Conceptual Graph Rules and Equivalent Rules: A Synthesis

Marie-Laure Mugnier

LIRMM (CNRS - University of Montpellier),
161, rue Ada, F-34392 Montpellier cedex, France
mugnier@lirmm.fr

Abstract. This paper is an extended abstract of the talk given at ICCS'09. Rules have long been considered as an essential component of knowledge-based systems. We focus here on conceptual graph rules and on the semantically equivalent knowledge constructs in logic and databases, namely rules with existential variables and tuple-generating dependencies. The aim of this presentation is to synthesize main decidability, complexity and algorithmic results obtained on this kind of rules. We emphasize the fact that the graph vision of rules has led to new results.

1 Introduction

Rules have long been considered as an essential component of knowledge-based systems. In this talk, we focus on rules in conceptual graphs (CG) and on the equivalent knowledge constructs in logic and databases. For precise definitions of all conceptual graph notions used in this presentation, we refer to [CM08].

A conceptual graph rule (in short $R : H \rightarrow C$) can be seen as a pair (H, C) of basic conceptual graphs, provided with a one to one correspondence between a subset of generic nodes in H and a subset of generic nodes in C. H and C are respectively called the *hypothesis* and the *conclusion* of the rule. The distinguished nodes in H and C are called *connection nodes*. Figure 1 shows a CG rule (pictured with Cogui[1]). The correspondence between connection nodes is visualized by dotted lines. The logical translation of this rule is $\forall x \forall y(Person(x) \wedge Person(y) \wedge siblingOf(x, y) \rightarrow \exists z(Person(z) \wedge Parent(z, x) \wedge Parent(z, y))$. This kind of logical rule is more general than the (positive) rules usually considered in logic programming or deductive databases. Indeed, there might be variables in the conclusion which are *existentially* quantified, hence the name $\forall\exists$-rule given to this kind of formula in [BLMS09].

A $\forall\exists$-rule has the same form as a very general kind of dependency studied in databases called tuple-generating dependency (TGD) [AHV95]. It can also be seen as an abstraction for ontological knowledge expressed in specific knowledge representation languages, f.i. the RDFS rules [Hay04], constraints in F-logic-Lite [CK06][CGK08], as well as some kinds of inclusions in description logics [BCM$^+$03].

[1] http://www.lirmm.fr/cogui

S. Rudolph, F. Dau, and S.O. Kuznetsov (Eds.): ICCS 2009, LNAI 5662, pp. 23–31, 2009.

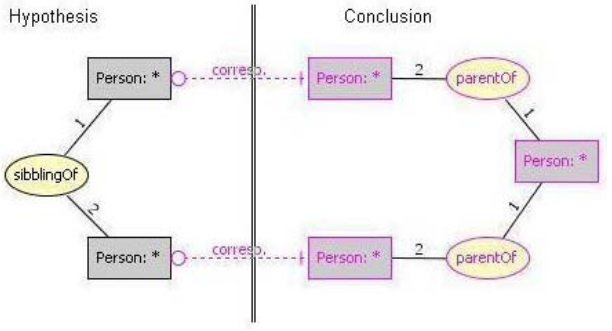

Fig. 1. A conceptual graph rule

Let us point out that, in conceptual graphs, concept types and relations are ordered by a specialization relation, and that the processing of this order is directly integrated in conceptual graph mechanisms. This feature does not add expressivity with respect to $\forall\exists$-rules, since specialization orders can be translated into simple rules of form $\forall x_1...x_k(t_1(x_1...x_k) \rightarrow t_2(x_1...x_k))$, where t_1 is a specialization of t_2, $k = 1$ if t_1 and t_2 are concept types, otherwise k is the arity of the relations. However, the specialization orders are managed with simple label comparisons, which leads to more efficient knowledge processing mechanisms.

The aim of this presentation is to synthesize theoretical and algorithmic results obtained on conceptual graph rules, as well as on the semantically equivalent knowledge constructs in logic and databases, namely $\forall\exists$-rules and tuple-generating dependencies. We emphasize the fact that the graph vision of rules has led to new results.

