Improving the Forward Chaining Algorithm for Conceptual Graphs Rules

Jean-François Baget
INRIA Rhône-Alpes
655, avenue de l'Europe
38334 Saint Isnmier, France
jean-francois.baget@inrialpes.fr

Abstract

Simple Conceptual Graphs (SGs) are used to represent entities and relations between these entities: they can be translated into positive, conjunctive, existential first-order logics, without function symbols. Sound and complete reasonings w.r.t. associated logic formulas are obtained through a kind of graph homomorphism called projection.

Conceptual Graphs Rules (or CG rules) are a standard extension to SGs, keeping sound and complete reasonings w.r.t. associated logic formulas (they have the same form as tuple generating dependencies in database): these graphs represent knowledge of the form "IF ... THEN".

We present here an optimization of the natural forward chaining algorithm for CG rules. Generating a graph of rules dependencies makes the following sequences of rule applications far more efficient, and the structure of this graph can be used to obtain new decidability results.

Introduction

Simple Conceptual Graphs (or SGs) have evolved since Sowa's reference book (Sowa 1984) as the cornerstone of a family of knowledge representation languages known as "Conceptual Graphs". SGs are used to represent entities as well as the relations between them. They can be translated into positive, conjunctive, existential first-order logics formulas, without function symbols. Sowa's graph-based inference operator has since been reformulated as a labeled graphs homomorphism called projection (Chein & Mugnier 1992). A projection of a SG H into a SG G means that all information encoded in H is already present in G; projection is sound (Sowa 1984) and complete (Mugnier & Chein 1996) w.r.t. the associated logical semantics.

A standard extension, proposed in (Sowa 1984), is obtained by using Conceptual Graphs Rules (or CG rules). These rules are also represented by graphs (one subgraph is identified as the hypothesis part, the remaining part being the conclusion). They represent knowledge of the form "IF ... THEN". Extending the logical semantics to translate CG rules (the obtained formulas are the same as the tuple generating dependencies studied in databases), the projection-based deduction mechanism of the CG rules model has been proven sound and complete w.r.t. deduction of the associated logic formulas (Salvat & Mugnier 1996).

An efficient backward chaining has been presented by (Salvat 1998), and its comparison with Prolog proved its efficiency (Coulondre & Salvat 1998). However, this algorithm does not cope well with some extensions built upon CG rules, and particularly the more expressive languages from the $\mathcal{SG}$ family (Baget & Mugnier 2002). In these models, algorithms used for deduction rely on forward chaining of rule applications.

In this paper, we present an optimization of the natural forward chaining algorithm that is highly adaptable to these extensions of CG rules. An initial treatment of a library of CG rules is used to generate the *graph of rules dependencies*. Using this graph makes the subsequent rule applications far more efficient, and its structure can be used to obtain new decidability results, extending those presented in (Baget & Mugnier 2002).

This paper is organized as follows: we first briefly recall basic definitions on SGs, and we present CG rules. To help readers unfamiliar with the CG formalisms, we recall their translation into first-order logics. After detailed motivations, we introduce the graph of rule dependencies, and discuss its efficiency (for optimization purpose as well as for extending existing decidable cases).

Simple Conceptual Graphs

The vocabulary available is encoded in a structure called the *support*.

Definition 1 (Support) *We call* support *a tuple* $\mathcal{S} = (\mathcal{M}, T_C, T_1, \ldots, T_k)$ *of pairwise disjoint partially ordered sets:* $\mathcal{M}$ *is the set of* markers, T_C *is the set of* concept types, *and* $T_i, 1 \leq i \leq k$ *is the set of* relation types *of arity* i.

The set of marker $\mathcal{M}$ contains a distinct element: the *generic marker* $*$, used to represent unnamed entities. The other markers (called *individual*) represent named entities. Individual markers are pairwise non comparable, and are more specific than the generic one. No assumption is needed on the partial orders encoding the types hierarchies. These partial orders will be denoted by $\leq$. These sets do not need to be finite, but we assume that the comparison of two elements can be computed in constant time.

To simplify definitions, we present here SGs as multiple directed hypergraphs[1] whose nodes (representing entities) are labeled by a concept type and a marker of $\mathcal{M}$ and hyperarcs (non empty tuples of nodes representing relations between entities) are labeled by a type of corresponding arity.

Definition 2 (SGs) *Let $\mathcal{S}$ be a support. A* simple conceptual graph *(or SG), defined over $\mathcal{S}$, is a tuple $G = (V, U, \lambda)$ where V is the set of* nodes*, $U \subseteq V^+$ is a multiset[2] of hyperarcs (we call them* relations*), and λ is a mapping that labels each node by a pair formed by a concept type of T_C and a marker of $\mathcal{M}$ (a node is said* generic *if labeled by $*$,* individual *otherwise), and labels each relation of size i by a relation type in T_i.*

We adopt for SGs the following graphical representation: each node is represented by a rectangle, and each relation $(x_1, \ldots, x_i)$ by an oval in which the relation type is written. For each of its arguments x_p, we draw a line between the rectangle representing x_p and the oval representing the relation, and write the number p next to this line. Finally, we write $T : M$ inside the rectangle representing a node whose type and individual marker are respectively T and M, and only T if the node is generic. The graph H in Fig. 1 is the drawing of the SG $H = (V, U, \lambda)$ defined as follows: $V = \{X, Y, Z\}$; $U = \{(Z, Y), (X, Y, Z)\}$; and $\lambda(X) = (t_1, *)$, $\lambda(Y) = (t_2, *)$, $\lambda(Z) = (t_3, *)$, $\lambda((Z, Y)) = r_2$, $\lambda((X, Y, Z)) = r_3$ (all nodes are generic).

