way, thus six models are actually obtained:

- the $\mathcal{SG}$ model for a KB composed of facts: $\mathcal{K} = (\mathcal{G},\ \emptyset,\ \emptyset,\ \emptyset)$;
- the $\mathcal{SR}$ model for a KB composed of facts and rules: $\mathcal{K} = (\mathcal{G},\ \mathcal{R},\ \mathcal{E},\ \emptyset)$
- the $\mathcal{SGC}$ model for a KB composed of facts and constraints: $\mathcal{K} = (\mathcal{G},\ \emptyset,\ \emptyset,\ \mathcal{C})$
- the $\mathcal{SRC}$ model for a KB composed of facts, inference rules and constraints: $\mathcal{K} = (\mathcal{G},\ \mathcal{R},\ \emptyset,\ \mathcal{C})$
- the $\mathcal{SEC}$ model for a KB composed of facts, evolution rules and constraints: $\mathcal{K} = (\mathcal{G},\ \emptyset,\ \mathcal{E},\ \mathcal{C})$
- the $\mathcal{SREC}$ model for $\mathcal{K} = (\mathcal{G},\ \mathcal{R},\ \mathcal{E},\ \mathcal{C})$

Since a fact has the same semantics as a rule with an empty hypothesis, the set $\mathcal{G}$ is used in models names only when both rule sets $\mathcal{R}$ and $\mathcal{E}$ are empty. Table 1 summarizes the complexity results obtained for the deduction problem and the consistency problem (if defined), when considering the general case and the specific cases of range-restricted rules, negative constraints or disconnected constraints. We will not detail these results here.

Open issues. The models combining rules and constraints raise a certain number of open issues.

- How could the constraint restoration mechanism be described in logics? In what does it lead to a notion of base consistency different from notions used in logical works also combining rules and constraints(for instance [LR98])?

	General case	$\mathcal{R}$ fes	$\mathcal{E}$ fes	$\mathcal{R} \cup \mathcal{E}$ fes	$\mathcal{R}$&$\mathcal{E}$ r.r.	$\mathcal{C}^-$	$\mathcal{R}$&$\mathcal{E}$ r.r., $\mathcal{C}^-$	$\mathcal{R}$&$\mathcal{E}$ r.r., $\mathcal{C}^d$
$\mathcal{SG}$-DED.	NP-C	/	/	/	/	/	/	/
$\mathcal{SGC}$-CONS.	Π_2^P-C	/	/	/	/	co-NP-C	co-NP-C	DP-C
$\mathcal{SGC}$-DED.	Π_2^P-C	/	/	/	/	DP-C	DP-C	DP-C
$\mathcal{SR}$-DED.	semi-dec.	dec.	/	dec.	NP-C	/	NP-C	NP-C
$\mathcal{SRC}$-CONS.	undec.	dec.	/	dec.	Π_2^P-C	co-semi-dec.	co-NP-C	DP-C
$\mathcal{SRC}$-DED.	undec.	dec.	/	dec.	Π_2^P-C	undec.	DP-C	DP-C
$\mathcal{SEC}$-DED.	semi-dec.	/	dec.	dec.	Σ_3^P-C	semi-dec.	Σ_2^P-C	Σ_2^P-C
$\mathcal{SREC}$-DED.	undec.	semi-dec.	undec.	dec.	Σ_3^P-C	undec.	Σ_2^P-C	Σ_2^P-C

Notations: fes (finite expansion set), *r.r.* (set of range-restricted rules), $\mathcal{C}^-$ (set of negative constraints), $\mathcal{C}^d$ (set of disconnected constraints)

Table 1: Summary of Complexity Results

- Evolution rules define actions enabling to transit from one world to others. Combined with constraints they could lead to a declarative programming language. One open issue is to translate the mechanism into logics. Another issue is to generalize evolution rules to transformation rules, i.e. rules which not only add information but also remove information.

These models are issued from recent work and need both to be studied from a theoretical viewpoint, specially with respect to their logical translation and their expressiveness, and from a practical viewpoint.

Acknowledgments. I would like to thank Michel Chein, Roland Ducournau and Anne Preller for their comments about this paper.

