Rules Dependencies in Backward Chaining of Conceptual Graphs Rules

Jean-François Baget, Eric Salvat

Rules Dependencies in Backward Chaining of Conceptual Graphs Rules

Jean-François Baget[1] and Éric Salvat[2]

[1] INRIA Rhône-Alpes jean-francois.baget@inrialpes.fr
[2] IMERIR salvat@imerir.com

Abstract. Conceptual Graphs Rules were proposed as an extension of Simple Conceptual Graphs (CGs) to represent knowledge of form "IF A THEN B", where A and B are simple CGs. Optimizations of the deduction calculus in this KR formalism include a Backward Chaining that unifies at the same time whole subgraphs of a rule, and a Forward Chaining that relies on compiling dependencies between rules.
In this paper, we show that the unification used in the first algorithm is exactly the operation required to compute dependencies in the second one. We also combine the benefits of the two approaches, by using the graph of rules dependencies in a Backward Chaining framework.

1 Introduction

Conceptual graphs (CG) rules [14] were proposed as an extension of simple CGs [13] to represent knowledge of form "IF A THEN B", where A and B are simple CGs. This graph-based knowledge representation (KR) formalism (named $\mathcal{SR}$ in [3]) was further formalized in [12]. Notwithstanding the interest of graphical representation of knowledge for an human interaction purpose, we are mainly motivated in using the graph structure of CGs to improve deduction algorithms, that are sound and complete w.r.t. the FOL semantics Φ [14] of the language. Using graph-theoretical operations to compute reasonings, instead of translating CGs into their equivalent formulaes and use a FOL solver, the algorithms presented in this paper explore a different optimization paradigm in KR.

Simple CGs [13] form the basic KR formalism (named $\mathcal{SG}$ in [3]) on which CG rules are built. The semantics Φ identifies them with formulaes in positive, conjunctive, existential FOL (without function symbols) [14]. Sound and complete reasonings in $\mathcal{SG}$ (a NP-hard problem) can be computed with a kind of graph homomorphism named *projection* [5]. This graph-theoretical operation is the kernel of reasonings for extensions of simple CGs (*e.g.* the $\mathcal{SG}$ family [3]).

In particular, projection is the elementary operation in Forward Chaining (FC) of CG rules [12], a graph-based algorithm computing deduction in $\mathcal{SR}$. Since CG Rules can be translated into FOL formulaes having the form of Tuple Generating Dependencies (TGDs) [7], $\mathcal{SR}$-DEDUCTION is semi-decidable.

A Backward Chaining (BC) framework is often used to avoid a major pitfall in FC: applying rules that are unrelated to the query. Though CG Rules deduction

can be computed using a PROLOG-like BC algorithm, successively unifying predicate after predicate in the equivalent FOL formulaes, [11] proposed to rely upon the structure of the graph and unify at the same time whole subgraphs of the rule (called pieces), effectively reducing the number of backtracks [7].

To optimize FC (more adapted to some extensions of simple CGs, *e.g.* [3]), [4] introduces the notion of neutrality (and its complementary notion, dependency) between rules: a CG Rule R_1 is neutral w.r.t. a rule R_2 if no application of R_1 on a CG can create a new application of R_2. Building the graph of rules dependencies (GRD) compiles enough information to reduce the number of checks for rule applicability as well as the cost of these checks in FC.

In this paper, we show that the criterium used to compute dependencies in [4] and the piece unification of [11] are very similar operations. In particular, we show that piece unification generalizes computation of dependencies to rules having individual markers in their conclusion (excluded in [4]). On the other hand, we generalize piece unification to any type hierarchy (and not only lattices, as in [11]). We propose solutions to use the GRD in a BC framework.

Organization of the paper Sect. 2 and 3 are respectively devoted to simple CGs (the $\mathcal{SG}$ language) and CG rules ($\mathcal{SR}$). We present the syntax, the semantics (via the translation Φ to FOL), and a sound and complete calculus (projection in the first case, basic FC in the latter) of both languages. The first enhancement of $\mathcal{SR}$-DEDUCTION, the BC based upon piece unification [12, 11], is presented in Sect. 4. The graph of rules dependencies (GRD) [4], its use in FC, and its relationships with piece unification, are presented in Sect. 5. Finally, in Sect. 6, we show how to efficiently use the GRD in a BC framework.
Caveat: For space requirements, no examples are included in this paper, they can be found in the references. For the same reason, non essential technical details will be printed in small.

2 Simple Conceptual Graphs

We recall fundamental results on simple CGs (without coreference links) [13, 14]. Sect. 2.1 presents their syntax, and Sect. 2.2, their semantics via the transformation Φ into FOL [14]. We use these formulas to define simple CGs' deduction problem ($\mathcal{SG}$-DEDUCTION in [3]). In Sect. 2.3, we use the graph homomorphism named projection [5] as a calculus for $\mathcal{SG}$-DEDUCTION. Up to a normality condition [10], projection is sound and complete w.r.t. the semantics Φ.

2.1 Syntax

Definition 1 (Vocabulary). *A* vocabulary *is a tuple* $(T_C, (T_R^1, \ldots, T_R^N), \mathcal{I}, \kappa)$ *where* $T_C, T_R^1, \ldots, T_R^N$ *are pairwise disjoint partially ordered sets (partial orders are denoted by* $\leq$*),* $\mathcal{I}$ *is a set, and* $\kappa : \mathcal{I} \rightarrow T_C$ *is a mapping. Elements of* T_C *are called* concept types*, elements of* T_R^i relation types *of* arity i*, elements of* $\mathcal{I}$ individual markers*, and* κ *is the* conformity relation*.*

Definition 2 (Simple CGs). *A* simple CG *over a vocabulary $\mathcal{V}$ is a tuple $G = (E, R, \epsilon, \gamma)$ where E and R are two disjoint sets, respectively of* entities *and* relations. *The mapping ϵ labels each entity of E by a pair of $T_C \times (\mathcal{I} \cup \{*\})$ (its* type *and* marker*). An entity whose marker is $*$ is called* generic*, otherwise it is an* individual. *For each individual $x \in E$, $\text{type}(x) = \kappa(\text{marker}(x))$. The mapping ϵ also labels each relation of R by a relation type (its* type*). We call* degree *of a relation the arity of its type. The mapping γ maps each relation of degree k to a k-tuple of E^k. If $\gamma(r) = (x_1, \ldots, x_k)$ we denote by $\gamma_i(r) = x_i$ the ith* argument *of r. If x and y are two arguments of r, x and y are* neighbours.

Simple CGs can be seen both as *bipartite multigraphs*, as in [5] ($\gamma_i(r) = e$ means that there is an edge labelled i between the concept node e and the relation node r); or as *directed multiple hypergraphs*, as in [2] ($\gamma(r) = (x_1, \ldots, x_k)$ is a directed hyperarc whose ends are the concept nodes $x1, \ldots, x_k$).

