Boolean formulas of simple conceptual graphs SGBF

Conference Paper · July 2011

DOI: 10.1007/978-3-642-29449-5_2

CITATION
1

READS
118

1 author:

Olivier Carloni

14 PUBLICATIONS **50** CITATIONS

SEE PROFILE

Boolean formulas of simple conceptual graphs ($\mathcal{SGBF}$)

Olivier Carloni
olivier.carloni@gmail.com

Abstract

This paper presents a conceptual graph formalism called *simple graph boolean formulas* which extends the $\mathcal{SG}$-family with boolean connectors. This formalism is used to define categories of objects in a classification service which can be turned into a legal content management system. We define the $\mathcal{SGBF}$-model of graph boolean formulas, present two decidable fragments of this formalism (relying on the first order logic BSR and guarded fragments), and describe the functional architecture of a generic classification service which can be used in the legal domain.

1 Introduction

Since the production of knowledge has substancially increased in the last previous years, organisations intend to mutualize knowledge or know-how of their members in systems that process data at a semantic level. This evolution has given rise to knowledge engineering, which aims at eliciting the semantics of data by way of a knowledge representation language controlled by ontologies which are semantic referentials that define the meaning of the language symbols.

Mondeca is a software publisher that is developing ITM (Intelligent Topic Manager), a knowledge management tool based on these principles. The core of this software is a three-level knowledge base (see. Figure 1):

- the highest level is a meta-model of all ITM knowledge bases. It consists of a representation ontology reflexively specifying the semantics of all representations used by ITM. It is common to all customers;
- the intermediate level is composed of models that specify the vocabulary used to describe customer data managed by ITM; it generally comprises a domain ontology describing the conceptual vocabulary used to annotate the content of the customer's resources, some (representation) ontologies defining primitives related to specific data structures enabling ITM services (primitives for representing the thesaurus or the logical organization of documents);
- the lowest level contains the instances: annotations describing the content of information managed by ITM (data, documents), terminological resources (thesaurus) used to index this information, description of the logical organization of documents, etc. This level is managed by the customer.

ITM provides several services as : indexing of documents from the thesaurus, creation of annotations controlled by a domain ontology, semi-automatic building of

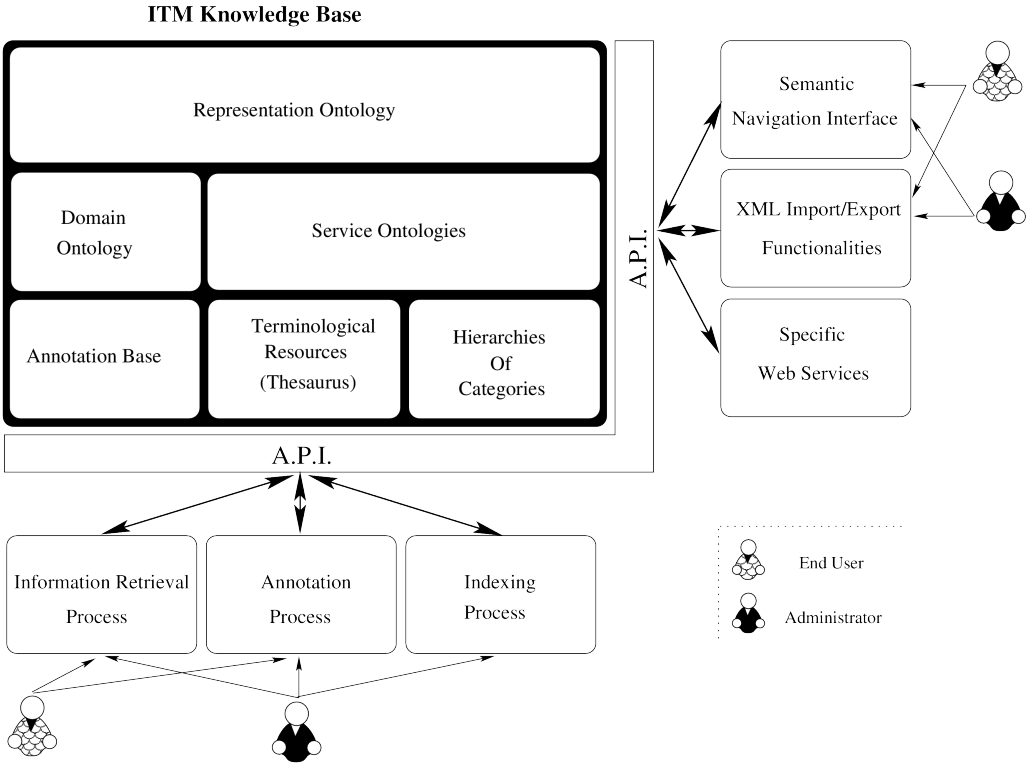


Figure 1: Workspaces and modules in ITM

document annotations, semantic navigation and search features through the annotation base. The rising number of customers in various fields (as tourism, terminological management, military, etc) requesting knowledge quality control, content enrichment or classification features led Mondeca to provide ITM with several reasoning services. As the knowledge representation formalism in ITM is quite similar to labeled graphs, Mondeca chose the conceptual graph formalism (see [19] and [20]) and especially the $\mathcal{SG}$-family [4] to define these reasoning services. The $\mathcal{SG}$-family provides ITM with a formal semantics in logic, inference rules and constraints, graph-based reasoning operations sound and complete with respect to the logical deduction (see [8] and [17]), and combinatorial-based efficient algorithms for these graph operations. The reader is referred to [7] for a complete description of ITM, its knowledge representation formalism and its relationships with the $\mathcal{SG}$-family.

One feature increasingly requested by customers is the ability to *classify* knowledge in hierarchically organized categories. This is particularly needed in the legal domain and especially by law publishers whose main activity is to publish the last up to date version of various regulations in reference books. Throughout the past few years, law publishers sought to diversify their offer by providing legal expertise services. A part of this diversification strategy relies on three other key actors of the legal domain : (1) the *lawmaker* whose work consists in maintaining the law up to date, (2) the *individual* which is expected to respect the law and (3) the *compliant-seller* whose work consists in selling products that must be compliant with the law. For each of these,

law publishers intend to provide law management or expertise services : (1) regulation management services to the *lawmaker*, (2) triggering services to alert the *individual* when his personal situation is concerned by a change in the law, and (3) search features for marketing prospection in order for the *compliant-seller* to know which individual is concerned by a given regulation.

ITM is equipped with a classification service providing the ability to define and manage hierarchies of categories. A category represents a collection of instances and is defined in a language allowing the combination of multiple criteria. Categories are organised in tree structured hierarchies provided with an inheritance mechanism. The query language to define categories is expressive enough to allow the conjunctive/disjunctive combinations of positive/negative criterias (a criterion may be another query). For example, a typical query might be "select the ships with electric engine and cooling system or provided with two twined engines with same cooling system but no adjacent tanks". This generic classification service can be set to meet some requirements of the legal domain. To do so, personal situations of individuals are represented as instances in the lowest level of the ITM knowledge base and rules of a regulation are formalized into categories. The classification of a situation-instance in a category means that the owner of this instance is affected by the rule corresponding to the category.

A formalism has been designed to meet the high expressivity needed to define categories while staying the closest possible to the "graph" spirit of the $\mathcal{SG}$-family and the ITM tool. This paper is devoted to this formalism, named $\mathcal{SGBF}$-model, which stands for $\mathcal{S}$imple $\mathcal{G}$raph $\mathcal{B}$oolean $\mathcal{F}$ormulas. The first section presents the $\mathcal{SG}$-model which is the lowest level of the $\mathcal{SG}$-family of conceptual graphs upon which is defined the $\mathcal{SGBF}$-model. The second section presents the $\mathcal{SGBF}$-model, its logical semantics and its relationships with the $\mathcal{SG}$-model. It is followed by a third section which discusses decision problems related to the formalism. The fourth section presents the classification service of ITM and its use in the legal domain. The paper ends with some research perspectives.

2 The $\mathcal{SG}$-model of conceptual graphs

The $\mathcal{SG}$-family defined in [4] is a hierarchy of conceptual graph formalisms organized in layers ranging from the less to the more expressive. The $\mathcal{SG}$-model is the bottom layer of this hierarchy and provides the elementary components and operations used in upper layers to define more complex structures and operations (as rules and constraints with their respective inference and validation methods). These components are the *support* and the *simple graph* (SG). A SG is a bipartite graph representing entities of the domain and their shared relations respectively as *concept nodes* edge-connected to *relation nodes*. Each node has a type defined in a structure called the *support* which is a rudimentary ontology. The support and SG definitions may vary through the litterature. The reader is referred to [8] about simple conceptual graphs and [4] for a study of the whole $\mathcal{SG}$-family.

2.1 Support

A support is a triple $S = (T_C, T_R, \mathcal{I})$. T_C and T_R are two finite sets respectively of concept types and relation types partially ordered by a relation $\leq_t$ where $x \leq_t y$ means that x is a subtype of y. T_R is partitionned into subsets T_R^1, T_R^2, ..., T_R^k of relation types

of arity 1...k respectively ($k \leq 1$). $\mathcal{I}$ is the set of individual markers. We denote by $*$ the generic marker, where $* \notin \mathcal{I}$ and by $\mathcal{M}$ the set of all markers $\mathcal{I} \cup \{*\}$. A partial order on $\mathcal{M}$ considers elements of $\mathcal{I}$ as pairwise incomparable, and $*$ as its greatest element.

The figure 2 shows a simple support with naval vocabulary. It is said that *tanker* is more specific than *ship* and that, if two entities are *adjacent* then they are *close*.

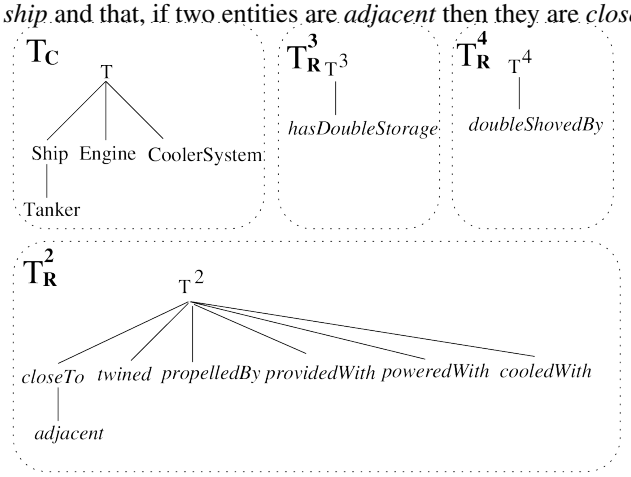


Figure 2: A support with a concept type hierarchy T_C and a binary relation type hierarchy T_R^2.