7 Appendix: Classical definitions about graphs and orders

Recalls and notations about orders. We shall recall here some basic definitions about orders. Let E be a set. A binary relation on E is a *preorder* (or a *quasi-order*) if it is reflexive and transitive. It is an *order* if it is also anti-symmetrical. We denote by $\leq$ any preorder or order. If needed, we index $\leq$ by the name of the set on which it is defined, for instance $\leq_E$. Given an order (or a preorder) $\leq$, we denote by $\geq$ the dual order (or preorder): for elements a and b, $a \leq b$ holds if and only if $b \geq a$ holds. In this paper the intuitive meaning of $a \leq b$ (resp. $b \geq a$) is always a is a *specialization* (resp. *generalization*) of b. Given an order, we note $a < b$ if $a \leq b$ and $a \neq b$. Given a preorder, if $a \leq b$ and $b \leq a$, a and b are *equivalent*; and we note $a < b$ if $a \leq b$ and a and b are not equivalent. Two elements a and b are said to be *comparable* if $a \leq b$ or $b \leq a$. Otherwise, they are *incomparable*.

Let us now consider orders only. A *partial* order is an order such that not all elements are necessarily comparable. b *covers* a if $a \leq b$ and there does not exist an element c, distinct of a and b, such that $a \leq c$ and $c \leq b$. Given a pair $\{a,\ b\}$, c is the *lowest greater bound* of $\{a,\ b\}$, if $a \leq c$ and $b \leq c$ and for all c' such that $a \leq c'$ and $b \leq c'$, $c < c'$. The *greatest lower bound* is defined in a dual way. An order on E is a *sup-semi-lattice* if every pair of elements of E has a lowest greater bound. An *inf-semi-lattice* is defined in a dual way. An order on E is a *lattice* if it is a sup-semi-lattice and an inf-semi-lattice. Notice that every sup-semi-lattice with a smallest element (an element e such that $\forall a \in E, e \leq a$) is a lattice. Dually, every inf-semi-lattice with a greatest element is a lattice.

Recalls about graphs (to be completed if necessary). By *path*, we mean a path that does not go twice through the same node. Such a path is sometimes called an elementary path.

References

[ACP87] S. Arnborg, D.G. Corneil, and Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8:277–284, 1987.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AvBN96] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of FOL. Research Report ML-96-03, Univ. of Amsterdam, 1996.

[Baa99] F. Baader. Logic-based knowledge representation. In M.J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, number 1600 in Lecture Notes in Computer Science, pages 13–41. Springer Verlag, 1999.

[Bag99] J.-F. Baget. A Simulation of Co-Identity with Rules in Simple and Nested Graphs. In *Proc. of ICCS'99*, pages 442–455. Springer, 1999.

[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.

[BBV97] C. Bos, B. Botella, and P. Vanheeghe. Modeling and Simulating Human Behaviors with Conceptual Graphs. In *Proc. of ICCS'97, LNAI 1257*, pages 275–289. Springer, 1997.

[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.

[BM02] J.-F. Baget and M.-L. Mugnier. The complexity of rules and constraints. *JAIR*, 16:425–465, 2002. http://www.jair.org/abstracts/baget02a.html.

[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.

[Bod93] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1–2):1–21, 1993.

[Bod96] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25:1305–1317, 1996.

[Bor96] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial intelligence*, 82:353–367, 1996.

[BP98] C. Bos-Plachez. *Modéliser pour simuler pour modéliser. Contribution à l'acquisition de connaissances comportementales.* PhD thesis, Université des Sciences et Technologies de Lille, 1998.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[BT01] J.-F. Baget and Y. Tognetti. Backtracking through biconnected components of a constraint graph. In *Proc. of IJCAI'01*, volume 1, pages 291–296, 2001.

[BV84] C. Beeri and M.Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[CC98] T. H. Cao and P. N. Creasy. Fuzzy order-sorted logic programming in conceptual graphs with a sound and complete proof procedure. In *ICCS*, volume 1453 of *Lecture Notes in Computer Science*, pages 270–286. Springer, 1998.

[CG95] O. Cogis and O. Guinaldo. A linear descriptor for conceptual graphs and a class for polynomial isomorphism test. In *Proc. of ICCS'95, LNAI 954*, pages 263–277. Springer, 1995.

[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/.

[CM97] M. Chein and M.-L. Mugnier. Positive Nested Conceptual Graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, ICCS'97 Proc., LNAI 1257*, pages 95–109. Springer Verlag, 1997.

[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/.

[CR97] Ch. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. In *Proc. of ICDT'97, LNCS, vol. 1186*, pages 56–70. Springer, 1997.

[CS98] S. Coulondre and E. Salvat. Piece Resolution: Towards Larger Perspectives. In *Proc. of ICCS'98, LNAI 1453*, pages 179–193. Springer, 1998.