Whatever the structure used to encode them, they share the same drawing. An entity e with $\epsilon(e) = (t, m)$ is represented by a rectangle enclosing the string "t: m". A relation r typed t is represented by an oval enclosing the string "t". If $\gamma(r) = (x_1, \ldots, x_k)$, then for $1 \leq i \leq k$, we draw a line between the oval representing r and the rectangle representing x_i, and write the number i next to it.

2.2 Semantics

In logics, semantics are provided to evaluate if a formula is true. Then we define H as a *logical consequence* of G iff H is true whenever G is true. These semantics are defined by sets over a domain. Simple CGs semantics are often expressed via a translation Φ through first-order logics [14], and deduction of simple CGs is defined as the logical consequence of the associated formulas. Model theoretic semantics of these formulas can be directly expressed on the simple CGs [10].

Interpretation of a vocabulary Let $\mathcal{V}$ be a vocabulary. Its interpretation $\Phi(\mathcal{V})$ is a FOL formula built as follows.

We first associate to each concept type $t_c \in T_C$ a distinct predicate name of arity 1, also noted t_c; to each relation type $t_r^i \in T_R^i$ a distinct predicate name of arity i, also noted t_r^i; and to each individual marker $i \in \mathcal{I}$ a distinct constant symbol, also noted i. Let us now consider the partial orders noted $\leq$ on $T_C, T_R^1, \ldots, T_R^N$. We call *covering relation* and note $\prec$ the smallest relation whose reflexo-transitive closure is $\leq$ (*i.e.* $t \prec t'$ iff $t \leq t'$, $t \neq t'$, and there is no t'' s.t. $t'' \neq t$, $t'' \neq t'$ and $t \leq t'' \leq t'$). A *covering pair* (t, t') in $\mathcal{V}$ is a pair of types in $T_C, T_R^1, \ldots$ or T_R^N s.t. $t \prec t'$. The interpretation of a covering pair (t, t') is the first-order logic formula $\phi((t, t')) = \forall x_1 \ldots \forall x_k (t(x_1, \ldots, x_k) \rightarrow t'(x_1, \ldots, x_k))$, where k is the arity of the predicate names t and t'. Finally, the interpretation $\Phi(\mathcal{V})$ of the vocabulary $\mathcal{V}$ is the conjunction of the interpretations $\phi((t, t'))$ of all covering pairs (t, t') in $\mathcal{V}$.

Interpretation of a simple CG Let $G = (E, R, , \epsilon, \gamma)$ be a simple CG over $\mathcal{V}$. The interpretation $\Phi(G)$ of G is the FOL formula built as follows.

Predicate names and constant symbols are obtained from types and markers as above. We first associate to each generic entity $e \in E$ a distinct variable name $f(e)$. If e is an individual, we define $f(e) = \mathit{marker}(e)$. The interpretation of an entity e with $\mathit{type}(e) = t$ is the predicate $\phi(e) = t(f(e))$. The interpretation of a relation $r \in R$ with $\mathit{type}(r) = t$ and $\gamma(r) = (e_1, \ldots, e_k)$ is the predicate $\phi(r) = t(f(e_1), \ldots, f(e_k))$. The interpretation $\Phi(G)$ of the simple CG G is the existential closure of the conjunction of the interpretations $\phi(x)$, for all $x \in E \cup R$.

Definition 3 (Deduction in $\mathcal{SG}$). *Let G and H be two simple CGs over a vocabulary $\mathcal{V}$. We say that G entails H in $\mathcal{V}$ (and note $G \models_{\mathcal{V}} H$) iff $\Phi(H)$ is a logical consequence of $\Phi(G)$ and $\Phi(\mathcal{V})$ (*i.e. $\Phi(\mathcal{V}), \Phi(G) \vdash \Phi(H)$*).*

2.3 Calculus

Definition 4 (Projection). *Let G and H be two simple CGs over a vocabulary $\mathcal{V}$, with $G = (E_G, R_G, \epsilon_G, \kappa_G)$ and $H = (E_H, R_H, \epsilon_H, \kappa_H)$. A* projection *from H into G (according to $\mathcal{V}$) is a mapping $\pi : E_H \to E_G$ such that:*

- *For each entity $e \in E_H$, $\mathit{type}(\pi(e)) \leq \mathit{type}(e)$. If, moreover, e is an individual, then $\mathit{marker}(\pi(e)) = \mathit{marker}(e)$.*
- *For each relation $r \in R_H$, with $\gamma_H(r) = (x_1, \ldots, x_p)$, there exists a relation $r' \in R_G$ such that $\mathit{type}(r') \leq \mathit{type}(r)$ and $\gamma_G(r') = (\pi(x_1), \ldots, \pi(x_k))$.*

Complexity As a generalization of GRAPH HOMOMORPHISM, PROJECTION is an NP-complete problem. As indicated in [2], thanks to a reduction from constraint networks (CSP) that preserves the structure of the projected graph [9], all structural restrictions on CSPs leading to polynomial algorithms (*e.g.* hypertrees decompositions in [8]) translate to structural restrictions on the projected simple CG leading to polynomial algorithms. In the same way, many backtracking optimization schemes studied in CSPs can be translated for projection of simple CGs (*e.g. Backmark + Forward Checking + NFC2* in [2]).

Normal form of a simple CG If two individuals e_1, e_2 of a simple CG have same marker i, then they have same type $t = \kappa(i)$ (thanks to the conformity relation κ, Def. 1). Then their interpretations are equal: $\phi(e_1) = \phi(e_2) = t(f(i))$, and their conjunction introduces a redundancy in $\Phi(G)$. Removing one of these predicates produces an equivalent formula Φ, and the graph operation on G used to obtain a simple CG G' s.t. $\Phi(G') = \Phi$ is the *join* of the two individuals e_1 and e_2. To join e_1 and e_2 in G, replace both entities in G by a single entity $e = \mathit{join}(e_1, e_2)$, having same type and marker than e_1 and e_2; then for every relation $r \in R$, replace each occurence of e_1 or e_2 in $\gamma(r)$ by e.

A simple CG G over $\mathcal{V}$ is said *normal* if all its individuals have distinct markers. A simple CG G is put into its *normal form* $\mathit{nf}(G)$ by successively joining all pairs of individuals having the same marker. Putting a simple CG into its normal form is linear in the size of the graph.