2.2 Simple graph

A SG G, defined over a support S is a finite bipartite graph (C, R, E, λ). Where C and R are the node sets respectively of concept nodes and relation nodes. E is a finite set of labelled edges (r, i, c) where $c \in C$, $r \in R$ and $i \in \mathbb{N}$ is the number of the edge. Edges (r, i, c) incident to a relation node are totally ordered with i ranging from 1 to the degree of the node and for $r \in R$ and $c \in C$ there is at most one $(r, i, c) \in E$.

Each node has a label given by the mapping λ. A relation node r is labelled by $type(r)$ an element of T_R called its type and the degree of r must be equal to the arity of $type(r)$. A concept node c is labelled by a pair $(type(c), marker(c))$ where $type(c)$ is an element of T_C called its type and $marker(c)$ is an element of $\mathcal{M}$ called its marker. The concept is said to be *generic* if $marker(c) = *$; and *individual* otherwise.

The set of node labels is partially ordered by a relation $\leq_\lambda$ such that for two nodes n_1 and n_2 $\lambda(n_1) \leq_\lambda \lambda(n_2)$ iff $type(n_1) \leq_t type(n_2)$ and, when n_1, n_2 are concepts, $marker(n_1)$ must be in addition less or equal than $marker(n_2)$.

The figure 3 shows four SGs E_1, E_2, E_3 and E_4 meaning respectively that (1) ship x is propelled by an electric engine and provided with an electric cooler system w, (2) ship x is provided with a cooler system w and shoved by two twined engines using the same cooler system, (3) tanker x has a storage capacity of two adjacent tanks and (4) the ship x has a tank t that is cooled by a cooler system z and adjacent to two other tanks u and w themselves adjacent to a fourth tank v. In these figures, the symbols x, t, , u, v, w and z denote individual markers (not variables).

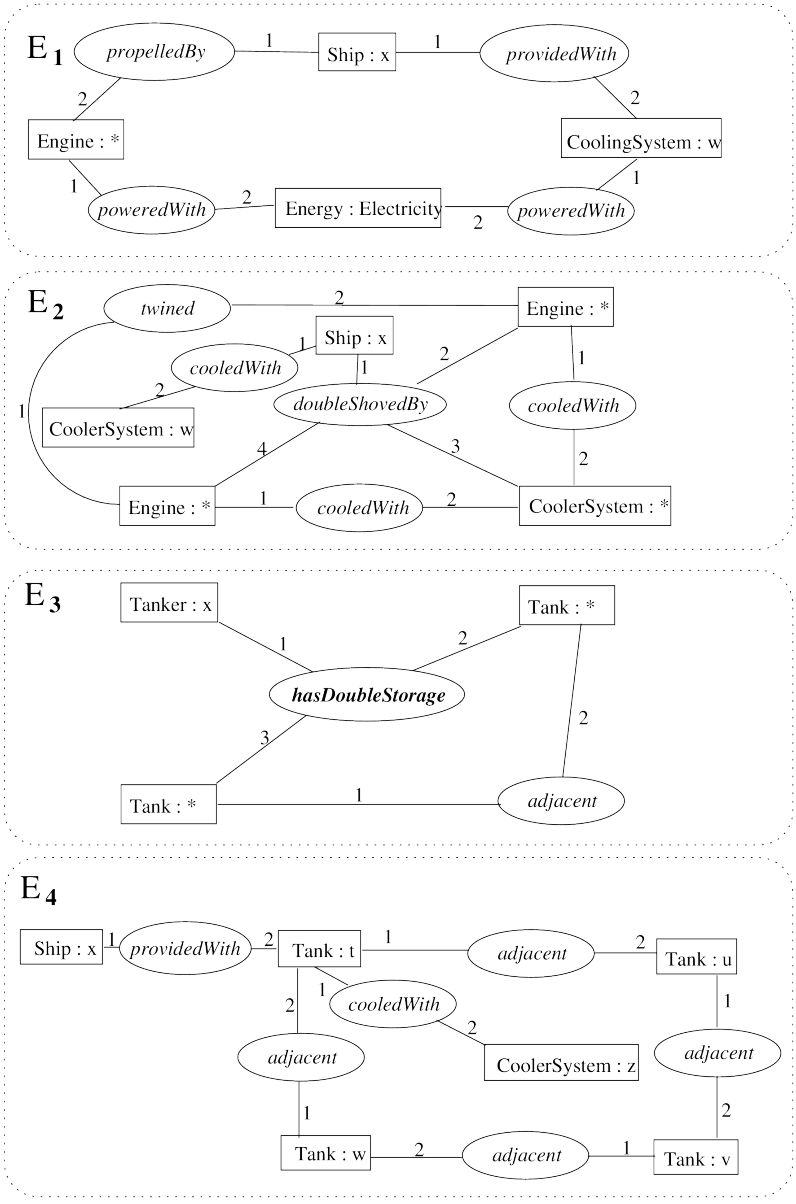


Figure 3: Three SGs.

2.3 Semantics

In the following, the term *logical formula* refers to a $\mathcal{FOL}$ formula where $\mathcal{FOL}$ refers to the classical first order logic without functions, with connectors $\{\wedge, \vee, \rightarrow, \neg\}$ and quantifiers $\{\forall, \exists\}$.

The mapping Φ maps a support as a set of universally quantified logical formulas and SGs to the existential and positive fragment $\mathcal{FOL}(\wedge, \exists)$ of $\mathcal{FOL}$. Given a support S, a constant is assigned to each individual of $\mathcal{I}$. For each k-ary relation type, is assigned one k-ary predicate and for each concept type one unary predicate. We consider that each constant or predicate names are identical to their corresponding element in $\mathcal{SG}$-model.

The set of formulas $\Phi(S)$ assigned to any support S contains all formulas $\forall x_1 ... x_k (t_1(x_1, ..., x_k) \rightarrow t_2(x_1, ..., x_k))$ such that $t_1 \leq_t t_2$ with $k = 1$ when $t_1, t_2 \in T_C$; and k denotes the arity of the relation type when $t_1, t_2 \in T_R$.

For example, the translation of the support shown in figure 2 returns :

- $\forall x(Tanker(x) \rightarrow Ship(x))$
- $\forall x \forall y(adjacent(x, y) \rightarrow closeTo(x, y))$

Given a SG G, in order to build the formula $\Phi(G)$ a term $term(c)$ is assigned to each concept c such that $term(c)$ is the constant corresponding to its marker or a distinct variable if the marker is generic. Let $\Phi^{\wedge}(G)$ be the conjunction of

- all predicates $p_c(term(c))$ such that $c \in C$ and p_c is the unary predicate assigned to $type(c)$
- and all predicates $p_r(term(c_1), term(c_2), ..., term(c_k))$ such that $r \in R$, p_r is the k-ary predicate assigned to $type(r)$ and c_i is the concept from $C(G)$ linked to r by an i-numbered edge from $E(G)$

$\Phi(G)$ is the formula $\exists \widetilde{x} \Phi^{\wedge}(G)$ where $\widetilde{x}$ is the set of variables appearing in $\Phi^{\wedge}(G)$.

The SG E_1 shown in figure 3 have the following $\mathcal{FOL}$ semantics, where X denotes here a constant: $\Phi(G_1) = (\exists y \exists z Ship(X) \wedge Engine(y) \wedge CoolerSystem(z) \wedge Energy(Electricity) \wedge propelledBy(X, y) \wedge propelledBy(X, z) \wedge poweredWith(y, Electricity) \wedge poweredWith(z, Electricity))$.

2.4 Standard operations

This subsection presents the usual elementary operations of the $\mathcal{SG}$-familly.

2.4.1 Subgraph

Given a SG G and a subset M of its markers, the subgraph induced by M is the subgraph G' induced by the set $C' \cup R'$ where C' is the set of concept nodes from G whose markers belong to M and R' is the set of relation nodes linked to *at least* one concept from C'. Note that according to the previous definition, a relation may have a different arity depending on wether it belongs to the SG or one of its subgraph. The notion of *sub-SG* has been defined in order to preserves the arity of relations.

Given a SG G and a subset M of its markers, the *sub-SG* of G induced by M is the subgraph G' of G induced by the set $R_M \cup C_M$ where (1) R_M is the set of all relations from G linked to at least one concept whose marker is in M and (2) C_M is the set of all concepts from G linked to a relation of R_M.

2.4.2 Merge

The *merge* $G \oplus H$ of two disjoint SGs G and H is the SG obtained by joining the node sets, the edge set and the labelling mapping of G with their homologous from H.

2.4.3 Subtraction

Given a SG G and one of its sub-SG H, the *subtraction* $G \ominus H$ of H from G is the subgraph of G induced by the set $R' \cup C'$ where (1) R' contains all relation nodes from G that are not in H and (2) C' contains all concept nodes from G linked to at least one relation node from R'.

2.4.4 Graph normal form

The normal form of a SG $G = (C, R, E, \lambda)$ is obtained by replacing each subset $C' \subseteq C(G)$ of concept nodes sharing the same marker m by a single concept node z such that (1) $z = \boxed{t : m}$ where t is the most specific type among all types of the concepts in C' (if such single type t exists) ; and (2) replacing by (r, i, z) each edge (r, i, c') in $E(R)$ where $c' \in C'$. According to the previous definition, the normal form is not computable when some concepts from C' have incomparable types. However, through the litterature (see [8] and [4]) some methods exists to solve such case. In the next section, the definition of the normal form is slightly modified in order to avoid this issue.

2.4.5 Graph unnormalized form

A simple graph is in *unnormalized form* according to a set M of markers if it does not contain any concept with marker from M linked to two distinct relations. Given a simple graph G and a set M of markers, the *unnormalized form* $uf(G, M)$ of G is the simple graph identical to G excepted that for each concept c linked by n edges (r_k, i_k, c) to n relation nodes r_k in G, n concepts c_k exist in $uf(G, M)$ such that (1) $c_k = \boxed{type(c) : marker(c)}$ and (2) c_k is linked by an edge (r_k, i_k, c_k) to the r_k relation node. The unnormalized form may be related to the notion of *piece* as defined in [17], although a piece is not usually restricted to a subset M of the markers of G.