2 Deduction with CG Rules

A conceptual graph vocabulary, also called *support*, contains finite ordered sets of concept types and of relations (as well as a set of individual markers, relation signatures, assertions of disjointness between concept types, ...). It can be seen as a very basic ontology. Basic conceptual graphs (BG) are used to express facts and queries. They are logically translated into existentially closed conjunctions of atoms. A BG itself can express a boolean database conjunctive query (i.e. with a yes/no answer) and, when generic concept nodes are distinguished to represent the answer part of the query, it is equivalent to a general conjunctive query. Let us consider conceptual graph knowledge bases (KBs) composed of a vocabulary, a set of facts (which can also be seen as a single fact) and a set of rules. Several fundamental problems on these KBs are computationally equivalent, namely *fact deduction* (is a fact deducible from a KB?), *rule deduction* (is a rule deducible from a KB?) and *boolean conjunctive query answering* in presence of incomplete knowledge (is a boolean conjunctive query deducible from a KB ?). Very simple polynomial reductions allow to go from one problem to another. Since a fact can

be seen as a rule with an empty hypothesis, fact deduction is a particular case of rule deduction. In turn, rule deduction can be reduced to fact deduction. The following transformation comes from [BV84] (and was applied to TGDs). Let K be the KB and R be the rule for which we want to know if it is deducible from the KB. Let $R' : H' \rightarrow C'$ be obtained from R by replacing, in each pair of corresponding connection nodes, the generic marker by a new individual marker that does not appear in K nor R. Let K' be obtained from K by adding the new fact H'. Then, R is deducible from K if and only if the fact C' is deducible from K'. Since a fact has the same form as a boolean conjunctive query, the equivalence of fact deduction and boolean query answering is immediate. From now on, we focus on *fact deduction*, which we simply call DEDUCTION (and all results concerning this problem can be immediately recast in terms of the other problems).

There are two classical ways of processing rules. *Forward chaining* starts from the facts and applies rules to facts to produce new facts. A derivation is a sequence of rule applications leading from an initial fact to an enriched fact. *Backward chaining* starts from a question, usually called a goal, and tries to build a derivation leading to an answer to this goal in a backward manner. We assume that the reader is familiar with both paradigms.

Conceptual graph rules are provided with sound and complete forward and backward chaining mechanisms, which operate directly on their graph form. For forward chaining, the basic notion is the BG homomorphism, classically called projection in the CG community (however, we do prefer to use the term homomorphism because it relates this notion to relational algebra and graph theory; moreover, the CG projection may be confused with the projection operator in relational algebra). The fundamental property is that BG homomorphism is sound and complete with respect to logical deduction [CM92]: given two BGs G and H built on a vocabulary $\mathcal{V}$, there is a homomorphism from G to H if and only if $\Phi(G)$ can be deduced from $\Phi(H)$ and $\Phi(\mathcal{V})$ (for the completeness part, either H has to be in a normal form, or a variant of homomorphism can be used to avoid this normality condition [CM04]). Homomorphism checking is NP-complete. A rule $R : H \rightarrow C$ can be applied to a fact F if there is a homomorphism h from H to F. Applying R to F according to h consists of "adding" C to F in a way defined by h (each connection node in C is merged with the image by h of the corresponding connection node in H). This yields a sound and complete mechanism: given a KB K composed of a vocabulary $\mathcal{V}$, a fact F and set of rules $\mathcal{R}$, and a query Q (also built on $\mathcal{V}$), there is a derivation from F to a fact F' using rules of $\mathcal{R}$, and a homomorphism from Q to F', if and only if $\Phi(Q)$ can be deduced from $\Phi(K)$ (i.e. $\Phi(\mathcal{V}) \cup \Phi(\mathcal{R}) \cup \{\Phi(F)\}$) [SM96].

Backward chaining relies on a *unification* operation, between part of a current goal and a rule conclusion. In logic programming, the conclusion of a rule consists of a single atom, thus unification involves one atom of the goal and the atom of a conclusion. [GW95] proposed a sound and complete backward mechanism for conceptual graph rules very similar to this mechanism. The goal is split into trivial subgraphs composed of a relation node and its neighbors. Then,

unification involves a trivial subgraph of the goal and an atom of the conclusion. In [SM96], a more complex unification operation is defined, which aims at exploiting the complex structure of a conceptual graph rule: it allows one to process conclusions and goals without decomposing them into trivial subgraphs. This mechanism will be detailed in section 4.

