Knowledge Representation and Reasonings Based on Graph Homomorphism*

M.L. Mugnier
LIRMM
161, rue Ada, 34392 Montpellier - France
mugnier@lirmm.fr
http://www.lirmm.fr/~mugnier/

Reference: Research Report LIRMM 00098 (August 2000)

Abstract

The main conceptual contribution in this paper is to present an approach to knowledge representation and reasonings based on labeled graphs and labeled graph homomorphism. Strengths and weaknesses of this graph-based approach are discussed. Main technical contributions are the followings. Fundamental results about the kernel of this approach, the so-called *simple graphs model* are synthesized. It is then shown that the basic deduction problem on simple graphs is essentially the same problem as conjunctive query containment in databases and constraint satisfaction; polynomial parsimonious transformations between these problems are exhibited. Grounded on the simple graphs model, a knowledge representation and reasoning model allowing to deal with facts, production rules, transformation rules, and constraints is presented, as an illustration of the graph-based approach.

Introduction

The main conceptual contribution in this paper is to present an approach to knowledge representation and reasonings based on labeled graphs and labeled graphs homomorphism. The kernel of this approach is the so-called *simple graphs model*, which has three essential characteristics: objects are *labeled*

*A preliminary version of this paper has been published in Proc. of ICCS-2000, B. Ganter and G.W. Mineau (eds), LNAI 1867, Springer, pp 172-192, 2000.

graphs; reasonings are based on graph operations and basically on labeled graph *homomorphism*; the model is *logically* founded. This basic model has been extended in several ways, keeping these fundamental properties.

Technical contributions in this paper are mainly the followings.

- Fundamental results about the simple graphs model are synthesized and revisited.
- It is shown that the basic deduction problem on simple graphs (comparing two graphs by subsumption) is essentially the same problem as conjunctive query containment in databases and constraint satisfaction, despite their very different formulation; polynomial parsimonious transformations between these problems are exhibited.
- Grounded on the simple graphs model, the *constrained derivation model*, which allows to deal with facts, production rules, transformation rules, and constraints, is presented.

A part of technical results provided in this paper are new, or have been published in French only. The paper is organized as follows. Next section is devoted to basic notions and results about the simple graphs model. Then, correspondences between the simple graph deduction problem and, on one hand the conjunctive query containment problem in databases and, on the other hand, the constraint satisfaction problem, are exhibited. Section 3 introduces the constrained derivation model. We finally discuss benefits and limitations of our approach.

1 The Simple Graphs Model

We assume the reader is familiar with the very basic notions about conceptual graphs, namely simple graphs (conceptual graphs without negation), their logical translation by Φ, and co-reference links [Sow84][Sow99].

1.1 Simple graphs

Basic ontological knowledge is encoded in a structure we call a **support**. We consider here a primitive version of a support with 3 components, say $S = (T_C, T_R, I)$. T_C and T_R are finite partially ordered sets and respectively denote the relation and concept type sets. T_C possesses a greatest element, noted $\top$. Relation types may have any arity greater or equal to 1, and two comparable relation types must have the same arity. I is a set of individual markers. The following partial order is defined on the set of markers $I \cup \{*\}$, where $*$ denotes the generic marker: $*$ is the greatest element and elements of I are pairwise noncomparable. $G = (R,\ C,\ E,\ l)$ denotes a **simple graph**, where R and C are the relation and concept node sets, E is the set of edges (with a total order on edges incident to a relation node), and l is the node labeling mapping. A simple graph is not necessarily connected. Also note that it may be empty. Relation node labels are partially ordered by the partial order on T_R. Concept node labels are partially ordered by the product of the partial order on T_C and the partial order on the set of markers, $I \cup \{*\}$ (a label $(t,\ m)$ is less or equal to a label (t', m') if $t \leq t'$ and $m \leq m'$).

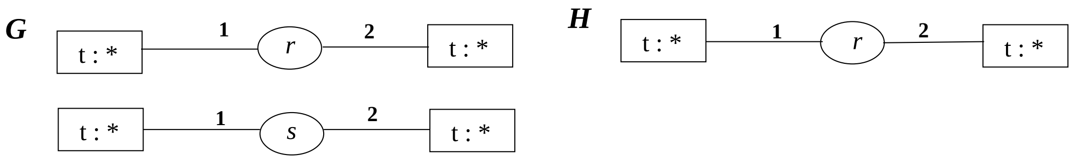
For simplicity reasons and because they do not add expressive power to simple graphs, we first exclude co-reference links from the basic model. However, we will show at the end of this section that all results extend to simple graphs with co-reference links.

1.2 The $\leq$ relation

The *specialization/generalization* relation (denoted by $\leq$ or $\geq$) — or *subsumption* relation — is the fundamental notion for reasoning with simple graphs. 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 homorphism.

In the original presentation of conceptual graphs [Sow84], simple graphs are *connected* graphs. Specialization operations (called canonical formation rules) are defined and it is proven that, given two (connected) simple graphs G and H, $H \leq G$ implies the existence of a projection from G to H. [CM92] proves the reciprocal. See also [MC93] or [Mug95] for a precise definition and a study of the notion of a specialization derivation.

In the proposed standard for conceptual graphs [Sow99], simple graphs are no more necessarily connected graphs (and may include co-reference links, but this aspect will be taken 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.



$\Phi(G)$ implies $\Phi(H)$, but there is no mean with the proposed canonical formation rules of deducing one graph from the other one.

Figure 1: H should subsume G

Indeed, the set of equivalence and specialization rules (or the set of equivalence and generalization rules) is logically *sound*, i.e. if H is obtained from G using these rules (say $H \leq G$) then $\Phi(H)$ implies $\Phi(G)$ — keeping into account $\Phi(S)$, the logical interpretation of partial orders on types in the support S. But it is *not complete*, in the sense that it does not allow to compute all logical deductions (see Figure 1).