2.4.6 Projection

The projection is the elementary reasoning operation of the $\mathcal{SG}$-familly. This operation is a kind of homomorphism that preserves the partial order defined on labels. Given G and H two SGs defined on a support S. A projection from G into H is a mapping π from $C(G)$ into $C(H)$ and from $R(G)$ into $R(H)$ which preserves edges as well as their numbering, and may specialize concept and relation node labels:

- $\forall (r, i, c) \in E(G) (\pi(r), i, \pi(c)) \in E(H)$
- $\forall x \in C(G) \cup R(H) \lambda_H(\pi(x)) \leq_\lambda \lambda_G(x)$

The projection operation is presented here only for informative purpose. It is not used in this paper which mainly focuses on relationships between $\mathcal{SGBF}$-model and $\mathcal{FOL}$.

3 The $\mathcal{SGBF}$-model of conceptual graphs

The $\mathcal{SGBF}$-model provides the expressivity to combine SGs with boolean connectors into expressions called *graph formulas*.

3.1 Boolean formulas of simple graphs

The definition of the $\mathcal{SGBF}$-model language relies on the $\{\wedge, \vee, \neg\}$ boolean connectors and on the notions of *variable marker*, *relational* support and *relational* simple graphs (which are specific types of support and SG).

3.1.1 Relational support

A *relational support* written $(T_R, \mathcal{I})$ is a support whose concept-type hierarchy is restricted to $\top$. Any support $S = (T_C, T_R, \mathcal{I})$ has a semantically equivalent *relational support* $rf(S) = (T'_R, \mathcal{I})$ (the *relational form* of S) identical to S excepted that the concept-type hierarchy under $\top$ has been moved under the $\top^1$ type of the unary relation-type hierarchy in T'_R such that the concept-type hierarchy is nothing but the type $\top$ (and thus does not appear anymore in the definition of the support). The figure 4 shows the relational form of the support of the figure 2.

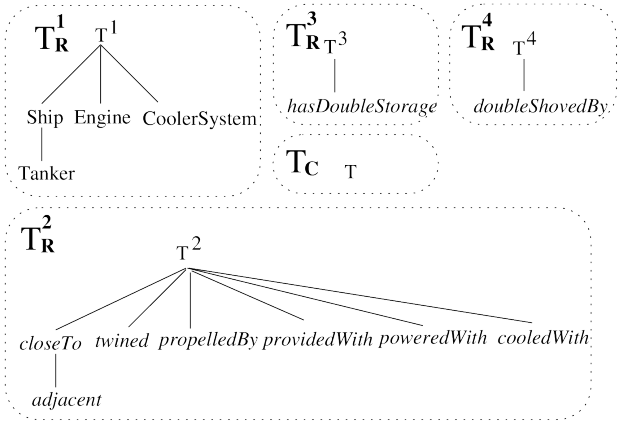


Figure 4: A relational support $S' = (T_R, \mathcal{I})$ which stands for the relational form of the support $S = (T_C, T_R, \mathcal{I})$ of the figure 2.

Given a support S, and its a *relational form* $rf(S)$, the language of all *relational simple graphs* defined over S is the language $\mathcal{SG}(rf(S))$ of all simple graphs defined over $rf(S)$. As shown in appendix, for any simple graph G defined over a support S there exists a semantically equivalent (i.e. with same logical semantics) simple graph G' defined over the relational form $rf(S)$ of the support S. Conversely, for any simple graph G' defined over a relational support S' there exists at least one (and possibly even more) simple graph G defined over a support S such that G and S are not necessarily relational, S' is the relational form $rf(S)$ of S, and G and S are respectively semantically equivalent to G' and S'. Thus, given a relational support S the language $\mathcal{SG}(rf(S))$ of all relational simple graphs defined over S has the same expressivity than the language $\mathcal{SG}(S)$ of all simple graphs defined over S.

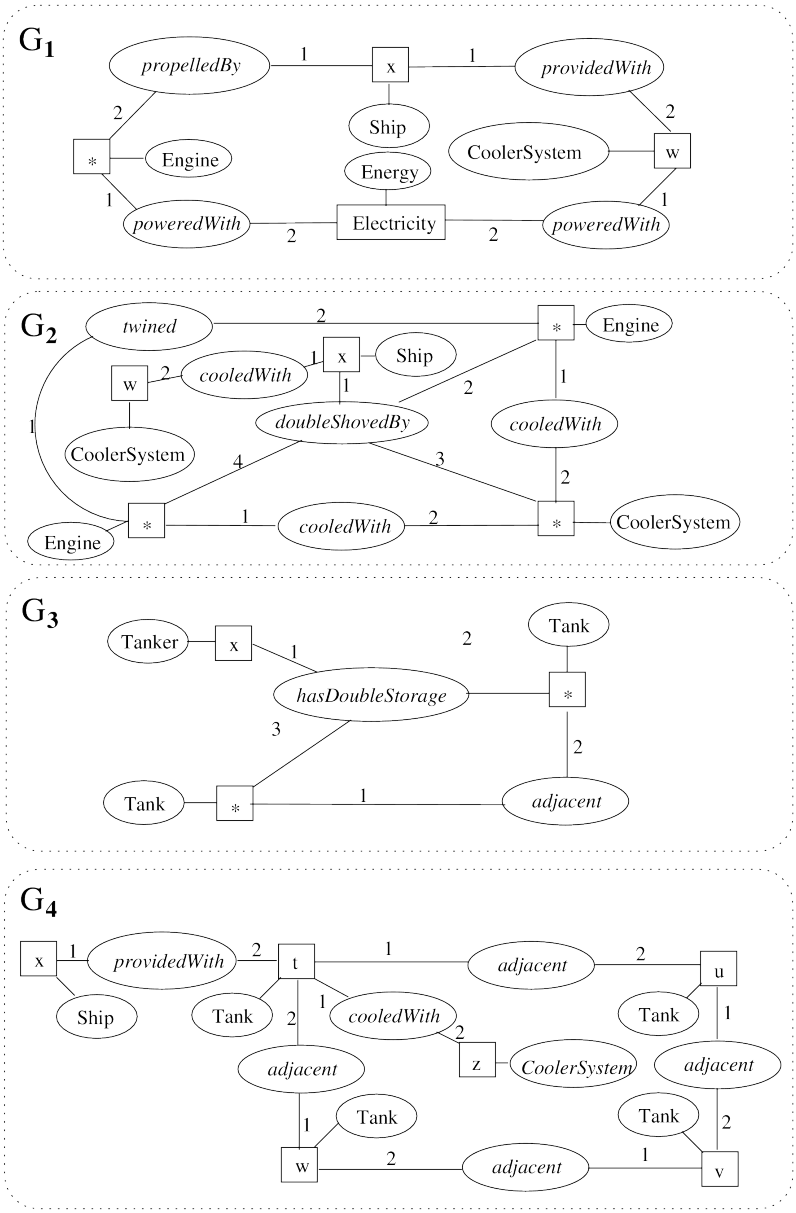
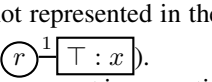
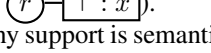


Figure 5: Three relational SGs corresponding to the SGs of figure 3.

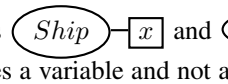
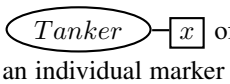
The figure 5 shows the relational SGs defined according to the support of the figure 2. These graphs are the relational version of those shown in figure 3. Note that in a *relational simple graph* defined over a support S, each concept type is $\top$. Indeed, since the support $rf(S)$ does not contain any other concept-type than $\top$, all concepts in simple graphs from $\mathcal{SG}(S)$ can not have nothing but the type $\top$. Since the concept-type $\top$ and the edge label 1 incident to any unary relation are ubiquitous in relational SGs, they are not represented in the figures (i.e. $\boxed{x}$ and  respectively stands for $\boxed{\top : x}$ and ).

Since (1) any support is semantically equivalent to its relational form and (2) simple graph languages defined over each of these supports have same expressivity, the notion of relational support can be used instead of that of support without loss of expressivity. Thus, supports and simple graphs defined in the remaining part of this paper are all relational.

3.1.2 Introduction of variables

We need to extend the definition of a SG in order to express that two generic concepts from two distinct SGs in a formula represent the same entity. This is done by using variable symbols. Given a set of variable symbols $\mathcal{V}$ disjoint from $\mathcal{I} \cup \{*\}$ and a SG (C, R, E, λ), the previous definition of $marker$ is now replaced by a total function from C into the set $\mathcal{M} = \mathcal{I} \cup \mathcal{V} \cup \{*\}$ which partial order still considers elements from $\mathcal{I}$ as pairwise incomparable ; and elements from $\mathcal{V} \cup \{*\}$ as pairwise equivalent and greater or equal than those from $\mathcal{I}$. Thus, $marker$ assigns to each concept of a graph either a variable, an individual, or a generic marker.

The notion of variable is taken into account in the logical semantics by updating the $term$ function in order to assign to a given variable from $\mathcal{V}$ a corresponding $\mathcal{FOL}$ variable with the same name.

For instance, in the unary-relations  and  of the graphs of figure 5, the symbol x denotes a variable and not an individual marker anymore.

3.1.3 v-normal form

Given a SG G and a subset M of the markers involved in G, the *sub-SG* of G induced by M is the subgraph G' of G induced by the set $R_M \cup C_M$ resulting from the union of (1) the set R_M of relations from G linked to at least one concept whose marker is in M and (2) the set C_M of all concepts from G linked to a relation of R_M.

A simple graph G is in *v-normal form* according to a set X of variables if the subgraph of G induced by $X \cup \mathcal{I}_G$ is in unnormalized form and the subgraph of G induced by $\mathcal{X}_G \setminus X$ is in normal form ; provided that $\mathcal{X}_G$ and $\mathcal{I}_G$ contains respectively (1) the variable markers and (2) the individual markers both appearing in G. The v-normal form $vnf(G, X)$ of a relational SG G is the graph equals to G excepted that the subgraph induced by $X \cup \mathcal{I}_G$ has been unnormalized while the subgraph induced by $\mathcal{X}_G \setminus X$ has been put in normal form where $\mathcal{X}_G$ and $\mathcal{I}_G$ are defined as above. Since any type of a relational SG concept node is $\top$, the normal form is always computable (types of concept are always comparable).