Theorem 1 (Soundness and completeness [10]). *Let G and H be two simple CGs over a vocabulary $\mathcal{V}$. Then $G \models_{\mathcal{V}} H$ if and only if there is a projection from H into $\mathit{nf}(G)$, the normal form of G, according to $\mathcal{V}$.*

3 Conceptual Graphs Rules

Conceptual graphs rules (CG rules) have been introduced in [14] as an extension of simple CGs allowing to represent knowledge of the form "IF H THEN C", where H and C are simple CGs. As for simple CGs, we first present their syntax in Sect. 3.1. In Sect. 3.2, their semantics (via the transformation Φ to FOL [14]) allows to define the CG rules deduction problem ($\mathcal{SR}$-DEDUCTION in [3]). Finally, as a sound and complete calculus for $\mathcal{SR}$-DEDUCTION, we present the Forward Chaining (FC) algorithm [12], based upon projection of simple CGs.

3.1 Syntax

Definition 5 (CG rules). *A* conceptual graph rule *(or CG rule) over a vocabulary $\mathcal{V}$ is a triple $\mathbb{R} = (\lambda, H, C)$ where $H = (E_H, R_H, \epsilon_H, \gamma_H)$ and $C = (E_C, R_C, \epsilon_C, \gamma_C)$ are two simple CGs over $\mathcal{V}$, and λ is a bijection between a distinguished subset of generic entities of E_H (called* connecting entities *of H) and a subset of generic entities of E_C (called connecting entities of C), s.t. $\lambda(e) = e' \Rightarrow type(e) = type(e')$. The simple CG H is called the* hypothesis *of $\mathbb{R}$, and C its* conclusion. *They are respectively denoted by $Hyp(\mathbb{R})$ and $Conc(\mathbb{R})$.*

This definition of CG rules clearly relates to a pair of λ-abstractions [12], without naming connecting entities.

The usual way to represent such a rule is by drawing two boxes next to each other. The box to the left is the hypothesis box, and the box to the right the conclusion box. Draw between these boxes an implication symbol $\Rightarrow$. Draw the simple CG H (as done in Sect. 2.1) in the hypothesis box and the simple CG G in the conclusion box. Finally, for each pair $(e, \lambda(e))$ of connecting entities, draw a dashed line (a coreference link) between the rectangle representing e and the rectangle representing $\lambda(e)$.

Another graphical representation of CG rules has been used in [3, 4], it relies on the representation of CG rules by colored graphs. Draw the simple CGs H and C, as indicated in Sect. 2.1. Rectangles and ovals representing the entities and relations of C are given a grey background, while rectangles and ovals originated from H keep their white bakckground. Join (as defined in Sect. 2.3) all pairs $(e, \lambda(e))$ of connecting entities (it is possible since they have the same type). The rectangles representing the resulting entities have a white backround.

3.2 Semantics

Interpretation of a CG Ruleset Let $\mathcal{R} = \{\mathbb{R}_1, \ldots, \mathbb{R}_k\}$ be a *CG ruleset* over $\mathcal{V}$, *i.e.* a set of CG rules over $\mathcal{V}$. Its interpretation $\Phi(\mathcal{R})$ is the conjunction of the FOL formulas $\Phi(\mathbb{R}_1), \ldots, \Phi(\mathbb{R}_k)$ interpreting its CG rules.

Predicate names and constant symbols are obtained from types and markers as in Sect. 2.2. Let us now build the interpretation $\Phi(\mathbb{R})$ of a CG rule $\mathbb{R}$. As done for simple CGs, we associate a variable name $f(e)$ to each generic entity $e \in E_H \cup E_C$. If $(e, \lambda(e))$ is a pair of connecting entities, then $f(e) = f(\lambda(e))$, otherwise these variable names are all distinct. If e is an individual, $f(e) = marker(e)$. As detailed in Sect. 2.2, a predicate

$\phi(e) = t(f(e))$ interprets each entity e typed t in $E_H \cup E_C$, and a predicate $\phi(r) = t(f(x_1), \ldots, f(x_k))$ interprets each relation $r \in R_H \cup R_C$, with $\gamma(r) = (x_1, \ldots, x_k)$ and $type(r) = t$. We note $\Phi_H(\mathbb{R})$ (resp. $\Phi_C(\mathbb{R})$ the conjunction of the interpretations $\phi(x)$, for all $x \in E_H \cup R_H$ (resp. for all $x \in E_C \cup R_C$). Let $x_1, \ldots, x_p$ be the variable names associated with generic entities of E_H, and $y_1, \ldots, y_q$ be those associated with generic entities of E_C, but not E_H. Then the interpretation of the CG rule $\mathbb{R}$ is the FOL formula $\Phi(\mathbb{R}) = \forall x_1 \ldots \forall x_p(\Phi_H(\mathbb{R}) \rightarrow (\exists y_1 \ldots \exists y_q \Phi_C(\mathbb{R})))$.

Definition 6 (Deduction problem in $\mathcal{SR}$). *Let G and H be two simple CGs over a vocabulary $\mathcal{V}$, and $\mathcal{R}$ be a CG ruleset. We say that $G, \mathcal{R}$ entails H in $\mathcal{V}$ (and note $G, \mathcal{R} \models_{\mathcal{V}} H$) iff $\Phi(H)$ is a logical consequence of $\Phi(G)$, $\Phi(\mathcal{R})$ and $\Phi(\mathcal{V})$ (*i.e. *$\Phi(\mathcal{V}), \Phi(\mathcal{R}), \Phi(G) \vdash \Phi(H)$).*

3.3 Calculus

Application of a CG rule Let $\mathbb{R} = (\lambda, H, C)$ be a CG rule and $G = (E, R, \epsilon, \gamma)$ be a simple CG over $\mathcal{V}$. The CG rule $\mathbb{R}$ is said *applicable* to G iff there is a projection π from $Hyp(\mathbb{R})$ into $nf(G)$, the normal form of G. In that case, the *application of* $\mathbb{R}$ *on* G *following* π produces a simple CG $G' = \alpha(G, \mathbb{R}, \pi)$ built as follows. We define the *disjoint union* of two graphs G_1, G_2 as the graph whose drawing is the juxtaposition of the drawings of G_1 and G_2. We first build the disjoint union of a copy of $nf(G)$ and of a copy of $Conc(\mathbb{R})$. Then, for each pair $(e, \lambda(e))$ of connecting entities in $\mathbb{R}$, we join (see Sect. 2.3) the entity x in the copy of $nf(G)$ obtained from $\pi(e)$ and the entity y in the copy of $Conc(\mathbb{R})$ obtained from $\lambda(e)$. Since $\epsilon(e) = \epsilon(\lambda(e))$, the label of x (*i.e.* the label of $\pi(e)$) is a specialization of the label of y, and $\epsilon(x)$ is used as the label of $join(x, y)$.