Let us propose another set of rules, which ensures the desirable soundness and completeness properties (this set has originally been defined in [CM95]; here, some operation names have been changed in order to avoid confusion with [Sow99]). There are four specialization operations and four inverse operations, called generalization operations. In both cases, the first operation produces a graph equivalent to the original graph (where "equivalent" can be understood in the sense of the $\leq$ relation or in a logical meaning). For all operations, the resulting graph is disjoint from the operand graph(s).

Specialization operations. Let G be a simple graph; a simple graph H can be obtained from G by:

- **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).
- **Join**: merge two concept nodes of G with same label.
- **Disjoint sum**: Let K be a simple graph disjoint from G; H is obtained by juxtaposing G and K.

Generalization operations. Let G be a simple graph, a simple graph H can be obtained from G by:

- **Relation duplicate**: duplicate a relation node of G (i.e. adds 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 union of the sets of edges incident to c_1 and c_2 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* $G \geq H$.

Notice both sets of specialization and generalisation operations are *minimal* (no rule can be simulated with the other rules) and correspond to very *elementary* operations.

1.3 Projection

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

A **projection** from a simple graph $G = (R_G, C_G, E_G, l_G)$ to a simple graph $H = (R_H, C_H, E_H, l_H)$ is a (total) mapping Π from R_G to R_H and from C_G to C_H which

1. preserves adjacency and local orders on edges incident to relation nodes: for each relation node r, if the *ith* neighbor of r is a concept node c, then the *ith* neighbor of $\Pi(r)$ is $\Pi(c)$
2. may decrease labels: $\forall x \in R_G \cup C_G \; l_G(x) \;\geq\; l_H(\Pi(x))$.

Note that projection is generally not a one-to-one mapping ($\Pi(G)$ is generally not isomorphic to G).

Projection is essentially a labeled graph homomorphism. Let us recall that a *graph homorphism* (or morphism) 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 that preserves node adjacency, i.e. if xy is an edge of E_G, then $\Pi(x)\Pi(y)$ is an edge of E_H. A *labeled graph homomorphism* is a graph homomorphism which in addition preserves labels (on nodes, and edges if the graph is labeled on both). A *projection* is a generalization of a labeled graph morphism; 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 $\Pi(x)$. When the partial order is trivial (all labels are noncomparable), one comes back to a classical labeled graph homomorphism. Note that this definition ensures that the node bipartition and the ordering on edges incident to a relation node are preserved.

Theorem 1 *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] and [CM92] — which apply to connected simple graphs — are trivially adapted.

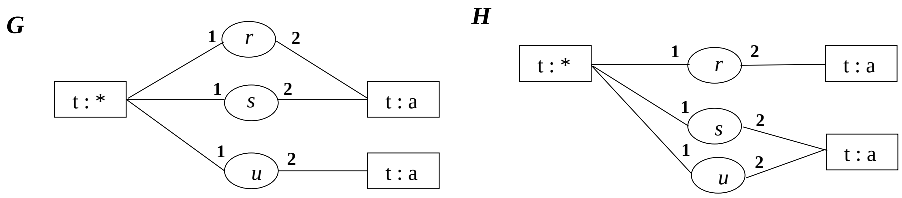
Finally, let us recall some results about the *structure* of the specialization relation:

1. $\leq$ is a preorder relation (and not an order) (e.g. [Jac88])
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 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
3. The restriction of $\leq$ to irredundant graphs is a *lattice.*

Proof. See [CM92] for property 2 and [CM95] for property 3.

1.4 Soundness and completeness of graph operations

The semantic Φ assigns a first-order formula $\Phi(G)$ to each graph G [Sow84]. $\Phi(G)$ is a positive, conjunctive and existentially closed formula. And to a support S is assigned a set of formulas, $\Phi(S)$, which corresponds to the interpretation of the partial orderings of T_R and T_C. For all types t_1 and t_2 such that t_1 covers t_2, 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.



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

G and H have same logical translation but they are noncomparable by projection.

Figure 2: The need for normality

Theorem 2 *(Soundness) Given two simple graphs G and H defined on S, if G projects to H then $\Phi(S), \Phi(H) \vDash \Phi(G)$.*

Proof. Similar to the proof of [Sow84] for connected graphs.

Theorem 3 *(Completeness) If H is in normal form (i.e. there are no two nodes with the same individual marker) and $\Phi(S), \Phi(H) \vDash \Phi(G)$ then G projects to H.*

Proof. See [CM95]. The proof is similar to that of [CM92] for connected graphs. Notice that if H is not in normal form the result is false: see Figure 2.

The normal form of a graph is easily computable (merge all individual nodes with same marker). However, depending on constraints on types associated with individual markers, it may be impossible to compute a type corresponding to the merging of two nodes. In the simplest case, when individual markers always appear with the same type in a concept node, any graph possesses a normal form, and the logical formula assigned to the normal form of a graph is trivially equivalent to that of the graph.

Two alternative logical semantics have been proposed in [PMC98] and [CMS98] [Sim98]. [PMC98] defines a new logic, in which deduction is equivalent to projection. [CMS98] [Sim98] proposes a variant of Φ, called Ψ, for which projection is sound and complete w.r.t. FOL deduction without any restriction. These semantics rely on the same intuitive idea: if several nodes represent the same entity, they *a priori* 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.

1.5 Co-reference links

Co-reference links do not add expressive power to the simple graph model. Indeed, there are two ways of translating them in FOL. If co-reference links allowed between *generic* concept nodes only, the same variable can be associated to co-referent nodes. If co-reference links can link generic and individual nodes, the equality operator is needed to express that terms associated with these nodes are equal. In both cases, the logical formula assigned to a graph with co-reference links is equivalent to that of the graph obtained by merging all co-referent nodes (provided that these nodes have the same type)[1].