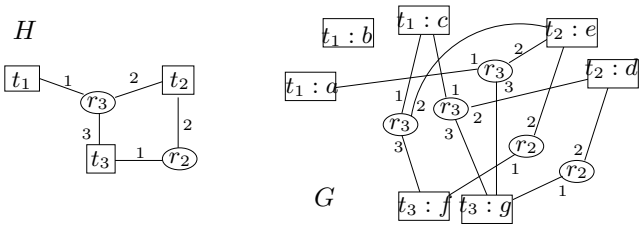

Figure 1: The drawing of two SGs G and H.

The basic inference operator for SGs is a labeled (hyper)graphs homomorphism called projection. Existence of a projection from a SG H into a SG G means that all information encoded in H is already present in G.

Definition 3 (Projection) *Let H and G be two SGs defined over the same support $\mathcal{S}$. A projection from H into G is a mapping $\pi : V(H) \rightarrow V(G)$ such that: for each node x of H, $\lambda(\pi(x)) \leq \lambda(x)$ (we also denote by $\leq$ the product order on the orders on T_C and $\mathcal{M}$), and for each relation $r = (x_1, \ldots, x_i)$ in H, there must exist a relation $r' = (\pi(x_1), \ldots, \pi(x_i))$ such that $\lambda(r') \leq \lambda(r)$.*

As an exercise, the reader can check that, assuming all types are pairwise non comparable in the support, there is exactly 2 projections from the graph H into the graph G, both represented in Fig. 1. One of them associates the nodes respectively labeled $t_1 : c$, $t_2 : d$ and $t_3 : g$ to the nodes respectively labeled t_1, t_2 and t_3. Note however that, contrary to what this example suggests, projection does not need to be an injective mapping.

Before defining our basic deduction problem (called, as in (Baget & Mugnier 2002) $\mathcal{SG}$-DEDUCTION), we must introduce the notion of *normal form*. A SG is said in normal form if all individual nodes have different markers (the same entity is represented by an unique node). A SG G is put into its normal form $nf(G)$ by fusioning all nodes sharing the same individual marker[3].

$\mathcal{SG}$-DEDUCTION
Data: A support $\mathcal{S}$ and two SGs G and H, defined over $\mathcal{S}$.
Question: Can H be deduced from the knowledge base $(\mathcal{S}, G)$, *i.e.* does there exist a projection from H into the normal form of G ?

Conceptual Graphs Rules

Conceptual graph rules (in short CG rules) express knowledge of the form "IF...THEN". It is convenient to represent them as colored SGs.

Definition 4 (CG Rules) *A* CG rule*, defined over a support $\mathcal{S}$, is a pair $R = (G, H)$ where G is a SG defined over $\mathcal{S}$, and H is a partial subgraph of G[4]. The SG H is called the* hypothesis *of the rule (Hyp(R)), and the other nodes and relations form its* conclusion *Ccl(R) (it is not necessarily a graph, since hyperarcs can lack their elements: we call it a* proto-graph*).*

CG rules are represented in the same way as SGs, excepted that we color rectangles and ovals to clearly see the nodes and relations that belong to the hypothesis or the conclusion. Here, elements of the hypothesis will remain in white, while elements of the conclusion will be shaded in gray (see Fig. 2).

We must now present the inference mechanism used in this KR model.

Definition 5 (Rule Application) *Let G be a SG and R be a CG rule, both defined over a support $\mathcal{S}$. Then R is said* applicable *to G if there exists a projection, say π, from Hyp(R) into $nf(G)$. In that case, we note $\alpha(G, R, \pi)$ the graph obtained by applying the rule R on the graph $nf(G)$ following the projection π. This is done in the following way: consider the proto-graph obtained by making the disjoint union of a copy of $nf(G)$ and a copy of Ccl(R). Then for each (proto)relation r in Ccl(R), for each node x of the hypothesis of R that is a ith argument of r, make the copy of $\pi(x)$ the ith argument of the copy of r.*

[1] The usual definition of SGs as bipartite graphs is simply obtained by considering the bipartite of incidence of our hypergraphs.

[2] There can be many occurrences of the same hyperarc, that can be labeled differently.

[3] We assume that all nodes sharing the same marker also share the same type, which is the type of the obtained node. Usually, a *conformity relation* defined in the support determines the type given to an individual node, according to its marker.

[4] Obtained from G by eventually removing some of its nodes and the relations for which one argument has been removed, then eventually removing some of the remaining relations.

The mechanism of rule application is illustrated in Fig. 2, where the SG G is already in normal form.

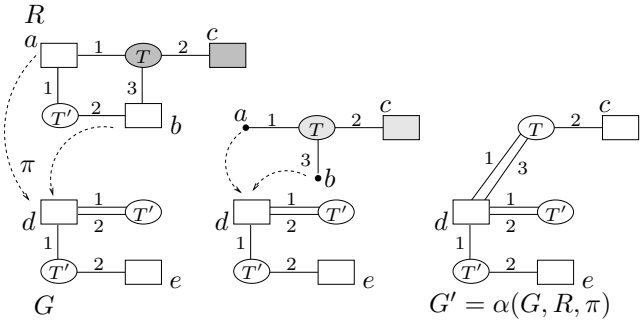

Figure 2: Applying a CG rule R to a SG G.