Deriving a simple CG with CG rules Let $\mathcal{R}$ be a CG ruleset and G, G' be two simple CGs over a vocabulary $\mathcal{V}$. We say that G' is *immediately derived from* G *in* $\mathcal{R}$ (and note $G \overset{\mathcal{R}}{\mapsto} G'$) iff there is a rule $\mathbb{R} \in \mathcal{R}$ and a projection π from $Hyp(\mathbb{R})$ into G such that $G' = \alpha(G, \mathbb{R}, \pi)$. We say that G' is *derived from* G *in* $\mathcal{R}$ (and note $G \overset{\mathcal{R}}{\rightsquigarrow} G'$) iff there is a sequence $G = G_0, G_1, \ldots, G_n = G'$ of simple CGs over $\mathcal{V}$ such that, for $1 \leq i \leq n$, $G_{i-1} \overset{\mathcal{R}}{\mapsto} G_i$.

Theorem 2 (Soundness and completeness [12]). *Let $\mathcal{R}$ be a CG ruleset, and G and H be two simple CGs over a vocabulary $\mathcal{V}$. Then $G, \mathcal{R} \models_{\mathcal{V}} H$ if and only if there is a simple CG G' such that $G \overset{\mathcal{R}}{\rightsquigarrow} G'$ and H projects into $nf(G')$.*

Forward Chaining of CG rules The Forward Chaining (FC) algorithm [12] immediately follows from theorem 2 and the property of confluence (Prop. 1).

Property 1 (Confluence). Let $\mathcal{R}$ be a CG ruleset, and G and H be two simple CGs over a vocabulary $\mathcal{V}$. Let us suppose that $G, \mathcal{R} \models_{\mathcal{V}} H$. Then for every simple CG G' such that $G \overset{\mathcal{R}}{\rightsquigarrow} G'$, the entailment $G', \mathcal{R} \models_{\mathcal{V}} H$ holds.

Any algorithm exploring all rule applications (Th. 2), *e.g.* using a breadth-first method, in any order (Prop. 1), will lead to a simple CG entailing the query H, if it exists. Such an algorithm, named FC, is proposed here (Alg. 1).

Algorithm 1: Forward Chaining

Data: A vocabulary $\mathcal{V}$, a CG ruleset $\mathcal{R}$, two simple CGs G and H over $\mathcal{V}$.
Result: YES iff $G, \mathcal{R} \models_{\mathcal{V}} H$ (infinite calculus otherwise).

```
ProjList ← ∅ ;
while TRUE do
    for ℝ ∈ ℛ do
        for π ∈ Projections(Hyp(ℝ), G) do
            ProjList ← ProjList ∪{(ℝ, π)} ;
    for (ℝ, π) ∈ ProjList do
        G ← α(G, ℝ, π) ;
        if Projects?(H, G) then return YES ;
```

Decidability Since FOL formulaes associated with CG rules have the same form as TGDs [7], $\mathcal{SR}$-DEDUCTION is semi-decidable (a sound and complete algorithm can compute in finite time whenever the answer is YES, but cannot always halt otherwise). Some decidable subclasses of the problem are proposed in [3]: let us suppose that, after the nth execution of the **while** loop in Alg. 1, the simple CG G obtained is equivalent to G as it was at the beginning of this loop. In that case, the algorithm could safely stop and answer NO. A CG ruleset ensured to have this behaviour is called a *finite expansion set*. Examples of finite expansion sets are *disconnected* CG rules (having no connecting entities) or *range restricted* CG rules (having no generic entity in the conclusion). Note that the union of two finite expansion rulesets is not necessarily a finite expansion ruleset.

4 Piece Unification and Backward Chaining

FC generates explicitely knowledge implicitely encoded in CG rules. By opposition, a Backward Chaining (BC) algorithm starts with the query H and rewrites it using *unification*. The interest of *piece unification* [12, 11] w.r.t. a PROLOG-like unification, is that it unifies at the same time a whole subgraph, instead of a simple predicate. Sect. 4.1 present preliminary definitions and Sect. 4.2 piece unification. A BC algorithm using piece unification is presented in Sect. 4.3.

4.1 Preliminary definitions

Definition 7 (Cut points, pieces). *Let $\mathbb{R} = (\lambda, H, C)$ be a CG rule over $\mathcal{V}$. A* cut point *of C is either a connecting entity (Def. 5) or an individual (Def. 2) of C. A* cut point *of H is either a connecting entity of H or an individual of H whose marker also appears in C. A* piece *P of C is a subgraph of C whose entities are a maximal subset of those of C s.t. two entities e_1 and e_2 of C belong to P if there is a path $e_1, x_1, \ldots, x_k, e_2$ where the x_i are not cut points of C.*

Conjunctive CGs When a CG rule $\mathbb{R}$ is applied to a simple CG G, in the resulting graph $\alpha(G, \mathbb{R}, \pi)$ the entities obtained from a join between a connecting entity of $Conc(\mathbb{R})$ and an entity of G may have a more specific label than the former connecting entity (Sect. 3.3). So to compute unification, we have to find which cut points of $Conc(\mathbb{R})$ have a common specialisation whith entities of the query. In [12, 11], such common specialisation of two entities e_1 and e_2 was typed by the *greatest lowerbound (glb)* of $type(e_1)$ and $type(e_2)$. The existence of the glb was ensured by using a lattice as the partial order on concept types. However, we do not impose here this partial order to be a lattice. To obtain a type having the same semantics, we will consider, as in [2, 6], conjunctive types.

A *conjunctive CG* is defined as a simple CG, but the type of an entity can be the conjunction of several types of T_C. The interpretation of an entity e with $\epsilon(e) = (t_1 \sqcap \ldots \sqcap t_p, m)$ is the conjunction $\phi(e) = t_1(f(e)) \wedge \ldots \wedge t_p(f(e))$. The partial order on T_C is extended to the partial order $\leq_\sqcap$ on conjunctive types: $t_1 \sqcap \ldots \sqcap t_p \leq_\sqcap t'_1 \sqcap \ldots \sqcap t'_q$ iff $\forall t'_i, \exists t_j$ with $t_j \leq t'_i$. The *join* operation (Sect. 2.3) between two entities e_1 and e_2 having different (conjunctive) types can now be defined: the type of the resulting entity $e = join(e_1, e_2)$ is the conjunction of the types of e_1 and e_2. If both e_1 and e_2 are individuals with same marker m, or generic entities with $m = *$, the marker of e is also m. If e_1 has individual marker m and e_2 is generic, the marker of e is m. The label $\epsilon(e)$ defined here is the *common specialization* of $\epsilon(e_1)$ and $\epsilon(e_2)$. The projection algorithm is the same as in Sect. 2.3, but relies on $\leq_\sqcap$ to compare conjunctive types. Normalization relies on the above-mentioned join operation. Up to these two differences, the soundness and completeness result (Th. 1) remains the same.