More generally, let us say that two nodes are *co-identical* if they refer to the same entity, i.e. they are generic nodes related by a co-reference link or individual nodes with the same individual marker. Formally, co-identity is an equivalence relation on concept node set of a graph, denoted by *co-ident*. The normal form of a simple graph with co-reference links is obtained by merging co-identical nodes. In other words, a graph is in normal form if the co-identity relation on its concept nodes is the identity. The logical formula associated with a graph is equivalent to that associated to its normal form.

Let us extend previous canonical formation rules in order to deal with co-reference links. 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.

- **Co-reference addition:** add a co-reference link between two generic concept nodes with same type.
- **Co-identical join:** merge two co-identical nodes of the same graph, i.e. merge two co-referent nodes or two nodes with the same individual marker.

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.

[1] Do nodes refering to the same entity necessarily have the same type? Since we do not want to tackle this question here, we consider simple ways of avoiding it: (1) an individual marker always appears with the same type; (2) coreferent nodes have the same type; (3) coreferent nodes are generic nodes (note however we could allow co-reference between individual and generic nodes with very slight changes in definitions).

- **Co-reference deletion:** delete a co-reference link.
- **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 nodes with the same label related by a co-reference link. 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.

Now, projection has to respect co-identity, i.e. two coreferent nodes have co-identical images (either the same image, or distinct images which are coreferent nodes or nodes with the same individual marker). Note that previous results about simple graphs (theorems 1, 2 and 3) extend to simple graphs with coreference links.

1.6 Note about projection and logical completeness

Our approach gives greater importance to projection than to the set of canonical formation rules; the price to pay is that graphs have to be put in normal form for ensuring logical completeness (w.r.t. Φ; let us recall that in [Sim98] an alternative semantic Ψ is proposed for which the completeness results hold without restriction). Suppose one rather aims at providing complete formation 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 in the following manner.

Property 1. *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 coreference links) G and H, H can be derived from G if and only if $\Phi(S), \Phi(H) \models \Phi(G)$.*

A similar property is obtained with the set of generalization rules added with co-identical join. These properties can be explained as follows. Let us consider the structuration of the simple graph space by the projection operation (a graph G is said to be more general than a graph H if G projects onto H). Let us consider two extremal transformations of a graph: the *normal* form (no co-identical join can be done) and the *anti-normal* form (see [GW95]: the graph is split into star graphs, composed of one relation and its neighbors, all distinct, related with co-reference links; in other words, no co-identical split can be done).

Following properties are immediate:

- The anti-normal form and the normal form of a graph are unique (up to isomorphism).
- If two graphs have the same normal form, they also have the same anti-normal form; to one normal form corresponds one anti-normal form, and reciprocally.
- G projects onto its normal form and the anti-normal form of G projects onto G
- All graphs with same anti-normal (or normal form) are logically equivalent by Φ.

If we possess both rules "co-identical join" and "co-identical split", we can search the whole space of graphs bounded by an anti-normal graph and the corresponding normal form. Note that the projection completeness theorem (theorem 3) could also 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) If G is in anti-normal form (i.e. no co-identical split can be done on G) and $\Phi(S), \Phi(H) \vDash \Phi(G)$ then G projects to H.*

To conclude this note, either one favours 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; or one favours logical completeness of the set of formation rules, and the correspondence with projection is lost.

1.7 Note about the proposed formation rules in [Sow99]

Previous rules present two advantages w.r.t. rules proposed in [Sow99]: logical completeness and simplicity. Specially, the equivalence rules of [Sow99], *copy* and *simplify*, could be decomposed into more elementary equivalence operations, respectively relation duplicate and co-identical split, and co-identical join and relation simplify (see Figure 3).

Let us consider for instance the *copy* rule definition (*simplify* is the inverse of copy): "the copy rule makes a copy of any subgraph v of u and adds it to u to form w. If c is any concept of v that has been copied from a concept d in u, then c must be a member of exactly the same coreference sets as

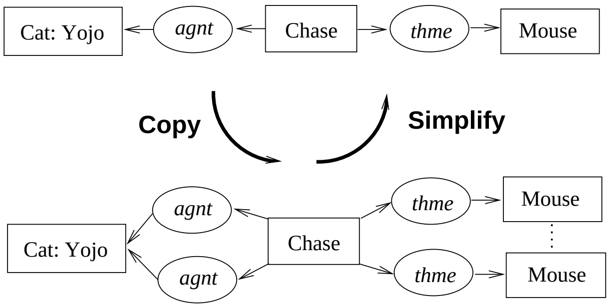


Figure 3: Proposed equivalence rules in [Sow99]

d. Some conceptual relations of v may be linked to concepts of u that are not in v; the copies of those relations must be linked to exactly the same concepts of u". Copy can be simulated by a sequence of relation duplicate (for all copied relations) followed by a sequence of co-identical splits (for all copied concepts).

Let us now come back to simple graphs without co-reference links, and to the basic deduction problem: given two simple graphs G and H, is there a projection from G to H? The next section shows that this problem is essentially the same as two other basic problems in databases and artificial intelligence.

2 Projection, Conjunctive-Query Containment and Constraint Satisfaction

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. **Constraint satisfaction problems** are a class of combinatorial problems widely investigated in artificial intelligence. What do these problems have in common, and what do they have in common with projection between simple conceptual graphs? This section shows that the three problems are essentially the same problem, even if they are formulated in very different ways.

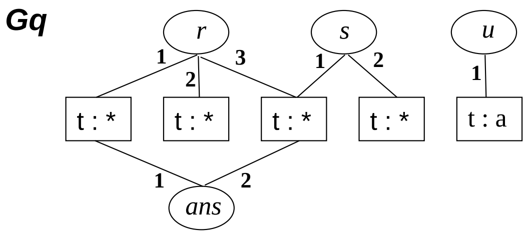
The key notion explaining correspondences between these problems is that of a labeled graph homomorphism. Let us recall that a labeled graph homomorphism from a labeled graph G to a labeled graph H is a mapping from G nodes to H nodes that preserves edges and labels. We will see that the three problems 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). 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 naturally formulated as a labeled graph of the same size (which in fact can be seen as a simple conceptual graph).