[DHL98] J. Dibie, O. Haemmerlé, and S. Loiseau. A Semantic Validation of Conceptual Graphs. In *Proc. of ICCS'98, LNAI 1453*, pages 80–93. Springer, 1998.

[Dic94] J.P. Dick. Using Contexts to Represent Text. In *Lecture Notes in AI, 835, Proc. ICCS'94*, pages 196–213, 1994.

[FFG01] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree decomposition. *ICDT'01*, 1973:22–38, 2001.

[FLS96] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proc. of AAAI'96*, pages 175–180, 1996.

[FV93] T. Feder and M.Y. Vardi. Monotone Monadic SNP and Constraint Satisfaction. In *Proceedings of the 25th ACM STOC*, pages 612–622, 1993.

[Gen00] D. Genest. *Extension du modèle des graphes conceptuels pour la recherche d'informations*. PhD thesis, Université Montpellier II, Dec. 2000.

[GLS99] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *PODS'99*, 1999.

[GLS01a] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: A survey. *MFCS'01*, 2136:37–57, 2001.

[GLS01b] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals and guards: Game-theoretic and logical characterizations of hypertree width. *PODS'01*, 2001.

[Grä99] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 1999.

[Guh91] R.V. Guha. Contexts: a formalization and some applications. PhD Thesis, Stanford University , 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.

[GW96] B.C. Ghosh and V. Wuvongse. Computational Situation Theory in the Conceptual Graph Language. In *Proceedings of ICCS'96*, Lecture Notes in AI, 1115. Springer, 1996.

[HM01] G. Hahn and G. MacGillivray. Graph homomorphisms:computational aspects and infinite graphs. Draft, 2001.

[HN90] P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. Combin. Th.*, (B)(48):33–42, 1990.

[HNZ94] P. Hell, J. Nešetřil, and X. Zhu. Duality and Polynomial Testing of Tree Homomorphisms. Manuscript, 1994.

[HT97] G. Hahn and C. Tardif. Graph homomorphism: structure and symmetry. In G. Hahn and G.Sabidussi, editors, *Graph Symmetry*, pages 107–166. Kluwer, 1997.

[Jac88] M. K. Jackman. Inference and the Conceptual Graph Knowledge Representation Language. In *Research and Development in Expert Systems IV*, 1988.

[KBvH01] A.M.C.A. Koster, H.L. Bodlaender, and S.P.M. van Hoesel. Treewidth:computational experiments. Draft, 2001.

[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/.

[Kon96] K. Konolige. Abductive Theories in Artificial Intelligence. In *Principles of Knowledge Representation*, pages 129–152. CSLI Publications, 1996.

[KV98] P. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. In *Proceedings of PODS'98*, 1998.

[KV00] P. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.

[LB94] M. Liquière and O. Brissac. A class of conceptual graphs with polynomial isoprojection. In *Supplemental proceedings of the International Conference on Conceptual Structures*, 1994. Revised version.

[Lec95] M. Leclère. *Les connaissances du niveau terminologique du modele des graphes conceptuels: construction et exploitation.* PhD thesis, Université Montpellier II, Dec. 1995.

[Lec96] M. Leclère. C-chic: Construction coopérative de hiérarchies de catégories. *Revue d'Intelligence Artificielle*, 10:57–100, 1996.

[Lec97] M. Leclère. Reasoning with type definitions. In *Proc. of ICCS'97, LNAI 1257*, pages 401–415. Springer, 1997.

[Lec98] M. Leclère. Raisonner avec des définitions de types dans le modèle des graphes conceptuels. *Revue d'Intelligence Artificielle*, 12:243–278, 1998.

[Leh92] F. Lehman, editor. *Semantics Networks in Artificial Intelligence.* Pergamon Press, 1992.

[LR96] A. Y. Levy and M.-C. Rousset. Verification of Knowledge Bases Based on Containment Checking. In *Proc. of AAAI'96*, pages 585–591, 1996.

[LR98] A. Y. Levy and M.-C. Rousset. Verification of Knowledge Bases Based on Containment Checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[LS98] M. Liquière and J. Sallantin. Structural machine learning with galois lattice and graphs. In *Proc. of ICML'98*, 1998.

[Mar97] P. Martin. CGKAT: a knowledge acquisition tool and an information retrieval tool which exploits structured documents, conceptual graphs and ontologies. In *Proc. of CGTOOLS'97*. Univ. of Washington, 1997.

[MC92] M.L. Mugnier and M. Chein. Polynomial algorithms for projection and matching. In *Proceedings of the 7th Workshop on Conceptual Graphs*, New Mexico State University, Las Cruces, New Mexico, 1992.