3.1.4 Definition

Given a relational support S, the language $\mathcal{SGBF}(S)$ of *graph formulas* over S is defined by the following rules :

- if $G \in \mathcal{SG}(S)$ and X is a subset of the variable markers appearing in G, then $G[X]$ is in $\mathcal{SGBF}(S)$
- if $A[X]$ and $B[Y]$ are in $\mathcal{SGBF}(S)$, then $(A \bullet B)[X \cup Y]$ is in $\mathcal{SGBF}(S)$ with $\bullet \in \{\wedge, \vee\}$
- if $A[X]$ is in $\mathcal{SGBF}(S)$, then $(\neg A)[X]$ is in $\mathcal{SGBF}(S)$

Given a graph formula $f \in \mathcal{SGBF}(S)$, the function $vars : \mathcal{SGBF}(S) \rightarrow 2^{\mathcal{V}}$ returns the set $vars(f)$ of variables appearing in the graphs of f.

The part $[X]$ of a graph formula $f = F[X]$ is referred as its *selection* part and represents significant variables which are selected in the *definition* part F of the formula. A graph formula $F[X]$ whose selection X is empty is written $F[]$ or F. A graph formula $F[\{x_1, x_2, ..., x_n\}]$ is also written $F[x_1, x_2, ..., x_n]$ and read as "entities x_1, x_2, ... and x_n verifying the definition F" while a graph formula $F[]$ (also written F) is read as "the formula whose definition is F". In this paper, logical formulas are usually denoted by the lowercase letter f, simple graphs by uppercase letters (E, G, H), supports by the letter S, graph formulas by lowercase letters (f, g, h, i) while their definition part and selection part by uppercase letters (respectively F, G, H and X, Y, Z).

$$F_1[x] = (G_1 \vee G_2)[x]$$
where G_1 and G_2 are the graphs from figure 5.

Figure 6: The graph formula $F_1[x]$.

$$F_2[x] = (G_3 \wedge \neg(G_1 \wedge \neg G_2))[x]$$
where G_1, G_2 and G_3 are the graphs from figure 5.

Figure 7: The graph formula $F_2[x]$.

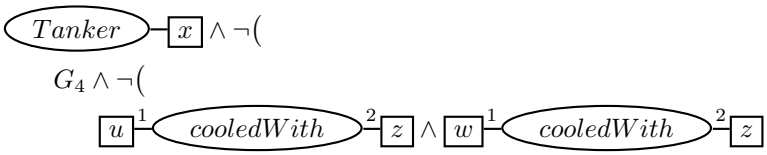
Figures 6, 7 and 8 show three graph formulas $F_1[x]$, $F_2[x]$ and $F_3[x]$. Each one is a boolean combination of graphs whose some of which comes from the figure 5.

(1) The disjunction $F_1[x]$ of the graphs G_1 and G_2 stands for `the ships` x `having either an electric engine or a cooler system and two independent engines using the same cooler system`.

(2) The conjunction $F_2[x]$ of the graph G_3 with the negation of $G_1 \wedge \neg G_2$ represents `the tankers` x `with a storage capacity of two adjacent tanks which are not propelled by an electric engine` v `and not provided with an electric cooler system unless the engine` v `shoving` x `is twined with another engine`.

(3) The formula $F_3[x]$ can be read as `the tankers` x `having no cooler system equipped tank` t `adjacent to two other tanks` u `and`

$F_3[x] = ($



$)[x]$

where G_4 is the graph from figure 5.

Figure 8: The graph formula $F_3[x]$.

`w adjacent to another one unless t, u and w share the same cooler system`.

3.2 Relationships with FOL

The next subsections present the $\mathcal{SGBF}$-model logical semantics and the translation from $\mathcal{FOL}$ to $\mathcal{SGBF}$-model.

3.2.1 From $\mathcal{SGBF}$-model to $\mathcal{FOL}$

Given a support S, an expression f from $\mathcal{SGBF}(S)$, the logical semantics $g2f(f)$ of f in $\mathcal{FOL}$ is given by the unary function $g2f$ such that :

- if $f = (A \bullet B)[X]$ then $g2f(f) = \exists\widetilde{x} g2f(A[X_A \cup \widetilde{x}]) \bullet g2f(B[X_B \cup \widetilde{x}])$ with $\bullet \in \{\wedge, \vee\}$ and $\widetilde{x} = vars(A) \cap vars(B) \setminus X$, $X_A = X \cap vars(A)$ and , $X_B = X \cap vars(B)$
- if $f = (\neg A)[X]$ then $g2f(f) = \neg g2f(A[X])$
- if $f = G[X]$ and G is a non-empty SG then $g2f(f) = \exists\widetilde{x} f_{_G}$ where $\widetilde{x} = free(f_{_G}) \setminus X$ contains new variable symbols and $f_{_G} = \Phi^{\wedge}(G)$[1] given that for any logical formula α, the function $free$ returns the set $free(\alpha)$ of *free* variable symbols in α.
- if f is the empty SG, then $g2f(f) = \top$ by convention

The $\mathcal{FOL}$ semantics for the graph formulas $F_1[x]$, $F_2[x]$ and $F_3[x]$ shown in figures 6 and 7 is respectively:

(1) $g2f(F_1[x]) = (\exists w$

$(\exists y Ship(x) \wedge Engine(y) \wedge CoolerSystem(w) \wedge Energy(Electricity) \wedge propelledBy(x, y) \wedge providedWith(x, w) \wedge poweredWith(y, Electricity) \wedge poweredWith(w, Electricity))$

[1]Note that since the $\mathcal{SGBF}$-model is based on the $\mathcal{SG}$-model, a SG from $\mathcal{SG}$-model or $\mathcal{SGBF}$-model has the same logical semantics.

$\vee$

$(\exists p \exists q \exists r Ship(x) \wedge Engine(p) \wedge Engine(q) \wedge CoolerSystem(r) \wedge CoolerSystem(w) \wedge doubleShovedBy(x,p,q,r) \wedge twined(p,q) \wedge cooledWith(p,r) \wedge cooledWith(q,r) \wedge cooledWith(x,w))$

)

(2) $g2f(F_2[x]) = ($

$(\exists y \exists z Tanker(x) \wedge Tank(y) \wedge Tank(z) \wedge hasDoubleStorage(x,y,z) \wedge adjacent(y,z))$

$\wedge \neg ($

$(\exists v \exists w Ship(x) \wedge Engine(v) \wedge CoolerSystem(w) \wedge Energy(Electricity) \wedge propelledBy(x,v) \wedge providedWith(x,w) \wedge poweredWith(v,Electricity) \wedge poweredWith(w,Electricity))$

$\wedge \neg ($

$(\exists q \exists r Ship(x) \wedge Engine(v) \wedge Engine(q) \wedge CoolerSystem(r) \wedge CoolerSystem(w) \wedge doubleShovedBy(x,v,q,r) \wedge twined(p,q) \wedge cooledWith(v,r) \wedge cooledWith(q,r) \wedge cooledWith(x,w))$

)

)

)

(3) $g2f(F_3[x]) = ($

$Tanker(x)$

$\wedge \neg (\exists u \exists v \exists z$

$(\exists t \exists w Ship(x) \wedge Tank(t) \wedge Tank(u) \wedge Tank(v) \wedge Tank(w) \wedge CoolerSystem(z) \wedge providedWith(x,t) \wedge adjacent(t,u) \wedge adjacent(u,v) \wedge adjacent(v,w) \wedge adjacent(w,t) \wedge cooledWith(t,z))$

$\wedge \neg ($

$cooledWith(u,z) \wedge cooledWith(w,z)$

)

)

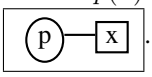
)

The three previous logical formulas are not quantifier closed. Indeed, as the variable x appear in the *selection* of $F_1[x]$, $F_2[x]$ and $F_3[x]$, the logical translation procedure avoid the existential quantification of x. More generally, variables appearing in the selection X of a graph formula $F[X]$ are free (i.e. unquantified) in the corresponding logical formula. When the selection is empty, the resulting logical formula is quantifier closed. The logical translation for each of the three previous examples $F_1[]$, $F_2[]$ and $F_3[]$ this time with an empty selection is $\exists x g2f(F[x])$ where $F \in \{F_1, F_2, F_3\}$.

3.2.2 From $\mathcal{FOL}$ to $\mathcal{SGBF}$-model

This section presents the mapping $f2g$ from $\mathcal{FOL}$ into $\mathcal{SGBF}$-model. Given a logical formula f, $f2g(f)$ returns a graph formula by applying the rules listed below. Let f^r be the variable renaming of f such that two variables appearing respectively in two distinct quantifiers have different names :

- if $f^r = A \to B$ then $f2g(f) = f2g(\neg A \vee B)$
- if $f^r = A \bullet B$ then $f2g(f) = (\alpha \bullet \beta)[X_\alpha \cup X_\beta]$ with $\bullet \in \{\wedge, \vee\}$, $\alpha[X_\alpha] = f2g(A)$ and $\beta[X_\beta] = f2g(B)$
- if $f^r = \neg A$ then $f2g(f) = (\neg\alpha)[X_\alpha]$ with $\alpha[X_\alpha] = f2g(A)$
- if $f^r = \forall x A$ then $f2g(f) = f2g(\neg\exists x \neg A)$
- if $f^r = \exists x A$ then $f2g(f) = (\alpha \wedge (top(x) \vee \neg top(x)))[X_\alpha \setminus \{x\}]$ where $\alpha[X_\alpha] = f2g(A)$ and $top(x)$ is the SG with only one concept node $\boxed{x}$
- if f^r is a predicate $p(t_1, ..., t_n)$ then $f2g(f) = G[X]$ where G is a SG with one concept $c_i = \boxed{m_i}$ for each term t_i and one relation node r of type p linked to each c_i by an edge (r, i, c_i). The marker m_i is the individual or variable from $\mathcal{I} \cup \mathcal{V}$ respectively corresponding to the i-th constant or variable t_i in p. X contains all variable markers appearing in G.