Let us call a **simplified support** a support $S = (T_C, T_R, I)$ where T_C and T_R are provided with the trivial partial order, i.e. all elements of T_C and T_R are non comparable. For establishing equivalencies between problems we will consider simplified supports; at the end of this section, we will reconsider the introduction of partial orders on types. The **Projection** problem is given by a simplified support S, and two simple graphs in normal form G and H relative to S, and asks if there is a projection from G to H. As already noticed, since S is a simplified support, the question comes down to ask wether there is a labeled graph homomorphism from G to H.

This section first presents polynomial transformations between Projection and Conjunctive query containment, then polynomial transformations between Projection and Constraint satisfaction. All mappings are parsimonious transformations (they preserve the set of solutions). Thus, algorithms for building a solution or for enumerating all solutions can be directly transferred from one domain to the other one.

2.1 Projection and Conjunctive-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-adic 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 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 $\{1, \ldots, n\}$,



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

Figure 4: Transformation from a query to a graph

$\mu(u_j) \in \alpha(r_j)$. 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 **Query containment** problem is the associated problem: given two queries q and q', does q contain q'? It is well-known in databases that the query containment problem can be reformulated as a homomorphism problem, where a homomorphism is defined as follows. A **homomorphism** from $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n)$ to $q' = ans'(u') \leftarrow r'_1(u'_1),\ ...\ r'_{n'}(u'_{n'})$ 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'.

2.1.1 Transformation from Query containment to Projection (Q2P).

Let $S = (T_C,\ T_R,\ I)$ be a simplified support built as follows. T_C is restricted to a single type. T_R is equal to $R \cup Ans$, where $Ans = \{ans_k|\ k \geq 1\}$ and k stands for the arity of ans. I is equal to dom. Elements of T_R are noncomparable. Then a simple graph (in normal form) $G_q = (R_q, C_q, E_q, l_q)$ can be assigned to q as follows (see Figure 4 for an example). R_q is a set of (n+1) nodes labeled by $r_1,\ ...,\ r_n,\ ans_k$. C_q is in bijection with the set of terms occurring in $u,\ u_1,\ ...,\ u_n$. The node of C_q assigned to a term e is generic if e is a variable and otherwise individual with the referent e. And if e is the i-th argument of r_j (or ans), then the concept node assigned to e is the i-th neighbor of the relation node assigned to r_j (or ans). Then, one can immediately check that, given two queries, q and q', every homomorphism from q to q' is a projection from G_q to $G_{q'}$, and reciprocally.

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 equivalencies.

2.1.2 Transformation from Projection to Query containment (P2Q).

A simplified support $S = (T_C,\ T_R,\ I)$ naturally induces a database; the database relation names are given by the union of T_C and T_R (the arity of an element of T_C is 1), and *dom* is a countable infinite set which contains the individual marker set I. A query q_G is built from a normal simple graph G in a manner similar to $\Phi(G)$; more specifically, $q_G = ans() \leftarrow r_1(u_1)\ ...\ r_n(u_n)$, where $r_1(u_1)\ ...\ r_n(u_n)$ are the atoms of $\Phi(G)$. Given two normal simple graphs G and H, it can be easily checked that any homomorphism from q_G to q_H is a projection from G onto H, and reciprocally.

Theorem 5 *The problems* Conjunctive query containment *and* Projection *are polynomially equivalent; furthermore by both transformations Q2P and P2Q, every solution to one problem is a solution to the other one.*

2.2 Projection and Constraint Satisfaction

The input of a constraint satisfaction problem (CSP) 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 5):

- a set of variables, $X = \{x_1,\ \dots\ , x_n\}$
- a set of constraints, $C = \{C_1,\ \dots\ , C_p\} \subset \mathcal{P}(X)$,
  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 an hypergraph.
- a set of variable domains, $D = D_1 \cup \dots \cup D_n$, where D_i is the domain of variable x_i.
- a set of constraint definitions, $R = \{R_1,\ \dots\ , R_p\}$; for every constraint C_i, let $(x_{i_1} \dots x_{i_q})$ be any ordering of C_i variables, then $R_i \subset D_{i_1} \times \dots \times D_{i_q}$.

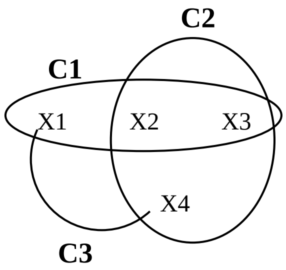
A *solution* to P is an assignment of values to the n variables such that all constraints are satisfied, more formally:

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

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

2.2.1 Transformation from Constraint Satisfaction to Projection (C2P).

A CSP can be recast as a Projection problem by the following transformation, say $C2P$ (see Figure 6). Consider a constraint network $P = (X,\ C,\ D,\ R)$. P is transformed into two simple graphs 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 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



R1

X1	X2	X3
a	a	b
a	b	a
b	a	a

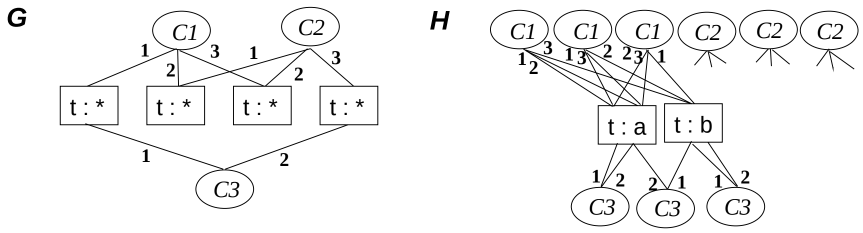
R2

X2	X3	X4
a	b	a
b	a	a
b	a	b

R3

X1	X4
a	a
b	a
b	b

Figure 5: An instance of a CSP



G and H are obtained from P of Fig. 5 by the transformation C2P.