The deduction problem requires the notion of derivation of a graph.

Definition 6 (Derivation) *Let $\mathcal{S}$ be a support, $\mathcal{R}$ be a set of CG rules, and G and G' be two SGs, all defined over $\mathcal{S}$. We say that G' is $\mathcal{R}$-*derived *from G if there exists a sequence (possibly reduced to G) of SGs $G = G_0, \ldots, G_k = G'$ such that, for $1 \leq p \leq k$, there is a rule $R \in \mathcal{R}$ and a projection π of Hyp(R) into G_{p-1} with $G_p = \alpha(G_{p-1}, R, \pi)$.*

We define now the deduction problem in this model using CG rules (called $\mathcal{SR}$, as in (Baget & Mugnier 2002)):
$\mathcal{SR}$-DEDUCTION
Data: A support $\mathcal{S}$, two SGs G and H, and a set of CG rules $\mathcal{R}$, all defined over $\mathcal{S}$.
Question: Can H be deduced from the knowledge base $(\mathcal{S}, G, \mathcal{R})$, *i.e.* does there exist an $\mathcal{R}$-derivation from G into a SG G' such that H projects into the normal form of G' ?

Relationships with FOL

Since (Sowa 1984), *semantics* of SGs are usually expressed through a translation to the positive, conjunctive, existential fragment of first-order logics (without function symbols); that fragment will be denoted by FOL$(\wedge, \exists)$. CG rules are translated to formulas corresponding to *tuple generating dependencies* in databases, as pointed out in (Coulondre & Salvat 1998). Knowledge expressed in a support $\mathcal{S}$, in a SG G or in a set of CG rules $\mathcal{R}$ can be translated to the formulas $\Phi(\mathcal{S})$, $\Phi(G)$ and $\Phi(\mathcal{R})$, as shown below. Though logical semantics are not in the scope of this paper, we think that these translations can help a reader unfamiliar with CGs.

Translating the support To each pair of types (t, t') of arity i in the support $\mathcal{S}$ (concept types are considered as relation types of arity 1), such that $t < t'$, we associate a formula $\phi(t, t') = \forall x_1 \ldots \forall x_i (t(x_1, \ldots, x_i) \rightarrow t'(x_1, \ldots, x_i))$. The interpretation $\Phi(\mathcal{S})$ of the support is the conjunction of these formulas $\phi(t, t')$, for all pairs (t, t') of comparable types in the support.

Translating SGs A SG G will be translated as follows: to each node x we associate the term $\sigma(x)$: a distinct variable if x is generic, and the constant M to each individual node having marker M. A node x typed by t will be interpreted by the formula $\phi(x) = t(\sigma(x))$. A relation $r = (x_1, \ldots, x_i)$ labeled by t will be interpreted by the formula $\phi(r) = t_1(\sigma(x_1), \ldots, \sigma(x_i)) \wedge \ldots \wedge t_p(\sigma(x_1), \ldots, \sigma(x_i))$. The interpretation of the graph G is then the formula $\Phi(G)$ obtained by making the existential closure of the conjunction of the formulas $\phi(r)$ and $\phi(x)$, for all relations r and all nodes x in G. By example, the interpretation of the graph H in Fig. 1 is the formula $\exists X \exists Y \exists Z (t_1(X) \wedge t_2(Y) \wedge t_3(Z) \wedge r_2(Z, Y) \wedge r_3(X, Y, Z))$.

Translating CG rules Let $R = (G, H)$ be a CG rules. As if translating the SG G, we build the formulas $\phi(r)$ interpreting each of its nodes and relations. We define $\Phi_H(R)$ as the conjunction of all $\phi(r)$, for nodes and relations r appearing in the hypothesis of the rule, and $\Phi_C(R)$ as the conjunction of all $\phi(r)$, for those appearing in the conclusion of the rule. The interpretation of a rule R is the formula $\Phi(R) = \forall x_1 \ldots \forall x_p (\Phi_H(R) \rightarrow (\exists y_1 \ldots \exists y_q \Phi_C(R)))$, where the x_i are the variables associated to nodes of the hypothesis, and the y_j are those associated to nodes of the conclusion. A set of rules is interpreted as the conjunction of the interpretations of its elements. By example, the interpretation of the CG rule R in Fig. 2 is the formula: $\Phi(R) = \forall X \forall Y (T'(X, Y) \rightarrow (\exists Z T(X, Z, Y)))$.