The previous procedure translates any $\mathcal{FOL}$ logical formula into a $\mathcal{SGBF}$-model graph formula. Any logical formula f with free variables $free(f)$ lead to a graph formula $F[free(f)]$ whose selection is the free variables of the logical formula. If the logical formula is quantifier closed then the selection of the resulting graph formula is empty. Furthermore, notice that the *fifth* translation step (of the $\exists$ quantifier) adds to the resulting conjunction a graph formula $N = (top(x) \vee \neg top(x))$. This step is necessary to guarantee that every logical formula may be translated into a graph formula. Indeed, if we remove N from this step, the translation rule would be "if $f = \exists x A$ then $f2g(f) = f2g(A)$", and some non-equivalent formulas as $\exists x \neg p(x)$ and $\neg\exists x p(x)$ would have the same corresponding graph formula $(\neg H)[]$ where $H[x] =$ 

.

3.3 Language Properties

3.3.1 Standard logical properties

Given a relational support S, and two graph formulas f and g from $\mathcal{SGBF}(S)$ we say that

- f is *logically satisfiable* (resp. *logically unsatisfiable*) iff the conjunction $\phi(S) \wedge g2f(f)$ of the logical translation of f and the support S is *satisfiable* (resp. *unsatisfiable*),
- f *logically entails* g iff $\phi(S) \wedge g2f(f) \wedge \neg g2f(g)$ is unsatisfiable,
- f and g are *logically equivalent* (which is written $f \equiv g$) iff each one *logically entails* the other.

Standard laws :

- Associativity : $(x \bullet (y \bullet z))[X] \equiv ((x \bullet y) \bullet z)[X]$
- Commutativity : $(x \bullet y)[X] \equiv (y \bullet x)[X]$
- Distributivity : $(x \bullet (y \circ z))[X] \equiv ((x \bullet y) \circ (x \bullet z))[X]$
- Double negation : $(\neg\neg x)[X] \equiv x[X]$
- Restricted de Morgan's : $(\neg(x \bullet y))[X] \equiv (\neg x \circ \neg y)[X]$ iff $vars(x) \cap vars(y) \subseteq X$
- Renaming : $(x \vee y)[X] = (x \vee y')[X]$ where y' is the graph formula y in which all variables from $vars(x) \cap vars(y) \setminus X$ are renamed such that $vars(x) \cap vars(y') \setminus X = \emptyset$

Graph laws :

- Graph merging : $(G \wedge H)[X] \equiv M[X]$ where M is the SG resulting from the merge $G \oplus H$ of G and H
- Graph decomposition : $G[X] \equiv (G_1 \wedge G_2 \wedge ... \wedge G_n)[X]$ where G_i are the connected components of G provided that $G[X]$ is in v-normal form

Graph properties :

- $G[X]$ is in *v-normal form* iff G is in v-normal form according to X

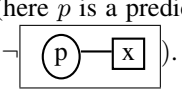
Figure 9: Algebraic laws and some properties of the $\mathcal{SGBF}$-model ; given $\bullet \neq \circ$ members of $\{\vee, \wedge\}$ and a relational support $S = (T_R, \mathcal{I})$ on which are defined two SGs G, H and three graph formula definitions x, y, z.

3.3.2 Algebraic laws

The figure 9 presents the algebraic laws of connectors of $\mathcal{SGBF}$-model. The disjunctive and conjunctive connectors $\{\vee, \wedge\}$ respect the usual laws of associativity, commutativity and distributivity. However connectors are no longer idempotent while using negations. For example, a graph formula $(\neg A \wedge \neg A)[X]$ is not *logically equivalent* to $(\neg A)[X]$ when $vars(A) \setminus X \neq \emptyset$. Indeed $g2f((\neg A \wedge \neg A)[X])$ returns $\exists \widetilde{x} \neg g2f(A) \wedge \neg g2f(A)$ with $\widetilde{x} = vars(A) \setminus X$ but not $\neg g2f(A[X])$ (which is the logical translation of $(\neg A)[X]$). In the other hand, a graph formula $(\neg A \wedge \neg A \wedge ... \wedge \neg A)[X]$ is *logically equivalent* to $(\neg A \wedge \neg A)[X]$. However, if the same repeated formula A does not contain any negation, $(A \wedge A \wedge ... \wedge A)[X]$ is *logically equivalent* to $A[X]$ (the same holds for $\vee$).

3.3.3 Expressivity

$\mathcal{FOL}$ and $\mathcal{SGBF}$-model have the same expressive power. Without loss of information: each graph formula can be translated into a logical formula and each logical formula may be translated into a graph formula. Given a logical formula f, $g2f(f2g(f))$ may return a formula different from f. However f and $g2f(f2g(f))$ are always semantically equivalent. Since $f2g$ adds expressions of form $(top(x) \vee \neg top(x))$ at the $\exists$-step of the translation, expressions of the same form may also appear in the logical formula $g2f(f2g(f))$. Such expressions do not change the semantics of the source formula f as they are always true and appear in a conjunction (a logical formula A being always equivalent to $A \wedge (top(x) \vee \neg top(x))$ whatever the quantification of x).

Universal quantification may be expressed in a graph formula by replacing the universal quantifier by a negated existentially quantified negation as shown in section 3.2.2. Given two logical formulas A and B, the quantifier closed implication $\forall \widetilde{x}(A \rightarrow B)$ where $\widetilde{x} = free(A) = free(B)$ may be rewritten as $\neg\exists\widetilde{x}(A \wedge \neg B)$ and can be translated to a graph formula $\neg(f2g(A) \wedge \neg f2g(B))$. A less trivial case is the formula $\forall x p(x)$ (here p is a predicate), whose corresponding graph formula is $\neg((\boxed{x} \vee \neg\boxed{x}) \wedge \neg$ ).

3.3.4 Relationships with conceptual graph propositions

graph formulas are very close to *nested conceptual graphs* with negation (see [21]) also known as *conceptual graph propositions* (see [3]). A conceptual graph proposition is defined as either a negated graph proposition or a box containing a SG and finitely many graph propositions. The language of a graph proposition is generated by the grammar $p ::= [gp^*]|\neg p$ where g denotes a SG. A box $[...]$ is called a *context* and has a conjunctive semantics. *Graph formulas* and *graph propositions* have the same expressivity and a similar $\mathcal{FOL}$ semantics. The fundamental differences are (1) the disjunction operator which does not explicitly exists in graph propositions and (2) the variables which are replaced in graph propositions by equality links named *coreference*. However, these are only syntactic differences since coreference links may correspond to homonymous variables and a disjunction $A \vee B$ may be represented as the semantically equivalent graph proposition $\neg[g_\emptyset \neg A' \neg B']$ provided that $g_\emptyset$ is the empty SG and A' and B' are respectively the graph propositions corresponding to A and B.

4 Graph arborescences, forests and clauses

Any graph formula may be rewritten without using disjunctions such that each of its conjunctions contains any number of negated other such conjunctions and not more than one positive graph. Such a *single-graph conjunctive formula* may be represented as an *arborescence* of graphs whose structure relies on the dependencies between positive graphs and negated formulas. Graph arborescences are collected in sets called *graph forests* which may be collected into collections called *clauses of forests*. A forest represents the conjunction of the formulas corresponding to its arborescences while a clause stands for the disjunction of the formulas corresponding to its forests. Graph arborescences, forests and clauses are used to represent graph formulas in a more compact way and to provide a much simpler basis to define a normal form for the graph formula formalism : the *indivisible normal form*.

4.1 Definitions

4.1.1 Graph arborescence

Given a relational support S, a graph arborescence defined over S is a couple $(\alpha, \mathcal{G})$ where $\mathcal{G}$ is a subset of $\mathcal{SG}(S)$ and $\alpha : \mathcal{G} \rightarrow \mathcal{G}$ is the "parent" function of the arborescence such that :

- there exists only one relational SG $R = root(A) \in \mathcal{G}$ for which $\alpha(R)$ is not defined.
- there does not exist any cycles in α, i.e. for any $n \in \mathbb{N}$ there does not exist any $G_1, G_2, ..., G_n \in \mathcal{G}$ such that $\alpha(G_1) = G_2$, $\alpha(G_2) = G_3$, ..., $\alpha(G_n) = G_1$.

4.1.2 Forest

A *graph forest* F is a set of graph arborescences.

4.1.3 Clause

A *clause* C is a collection of forests.

4.1.4 Selection

A graph arborescence $A = (\alpha, \mathcal{G})$, a forest F and a clause C may be associated to a subset X of the variables occurring in the graphs they contain. In such a case, they are respectively written $a = A[X]$, $f = F[X]$ and $c = C[X]$ where A, F and C are the *definition* part and X is the *selection* part of respectively a, f and c.

4.2 Elementary operators

The following define some operators on graph arborescences.

4.2.1 Operators on variables

Variables of arborescences, forests and clauses

We extend $vars$ such that for (1) a given graph arborescence $A = (\alpha, \mathcal{G})$, (2) a given forest $F = \{A_1, A_2, ..., A_n\}$ and (3) a given clause $C = \{F_1, F_2, ..., F_n\}$, $vars(A)$ respectively returns (1) the set of variables occurring in $\mathcal{G}$, (2) the set $\bigcup_{A_i \in F} vars(A_i)$ of variables occurring in all arborescences $A_i \in F$ and (3) the set $\bigcup_{F_i \in C} vars(F_i)$ of variables occurring in all forests $F_i \in C$.

Bound variables

We define the operator $\mathcal{B}$ which return the set $\mathcal{B}(G, A[X])$ of *bound variables* of a graph $G \in \mathcal{G}$ according to an arborescence $A[X]$ where $A = (\alpha, \mathcal{G})$ such that $\mathcal{B}(G, A[X])$ contains the variables from G appearing in X or the other graphs of A. More formally, $\mathcal{B}(G, A[X]) = (X \cap vars(G)) \bigcup_{G' \in \mathcal{G} \setminus \{G\}} vars(G) \cap vars(G')$.

We extend the previous operator $\mathcal{B}$ in order for the first argument to be a subarborescence of the second argument. $\mathcal{B}(A', A[X])$ returns the variables that the graphs

of A' share with X or the graphs from $A[X]$ not in A'. More formally, given a subarborescence $A' = (\alpha_{A'}, \mathcal{G}_{A'})$ of an arborescence $A[X]$ where $A = (\alpha_A, \mathcal{G}_A)$, $\mathcal{B}(A', A[X]) = (X \cap vars(A')) \bigcup_{G \in \mathcal{G}_A \setminus \mathcal{G}_{A'}} vars(G) \cap vars(A')$.

Renaming of variables
Given an arborescence $A[X]$, the renaming $\rho(A[X])$ of A according to X returns a new arborescence equals to A whose each variable not in X has a name which does not appear anymore in any other arborescence.