Figure 6: Transformation from CSP to Projection

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 contraints (maps relation nodes of G onto relation nodes of H), i.e. a projection from G to H.

More specifically, a simplified support $S = (T_C, T_R, I)$ is built from P as follows: T_C is restricted to one element, say $\top$, T_R is equal to C, each relation type having the same arity as the corresponding constraint, and I is equal to D. G and H are then defined as follows. The concept node set of G is equal to X. Each concept node is labeled by $(\top,\ *)$. There is one relation node for each constraint, labeled by the corresponding relation type. The i-th neighbor of a relation node is the concept node corresponding to the i-th variable of the constraint. The concept node set of H is D. Each concept node is labeled by $(\top,\ a)$, where a is the associated value of D. There is one relation node for each tuple of a constraint definition; this relation node is labeled by the name of the constraint, and its i-th neighbor is the concept node corresponding to the i-th value. Notice that both graphs G and H are in normal form. It can be easily checked that any solution to the CSP is a projection from G to H, and reciprocally.

2.2.2 Transformation from Projection to Constraint Satisfaction (P2C)

In turn, a Projection problem can be recast as a CSP by the following transformation, say *P2C*. There is one variable for each concept node of G, and one constraint for each relation node of G. The set D of domain values is built from the concept node labels of H: there is one value for each distinct label in H. The domain D_i of a variable x_i is the subset of D to *a priori* possible images for the concept node c_i by a projection from G to H, i.e. $D_i = \{l = (t, m) \ s.t. \ label(c_i) \geq l\}$. The definition of a constraint C_i coming from a relation node r_i is the set of tuples given by the relation nodes of H with same label as r_i, i.e. if C_i is a constraint over variables $(x_{i_1} \ ... \ x_{i_q})$, then its definition is $R_i = \{(l_{i_1} \ ... \ l_{i_q})$ where $l_{i_j} = label(c_{i_j})$ s.t. $c_{i_1} \ ... \ c_{i_q}$ are neighbors, in this order, of a relation node r in H with $label(r) = label(r_i)\}$. It can be easily checked that any projection from G to H is a solution to the CSP, and reciprocally.

Theorem 6 *The problems* CSP *and* Projection *are polynomially equivalent; furthermore by both transformations C2P and P2C, every solution to one problem is a solution to the other one.*

Note about general supports. What if we consider simple conceptual graphs on general supports instead of simplified supports? The translations C2P and Q2P from CSP and Conjunctive query containment to Projection still hold, since a simplified support is a support. The translation P2C can be adapted in a straightforward way: the tuples of the definition of a constraint C_i coming from a relation node r_i of G are given by relation nodes of H with a label *less or equal* to the label of r_i (instead of *equal to*). For P2Q, G and H on a support S can be translated into G' and H' on a simplified support S' such that every projection from G to H is a projection from G' to H', and reciprocally. See for instance [Bag99]; the size of the new input $(S', \ G', \ H')$ is linear in the size of the old input $(S, \ G, \ H)$.

Relationships with published works. The translation Q2P has been published in [CMS98]. A first version of the correspondences C2P and P2C can be found in [CM95] (not published) or in [MC96] (in French). In these previous versions, correspondences are done between labeled graph morphism and binary CSP, but they rely on the same ideas. Independently, in [KV98] it is shown that conjunctive query containment and CSP are essentially the same problem because they can be recast as a relational structure homomorphism problem.

3 The constrained derivation model

It would be out of the scope of this paper to present in detail this extension to the simple graph model (see [BGM99]). The aim of the following presentation is rather to illustrate our approach. The constrained derivation model is composed of three kinds of objects: simple graphs, rules and constraints. Simple graphs are the basic constructs from which rules and constraints are defined. Projection is the basic operation from which reasonings with rules and constraints are defined.

- In this model, a *simple graph* is interpreted as a fact or a query.
- A *rule* allows one to add new knowledge. It is composed of an hypothesis and a conclusion, and is used in the following classical way: given a simple graph, if the hypothesis of the rule projects to the graph, then the information contained in the conclusion is added to the graph. Rules are split into *production rules* and *transformation rules*.
- A *constraint* defines conditions for a simple graph to be valid. It is composed of a condition part and a mandatory part. Roughly said, a graph satisfies a constraint if for every projection of its condition part, its mandatory part also projects to the graph.

The deduction problem in this framework can be outlined as follows: given a *knowledge base* (composed of facts, production rules, transformation rules and constraints) and a *query*, is it possible for this knowledge base to *evolve* in such a way that the query is satisfied? More specifically, facts describe an initial world. Production rules complete world descriptions. Transformation rules define possible transitions from a world to other ones. Constraints are used to check validity of worlds. A successor of a valid world is obtained by a single application of a transformation rule on this world. Solving the problem is to find a chain of valid worlds evolving from the initial one such that the query projects to the last one.

All examples of this section are taken from the modelization of the Sysiphus-I case study (a problem of allocation of offices to members of a compagny) with the constrained derivation framework [BGM99].