We have now all the tools to express the "soundness and completeness" results that logically found deduction in the $\mathcal{SG}$ and $\mathcal{SR}$ models:

Theorem 1 ((Sowa 1984; Mugnier & Chein 1996)) *Let H and G be two SGs defined over $\mathcal{S}$. Then H can be deduced from $(\mathcal{S}, G)$ if and only if $\Phi(H)$ is a logical consequence of $\Phi(\mathcal{S})$ and $\Phi(G)$.*

Theorem 2 ((Salvat & Mugnier 1996)) *Let $\mathcal{R}$ be a set of CG rules, and H and G be two SGs, all defined over a support $\mathcal{S}$. Then H can be deduced from $(\mathcal{S}, G, \mathcal{R})$ if and only if $\Phi(H)$ is a logical consequence of $\Phi(\mathcal{S})$, $\Phi(\mathcal{R})$ and $\Phi(G)$.*

Complexity and decidability

Let us now recall complexity and decidability results about these two deduction problems:

Theorem 3 (Complexity) $\mathcal{SG}$-DEDUCTION *is an NP-complete problem.*

This theorem has been initially proven in (Chein & Mugnier 1992), with a CLIQUE reduction. It can be more convenient to point out that $\mathcal{SG}$-DEDUCTION is a trivial generalization of GRAPH HOMOMORPHISM, itself a well known generalization of GRAPH K-COLORING: both are well known NP-complete problems.

Theorem 4 ((Un)Decidability) $\mathcal{SR}$-DEDUCTION *is a semi-decidable (but not decidable) problem.*

This was proven by (Coulondre & Salvat 1998), reducing the problem to IMPLICATION OF TUPLE GENERATING DEPENDENCIES. (Baget 2001) shows that it is the payback for expressivity: indeed, $\mathcal{SR}$-DEDUCTION is a computation model (Turing Machines can be encoded with these rules).

Decidability results exploit the notion of completeness (no rule application can add new information to the graph) (Baget & Mugnier 2002), allowing to define a generic criterion (finite expansion sets) for decidability.

Definition 7 (Complete Graph) *An SG G is said* complete *with respect to a set of rules $\mathcal{R}$ if for every rule $R \in \mathcal{R}$, for every projection π of R into $nf(G)$, the SG $\alpha(G, R, \pi)$ can be projected into $nf(G)$.*

If we can derive a complete graph, then it is equivalent to all other complete graphs that can be derived. The irredundant graph (see (Baget & Mugnier 2002)) noted $G^{\mathcal{R}}$ is the smallest representant of this equivalence class.

Definition 8 (Finite Expansion Sets) *A set of CG rules $\mathcal{R}$ is called a* finite expansion set *if for every SG G, a complete SG can be $\mathcal{R}$-derived from G.*

If we restrict our knowledge base to some range-restricted set of rules, then $\mathcal{SR}$-DEDUCTION becomes a decidable problem. Two example of finite expansion sets have been studied in (Baget & Mugnier 2002). *Range restricted rules* are rules such that no generic node belong to their conclusion (they are named by analogy with Datalog rules in which all variables of the head must appear in the queue (Abiteboul, Hull, & Vianu 1995)). *Disconnected rules* are such that no path exists between nodes of the conclusion and those of the hypothesis. Using any of these restrictions makes $\mathcal{SR}$-DEDUCTION an NP-complete problem. However, considering a set of rules that is the union of range-restricted rules and disconnected rules leads to a semi-decidable $\mathcal{SR}$-DEDUCTION.

Motivations

Extensions of the $\mathcal{SG}$ end $\mathcal{SR}$ composing the $\mathcal{SG}$ family (Baget & Mugnier 2002) have been initially proposed in (Baget, Genest, & Mugnier 1999) as a convenient way to model and solve the SISYPHUS I problem proposed by the Knowledge Acquisition community. But though the language proposed enabled an elegant modelization of the problem, algorithmic efficiency was lacking. Moreover, our first experiments, using both the naive forward chaining algorithm (FC) and the efficient backward chaining one (BC) (Salvat 1998) available on the platform CoGITaNT (Genest & Salvat 1998) to solve $\mathcal{SR}$-DEDUCTION, showed that FC was much quicker.

Let us explain this result. In the $\mathcal{SEC}$ model (an extension of $\mathcal{SR}$), rules applications can be seen as elementary evolutions of a world. Also present in the knowledge base are constraints, that are used to check the integrity of the world at each step of its evolution. In this model, the deduction problem asks whether there exists a sequence of rules applications that generates only graphs satisfying the constraints, and where the last one answers to the query. Using FC, it is possible to cancel a rule application and backtrack as soon as a constraint violation is observed. No efficient pruning could be developed for BC: most of the time, a generated sequence of rules applications leading to the answer was found violating a constraint only when applying it to the initial graph. Such a problem should be encountered as soon as an external mechanism is used to forbid some rules applications sequences.

However, FC, though better than BC, was still an inefficient algorithm: though SISYPHUS I can be considered as a "toy example", the program based upon this algorithm ran 6 long days to enumerate all solutions. We considered three different ways to optimize this algorithm:

1. Optimize projection itself. Thanks to the close relationship exposed in (Mugnier 2000) between $\mathcal{SG}$-DEDUCTION and CSP (Constraint Satisfaction Network), it is possible to adapt backtrack enhancements developed in the CSP community to $\mathcal{SG}$-DEDUCTION (Baget 2003).
2. Reduce the number of projections computed at each step of FC.
3. Reduce the size of these projections.

The algorithm presented here relies on an initial treatment of the set of rules to answer these two last points. Moreover, the structure of the graph of rules dependencies initially generated can be used to extend the decidable cases when mixing finite expansion sets.