Compatible partitions A set of entities E is *join compatible* iff there is a concept type of T_C more specific than all types in E and there is at most one individual marker in E. Let G be a simple or conjunctive CG and E be a join compatible subset $\{e_1, \ldots, e_p\}$ of entities of G. The *join of G according to E* is the conjunctive CG obtained by joining e_1 and e_2 into e, then by joining G according to $\{e, e_3, \ldots, e_p\}$, until this subset contains a single entity e: we note $e = join(E)$. Let S and S' be two disjoint sets of entities. Let $P = (P_1, \ldots, P_n)$ and $P' = (P'_1, \ldots, P'_n)$ be two ordered partitions, resp. of S and S' (a partition of X is a set of pairwise disjoint sets whose union equals X). P and P' are *compatible partitions* of S and S' iff $P_i \cup P'_i$ is a join compatible set, for $1 \leq i \leq n$.

Definition 8 (Specialization according to a compatible partition). *Let G and G' be two simple or conjunctive CGs over $\mathcal{V}$. Let E and E' be respective subsets of entities of G and G'. Let $P = (P_1, \ldots, P_n)$ and $P' = (P'_1, \ldots, P'_n)$ be two compatible partitions of E and E'. The* specialization of G according to (P, P') *is the conjunctive CG $sp(G, (P, P'))$ built from G by building the join of G according to P_i, for $1 \leq i \leq n$, then by replacing the label of each $join(P_i)$ with its common specialization $join(P'_i)$.*

The *join of G and G' according to compatible partitions P and P'* is the conjunctive CG obtained by making the disjoint union of the specialization of G according to (P, P') and of the specialization of G' according to (P, P'), then by joining each $join(P_i)$ with $join(P'_i)$.

4.2 Piece Unification

Definition 9 (Piece unification). *Let Q be a simple (or conjunctive) CG (denoting a query) and $\mathbb{R} = (\lambda, H, C)$ be a CG rule over $\mathcal{V}$. Q and $\mathbb{R}$ are said* unifiable *iff there is a* piece unification *between Q and $\mathbb{R}$,* i.e. *a triple $\mu = (P^C, P^Q, \Pi)$ where:*

- *P^C and P^Q are two compatible partitions, resp. of a subset of cut points of C and a of subset of entities of Q that will be considered as cut points of Q;*
- *Π is a projection from a non-empty set of pieces of $\mu(Q) = sp(Q, (P^C, P^Q))$ (cut points of $\mu(Q)$ are entities resulting from the join of cut points of Q) into $\mu(\mathbb{R}) = sp(C, (P^C, P^Q))$ such that $\Pi(join(P_i^Q)) = join(P_i^C)$.*

Rewriting of a query If a query Q is unifiable with a CG rule $\mathbb{R}$, then an unification μ between Q an $\mathbb{R}$ determines a rewriting of Q (that can become a conjunctive CG). To put it simply, we remove from the new query the conclusion of $\mathbb{R}$ and add its hypothesis, that has still to be proven (by unification with another rule or with the facts graph G that can be considered as a CG rule with an empty hypothesis).

More precisely, let Q be a simple (or conjunctive) CG, $\mathbb{R} = (\lambda, H, C)$ be a CG rule, and $\mu = (P^C, P^Q, \Pi)$ be a piece unification between Q and $\mathbb{R}$. We call unification result of μ on Q and note $\beta(Q, \mathbb{R}, \mu)$ the conjunctive CG buit as follows:

1. Let S^C and S^Q be the subpartitions of P^C and P^Q formed respectively from the codomain and the domain of Π;
2. Let S^H be a partition of the subset of cut points of H that correspond to the partition S^C of cut points of C (if e is an entity of a partition S_i^C of S^C, the entities $g_1, \ldots, g_q$ of H that correspond to e, *i.e.* either $q = 1$ and $\lambda(g_1) = e$ or $g_1, \ldots, g_q$ and e have the same individual marker, belong to the partition S_i^H);
3. Build the conjunctive CGs $Q' = sp(Q, (S^H, S^Q))$ and $H' = sp(H, (S^H, S^Q))$;
4. Let P be a piece of Q whose entities are in the domain of Π. We remove from Q' all relations of P and all entities of P that are not cut points of Q';
5. We finally join Q' and H' according to (S^H, S^Q).

Definition 10 (Resolution). *Let H be a simple CGs, and $\mathcal{R}$ be a CG ruleset (that includes the facts CG G as a rule with an empty hypothesis) over $\mathcal{V}$. We call* resolution *of H in $\mathcal{R}$ a sequence $H = H_1, H_2, \ldots, H_{p+1}$ of conjunctive CGs such that, for $1 \leq i \leq p$, there is a piece unification μ between H_i and a rule $\mathbb{R} \in \mathcal{R}$, $H_{i+1} = \beta(H_i, \mathbb{R}, \mu)$ and H_{p+1} is the empty CG.*

[12, 11] proves that if $H = H_1, H_2, \ldots, H_{p+1} = \emptyset$ is a resolution of H in $\mathcal{R}$ using successively the rules $\mathbb{R}_{i_1}, \ldots \mathbb{R}_{i_p} = \mathbb{G}$, then there is a derivation sequence (see Sect. 3.3) $G = G_1, \ldots, G_p$ that successively applies the rules $\mathbb{R}_{i_1}, \ldots \mathbb{R}_{i_{p-1}}$ in reverse order, and such that H projects into G_p. Conversely, from a derivation sequence, we can extract a subsequence that corresponds to a resolution using the same rules in reverse order. The soundness and completeness theorem is a consequence of this correspondences between Forward and Backward Chaining.

Theorem 3 (Soundness and completeness [12]). *Let G and H be two simple CGs, and $\mathcal{R}$ be a CG ruleset over $\mathcal{V}$. Then $G, \mathcal{R} \models_{\mathcal{V}} H$ if and only if there is a resolution of H in $\mathcal{R} \cup \{\mathbb{G}\}$ ($\mathbb{G} = (\lambda, \emptyset, G)$ is a CG rule equivalent to G).*

4.3 Backward Chaining

The following algorithm (Alg. 2) provides a version of Backward Chaining using the resolution and piece unification presented in Sect. 4.2.