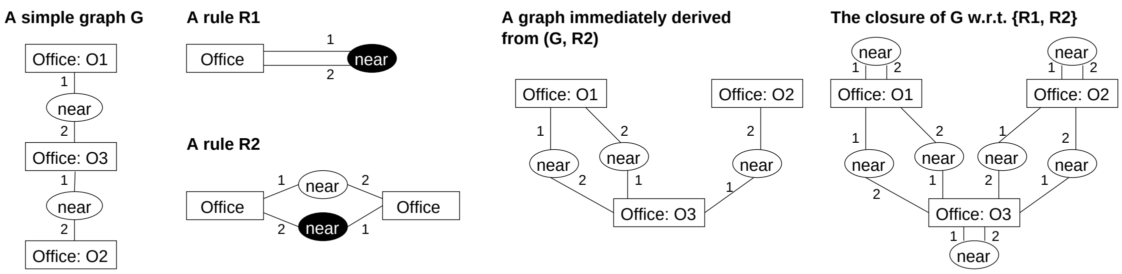


Figure 7: Rules, derivation and closure

3.1 Rules

A rule "If H then C" is classically defined as a couple of simple graphs, H and C, respectively called hypothesis and conclusion of the rule, sharing some generic concept nodes (called here frontier nodes). Briefly said, the logical semantics Φ is extended to rules in the following way: let x_1 ... x_p be the variables assigned to frontier nodes; let $\Phi'(H)$ and $\Phi'(C)$ be formulas obtained from $\Phi(H)$ and $\Phi(C)$ by removing existential quantification of x_1 ... x_p. Then, given a rule R, $\Phi(R) = \forall x_1 \ldots \forall x_p \; \Phi'(H) \rightarrow \Phi'(C)$.

[Bag99] vizualises a rule as a simple graph provided with a coloring of its nodes with two colors, say 0 and 1. In drawings, 0-colored nodes are painted in white, and the others in black. The subgraph induced by the color 0 nodes must be a syntactically correct simple graph. Nodes with color 0 are the hypothesis nodes and make up 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. See in Figure 7 rules R_1 and R_2 expressing that "near" is a reflexive and symmetric relation.

A rule R is *applicable* to a simple graph 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 simple graph 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 to $\Pi(c)$. In this case, we say that G' is *immediately derived* from (G, R) (see Figure 7). Let $\mathcal{R}$ be a set of rules. 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 $G = G_0, G_1, \ldots, G_k = G'$ such that each G_{i+1} is immediately derived from $(G_i, \mathcal{R})$.

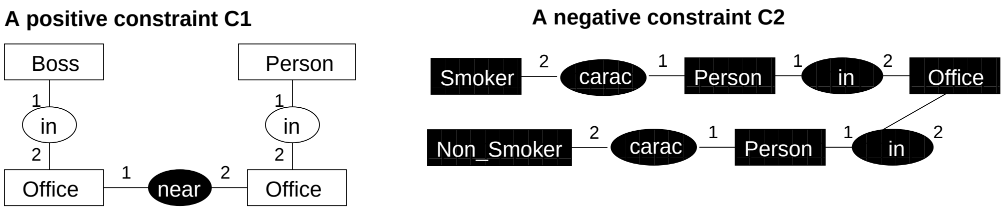


Figure 8: Positive and negative constraints

Theorem 7 *(Soundness and compleness of forward chaining) Let G and H be simple graphs and $\mathcal{R}$ be a set of rules, all relative to a support S.*

- *if H is derived from $(G, \mathcal{R})$ then $\Phi(S),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(H)$*
- *provided that G and all rules of $\mathcal{R}$ are in normal form, and the graph obtained at each derivation step is put into normal form: if $\Phi(S),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(H)$ then there exists a simple graph H' that can be derived from $(G, \mathcal{R})$ such that H projects onto H'.*

Proof. See [SM96]. Note that [SM96] also defines backward chaining operations, and proves associated soundness and completeness results.

3.2 Constraints

We consider *positive* and *negative* constraints. A positive constraint expresses that "if information A is present, then information B must also be present". A negative constraint expresses that "if information A is present, then information B must be absent". Formally, a constraint is defined in the same way as a rule, as a simple graph provided with two colors, 0 and 1. Nodes colored in 0 form the condition part of the constraint. A given graph G satisfies a *positive* constraint C if *every* projection from the condition part of C into G can be extended to a projection of the whole graph C into G. And G satisfies a *negative* constraint C if *no* projection from the condition part of C into G can be extended to a projection of the whole graph C into G. A graph G is said to be valid w.r.t. a set of constraints $\mathcal{C}$ if it satisfies all constraints of $\mathcal{C}$.

Fig. 8 presents two constraints: a positive constraint *C1*, expressing that "a boss office should be near all offices" (more litterally: "if a *boss* is *in* an *office* and a *person* is *in* an *office*, then this latter office must be *near* the *boss office*"), and a negative constraint, *C2*, expressing that "a smoker and a non smoker can not share an office" (the condition part is here empty).

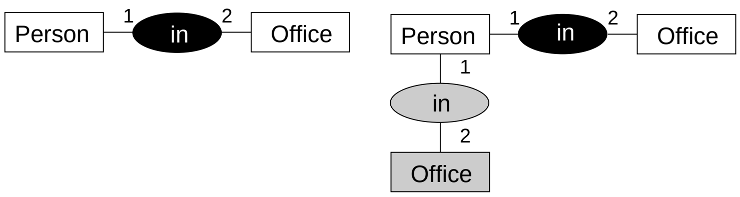


Figure 9: Transformation rules

Note that more generally we could restrict negative constraints to negative constraints with an empty condition. Indeed, given a constraint C and a graph G, the statement "no projection from the condition part of C to G can be extended to a projection of C into G" is equivalent to "there is no projection from C to G". Also note that a more general form of constraint can be handled, corresponding to the following intuitive semantics "if information A is present and information B is absent then information C must be present and information D must be absent". It is defined as a simple graph provided with four colors, one for each part A, B, C and D.