Rules Dependencies

Let us first briefly present a version of the naive FC. At each step of its execution, we compute all projections of all rules hypothesis in the current graph G, and store these couples (R_i, π_i). Then we compute the derivation $G = G_0, \ldots, G_k$ such that for $1 \leq i \leq k$, $G_i = \lambda(G_{i-1}, R_i, \pi'_i)$, where π'_i is a projection determined by π_i in the subgraph of G_{i-1} obtained from G. Note that the order of applications is not important, since different orderings lead to equivalent (and even isomorphic) graphs. The obtained SG G_k is the new current graph, and this step is repeated until the query can be projected into the current graph.

Neutrality Since applying the same rule twice following the same projection creates only redundant, useless information, it is immediate to point out that, at step i, a rule application of R following some projection must use a node that was added at step $i-1$ to be of any use. It means that some node in the hypothesis of R must be projected into a node added at step $i-1$, *i.e.* a node belonging to the conclusion of a rule in $\mathcal{R}$. Simply put, let R_1 and R_2 be two rules: if no node x_2 in the hypothesis of R_2 can be projected into a node x_1 of the conclusion of R_1, then no application of the rule R_1 into a graph can create a new application of R_2 into this graph. Let us formalize and generalize this basic idea.

Definition 9 (Neutral) *Let R and R' be two CG rules defined over a support $\mathcal{S}$. We say that R is* neutral *for R' if, for every graph G that can be defined over $\mathcal{S}$, for every graph $G' = \alpha(G, R, \pi)$, the set of all projections from R' into G' is still the same as the set of all its projections into G.*

Graphs of rules dependencies Let us now build a complete[5] directed graph $G(\mathcal{R})$ whose nodes are the rules of $\mathcal{R}$.

[5] There is an arc between each pair of nodes, loops included.

Now let us remove *some* of the arcs (R, R') such that R is neutral to R'. We obtain a *graph of rules dependencies*. We modify then the algorithm FC in the following way, obtaining the algorithm FCD (Forward Chaining with Dependencies). At the first step of the algorithm, all rules of $\mathcal{R}$ are checked for applicability. At subsequent steps, the only rules that are checked for applicability are the rules R such that there exists a rule R' applied during the previous step with (R', R) being an arc of $G(\mathcal{R})$. The following property, whose proof is immediate, points out the equivalence between the two algorithms FC and FCD. Note also that if no arc is removed from $G(\mathcal{R})$, FCD behaves exactly as FC.

Property 1 *For any positive integer i, the SG obtained at step i of the algorithm FC is equivalent to the SG obtained at step i of the algorithm FCD.*

As FC, FCD is thus sound and complete w.r.t. $\mathcal{SR}$-DEDUCTION. Note that FCD does not require to remove all arcs corresponding to neutral rules couples, but only those arcs can be removed or completeness would be lost. The task is thus to remove only those arcs, but the greater number possible (eventually all) to achieve a better efficiency.

From a weak to an optimal neutrality condition Let us formalize the neutrality condition presented as an example before Def. 9. It is immediate to check that if no label in the conclusion of R_1 is lesser than a label in the hypothesis of R_2, then R_1 is neutral to R_2. However, this characterization of neutrality does not detect enough neutrals (the criterion is sufficient but not necessary):

R_1	IF $[A : *]$	THEN $[B : *]$
R_2	IF $[B : *] \rightarrow (r) \rightarrow [C : *]$	THEN ...

The above criterion does not consider R_1 as a neutral to R_2 (the node typed B in Hyp(R_2) can be projected into the node of Ccl(R_2) with the same label), even if the hypothesis of R_2 cannot be projected into the SG restricted to the node typed B. This is the basic idea behind the main theorem: it is not sufficient to project a node into an other, its neighbors must also be projected. However, the following characterization of neutrals does not take normal form into account. Indeed, it would detect too much neutrals if SGs are put into their normal form, as it should be, between rule applications. Then too much arcs would be removed in the rules dependencies graph, and the FCD algorithm would be incomplete. Solutions to restore completeness are proposed in the next section.

Theorem 5 *Let R and T be two CG rules, such that applying R on any SG produces a SG in normal form. Then R is a trigger for T (*i.e. *R is not neutral for T) if and only if there exists:*

- *a projection π from a non empty subgraph H of Hyp(T) into Ccl(R) (we note then $N(H)$ the nodes of Hyp(T) that are not in H but are in its neighborhood),*
- *a partition $\oplus_N = \{N_1, \ldots, N_k\}$ of nodes of $N(H)$,*
- *a partition $\oplus_F = \{F_1, \ldots, F_{k+1}\}$ of nodes of the* frontier[6] *F of R,*

answering the following conditions:

1. *for each relation n incident to a node of H but that is not in H, there exists a relation f of same arity in R (we say that f is the* triggering support of *n) such that:*
   - *$\lambda(f) \leq \lambda(n)$;*
   - *for $1 \leq i \leq$ arity(n):*
     - *if the ith argument of n is a node $h \in H$, then the ith argument of f is $\pi(h)$;*
     - *if the ith argument of n is a node of N_j, then the ith argument of f is a node in F_j;*
2. *for $1 \leq j \leq k$, there exists a node whose label is more specific than the labels of all nodes in N_j and F_j.*

Proof: Intuitively, the projection π expresses that a part of Hyp(T) must be projected in a part of the SG that has been added when applying R, while the partitions show that nodes in N_j and those in F_j should be able to project into the same node of G.