Algorithm 2: Backward Chaining

Data: A vocabulary $\mathcal{V}$, a CG ruleset $\mathcal{R}$, two simple CGs G and H over $\mathcal{V}$.
Result: If Backward Chaining halts on YES, then $G, \mathcal{R} \models_{\mathcal{V}} H$, if it halts on NO, then $G, \mathcal{R} \not\models_{\mathcal{V}} H$ (but it can run infinitely).

```
UnifList ← NewFilo() ;
for ℝ ∈ ℛ ∪ {𝔾} do
    for μ ∈ Unifications(ℝ, H) do
        UnifList ← AddFilo(UnifList, (μ, ℝ, H)) ;
while UnifList ≠ ∅ do
    (μ, ℝ, H) ← FiloRemove(UnifList) ;
    H' ← Rewrite(μ, ℝ, H) ;
    if H' = ∅ then return YES ;
    for ℝ' ∈ ℛ do
        for μ' ∈ Unifications(ℝ', H') do
            UnifList ← AddFilo(UnifList, (μ', ℝ', H')) ;
return NO ;
```

Forward Chaining and Backward Chaining It is well known (*e.g.* [1]) in Logic Programing that, from BC or FC, no algorithm is always better than the other. The main differences are that 1) FC enriches the facts until they contain an answer to the query while BC rewrites the query until all its components have been proven; 2) FC derivation is a confluent mechanism, while BC rewritings depends upon the order of these rewritings, and thus requires a backtrack; and 3) FC enumerates all solutions to the query by applying rules breadth-first, while BC *usually* (as we did in Alg. 2) tries to find them quicker by rewriting the query depth-first (eventually missing solutions). It is to be noted that a breadth-first version of BC, that misses no solution, can be implemented by replacing the Filo structure of *UnifList* in Alg. 2 by a Fifo structure. In that case, completeness is achieved at the expense of efficiency. Finally, [7] compares BC using piece unification with the standard PROLOG BC that unifies only one predicate at a time. Though piece unification leads to fewer backtracks in the query rewriting mechanism, it does not translate to the overall efficiency of the algorithm, since these backtracks are hidden in the computation of unifications, that relies on a projection. Optimization of projection (Sect. 2.3) and compilation of unifications in the graph of rules dependencies (Sect. 6) are solutions to this problem.

5 Rules Dependencies in Forward Chaining

The notions of neutrality/dependency between CG rules were introduced in [4] to enhance the basic FC (Alg. 1). The basic idea can be expressed as follows:

suppose that the conclusion of $\mathbb{R}_1$ contains no entity or relation that is a specialization of an entity or a relation in the hypothesis of $\mathbb{R}_2$. Then an application of $\mathbb{R}_1$ on a given simple CG does not create any new application of $\mathbb{R}_2$. This is a simple case of neutrality between rules. A general definition is provided in Sect. 5.1.We present in Sect. 5.2 a characterization of dependency (the inverse notion of neutrality), based upon piece unification, that generalizes the characterization of [4]. Finally, in Sect. 5.3, we show that encoding all dependencies of a CG ruleset (in the graph of rules dependencies [4]) enhances FC.

5.1 Neutrality and Dependency

Though the definition of neutrality and dependency expressed below seems strictly identical to [4], it is indeed more general. A component of this definition is rule application (Sect. 3.3). In this paper, the graph on which the rule is applied is put into normal form, and not in [4]. As a consequence, the algorithm was not complete for CG rulesets containing rules having individuals in the conclusion. Since our definition of derivation takes into account the need to put a simple CG into its normal form after each application of a rule, the following definition of neutrality/dependency is more adapted to $\mathcal{SR}$-DEDUCTION.

Definition 11 (Neutrality, Dependency). *Let $\mathbb{R}_1$ and $\mathbb{R}_2$ be two CG rules over a vocabulary $\mathcal{V}$. We say that $\mathbb{R}_1$ is* neutral *w.r.t. $\mathbb{R}_2$ iff, for every simple CG G over $\mathcal{V}$, for every projection π of $Hyp(\mathbb{R}_1)$ into G, the set of all projections of $Hyp(\mathbb{R}_2)$ into $\alpha(G, \mathbb{R}_1, \pi)$ and the set of all projections of $Hyp(\mathbb{R}_2)$ into G are equal. If $\mathbb{R}_1$ is not neutral w.r.t. $\mathbb{R}_2$, we say that $\mathbb{R}_2$* depends *upon $\mathbb{R}_1$.*

5.2 Piece Unification and Dependency

Since we have changed the definition of derivation used in [4] the characterization of dependency must take that change into account. We prove here that this updated characterization corresponds to the piece unification of [12, 11], for CG rules that are not trivially useless. A CG rule $\mathbb{R}$ is said *trivially useless* if, for every simple CG G, for every projection π of $Hyp(\mathbb{R})$ on G, $G = \alpha(G, \mathbb{R}, \pi)$. We can remove in linear time all trivially useless rules from a CG ruleset.

Theorem 4. *Let $\mathbb{R}_1$ and $\mathbb{R}_2$ be two CG rules over a vocabulary $\mathcal{V}$, where $\mathbb{R}_1$ is not trivially useless. Then $\mathbb{R}_2$ depends upon $\mathbb{R}_1$ if and only if $Hyp(\mathbb{R}_2)$ and $\mathbb{R}_1$ are unifiable (see Def. 9).*

Let us introduce the *composition of unification and projection* (noted $\odot$). Let G and H be a simple CG, and $\mathbb{R}$ be a CG rule over $\mathcal{V}$. Let $\mu = (P^C, P^Q, \Pi)$ be a unification between H and $\mathbb{R}$. Let π be a projection from $Hyp(\mathbb{R})$ into G. We say that μ and π are *composable* iff for each compatible partition P_i^H whose join belongs to the domain of Π, the entities of $Hyp(\mathbb{R})$ associated (by λ^{-1} or by sharing the same individual marker) with the compatible partition P_i^C of $Conc(\mathbb{R})$ are all mapped by π into the same entity noted $f(P_i^H)$. If μ and π are

composable, then we note $\mu \odot \pi : H \to \alpha(G, \mathbb{R}, \pi)$ the partial mapping defined as follows: if e is a cut point of P_i^H in the domain of Π, then $\mu \odot \pi(e) = f(P_i^H)$, otherwise, if e is an entity in the domain of Π that is not a cut point, $\mu \odot \pi(e)$ is the entity of $\alpha(G, \mathbb{R}, \pi)$ that corresponds to $\Pi(e)$ in $Conc(\mathbb{R})$. It is immediate to check that $\mu \odot \pi$ is a partial projection from H into $\alpha(G, \mathbb{R}, \pi)$.

Proof. Let us successively prove both directions of the equivalence:

($\Leftarrow$) Suppose that $Hyp(\mathbb{R}_2)$ and $\mathbb{R}_1$ are unifiable, and note μ such an unification. Let us consider the conjunctive CG $G = \beta(Hyp(\mathbb{R}_2), \mathbb{R}_1, \mu)$. We transform it into a simple CG by replacing all its conjunctive types by one of their specializations in T_C (it exists, by definition of compatible partitions, Sect. 4.1). There exists a projection π from $Hyp(\mathbb{R}_1)$ into G: if e has been joined in G, $\pi(e)$ is this join, and $\pi(e) = e$ otherwise. This mapping π is a projection. It is immediate to check that μ and π are composable (see above). Then $\mu \odot \pi$ is a partial projection from $Hyp(\mathbb{R}_2)$ into $G' = \alpha(G, \mathbb{R}_1, \pi)$ that uses an entity or relation of G' that is not in G (or $\mathbb{R}_1$ would have been trivially useless). Since BC is sound and complete, $\mu \odot \pi$ can be extended to a projection π' of $Hyp(\mathbb{R}_2)$ into G', and π' is not a projection from $Hyp(\mathbb{R}_2)$ into G. Then $\mathbb{R}_2$ depends upon $\mathbb{R}_1$.
($\Rightarrow$) Suppose that $H = Hyp(\mathbb{R}_2)$ and $\mathbb{R}_1$ are not unifiable. Let us consider a simple CG G, and a projection π from $H = Hyp(\mathbb{R}_1)$ into G. If there is a projection from H) into $\alpha(G, \mathbb{R}_1, \pi)$ that is not a projection of H into G, it means that there is a solution to the query H that requires the application of $\mathbb{R}_1$. Since H and $\mathbb{R}_1$ are not unifiable, such a solution could not be found by BC, which is absurd. □

5.3 Graph of Rules Dependencies in Forward Chaining

In this section, we present an enhancement of FC (Alg. 1) that relies upon the graph of rules dependencies (GRD) [4].