3.3 Constrained derivation

Consider that asserted facts (represented by a simple graph F) and production rules (represented by a set of rules $\mathcal{R}$) describe a world, say W. The associated deduction problem (given a query Q and a world W, is Q deducible from W, i.e. does there exist a derivation from $(F, \mathcal{R})$ to a graph F' such that Q can be projected into F'?) is a semi-decidable problem [CS98]. Let us restrict ourselves to decidable cases. Such a case occurs for instance if we restrict production rules to rules that do not add new concept nodes (the conclusion of a rule is composed of frontier nodes and relation nodes; or in logical terms, the formula assigned to the conclusion does not add new variables). It is then possible to compute the *closure* of F w.r.t. $\mathcal{R}$. Intuitively, a graph G is closed w.r.t. $\mathcal{R}$ if no application of a rule of $\mathcal{R}$ to G adds information. More formally, for any rule R of $\mathcal{R}$, if R can be applied onto G (there is a projection Π from the hypothesis of R into G) then the resulting graph is equivalent to G (Π can be extended to a projection of the whole graph R into G). Given a graph F and a set of rules $\mathcal{R}$, a closure of F w.r.t. $\mathcal{R}$ is a graph derived from $(F, \mathcal{R})$ closed w.r.t. $\mathcal{R}$. All closures are equivalent for projection; we call *the* closure of F by $\mathcal{R}$ the (unique) smallest graph of this set (see Figure 7). Now, a world $W = (F, \mathcal{R})$ is said to be valid w.r.t. a set of constraints $\mathcal{C}$ if the closure of F by $\mathcal{R}$ is valid w.r.t. $\mathcal{C}$.

Let us add a set of rules, called *transformation rules*, whose role is to generate new worlds. Transformation rules are rules as defined above but the intuitive semantic becomes "if information A is present in a world, then generate a new world by adding information B to this world". See Figure 9: the rule on the left says "if there are a *person* and an *office*, try to put that *person* into that *office*".

A more general form of transformation rule can be handled, with hypothesis split into a positive part ("if information A is present") and a negative part ("and if information B is absent"). Such a rule is encoded by a graph with three colors. It is applicable to a graph if there is a projection from the positive part of the hypothesis onto the graph that cannot be extended to a projection of the whole hypothesis ("A is found but we cannot find B"). See Figure 9: the rule on the right says "if there are a *person* who is not already assigned to an office, and an *office*, try to put that *person* into that *office*". Note that it would not be possible to generalize in the same way production rules because the derivation mechanism would become non monotonic (for instance the "unique closure" property would not hold true anymore).

Let $W = (F, \mathcal{R})$ be a valid world w.r.t. a set of constraints $\mathcal{C}$ and let $\mathcal{T}$ be a set of transformation rules; a *successor* of W by $\mathcal{T}$ is a valid world obtained by appling one rule of $\mathcal{T}$ onto the closure of F by $\mathcal{R}$. A world W' is a *descendant* of a (valid) world W if there exists a sequence $W_0 = W$ ... $W_k = W'$ such that each W_{i+1} is a successor of W_i, $0 \leq i < k$. Such a sequence is called a constrained derivation from W to W'. The global deduction problem can be expressed as follows. The input is a world $W = (F, \mathcal{R})$, a set of constraints $\mathcal{C}$, a set of transformation rules $\mathcal{T}$ and a query Q. The question is whether there exists a constrained derivation from W to a world W' such that Q projects to W'. Note that every world of a constrained derivation has to be valid, including W.

Finally, let us point out that the constrained derivation model could handle nested graphs [CMS98] instead of simple graphs.

4 Strengths and Weaknesses of this Graph-based Approach

Let us first answer some questions about our approach: why are we representing knowledge with labeled graphs, and specially with conceptual graphs? Why are we doing reasonings with graph operations instead of logical operations? More specifically, why are we basing reasonings on graph homomorphism and not on diagrammatic rules *a la Peirce*?

Why labeled graphs? Different kinds of labeled graphs have long been used to represent knowledge. One of the more attractive features of labeled graphs in general for knowledge acquisition and representation is their *visual* aspect. Basically a simple conceptual graph is a bipartite labeled graph, where one class of nodes represents entities and the other represents relationships between these entities. It is thus easily editable and interpretable by a person, in particular if it has a small number of nodes and edges or if it has a special structure enhanced by the drawing. As noticed in [BBV97] for instance, in a knowledge modeling context, "Experts appreciate graphic notations. The very simple graphic concepts of Conceptual Graphs (boxes, ovals and arrows) and the use of labels chosen in the expert language make modeled knowledge understandable by the expert with no effort, whereas they were afraid by textual formal languages".

Why conceptual graphs? Several important features distinguish conceptual graphs from other labeled graph representations:

- a clear separation is made between ontological knowledge and factual knowledge;
- they are not limited to binary relations;
- they are provided with a logical semantics.

Why graph-based reasonings? Conceptual graphs are not only a *visual* language but also a *formal* knowledge representation model. They are provided with reasoning operations which are sound and complete with respect to deduction in first order logics (see above for simple graphs, [CMS98] for nested graphs, and [Sow84][Wer95][KS97] for more general graphs equivalent to first order logic).

One commonly held opinion is that conceptual graphs are — only — a graphical interface for logics. Indeed, a justified question is to ask why not simply translate conceptual graphs into logics, use logical techniques for doing reasonings, and come back to graphs for presenting results. A related question is why base reasonings on graph homomorphism (projection) rather than on elementary graphical rules (canonical formation rules for simple graphs and diagrammatic rules *a la* Peirce for general conceptual graphs). The main argument justifying graph-based reasonings is related to knowledge modeling. Another argument is related to computational aspects.

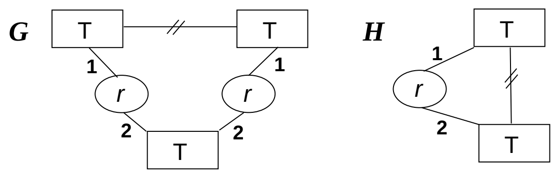
Benefits from a modeling viewpoint. Let us summarize main qualities of the model we propose from a modeling viewpoint (for a more detailed discussion, see [BGM99]). Objects are easily understandable by an end-user *as well as reasonings*, and this for two reasons: projection is a graph-matching operation easily interpretable, and the same language is used at interface and operational levels. This property can provide a precious help to verify the validity of an expertise modeling for instance. Indeed, it is in general very difficult to ensure that the expert reasoning is correctly modeled [GS95]. Our model can aid by giving the expert the possibility to run the system and to follow reasonings step by step in order to check that what is done is correct from his/her viewpoint.