[MC93] M.L. Mugnier and M. Chein. Characterization and Algorithmic Recognition of Canonical Conceptual Graphs. In *LNAI 699, Proc. ICCS'93*, pages 294–311. Springer Verlag, 1993.

[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/.

[McC93] J. McCarthy. Notes on Formalizing Contexts. In *Proc. IJCAI'93*, pages 555–560, 1993.

[Min75] M. Minsky. A framework for representing knowledge. In P.Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[MM97] G. W. Mineau and R. Missaoui. The Representation of Semantic Constraints in Conceptual Graphs Systems. In *Proc. of ICCS'97, LNAI 1257*, pages 138–152. Springer, 1997.

[Mug92] M.L. Mugnier. *Contributions algorithmiques pour les graphes d'héritage et les graphes conceptuels*. PhD thesis, Université Montpellier II, Oct. 1992.

[Mug95] M.-L. Mugnier. On Generalization / Specialization for Conceptual Graphs. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:325–344, 1995. available at http://www.lirmm.fr/˜mugnier/.

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

[Naz93] A. Nazarenko. Representing Natural Language Causality in Conceptual Graphs . In *Lecture Notes in AI, 699, Proc. ICCS'93*, pages 205–222, 1993.

[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.

[Ohr97] P. Ohrstrom. C.S. Peirce and the Quest for Gamma Graphs. http://www.hum.auc.dk/˜poe/ARTIKLER/GammaHTML/GammaGraphs.html, 1997.

[PMC98] A. Preller, M.-L. Mugnier, and M. Chein. Logic for nested graphs. *Computational Intelligence: An International Journal*, 14-3:335–357, 1998.

[Qui68] M. Quillian. Semantic memory. In M. Minsky, editor, *Semantic information processing*, pages 216–270. MIT Press, Cambridge, 1968.

[Rob92] D.D. Roberts. The Existential Graphs. In F. Lehman, editor, *Semantics Networks in Artificial Intelligence*, pages 639–663. Pergamon Press, 1992.

[RS86] N. Robertson and P.D. Seymour. Graph minors ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[Sal93] E. Salvat. Algorithmes de projection de graphes conceptuels. Master's thesis, Université Montpellier II, 1993. DEA.

[Sal97] E. Salvat. *Raisonner avec des opérations de graphes: graphes conceptuels et règles d'inférence.* PhD thesis, Université Montpellier II, 1997.

[Sal98] E. Salvat. Theorem proving using graph operations in the conceptual graphs formalism. In *Proc. of ECAI'98*, pages 356–360, 1998.

[SCM98] G. Simonet, M. Chein, and M.-L. Mugnier. Projection in Conceptual Graphs and Query Containment in nr-Datalog. R.R. LIRMM, 1998.

[Sha79] S. C. Shapiro. The sneps semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, 1979.

[Sha00] S. C. Shapiro. An Introduction to SNePS 3. In *Proc. of ICCS'00*, pages 510–524. Springer, 2000.

[Sim98] G. Simonet. Two FOL Semantics for Simple and Nested Conceptual Graphs. In *Proc. of ICCS'98, LNAI 1453*, pages 240–254. Springer, 1998.

[SM96] E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *Proc. of ICCS'96, LNAI 1115*, pages 248–262. Springer, 1996.

[Smu68] R. M. Smullyan. *First-Order Logic.* Springer Verlag, Berlin, 1968.

[Sow] J. F. Sowa. Semantic networks. *http://www.jfsowa.com/pubs/semnet.htm.*

[Sow76] J. F. Sowa. Conceptual Graphs. *IBM Journal of Research and Development*, 1976.

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

[Sow94] J. F. Sowa. Logical Foundations for Representing Object-Oriented Systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 1994.

[Thu14] A. Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skr. Viedensk. Selsk. I*, 10, 1914.

[TY84] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13(3):566–579, 1984.

[Var84] M.Y. Vardi. The implication and finite implication problems for typed template dependencies. *Journal of Computer and System Science*, 28(1):3–28, 1984.

[vB97] J. van Benthem. Dynamic bits and pieces. ILLC Research Report LP-1997-01, Univ. of Amsterdam, 1997.

[Wer95a] M. Wermelinger. Conceptual Graphs and First-Order Logic. In *Proc. of ICCS'95, LNAI 954*, pages 323–337, 1995.

[Wer95b] M. Wermelinger. Teorica básica das estruturas conceptuais. Master's thesis, Universidade Nova de Lisboa, 1995.