4.2.2 Operators on filiation or arborescences

Given an arborescence $a = A[X]$ where $A = (\alpha, \mathcal{G})$ and two SGs $G, G' \in \mathcal{G}$ we consider the following elementary operators :

- $\alpha(G)$ returns the parent of G
- $children(G, A)$ returns the set $\{H \in \mathcal{G} | \alpha(H) = G\}$
- $sub(G, A)$ returns the subarborescence A' from A such that $root(A') = G$
- $subs(G, A)$ returns the set containing each subarborescence A' from A such that $root(A') \in children(G, A)$. Furthermore, $subs(A)$ stands for $subs(root(A), A)$ and returns the *top-subarborescences*.

Furthermore, we extend the *merge* operator $\oplus$ to make it applicable to arborescences. Thus, the merge of two arborescences A_1 and A_2 is the arborescence $A = A_1 \oplus A_2$ such that (1) its root $root(A)$ is the merge $root(A_1) \oplus root(A_2)$ of the roots of A_1 and A_2 and (2) its top-subarborescences $subs(A) = subs(A_1) \cup subs(A_2)$ is the union of the top-subarborescences of A_1 and A_2.

4.2.3 Translation to graph formula

The *semantics of a graph arborescence* $A[X]$ in graph formula, is a graph formula $\mathcal{A}2g(A[X]) = H[X]$ such that $H = root(A) \wedge \neg\mathcal{A}2g(A_1) \wedge \neg\mathcal{A}2g(A_2) \wedge ... \wedge \neg\mathcal{A}2g(A_n)$ where $A_i \in subs(A)$.

The *semantics of a forest* $F[X]$ with $F = \{A_1, A_2, ..., A_n\}$ in graph formula is the graph formula $\mathcal{F}2g(F[X]) = H[X]$ such that $H = \mathcal{A}2g(A_1[X \cap vars(A_1)]) \wedge \mathcal{A}2g(A_2[X \cap vars(A_2)]) \wedge ... \wedge \mathcal{A}2g(A_n[X \cap vars(A_n)])$ of the formulas resulting from the translation of the arborescences in $F[X]$.

The *semantics of a clause* $C[X]$ with $C = \{F_1, F_2, ..., F_n\}$ in graph formula is the disjunction $\mathcal{C}2g(C[X]) = H[X]$ such that $H = \mathcal{F}2g(F_1[X \cap vars(F_1)]) \vee \mathcal{F}2g(F_2[X \cap vars(F_2)]) \vee ... \vee \mathcal{F}2g(F_n[X \cap vars(F_n)])$ of the formulas resulting from the translation of the forests in $C[X]$.

Note that the semantics of an arborescence A'' resulting from the merge $A''[X' \cup X] = A' \oplus A$ of two arborescences A' and A is equivalent to the conjunction of the graph formulas corresponding to $A'[X']$ and $A[X]$.

4.2.4 Translation to graph arborescence

Any graph formula $f = F[X]$ may be translated into a graph arborescence $g2\mathcal{A}(f) = A[X]$ provided that f is in a specific form called *the single-graph conjunctive form*.

A graph formula $f = F[X]$ is a single-graph conjunctive formula iff $F = G \wedge \neg F_1 \wedge \neg F_2 \wedge ... \wedge \neg F_n)$ where G is a SG and all F_i are single-graph conjunctive formulas. Any graph formula $f = F[X]$ has a single-graph conjunctive form obtained by :

- computing the conjunctive form of F : this is done by recursively rewriting each expression $(d \vee e)$ into $(\neg(G_\emptyset \wedge \neg d \wedge \neg e)) \wedge \boxed{\boxed{v_1}\boxed{v_2}...\boxed{v_n}}$ where $G_\emptyset$ is the empty graph and each variable v_i is shared by d and e (i.e. $v_i \in vars(d) \cap vars(e)$ for $1 \leq i \leq n$)
- adding an empty-graph into any conjunction : each conjunction $(d \wedge e)$ is rewritten $((d \wedge e) \wedge G_\emptyset)$
- merging all graphs not separated by negations : this is done by replacing until stabilisation of F each expression $(G_1 \wedge (G_2 \wedge e))$ by $(G \wedge e)$ where e is a graph formula, G_1, G_2 are SGs and $G = G_1 \oplus G_2$ is the SG resulting of the merge of G_1 with G_2

Given a graph formula $f = F[X]$, its single-graph conjunctive form $h = H[X]$ where $H = G \wedge \neg H_1 \wedge \neg H_2 \wedge ... \wedge \neg H_n$, $g2\mathcal{A}(f)$ is the graph arborescence $A[X]$ whose selection is equal to that of f, the root is G and the top-subarborescences are the graph arborescences $g2\mathcal{A}(H_i)$ corresponding to all the H_i (i.e. $root(g2\mathcal{A}(f)) = G$ and $subs(g2\mathcal{A}(f)) = \{g2\mathcal{A}(H_i) | 1 \leq i \leq n\}$). A graph formula f is always logically equivalent to the translation $\mathcal{A}2g(g2\mathcal{A}(f))$ in graph formula of its corresponding arborescence $g2\mathcal{A}(f)$. The figure 10 shows the two arborescences $A_2[x] = g2\mathcal{A}(F_2[x])$ and $A_3[x] = g2\mathcal{A}(F_3[x])$ corresponding respectively to the graph formulas $F_2[x]$ and $F_3[x]$.

5 The indivisible normal form

This subsection presents the notion of *indivisibility* of graphs and arborescences. Informally, the indivisibility of graphs and arborescences is closely related to the inability to partition them into a semantically equivalent set of more compact subarborescences or subgraphs.

5.1 Indivisibility

A clause is said to be *indivisible* if its forests or their arborescences can not be decomposed into more elementary forests or arborescences while preserving their graph formula semantics. *Indivisible* clauses stand for a *normal form* of the graph formula formalism.

5.1.1 Graph indivisibility

A simple graph $G[X]$ is *indivisible* iff (1) G is connected (i.e. G has only one connected component), (2) $G[X]$ is in v-normal form. Note that in an indivisible SG $G[X]$, (1) two distinct concepts may share the same variable x only if $x \in X$ and (2) a concept is linked to two distinct relations iff its marker is a variable $y \notin X$.

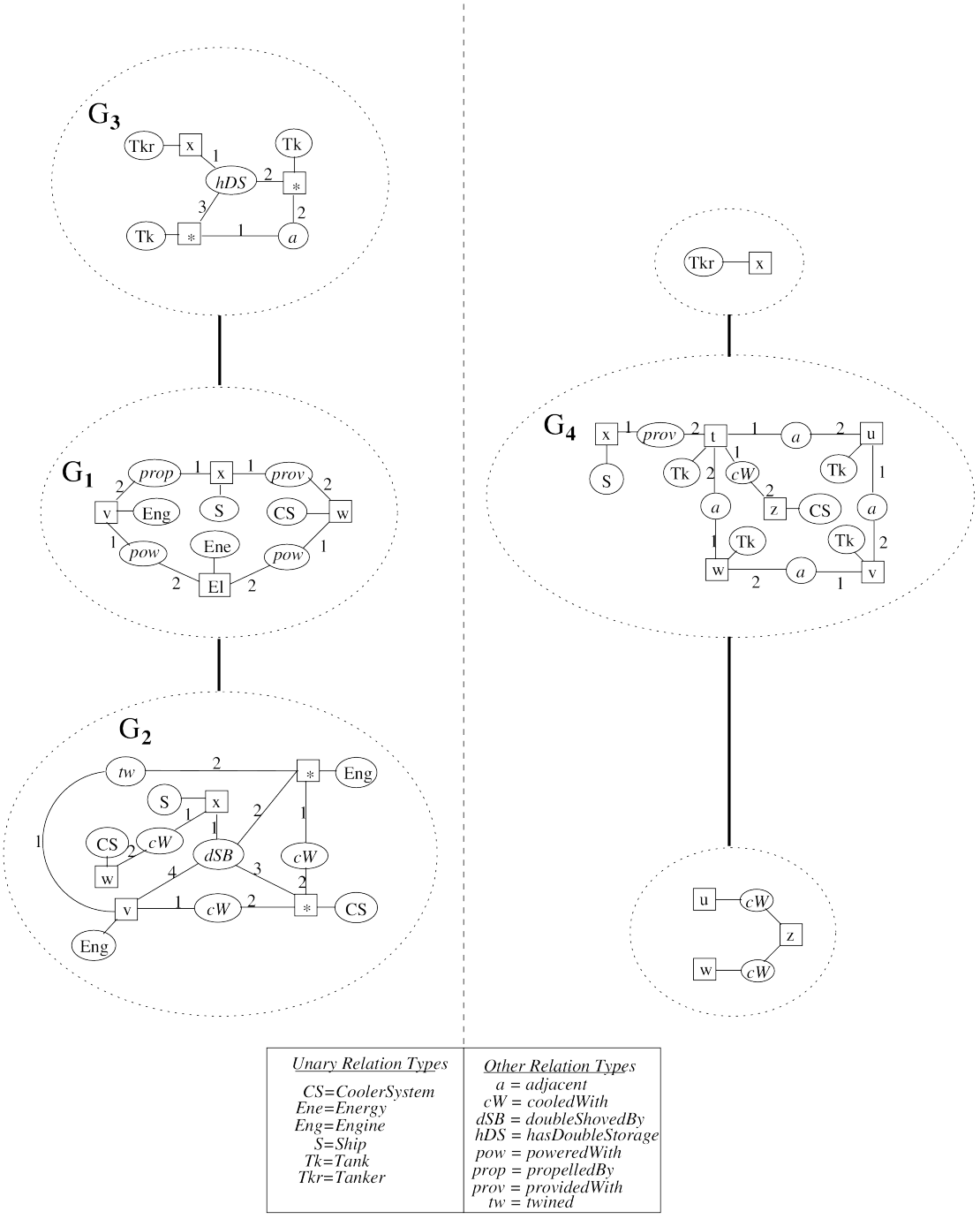


Figure 10: The two arborescences $A_2[x]$ and $A_3[x]$ corresponding respectively to the graph formulas $F_2[x]$ and $F_3[x]$

5.1.2 Arborescence top-indivisibility

Given a graph arborescence $A[X]$, the undirected graph $\mathcal{D}(A[X]) = (V, E)$ of *variable top-dependencies* is defined as follows :

- each top subarborescence in $subs(A[X])$ is a vertex of $\mathcal{D}(A[X])$
- $root(A)$ is a vertex of $\mathcal{D}(A[X])$ iff it is not the empty graph
- there exists an undirected edge $\{x, y\}$ between to vertices x and y of $\mathcal{D}(A[X])$ iff x and y share at least one variable not in X (i.e. $\{x, y\} \in E$ iff $vars(x) \cap vars(y) \setminus X \neq \emptyset$)

A graph arborescence $A[X]$ is said to be *top-indivisible* iff :

- the graph $R[X]$ is *indivisible* where $R = root(A)$
- $\mathcal{D}(A[X])$ contains only *one connected component*.