Benefits from a computational viewpoint. Having knowledge represented by graphs and reasonings based on graph theoretic notions allows one to moor to combinatorial algorithms, and to benefit from results in this domain. Also, using graphs whereas the dominating formalism is logic[2] provides a different viewpoint. For instance, just to give some immediate notions, a path, a cycle or a distance are natural notions on graphs and are not in logics. See for instance the algorithm for backward chaining of production rules based on the graph notion of a *piece* in [SM96] or the complexity result in [BMT99] (the deduction problem of a description logic, whose complexity was unknown, is shown to be solvable in polynomial time by a translation into a graph homomorphism problem).

Limitations. One question we have been exploring can be summarized as follows "how far is it possible to go in knowlegde representation and reasonings with labeled graphs and graph matching operations (with graph homomorphism in the foreground)?". Indeed, the main hypothetical limitation of our work is expressivness. At present time we are not able to give a definitive answer to this question, but can provide some partial results.

Simple graphs represent positive and conjunctive knowledge. And projection corresponds to deduction on this knowledge. It seems that, taken alone, this kind of operation (that compares two graphs by a *global* matching, instead of trying to obtain one from another by a derivation sequence) is fundamentally unable to deal with negation or disjunction in a complete way, even limited as shown below. Suppose for instance one wants to deal with *difference links*, expressing that two entities represent distinct entities

[2] and we do no intend here to question this pre-eminence; logic is naturally the reference formalism as soon as one aims at modelizing reasonings.



$\Phi(G)$ implies $\Phi(H)$, but there is no projection from H to G

Figure 10: Projection and difference links

(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. One may represent difference as a relation type, say *dif* with associated production rules giving it the semantics of $\neg$*equal* ($\forall x \forall y dif(x, y) \rightarrow dif(y, x)$ and $\forall x \forall y \forall z dif(x, y) \wedge (y = z) \rightarrow dif(x, z)$).

In Figure 10 [3], difference links are represented by crossed lines. $\Phi(G)$ implies $\Phi(H)$ but there is no projection from H to G. Indeed, $\Phi(G) = \exists x \exists y \exists z\, r(x,\ z) \wedge r(y,\ z) \wedge \neg(x = y)$ and $\Phi(H) = \exists x \exists y\, r(x,\ y) \wedge \neg(x = y)$, and by adding to $\Phi(G)$ the subformula $((x = z) \vee \neg(x = z)) \wedge ((y = z) \vee \neg(y = z))$, one keeps an equivalent formula from which $\Phi(H)$ can be trivially deduced.

One approach for obtaining a model equivalent to FOL (or more), while keeping projection as the basic operation is that of [KS97]. This work proposes a sound and complete reasoning procedure for general conceptual graphs, based on *tableaux*, which decomposes conceptual graphs into simple graphs, and combines projection onto simple graphs with tableaux rules.

The constrained derivation model outlined in this paper clearly allows some forms of negation and disjunction, while keeping objets and operations easily interpretable. The study of its expressivness is not achieved yet.

[3] This counter-example is from Geneviève Simonet (personal communication).

References

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

[Bag99] J.-F. Baget. A simulation of co-identity with rules in simple and nested graphs. In *Proc. ICCS'99, LNAI 1640*, pages 442—455. Springer Verlag, 1999.

[BBV97] C. Bos, B. Botella, and P. Vanheeghe. Modeling and simulating human behaviors with conceptual graphs. In *ICCS'97 Proc., LNAI 1257*, pages 275–289. Springer Verlag, 1997.

[BGM99] J.-F. Baget, D. Genest, and M.-L. Mugnier. Knowledge acquisition with a pure graph-based knoweldge representation model — application to the sisyphus-i case study. In *Proc. KAW'99*, 1999.

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

[CM92] M. Chein and M.L. Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, 1992.

[CM95] M. Chein and M.L. Mugnier. Conceptual Graphs are also Graphs. Research Report 95-004, LIRMM, Jan. 1995. 17 pages.

[CMS98] M. Chein, M.-L. Mugnier, and G. Simonet. Nested Graphs: a Graph-based Knowledge Representation Model with FOL semantics. In *Proc. KR'98*, pages 524–534. Morgan Kaufmann, 1998.

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

[GS95] B.R. Gaines and M.L. Shaw. Knowledge and requirements engineering. In *Proc. KAW'95*, 1995.

[GW95] B.C. Ghosh and V. Wuwongse. A direct Proof Procedure for Definite Conceptual Graph programs. In *LNAI 954, Proceedings of ICCS'95*, pages 158–172. Springer Verlag, 1995.

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

[KS97] G. Kerdiles and E. Salvat. A sound and complete proof procedure based on tableaux and projection. In *Proc. of ICCS'97, LNAI 1257*, pages 371–385. Springer Verlag, 1997.

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

[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.

[Mug95] M.L. Mugnier. On generalization/specialization for conceptual graphs. *J. Expt. Theor. Artif. Intell.*, 7(3):325–344, 1995.

[PMC98] A. Preller, M.-L. Mugnier, and M. Chein. Logic for Nested Graphs. *Computational Intelligence*, 14(3), 1998.

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

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

[SM96] E. Salvat and M.L. Mugnier. Sound and complete forward and backward chainings of graph rules. In *ICCS'96 Proc., LNAI 1115*, pages 248–262. Springer Verlag, 1996.

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

[Sow99] J. F. Sowa. Conceptual Graphs: Draft Proposed American National Standard. In *ICCS'99 Proc., LNAI 1640*, pages 1–65. Springer Verlag, 1999.

[Wer95] M. Wermelinger. Conceptual Graphs and First-Order Logic. In *Proc. of ICCS'95, LNAI 954*, 1995.