Building the Graph of Rules Dependencies Let $\mathcal{R}$ be a CG ruleset over $\mathcal{V}$. We call *graph of rules dependencies* (GRD) of $\mathcal{R}$, and note $GRD_{\mathcal{V}}(\mathcal{R})$ the (binary) directed graph whose nodes are the rules of $\mathcal{R}$, and where two nodes $\mathbb{R}_1$ and $\mathbb{R}_2$ are linked by an arc $(\mathbb{R}_1, \mathbb{R}_2)$ iff $\mathbb{R}_2$ depends upon $\mathbb{R}_1$. In that case, the arc $(\mathbb{R}_1, \mathbb{R}_2)$ is labelled by the non-empty set of all unifications between $Hyp(\mathbb{R}_2)$ and $\mathbb{R}_1$. By considering the simple CG G encoding the facts as a CG rule with an empty hypothesis and the simple CG H encoding the query as a CG rule with an empty conclusion, we can integrate them in the graph of rules dependencies, obtaining the graph $GRD_{\mathcal{V}}(\mathcal{R}, G, H)$. Finally, we point out that if a rule $\mathbb{R}$ is not on a path from G to H, then no application of $\mathbb{R}$ is required when solving $\mathcal{SR}$-DEDUCTION [4]. The graph $SGRD_{\mathcal{V}}(\mathcal{R})$ obtained by removing all nodes that are not on a path from G to H, called the *simplified GRD*, is used to restrain the number of unnecessary rules applications.

The problem $\mathcal{SR}$-DEPENDENCY (deciding if a CG rule $\mathbb{R}_2$ depends upon a CG rule $\mathbb{R}_1$) is NP-complete (since a unification is a polynomial certificate, and when $\mathbb{R}_1$ is disconnected, a unification is exactly a projection). Building the GRD is thus a costly operation, that requires $|\mathcal{R}|$ calls to a NP-hard operation.

Using the Graph of Rules Dependencies in Forward Chaining The GRD (or its simplified version) can be used to enhance FC (Alg. 1) as follows. Let us consider a *step* of FC (an execution of the main **while** loop). The *PartialProjList* contains all partial projections from the hypothesis of the CG rules in $\mathcal{R}$ into G. If one of these partial projections can be extended to a full projection π of the hypothesis of a rule $\mathbb{R}$, then $\mathbb{R}$ is applicable and the only rules that will be applicable on $\alpha(G, \mathbb{R}, \pi)$ (apart from those already in *PartialProjList*) are the successors of $\mathbb{R}$ in the GRD. Moreover, the operator $\odot$ is used to efficiently generate partial projections of the hypothesis of these rules.

Algorithm 3: Forward Chaining using Rules Dependencies

```
Data: A vocabulary V, a CG ruleset R, two simple CGs G and H over V.
Result: YES iff G, R |=_V H (infinite calculus otherwise).
D ← SimplifiedRulesDependenciesGraph(R, G, H) ;
PartialProjList ← NewFifo() ;
for R ≠ H ∈ Successors(D, G) do
    for μ ∈ Unifications(D, G, R) do
        PartialProjList ← AddFifo(PartialProjList, (R, μ)) ;
while TRUE do
    (R, π) ← FifoRemove(PartialProjList) ;
    for π' ∈ ExtendPartialtoFullProjections(Hyp(R), G, π) do
        G ← α(G, R, π') ;
        if Projects?(H, G) then return YES ;
        for R' ≠ H ∈ Successors(D, R) do
            for μ ∈ Unifications(D, R, R') do
                if Composable(μ, π') then
                    PartialProjList ← AddFifo(PartialProjList, R', μ ⊙ π') ;
```

Evaluating the algorithm With respect to the standard FC, FC with rules dependencies (FCRD, Alg. 3) relies on three different optimizations:

1. using the simplified GRD allow to ignore some CG rules during derivation;
2. though FC, at each step, checks applicability of all rules in $\mathcal{R}$, FCRD only checks the successors of the rules applied at the previous step;
3. the operator $\odot$, by combining projections and unifications into a partial projection, reduces the search space when checking applicability of a rule.

Though generating the GRD is a lengthy operation, it can be done once and for all for a knowledge base $(G, \mathbb{R})$, leaving only to compute the n unifications of the query Q at run time. Moreover, even if the KB is used only once, the cost of the operations required to compute the GRD is included in the two first steps (the main **while** loop) of the FC agorithm.

Finally, the GRD has been used in [4] to obtain new decidability result. If the GRD (or the simplified GRD) has no circuit, then $\mathcal{SR}$-DEDUCTION is decidable. Moreover, if all strongly connected components of the GRD (or simplified GRD) are finite expansion sets (see Sect. 3.3), then $\mathcal{SR}$-DEDUCTION is decidable.

6 Rules Dependencies in Backward Chaining

The identification of dependencies and unifications (Th. 4) naturally leads to the following question: how to efficiently use the GRD in a Backward Chaining framework ? We consider the three interests of the simplified GRD in a FC framework, at the end of Sect. 5.3, and show how they translate to a BC framework (Sect. 6.1). In Sect. 6.2,we provide an update of BC (Alg. 2) that relies on the simplified GRD. Further works on that algorithm are discussed in Sect. 6.3.

6.1 Reducing the number of searches for unification

The simplified GRD can be used as in Forward Chaining to remove rules that are not involved in reasonings: if there is no derivation sequence from G into a solution of H that involves the rule $\mathbb{R}$, then the correspondence between FC and BC proves that no rewriting of H into $\emptyset$ involves that same CG rule $\mathbb{R}$. We should note that, if there is a path from $\mathbb{R}$ to H, but no path from G to $\mathbb{R}$ in the GRD, simplifying the GRD removes this rule though the standard Backward Chaining may try to use it in a rewriting sequence.