We prove first that, given such π, $\oplus_N$ and $\oplus_F$ between two rules R and T, then R is a trigger of T (meaning R is not neutral for T). Proof of the ($\Rightarrow$) part of the equivalence rely on the construction of a SG G such that one application of R creates a new application of T.

For the second part of the equivalence ($\Leftarrow$), we suppose a graph G such that applying R creates a new application of T. Then we build projection π, the partitions $\oplus_N$ and $\oplus_F$, and finally check that conditions 1. et 2. are satisfied.

($\Rightarrow$) Suppose there exists such a projection π and such partitions $\oplus_N$ and $\oplus_F$ between R and T. Let us build the initial graph G and the graph G' obtained by applying R on G:

1. the graph G is initially defined as the hypothesis of R;
2. for each $N_j \in \oplus_N$, nodes of F_j in the frontier of R are fusionned, and the label s_j of the resulting node is more specific than the labels of nodes in F_j and N_j. Condition 2 asserts the existence of such a label, but its not unique: however, we can chose any potential candidate.
3. let us add to this graph (it is a disjoint union) the subgraph H' of the hypothesis of T containing all nodes that are neither in H, nor in $N(H)$.
4. for each relation $r = (x_1, \ldots, x_q)$ incident to a node in H' or a node in $N(H)$, we add a relation r' of same type and same arity in G such that, for every $1 \leq i \leq q$:
   - if x_i is a node of H', then the ith argument of r' is the node corresponding to x_i in G.
   - otherwise, x_i is a node of $N(H)$ (we suppose it belongs to the N_j partition), then the the ith argument of r' is the node s_j.

Let us now consider the projection π_1 from the hypothesis of R into this SG G defined as follows (there may be more projections, pointless for this proof). The subgraph of G

[6] The *frontier* is composed of nodes of the hypothesis that are incident to a relation in the conclusion.

obtained after phase 2. of its construction is a specialization of the hypothesis of R, and π_1 is a projection of Hyp(R) into this subgraph.

- if $x \in F_j$ (for $1 \leq j \leq k$), then $\pi_1(x) = s_j$;
- otherwise, $\pi_1(x) = Id(x)$.

This projection π_1 allows us to build the graph G', obtained by applying the rule R to G following π_1. We finish this part of the proof by constructing a projection π_2 of the hypothesis of T into G' that is not a projection into G. Note that *we do not put G' into its normal form*.

Let π_2 be the mapping associating a node of G' to each node of Hyp(T) defined by:

- if $x \in H$, then $\pi_2(x) = \pi(x)$ (more precisely, it is the node of G' that was added when applying R and that corresponds to $\pi(x)$);
- if $x \in N(H)$, then it belongs to a partition N_j, and $\pi_2(x) = s_j$;
- if $x \in H'$, then $\pi_2(x) = Id(x)$ (it is the node corresponding to x that was added during phase 3. of the construction of G).

If this mapping is a projection, then it is a projection that is not entirely in G (since H is non empty, there is at least one node whose image has been added by the application of R). It remains now to prove that π_2 is a projection.

Firstly, we point out that the restriction of π_2 to the subgraph generated by nodes of H and H' is a projection. Indeed, π is a projection the subgraph generated by H into the part of G' that was added when applying R; and *Id* is a projection from H' into the part of G' obtained from itself. Since there is no relation that is incident to both a node of H and a node of H', these two projections define a projection from the subgraph generated by nodes of H and H' into G'.

It remains now to extend this projection to the nodes from $N(H)$. First, π_2 maps each node x of $N(H)$ into a node whose label is more specific than the label of x (see definition of s_j at phase 2.). The last step is to prove that, for every relation $r = (x_1, \ldots, x_p)$ incident to a node in $N(H)$, there is a relation $r' = (\pi_2(x_1), \ldots, \pi_2(x_p))$ in G', whose label is more specific than the label of r. We obtain the three following cases:

1. Arguments of r belong to $N(H), H$ and H'. This is impossible, otherwise the argument in H' would have been put in $N(H)$.
2. Arguments of r belong to $N(H)$ and H. The existence of r' is ensured by the second condition of the theorem.
3. Arguments of r belong to $N(H)$ or $N(H)$ and H'. The existence of r' is ensured by the construction of the graph G' (phase 4.).

In conclusion, π_2 is a projection from T into G' that is not a projection in G. The existence of such a graph G proves that R is not neutral for T.

($\Leftarrow$) We suppose now that R is a trigger of T. Then there exists a graph G, such that, for any SG G' obtained by applying R to G following a projection π_1, there exists a projection π_2 from the hypothesis of T into G' that is not entirely into G. We will build a projection π and two partitions $\oplus_N$ and $\oplus_F$ that satisfy the two conditions of the theorem.

We note H the subgraph of the hypothesis of T whose node images following π_2 are the nodes of G that where added when applying R to G following π_1. We point out that H is non empty, otherwise the projection π_2 would have images only in G. We also remark that the restriction of π_2 to H defines a projection from this non empty subgraph of the hypothesis of T into the conclusion of R. Let us call π this projection.

We consider now the subgraph G_f of G that is generated by the images following π_1 of the frontier of R. We consider the partition $\oplus_F$ of G_f induced by π_1 (two nodes belong to the same partition if they are mapped by π_1 into the same node), and the partition $\oplus_N$ of the nodes of $N(H)$ (the neighborhood of H) induced by π_2. We reorder these partitions and write F_i, N_i if these partitions have been created by the same node.