It is easily checked that forward chaining may not halt, even with criteria avoiding redundant applications of rules. Backward chaining may not halt either. The fundamental reason is that DEDUCTION is not decidable, but only semi-decidable. This has been first proven for TGDs in [BV84]. Two other proofs for CGs can be found in [Bag01] (with a reduction from the halting problem of a Turing machine, which proves that DEDUCTION with rules is a computation model) and in [BM02] (with a reduction from the word problem in a semi-thue system). It thus important to define large and meaningful cases in which the problem is decidable. Decidable cases may be defined by an *abstract* property which guarantees decidability. However, such an abstract property is generally not provided with a finite procedure allowing to determine whether a given set of rules has the property or not. The next step is thus to exhibit *concrete* cases, which fulfill the abstract property and can be recognized by a finite procedure. The conditions defining concrete cases may be relative to each rule independently or to a set of rules.

Obviously, if the forward chaining is guaranteed to halt with a kind of rules, then DEDUCTION is decidable in this particular case. This leads to the following abstract property: a set of rules is called a *finite expansion* set if it is guaranteed, for any fact, that after a finite number of rule applications, all further rule applications will become redundant, i.e. will produce facts equivalent to the current fact; DEDUCTION is decidable for finite expansion sets of rules [BM02]. Two concrete cases of finite expansion sets of rules are range-restricted rules and disconnected rules. A *range-restricted* rule is such that all concept nodes of the conclusion are either connection nodes or nodes with an individual marker (in logical terms: it is a $\forall\exists$-rule without existentially quantified variable in the conclusion, which corresponds to a range-restricted rule in positive Datalog). A *disconnected* rule has no connection nodes. DEDUCTION with a set of range-restricted rules or a set of disconnected rules is NP-complete (assuming that the arity of relations is bounded), thus is in the same complexity as DEDUCTION with facts only, which involves a simple homomorphism test.

In [BLMS09], a similar abstract property is exhibited in relation with backward chaining. Given a goal Q and a rule R, a *rewriting* of Q with R is a graph obtained by a unification of Q with the conclusion of R. A set of rules $\mathcal{R}$ is called a *finite unification* set if it is guaranteed, for any goal, that there is a finite set $\mathcal{Q}$ of rewritings of Q with rules in $\mathcal{R}$, such that any other possible rewriting of Q is more specific than an element of $\mathcal{Q}$. Two concrete cases of finite unification sets are exhibited in [BLMS09]: atomic hypothesis rules and domain restricted rules. In an *atomic hypothesis rule*, the hypothesis contains a single atom. These rules are particularly well adapted to express necessary properties of concepts or relations in ontological languages, without any restriction on the form of the conclusion (i.e. rules of form $C(x) \rightarrow P$ or $r(x_1 \ldots x_k) \rightarrow P$, where C is a concept

type, r a k-ary relation and P any set of atoms). The second kind of rules does not put any restriction on the form of the hypothesis but constrains the form of the conclusion: in a *domain restricted* rule, each atom of the conclusion contains all or none of the variables in the hypothesis. The complexity of DEDUCTION for these particular kinds of rules has not been studied yet.

Other decidable cases are not based on individual properties of rules but on interactions between rules and will be presented in the next sections.

3 Equivalent Problems in Databases

Tuple-generating dependencies (TGDs) are a very general class of dependencies, encoding most dependencies in databases [AHV95]. They have exactly the same logical form as $\forall\exists$-rules. If a TGD is not satisfied by a database instance, it is possible to repair the database instance by extending it with new atoms. The procedure that enforces the validity of a set of TGDs is called the *chase*: it is equivalent to forward chaining. The chase was first introduced for the *TGD implication problem*: given a set of TGDs T, and a TGD t, is t implied by T? (this problem is the same as the above rule deduction problem). A related problem is the *query containment problem* under a set of TGDs: given a set of TGDs T, and two conjunctive queries q_1 and q_2, is the the set of answers to q_1 included in the set of answers to q_2 for any database satisfying T (i.e. satisfying each TGD in T) ? A problem more recently introduced is *query answering on incomplete data* [CLR03]: given a set of TGDs T, a database instance D, that may not satisfy T, a conjunctive query q and a tuple of values t, is t an answer to q in a database instance obtained from D by enforcing T ? All these problems can be proven equivalent to DEDUCTION.