Informally, if the graph $\mathcal{D}(A[X])$ has only *one connected component* then for any vertices $a \neq b$ in $\mathcal{D}(A[X])$, we can always find a path P starting at a and ending at b such that each vertex in P (i.e. a SG or a top-subarborescence) shares with each of its neighbours some homonymous variable not appearing in X. This means that $root(A)$ and its subarborescences are "stuck" together by their shared variables and can not be separated one from the other without changing the semantics of the arborescence.

5.1.3 Arborescence indivisibility

A graph arborescence $A[X]$ where $A = (\alpha, \mathcal{G})$ is said to be *indivisible* iff :

- $A[X]$ is *top-indivisible*
- each top-subarborescence $A'[X']$ in A (if any) is *indivisible* where $A' \in subs(A)$ and $X' = \mathcal{B}(A', A[X])$

5.1.4 Forest indivisibility

A forest $F[X] = \{A_1, A_2, ..., A_n\}$ is said to be *indivisible* iff each of its arborescences $A_i[X \cap vars(A_i)]$ is indivisible.

5.1.5 Clause indivisibility

A clause $C[X] = \{F_1, F_2, ..., F_n\}$ is said to be *indivisible* iff each of its forests $F_i[X \cap vars(F_i)]$ is indivisible.

5.1.6 Observations

Indivisible graph arborescences are used to structure graph formulas in layers of irreducible negations organized according to their variable dependencies. Given a SG G, all subarborescences starting at one child of G (if any) are considered as negated relatively to G (which may be itself the start of a subarborescence negated relatively to its parent if any). Such negations are said irreducible in the sense that we can not refactor the arborescence into a shorter one without changing its semantics (we can not remove these negations). The deeper an indivisible graph arborescence is, the more irreducible

negations are nested in the corresponding graph formula, and the more complex is the formula.

An indivisible forest may be considered as a conjunctive clause in which each graph arborescence is kept separated from the other in order to preserve the *indivisibility* condition. Indeed, the merge of all these graph arborescences into a single one (by merging all roots into a single parent of all merged roots subarborescences) lead to a graph arborescence that is not anymore *top-indivisible*.

An indivisible clause $C[X]$ may be considered as a disjunctive clause in which forests are totally independent in the sense that if we rename variables out of X in one forest without minding the others, this does not affect the semantics of the clause.

5.2 Rewriting rules

This subsection presents the rules whose repeated application on an arborescence leads to its indivisible form while preserving its semantic. The preservation of the semantics is proven in appendix C. The application of a rule on a specific graph G of a given arborescence A always returns a clause of forests containing the arborescences induced by the rewriting of A.

5.2.1 The top splitting rule TS

Given an arborescence $A[X]$ and a graph G in $A[X]$, let A_G be the subarborescence $sub(G, A)$ starting at G in A, if A_G is already top-indivisible then the application $TS(G, A[X])$ of the *top splitting rule* at G in the arborescence A returns the clause $C = \{\{A[X]\}\}$. Otherwise, A_G is not top-indivisible, and there exists two subarborescences A^1_G and A^2_G such that :

- A^1_G is top-indivisible such that $root(A^1_G) \neq G_\emptyset$,
- $vars(A^1_G) \cap vars(A^2_G) \subseteq X$
- A_G is equal to $A^1_G \oplus A^2_G$

In that case, the application $TS(G, A[X])$ returns a clause C whose content varies according to the following cases :

<u>G is root</u>
If G is the root of A then the clause $C = \{\{A^1_G, A^2_G\}\}$ consists in a single forest containing A^1_G and A^2_G.

<u>G is a root child</u>
If G is a root child in A then the clause $C = \{\{\rho(B_1[X])\}, \{\rho(B_2[X])\}\}$ consists in two forests, each one containing respectively the renamings $\rho(B_1[X])$ and $\rho(B_2[X])$ of the two arborescences B_1 and B_2 whose root is that of A and whose top-subarborescences are those of A among which A_G has been respectively replaced by A^1_G and A^2_G. In the figure 12 the quotes and double-quotes stands for the two different renamings of the resulting arborescences.

<u>Otherwise : G is any other node</u>
Let G be a child of a graph F whose parent is E in A then the clause $C = \{\{B[X]\}\}$

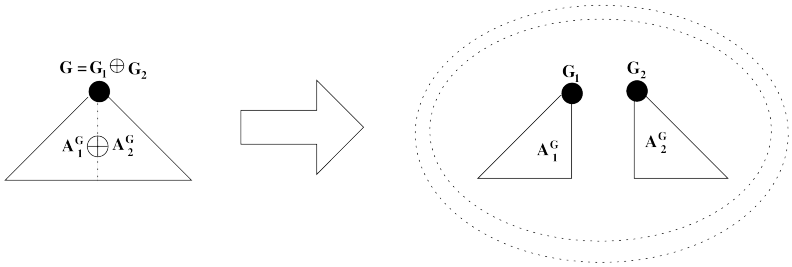


Figure 11: The *root* application case of the TS splitting rule.

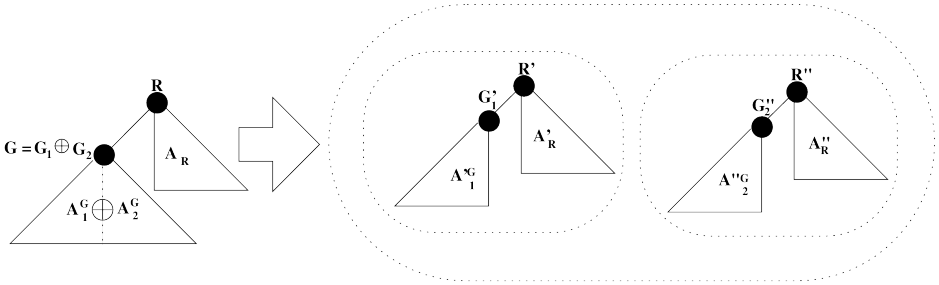


Figure 12: The *root-child* application case of the TS splitting rule.

consists in a single forest containing one arborescence $B[X]$ equals to $A[X]$ excepted that the subarborescence $A_F = sub(F, A)$ starting at F has been replaced in B by the two arborescences obtained by applying the *top-splitting rule* $TS(G, A_F[X_F])$ at G in the subarborescence A_F with $X_F = \mathcal{B}(A_F, A[X])$.

5.2.2 The empty-graph elimination rule

Given a graph arborescence $A[X]$ and a graph G, if G is not empty then the application $EGE(G, A[X])$ of the *empty-graph elimination rule* at G in the arborescence A returns the clause $C = \{\{A[X]\}\}$. Otherwise G is empty and $EGE(G, A[X])$ returns a clause C whose content varies according to the following cases :

G is the root
$C = \{\{A[X]\}\}$ consists in the single forest containing $A[X]$.

G is a root child
If G is an empty child of the root F of A then

- if G is a leaf in A then $C = \{\{A_\emptyset\}\}$ where $A_\emptyset$ denotes the empty arborescence
- otherwise : $C = \{\{A_G\}\{\rho(A_F \oplus A_H)\}\}$ provided that G is the parent of a non-empty set $\{A_H\} \cup \Xi$ of subarborescences ; and A_G, A_F are both equal to A excepted that $subs(G, A_G) = \Xi$ and $subs(A_F) = subs(A) \setminus \{sub(G, A)\}$

Otherwise : G is any other node

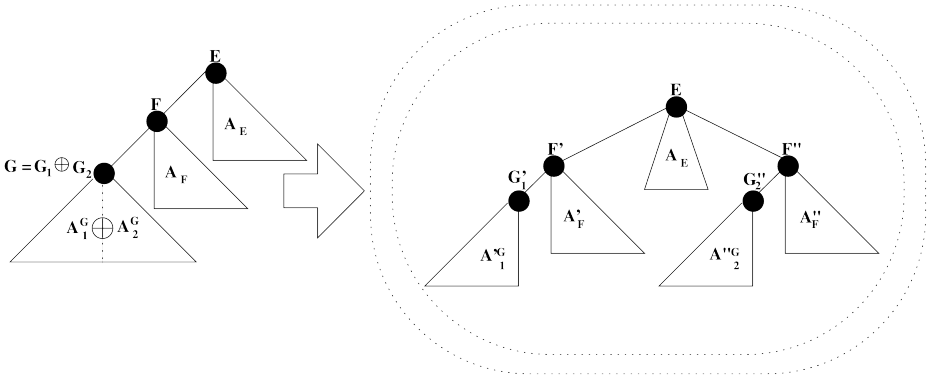


Figure 13: The *any-other-node* application case of the TS splitting rule.

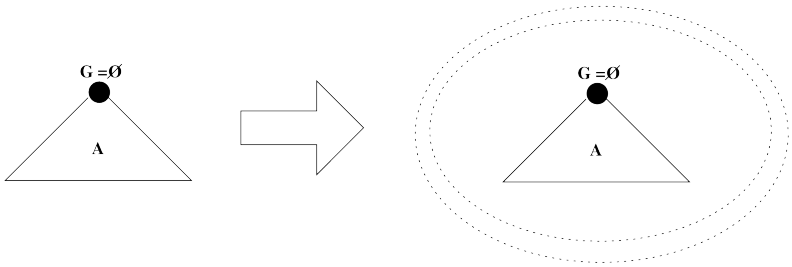


Figure 14: The *root* application case of the EGE splitting rule.