It is now easy to check that π, $\oplus_F$ and $\oplus_N$ satisfy the two conditions of the theorem. □

An important consequence of this theorem is that it allows us to give the complexity of this problem.

$\mathcal{SR}$-NEUTRALITY
Data: Two CG rules R and R'.
Question: Is R neutral for R' ?

Theorem 6 (Complexity) *$\mathcal{SR}$-NEUTRALITY is a co-NP-complete problem.*

The proof is direct. The projection π as well as the two partitions is a polynomial certificate that R is not neutral for T. When the CG rules are disconnected, R is not neutral for T if and only if there is a projection from Hyp(T) into Ccl(R), hence the completeness. $\mathcal{SR}$-NON-NEUTRALITY being NP-complete, $\mathcal{SR}$-NEUTRALITY is co-NP-complete.

Using the graph of rules dependencies

Strengths We have presented an algorithm, FCD, that improves the standard Forward Checking as long as enough neutrals are found. Not only does it reduce the number of projection checks at each step of the algorithm, but it is also possible to store in the arcs of the graph of rules dependencies the partial projections from the hypothesis of the destination to the conclusion of the origin. This reduces the size of computed projections.

The initial cost of FCD can be high: there is $|\mathcal{R}|^2$ NP-hard problems to compute. First, the huge overhead cost induced by building the rules dependencies graph is quickly compensated: if Forward Chaining execution is longer than two steps, the overhead cost is compensated. Using our algorithm to solve the SISYPHUS I problem, we managed to generate all solutions in less than 2 hours. Then, if we consider a set of rules as a library, the rules dependencies graph should only been built once. Its cost is thus divided between all "users" of that library.

Second, structural arguments on the rules dependencies graph can be used to obtain new decidability results, or to extend existing ones, as shown by the two following theorems.

Theorem 7 *If the rules dependencies graph has no circuits (note that a loop is considered as a circuit), then $\mathcal{SR}$-*DEDUCTION *is decidable.*

Proof: See that no rule R can be applied at two different steps of the FCD algorithm. Otherwise, it would mean that the first application of R triggered a rule that triggered a rule ... that triggered R. There would then be a circuit in the rules dependences graph. The number of steps required by FCD is thus bounded by the number of rules. □

Theorem 8 *If each strongly connected component is formed of a finite expansion set, then $\mathcal{SR}$-*DEDUCTION *is decidable.*

Proof: The graph of strongly connected components, hereafter called the components graph, (each component is a node, there is an arc between two different components if there is an arc between rules belonging to these components) is a graph without circuits. The rules that must be checked for applicability at the second step of FCD belong to the strongly connected components $C_1, \ldots, C_p$. **Tag** Let us try to apply the rules of a C_i such that C_i has no predecessor in $C_1, \ldots, C_p$ in the components graph. C_i exists, since the components graph has no circuits. If no rule of C_i is applicable, remove C_i from the set of components and repeat **Tag** until we find a component C containing an applicable rule (if none is found, the algorithm ends). Though we keep in memory all rules triggered by all applications, we apply first the rules belonging to C (the order does not matter). Since C is a finite expansion set of rules, we obtain a closed graph. This ends the second step of the algorithm. Following steps are identical, using memorized triggered rules. See (as in the previous proof), that no rule of C will ever be triggered again.

Note that, though our proof rely on a particular ordering of rules applications, any ordering (and in particular the standard FCD algorithm) will lead to the same closed graph. □

Finally, the question H can be seen as a CG rule with an empty conclusion, and the SG G as a CG rule with an empty hypothesis. They can thus be integrated in the rules dependencies graph. See that rules that are not on a path from G to H will be of no use to solve the deduction problem. Removing these rules from the graph may modify its structure, and can lead to a decidable case.

Weaknesses The main problem with FCD is that using the optimal neutrality condition leads to loose completeness as soon as SGs are put into their normal form during the derivation. Three solutions can be adopted:

- drop this "optimal" criterion and use the weaker one, safe w.r.t. normalization;
- restrict ourselves to rules that never require any normalization after their application, they are the rules that have only generic nodes in their conclusion;
- it is possible to keep the optimum criterion without any restriction on rules used. Let us observe that a rule having an individual node in its conclusion is exactly equivalent (in the sense that their application generates the same graphs), as soon as a node with the same marker is present in the current graph, to a rule where this node belongs to the conclusion. Then as soon as an individual marker appears in the current graphs, rules where this marker appears in conclusion must be modified, and then the arcs for this rule must be computed again in the graph of rules dependencies. Though this work could be prepared at compile time, we must point out that a rule with k different individual markers in its conclusion can be replaced by 2^k different rules.

Conclusion

In this paper, we have presented an original method that optimizes the naive Forward Chaining algorithm used to compute deduction in a conceptual graphs enriched with rules model. This method uses a costly initial treatment of the rules base to reduce the number of rules applicability checks, to reduce the cost of each of these checks, and to achieve a goal-oriented form of forward checking, all this without additional overhead cost at runtime. Moreover, the rules dependencies graph built at compile time can be used to assert that the deduction problem with a particular set of rules is decidable, or that the deduction problem with a particular set of rules and a given query is decidable.