Interestingly, very recent results have exhibited classes of TGDs for which the problem is decidable even if the chase does not halt [CGK08]. The abstract property is that when all facts generated by a set of rules have a "bounded treewidth" then DEDUCTION is decidable. Note that this class of rules includes finite expansion sets, but not finite unification sets. Concrete cases of rules satisfying this abstract property are the guarded TGDs, and their generalization to weakly guarded TGDs. A TGD is *guarded* if its body (i.e. hypothesis) includes an atom, called guard, that contains all variables occurring in the body. *Weakly guarded* TGDs are an extension of guarded TGDs that requires guards to contain only some variables in the body (see [CGK08] for a precise definition). This property cannot be checked independently on each rule, but requires to consider the whole set of rules. It is shown that the problem is EXPTIME-complete (with the assumption of bounded predicate arities) for weakly guarded TGDs, and even for guarded TGDs. It becomes NP-complete when, moreover, the number of predicates appearing in the TGDs is bounded.

4 Graph Rules: The Added Value

In this section, we focus on backward chaining and on how the graph structure allows to obtain new results. Graphs are a natural construct for representing

complex structures. In previously cited results on TGDs, conclusions of rules are restricted to one atom, as it is usually the case in logic programming. This restriction does not lead to a loss of expressivity since any set of rules can be rewritten (in linear time) as a set of rules with one atom in conclusion. Indeed, a rule $H \rightarrow C$ can be equivalently encoded by a set of rules $\{H \rightarrow R(t_1, ..., t_k), (R(t_1, ..., tk) \rightarrow A_c)_{A_c \in C})\}$, where R is a new predicate assigned to the rule and $t_1...t_k$ are the terms occurring in C. However, beside loss in readability, this rewriting leads to a loss in efficiency and weaker decidability results [BLMS09].

The sound and complete backward chaining outlined in this section is based on the notions of a *piece* and the associated *piece-based unification* [SM96]. Let us mention that these notions have been defined for conceptual graph rules obeying two constraints. First, two corresponding connection nodes have the same type. Secondly, an individual marker always occurs with the same concept type. As a consequence of these restrictions, a rule application to a fact F never restricts labels of existing nodes in F. It only adds new nodes to F. These restrictions do not lead to a loss in expressivity in the sense that concept types can be equivalently represented as unary relations. However, to work directly on general conceptual graph rules, the notions of piece and piece-based unification presented hereafter would need to be extended. These restrictions do not apply in a logical setting, since there is no distinction between concept types and relations, which are all predicates.

A *cutpoint* of a rule is either a connection node or a node with an individual marker. A *piece* of a rule is a (non empty) subgraph of its conclusion, in which any two nodes are connected by a path that does not go *through* a cutpoint (however, a cutpoint can be an extremity of such a path), and to which no more nodes can be added while preserving this property. A way of understanding pieces is as follows: assume that all cutpoints of the rule conclusion are deleted; each connected component obtained after this deletion belongs to a separate piece; each piece itself is obtained from such a connected component by adding again the cutpoints linked to its nodes (if any) as well as the associated edges. For instance, the rule in Figure 1 has two cutpoints and a single piece. These graph notions can be translated into logical notions in a straightforward way (see [BLMS09]). For instance, the rule $R = \forall x \forall y (p(x, y) \rightarrow \exists z \exists t \exists u (p(x, z) \wedge p(z, t) \wedge p(t, x) \wedge p(x, u)))$ has one cutpoint, which is x, and two pieces defined by the sets of atoms $\{p(x, z), p(z, t), p(t, x)\}$ and $\{p(x, u)\}$.

The idea behind piece is that a piece can be seen as a "unit" of knowledge brought by a rule application in forward chaining. Indeed, a rule R can be decomposed into an equivalent set of rules with the same hypothesis and exactly one piece in conclusion. More precisely, any rule $R : H \rightarrow C$, such that C contains k pieces $C_1, \ldots, C_k$, is equivalent to the set of rules $\{H \rightarrow C_i\}_{1 \leq i \leq k}$. Moreover, the conclusions of these rules cannot be further decomposed while keeping a set of conceptual graph rules with the same semantics as R (provided of course that H is not modified, otherwise see the above decomposition).