The second optimization brought by the GRD to Forward Chaining consists in reducing the number of checks for applicability of a rule. To translate that feature to Backward Chaining, we must ask if, after unifying a query with a rule and rewriting this query w.r.t. this unification, we need to compute the unifications of this new query with all the rules in the CG ruleset $\mathcal{R}$. By giving a negative answer to this question, Th. 5 shows that the GRD can be used during BC for added efficiency.

Theorem 5. *Let H be a simple CG, and $\mathcal{R}$ be a CG ruleset over a vocabulary $\mathcal{V}$. Let μ be an unification between H and $\mathbb{R} \in \mathcal{R}$. Let $H' = \alpha(H, \mathbb{R}, \mu)$ be the rewriting of H according to μ. The following property holds: if $\mathbb{R}'$ and H' are unifiable then $\mathbb{R}'$ is a predecessor of H or $\mathbb{R}$ in $GRD(\mathcal{R}, G, H)$.*

Proof. Suppose $\mathbb{R}'$ and H' are unifiable, by a unification μ'. We note $H'' = \beta(H', \mathbb{R}', \mu')$. Let us consider the simple CG G' that specializes the conjunctive CG H'', built in the same way as in the proof of Th. 4. Since G' proves H'', the correspondence between FC and BC implies that there exists a derivation sequence $G', G'' = \alpha(G', \mathbb{R}', \pi_1), G''' = \alpha(G'', \mathbb{R}, \pi_2)$ such that H projects into G'''. Since FC with rules dependencies is complete, it means that either H depends upon $\mathbb{R}'$, or that $\mathbb{R}$ depends upon $\mathbb{R}'$. □

6.2 Backward Chaining with Rules Dependencies

The following algorithm uses the graph of rules dependencies in a Backward Chaining framework to include the two optimizations discussed in Sect. 6.1.

Algorithm 4: Backward Chaining using Rules Dependencies

Data: A vocabulary $\mathcal{V}$, a CG ruleset $\mathcal{R}$, two simple CGs G and H over $\mathcal{V}$.
Result: If Backward Chaining halts on YES, then $G, \mathcal{R} \models_{\mathcal{V}} H$, if it halts on NO, then $G, \mathcal{R} \not\models_{\mathcal{V}} H$ (but it can run infinitely).

```
D ← SimplifiedRulesDependenciesGraph(R, G, H) ;
UnifList ← NewFilo() ;
for ℝ ∈ Predecessors(D, H) do
    for μ ∈ Unifications(D, ℝ, H) do
        UnifList ← AddFilo(UnifList, (μ, ℝ, H)) ;
while UnifList ≠ ∅ do
    (μ, ℝ, H) ← FiloRemove(UnifList) ;
    H' ← Rewrite(μ, ℝ, H) ;
    if H' = ∅ then return YES ;
    for ℝ' ∈ Predecessors(ℝ) do
        for μ' ∈ ComputeNewUnifications(ℝ', H') do
            UnifList ← AddFilo(UnifList, (μ', ℝ', H')) ;
return NO ;
```

6.3 Further work: combining unifications

Finally, we point out that we have not translated in this BC framework the third optimization of FC brought by the GRD. In FC, the composition operator $\odot$ between the current projection and the unifications between the current rule and its successors is used to reduce the size of projections that have to be computed during the following execution of the main **while** loop. A similar operator, composing unifications into a partial unification, would be required to achieve the same optimization result in BC.

7 Conclusion

In this paper, we have unified two optimization schemes used for computing deduction with conceptual graphs rules [14, 12] ($\mathcal{SR}$-DEDUCTION), namely piece unification in Backward Chaining [12, 11], and the graph of rules dependencies in Forward Chaining [4]. Our main contributions are listed below:

1. **Unification of syntax:** [12, 11] defines simple CGs as bipartite multigraphs and CG rules as pairs of λ-abstractions, while [4] defines them as directed hypergraphs and coloured CGs. We unify these different syntaxes in a common structure (Sect. 2.1 and 3.1).
2. **Generalization of piece unification:** the definition of piece unification in [12, 11] does no longer rely on concept types being ordered by a lattice.
3. **Generalization of dependencies:** the definition of dependencies in [4] does not take into account the modification of structure induced by normalization, and thus is restricted to CG rules having no individual in the conclusion for completeness of algorithms. This restriction is dropped here.

4. **Identification of piece unification and dependencies:** Up to the generalizations above, we prove that piece unification and neutrality (the inverse of dependency)are equivalent (Th. 4 in Sect. 5.2).
5. **Use of the graph of rules dependencies in a Backward Chaining framework:** we show how the optimizations allowed by the graph of rules dependencies of [4] in a Forward Chaining framework can be adapted to the Backward Chaining framework of [12, 11] (Th. 5 in Sect. 6).

Perspectives Though the graph of rules dependencies already increases efficiency in both FC and BC, we are now considering the following problems:

1. **Traversals of the graph of rules dependencies:** FC and BC rely respectively on a breadth-first and a depth-first traversal of the GRD. Different types of traversals can be tested.
2. **Rewriting of a CG ruleset:** Some transformations of rules preserve their semantics (*e.g.* a rule with k pieces is equivalent to k rules with one piece). What transformations can give a more efficient FC or BC?
3. **Finding a composition operator for unifications:** (Sect. 6.3)

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. J.-F. Baget. Simple Conceptual Graphs Revisited: Hypergraphs and Conjunctive Types for Efficient Projection Algorithms. In *Proc. of ICCS'03*, volume 2746 of *LNAI*. Springer, 2003.
3. J.-F. Baget and M.-L. Mugnier. The Complexity of Rules and Constraints. *JAIR*, 16:425–465, 2002.
4. Jean-François Baget. Improving the forward chaining algorithm for conceptual graphs rules. In *Proc. of KR2004)*, pages 407–414. AAAI Press, 2004.
5. M. Chein and M.-L. Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, 1992.
6. M. Chein and M.-L. Mugnier. Types and Coreference in Simple Conceptual Graphs. In *Proc. ICCS'04*, volume 3127 of *LNAI*. Springer, 2004.
7. S. Coulondre and E. Salvat. Piece Resolution: Towards Larger Perspectives. In *Proc. of ICCS'98*, volume 1453 of *LNAI*, pages 179–193. Springer, 1998.
8. G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 394–399. Morgan Kaufmann, 1999.
9. M.-L. Mugnier. Knowledge Representation and Reasoning based on Graph Homomorphism. In *Proc. ICCS'00*, volume 1867 of *LNAI*, pages 172–192. Springer, 2000.
10. M.-L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1):7–56, 1996.
11. E. Salvat. Theorem proving using graph operations in the conceptual graphs formalism. In *Proc. of ECAI'98*, pages 356–360, 1998.

12. E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *Proc. of ICCS'96*, volume 1115 of *LNAI*, pages 248–262. Springer, 1996.
13. J. F. Sowa. Conceptual Graphs. *IBM Journal of Research and Development*, 1976.
14. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.