It is difficult to theoretically evaluate the performance of this algorithm. In the worst case, all rules trigger all rules (the query being considered as a rule), numerous and very small partial projections are a triggering proof for each pair of rules. In that case, FCD behaves as FC and the compilation was a waste of time. For an average case study, we need a probabilistic model to generate the support, the SGs, and the rules. It would be very difficult to evaluate the respective relevance of the many variables defining this random generator. We give here some characteristics that reduce our algorithm efficiency (on the other hand, opposite characteristics improve that efficiency):

1. the concept type hierarchy is a lattice, or close to a lattice (need to add/remove only some comparisons to create a lattice);
2. types labeling relations and nodes in the hypothesis of rules are very generic (much closer to the top of the hierarchy than to the bottoms);
3. most of the relations in the conclusion of rules have most of their arguments in the hypothesis.

These remarks have important consequences: 1) FCD is quite inefficient for CG models that require a lattice of concept types; 2) and 3) can be solved by "programming with rules best practices". Many specific rules are better than one generic rule, and unrelated conclusions that can be drawn from the same hypothesis must be split between different rules. Automatic rewriting of rules could take care of this problem.

Finally, evaluation of our algorithm should be done through comparison with other methods or formalisms. How do the H and $N(H)$ used in our main theorem relate to the pieces (Salvat 1998) used to optimize backward chaining ? How do our algorithm relate to the magic sets approach to

optimize the chasing algorithm solving constrained tuple-generating dependencies in databases (Maher & Srivastava 1996) ?

Acknowledgments

We would like to thank Marie-Laure Mugnier, Marie-Christine Rousset and Pierre Marquis for their precious comments and advices on an earlier version of this work, as well as the anonymous referees for their helpful critics and suggestions.

References

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

Baget, J.-F., and Mugnier, M.-L. 2002. Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research* 16:425 – 465. `http://www.cs.washington.edu/research/jair/contents/v16.html`.

Baget, J.-F.; Genest, D.; and Mugnier, M.-L. 1999. Knowledge Acquisition with a Pure Graph-Based Knowledge Representation Model – Application to the SISYPHUS-I Case Study. In Gaines, B. R.; Musen, M. A.; and Kremer, R. C., eds., *Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Banff, Alberta, Canada, October 16–21, 1999*. Online proceedings at `http://sern.ucalgary.ca/KSI/KAW/KAW99/papers.html`.

Baget, J.-F. 2001. *Représenter des connaissances et raisonner avec des hypergraphes: de la projection à la dérivation sous contraintes*. Ph.D. Dissertation, Université de Montpellier II.

Baget, J.-F. 2003. Simple conceptual graphs revisited: Hypergraphs and conjunctive tupes for efficient projection algorithms. In de Moor, A.; Lex, W.; and Ganter, B., eds., *Conceptual Structures for Knowledge Creation and Communication, 11th International Conference on Conceptual Structures, ICCS 2003, Dresden, Germany, July 21–25, 2003, Proceedings*, volume 2746 of *Lecture Notes in Artificial Intelligence*, 229 – 242. Springer.

Chein, M., and Mugnier, M.-L. 1992. Conceptual Graphs: fundamental notions. *Revue d'Intelligence Artificielle* 6(4):365–406.

Coulondre, S., and Salvat, Éric. 1998. Piece Resolution: Towards Larger Perspectives. In Mugnier and Chein (1998), 179 – 193.

Genest, D., and Salvat, Éric. 1998. A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT. In Mugnier and Chein (1998), 154 – 161.

Maher, M. J., and Srivastava, D. 1996. Chasing constrained tuple-generating dependencies. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, 128–138. ACM Press.

Mugnier, M.-L., and Chein, M. 1996. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle (numéro spécial "Graphes Conceptuels")* 10(1).

Mugnier, M.-L., and Chein, M., eds. 1998. *Conceptual Structures: Theory, Tools and Applications, 6th International Conference on Conceptual Structures, ICCS '98, Montpellier, France, August 10-12, 1998, Proceedings*, volume 1453 of *Lecture Notes in Computer Science*. Springer.

Mugnier, M.-L. 2000. Knowledge Representation and Reasonings Based on Graph Homomorphism. In Ganter, B., and Mineau, G. W., eds., *Conceptual Structures: Logical, Linguistic, and Computational Issues, 8th International Conference on Conceptual Structures, ICCS 2000, Darmstadt, Germany, August 14–18, 2000, Proceedings*, volume 1867 of *Lecture Notes in Computer Science*, 172 – 192. Springer.

Salvat, Éric., and Mugnier, M.-L. 1996. Sound and Complete Forward and Backward Chainings of Graphs Rules. In Eklund, P. W.; Ellis, G.; and Mann, G., eds., *Conceptual Structures: Knowledge Representation as Interlingua, 4th International Conference on Conceptual Structures, ICCS '96, Sydney, Australia, August 19-22, 1996, Proceedings*, volume 1115 of *Lecture Notes in Computer Science*, 248 – 262. Springer.

Salvat, Éric. 1998. Theorem Proving Using Graph Operations in the Conceptual Graph Formalism. In Prade, H., ed., *13th European Conference on Artificial Intelligence, Brighton, UK, August 23-28 1998, Proceedings*, 356 – 360. John Wiley and Sons.

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