We do not recall the definition of piece-based unification, which would require more technical developments (see [SM96] [CM08] in the CG framework, and [BLMS09] for $\forall\exists$-rules). The important point for the backward chaining mechanism is that piece-based unification allows it to be guided by the structure of rules and goals. An experimental comparison between piece-based backward chaining and Prolog resolution was led in [CS98].

Another important point is that it allows to characterize exactly the notion of *dependency* between rules. Generally speaking, compiling a knowledge base consists in preprocessing it off-line, so that the compiled form obtained can be used on-line to accelerate reasoning tasks (e.g. query answering). Concerning rules, a classical compilation technique consists in precomputing a graph encoding dependencies between rules. This technique allows us to improve the efficiency of forward and backward chaining mechanisms.

Given rules R_1 and R_2, R_2 is said to depend on R_1 if the application of R_1 on a fact may trigger a *new* application of R_2, i.e. if there exists a fact F to which R_1 can be applied leading to a fact F', such that there is a homomorphism from the hypothesis of R_2 to F', that is not a homomorphism from the hypothesis of R_2 to F. It is easy to define *necessary* conditions for a rule to depend from another: f.i. if R_2 depends on R_1 then there is an atom in H_2 that can be unified (in the logical classical meaning) with an atom in C. Characterizing dependency by effectively computable *necessary and sufficient* conditions is less obvious.

Piece-based unification yields such an effective characterization: R_2 depends on R_1 if and only if there is a piece-based unification of H_2 with R_1 (see [Bag04]) for an equivalent characterization restricted to rules without constants, [BS06] for this result on conceptual graph rules, and [BLMS09] for this result in a logical framework).

Given a set of rules $\mathcal{R}$, the *graph of rule dependencies* (GRD) of $\mathcal{R}$ is the directed graph with node set $\mathcal{R}$, and such that there is a (directed) edge from R_1 to R_2 if and only if R_2 depends on R_1 ("an application of R_1 may trigger a new application of R_2"). As far as we know, piece unification yields the first effective characterization of this graph. Very recently, in the context of databases, [DNR08] defined a notion equivalent to the GRD on TGDs ("the chase graph"), but no constructive characterization of this graph was provided in this paper.

The GRD has two interests. It allows to speed up forward or backward chaining and it leads to new decidability results. About the first point, a simple use of the GRD is the following. Assume that the set of facts is considered as a single graph, say F, and classified in the GRD as a rule with an empty hypothesis. F is necessarily a source (node without ingoing edge) in the GRD. The query or goal, say Q, can also be classified in the GRD as a rule with an empty conclusion. Q is necessarily a sink (node without outgoing edge) in the GRD. Then, to answer a given Q, the only rules to consider are the rules corresponding to nodes in the GRD on a path from F to Q. Let us consider a basic forward chaining algorithm, that proceeds in a breadth-first way, i.e. at each step it computes all new

(and non redundant) rule applications w.r.t the current fact, then applies them producing a new fact. The rules to consider in the first step are the successors of F in the GRD. Then, the rules to consider at a given step i, $i > 1$, are the rules successors of rules applied at step $i - 1$. For further improvements of the forward and backward chaining mechanisms, see [Bag04] (forward chaining) and [BS06] (both mechanisms).

Concerning decidability results, the structure of the GRD provides information of how the rules interact with each other. It can be easily checked that if the GRD has no circuit (including no loop), then DEDUCTION is decidable. This result can be extended to a GRD in which each strongly connected component[2] is a finite expansion set of rules, i.e. circuits inside a finite expansion set of rules are allowed [Bag04]. A similar result holds for a GRD in which each strongly connected component is a finite unification set of rules [BLMS09]. Note that DEDUCTION may not be decidable in a GRD where each strongly connected component is either a finite expansion set or a finite unification set. However, there is a way of combining both notions that guarantees a finite procedure [BLMS09]: assume that the set of rules $\mathcal{R}$ can be partitioned into two sets, $\mathcal{R}_1$ and $\mathcal{R}_2$, such that $\mathcal{R}_1$ is a finite expansion set, $\mathcal{R}_2$ is a finite unification set, and there is no edge from a rule in $\mathcal{R}_2$ to a rule in $\mathcal{R}_1$ in the GRD; in this case, one can first use forward chaining on F with $\mathcal{R}_1$, which leads to a fact F'; then, backward chaining is used on F' and $\mathcal{R}_2$ to compute a set of rewritings of Q and check if there is a homomorphism from a rewriting in this set to F'; the mechanism obtained halts in all cases, and is sound and complete [BLMS09]. This result can be extended by combining it with the results in [CGK08]: see [BLMS09], which also provides a map of all known decidable cases.

Let us end by emphasizing the role of the piece notion in these results, which is a natural notion when rules are considered in their graph form because it relies on the path notion. Of course, it can be translated into a logical setting (see [BLMS09]) but it does not rely on logical notions.

5 Conclusion

In this presentation, we have synthesized main decidability and complexity results obtained on a kind of rules which takes several forms in the literature, namely conceptual graph rules, $\forall\exists$-rules and TGDs. We have shown that the graph vision of rules can lead to new notions and results. Some of the decidability results for concrete cases still have to be completed by complexity results. Further work also includes the design of forward and backward chaining algorithms exploiting as much as possible the graph of rule dependencies.

[2] Two nodes x and y are in the same strongly connected component if there is a path from x to y and a path from y to x; strongly connected components in the GRD represent maximal sets of rules that mutually depend, directly or indirectly, on each other.

References

[AHV95] Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)

[Bag01] Baget, J.-F.: Représenter des connaissances et raisonner avec des hypergraphes: de la projection à la dérivation sous contraintes. PhD thesis, Université Montpellier II (November 2001)

[Bag04] Baget, J.-F.: Improving the forward chaining algorithm for conceptual graphs rules. In: KR, pp. 407–414. AAAI Press, Menlo Park (2004)

[BCM+03] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)

[BLMS09] Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: Extending decidable cases for rules with existential variables. In: Proc. of IJCAI 2009 (to appear, 2009)

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

[BS06] Baget, J.-F., Salvat, E.: Rules dependencies in backward chaining of conceptual graphs rules. In: Schärfe, H., Hitzler, P., Øhrstrøm, P. (eds.) ICCS 2006. LNCS, vol. 4068, pp. 102–116. Springer, Heidelberg (2006)

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

[CGK08] Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: KR, pp. 70–80 (2008)

[CK06] Calì, A., Kifer, M.: Containment of conjunctive object meta-queries. In: VLDB, pp. 942–952 (2006)

[CLR03] Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: PODS, pp. 260–271. ACM Press, New York (2003)

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

[CM04] Chein, M., Mugnier, M.-L.: Concept types and coreference in simple conceptual graphs. In: Wolff, K.E., Pfeiffer, H.D., Delugach, H.S. (eds.) ICCS 2004. LNCS (LNAI), vol. 3127, pp. 303–318. Springer, Heidelberg (2004)

[CM08] Chein, M., Mugnier, M.-L.: Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs. In: Advanced Information and Knowledge Processing. Springer, Heidelberg (2008)

[CS98] Coulondre, S., Salvat, E.: Piece Resolution: Towards Larger Perspectives. In: Mugnier, M.-L., Chein, M. (eds.) ICCS 1998. LNCS (LNAI), vol. 1453, pp. 179–193. Springer, Heidelberg (1998)

[DNR08] Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: PODS, pp. 149–158 (2008)

[GW95] Ghosh, B.C., Wuwongse, V.: A Direct Proof Procedure for Definite Conceptual Graphs Programs. In: Ellis, G., Rich, W., Levinson, R., Sowa, J.F. (eds.) ICCS 1995. LNCS (LNAI), vol. 954, pp. 158–172. Springer, Heidelberg (1995)

[Hay04] Hayes, P. (ed.): RDF Semantics. W3C Recommendation. W3C (2004)

[SM96] Salvat, E., Mugnier, M.-L.: Sound and Complete Forward and Backward Chainings of Graph Rules. In: Eklund, P., Mann, G.A., Ellis, G. (eds.) ICCS 1996. LNCS (LNAI), vol. 1115, pp. 248–262. Springer, Heidelberg (1996)