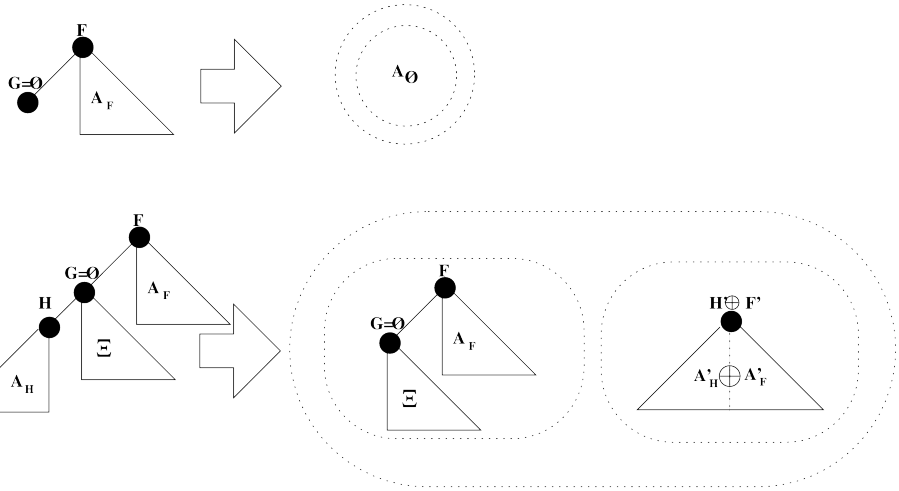


Figure 15: The *root-child* application case of the EGE splitting rule.

Let G be a child of a graph F whose parent is E in A and $R = EGE(G, A_F[X_F])$ the result of the *empty-graph elimination rule* application at G in the subarborescence A_F with $X_F = \mathcal{B}(A_F, A[X])$ then the clause $C = \{\{B[X]\}\}$ consists in a single forest containing one arborescence $B[X]$ equals to $A[X]$ excepted that :

- if the only arborescence in R is the empty arborescence $A_\emptyset$ then the subarborescence $A_F = sub(F, A)$ starting at F in A has been removed from B,
- otherwise A_F has been replaced in B by the two arborescences in R.

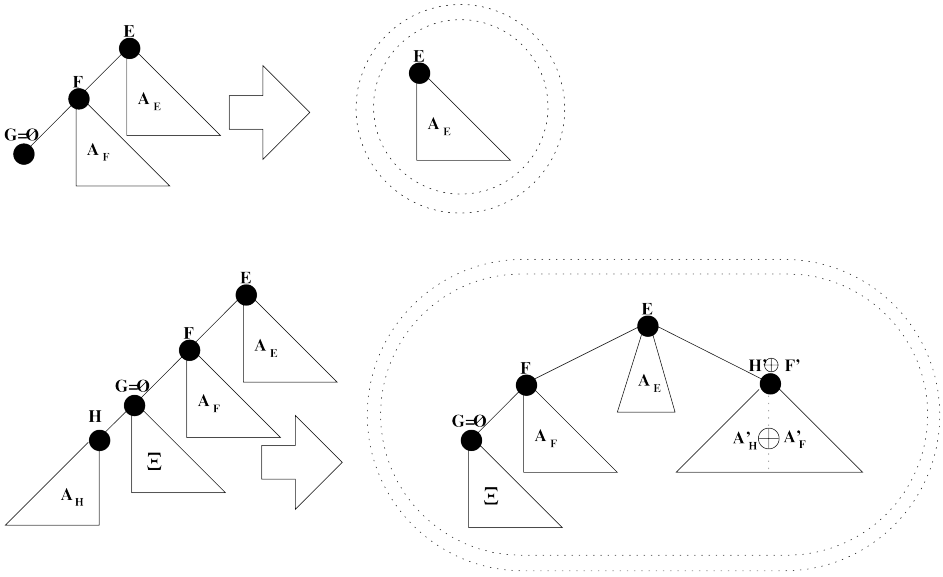


Figure 16: The *any-other-node* application case of the EGE splitting rule.

5.2.3 Extending the rules

In order to compose each of the two previous rules with the other, the previous rules are extended to make them applicable to a given clause. The application $S(C)$ of a rewriting rule S on a given clause C returns a clause R such that (1) R is set equal to C and (2) for any forest $F \in R$, arborescence $A \in F$ and graph G in A :

- let R' be the result of the application $S(G, A)$ of the rule S on G in A
- replace in R the forest F by as many new forests F_i identical to F than there exist forests F'_i in R' such that A is replaced in F_i by the arborescences from F'_i. More formally, $R := (R \setminus F) \cup \{F_i | F_i = (F \setminus A) \cup F'_i \wedge F'_i \in R'\}$.

5.3 The procedure

Any graph formula $F[X]$ has a semantically equivalent indivisible clause $inf(F[X])$ computable by an algorithm $inf()$ which migrates disjunctive contents as more as possible to the top and negative contents as deeper as possible by applying until stabilization the two previous rewriting rules on the clause corresponding to $F[X]$. This

algorithm always terminates and returns the clause $C[X] = inf(F[X])$ such that $\mathcal{C}2g(C[X])$ is logically equivalent to $F[X]$ since each application of the rewriting rules preserves the semantics of the clause.

Given a graph formula $F[X]$, the *indivisible normal form* $inf(F[X])$ of $F[X]$ is obtained by :

(1) setting R equal to $\{\{g2\mathcal{A}(F[X])\}\}$

(2) applying the rule $TS \circ EGE$ resulting from the composition of the two previous rules TS and EGE until stabilization of R (i.e. until further applications of the two previous instructions do not change anymore the contents in R)

(3) removing from R forests containing the empty arborescence $A_\emptyset$

6 Decision problem

Given a graph formula f, the problem to determine if f is satisfiable is not decidable for the general case since the $\mathcal{SGBF}$-model and $\mathcal{FOL}$ have the same expressivity. If it were, this would lead to the contradiction that the satisfiability problem is decidable in $\mathcal{FOL}$ (which is not). As the satisfiability problem has been proven decidable for some specific fragments of $\mathcal{FOL}$, one way to find decidable fragments for the language of graph formulas is to transfer directly in graph formula the syntactic restrictions a logical formula has to respect in order to belong to one of the decidable fragments of $\mathcal{FOL}$. This lead us to define two different decidable fragments of $\mathcal{SGBF}$-model relying on the $\mathcal{FOL}$ guarded fragment and the $\mathcal{FOL}$ BSR fragment. In the following, these fragments are defined in terms of *indivisible clauses* and not in terms of graph formula because indivisible clauses are easier to manipulate as they are much more compact than graph formulas (while staying semantically equivalent). Thus, in order to check if a given graph formula f is satisfiable, the reasoning system described in this paper translates it into an indivisible clause C and selects the decision procedure defined for the fragment to which belongs C. This shows the benefits of the *indivisible normal form* : which is used as a "hub" between several decidable fragments of the graph formula language.

6.1 The BSR fragment

The *Bernays-Schnfinkel-Ramsey fragment* (i.e. BSR-fragment) is the fragment of $\mathcal{FOL}$ containing all formulas whose prenex normal form has a $\exists^*\forall^*$ quantifier prefix. It has been firstly proven decidable in [6] for formulas without equality, then extended in [16] to those with equality. The *BSR-fragment* is also known as *Effectively Propositional Logic* (i.e. EPR) which is a widely used formalism in the automated reasoning community since a significant number of efficient theorem provers have been recently developped for this class of formulas (see [9], [10] and [14]).

One important aspect of the BSR-fragment is that the *herbrand universe* of a $\exists^*\forall^*$ formula f is finite ; making it translatable[2] into an equisatisfiable *propositional* formula, and hence the name *effectively propositional* also used to refer to that fragment. Indeed, since formulas in $\mathcal{FOL}$ do not have any function, the skolemized form of a formula $f = \exists x_1 \exists x_2 ... \exists x_m \forall y_1 \forall y_2 ... \forall y_n f'$ (where f' is quantifier free) is the formula

[2]This process of turning an arbitrary $\exists^*\forall^*$ formula into an equisatisfiable propositional formula is also known as *instantiation* or *grounding* in the EPR community.

$f_1 = \forall y_1 \forall y_2 ... \forall y_n f'_1$ where f'_1 is a conjunction of disjunctions of literals from f in which each variable x_i has been replaced by a new constant. Since each term in a predicate of f_1 is either a constant or a variable y_i, the herbrand universe of f being the collection of all the constants involved in f_1 is therefore finite. The f_1 formula can be turned into a equisatisfiable quantifier free formula f_2 by replacing each predicate p involved in f_1 by the disjunction of the predicates resulting from all possible replacements in p of the y_i variables by constants from the herbrand universe of f. Once the replacements are done and the $\forall$ quantifiers eliminated, all predicates in f_2 are referred as *ground* since their terms are nothing but constants and can be considered as propositional symbols. Thus the satisfiability of the formula f can be checked by using the standard propositional logic methods to verify that of f_2. However, since the size of the propositional formula f_2 may be exponentially larger than the initial formula f (because a lot of distinct replacements are possible), this method does not lead to interesting performences. Therefore, a significative number of approaches have been proposed to decide the satisfiability of a BSR formula much more efficiently than previously ; while taking advantage of the herbrand universe finiteness (see [18], [5], [11] and [13]).

6.1.1 The $\mathcal{SGBF}$-model BSR fragment

The following define the restrictions that graph arborescences, forests and clauses have to respect in order to be translatable into a logical formula of the BSR-fragment.

BSR graph arborescence
A graph arborescence $A[X]$ is ***BSR*** iff :

- X is empty
- the depth of $A[X]$ does not exceed 3 levels
- and any third level SG in $A[X]$ is restricted to one single relation r surrounded by concept nodes linked to r whose each variable appears in another SG of $A[X]$

BSR forest
A forest $F[X]$ is ***BSR*** iff (1) X is empty and (2) each of its arborescences is BSR.

BSR clause
A clause $C[X]$ is ***BSR*** iff (1) X is empty and (2) each of its forests is guarded.

Example
The clause $C_3[x]$ of the figure 17 is BSR since each arborescence in each of its forests has a depth less or equal to 3 and when a third level SG exists it is restricted to one relation $\boxed{x}\overset{1}{-}(cW)\overset{2}{-}\boxed{u}$ or $\boxed{x}\overset{1}{-}(cW)\overset{2}{-}\boxed{w}$ such that each of the variables x, u or x, w (or a renaming) appears in the parent graph G_4 (or a renaming). Note that the arborescence $A_3[x]$ does not comply to the BSR definition although it is equivalent to the clause $C_3[x]$. This shows the advantage of the *indivisible normal form* which helps to reveal the BSR nature of a graph formula or a clause.

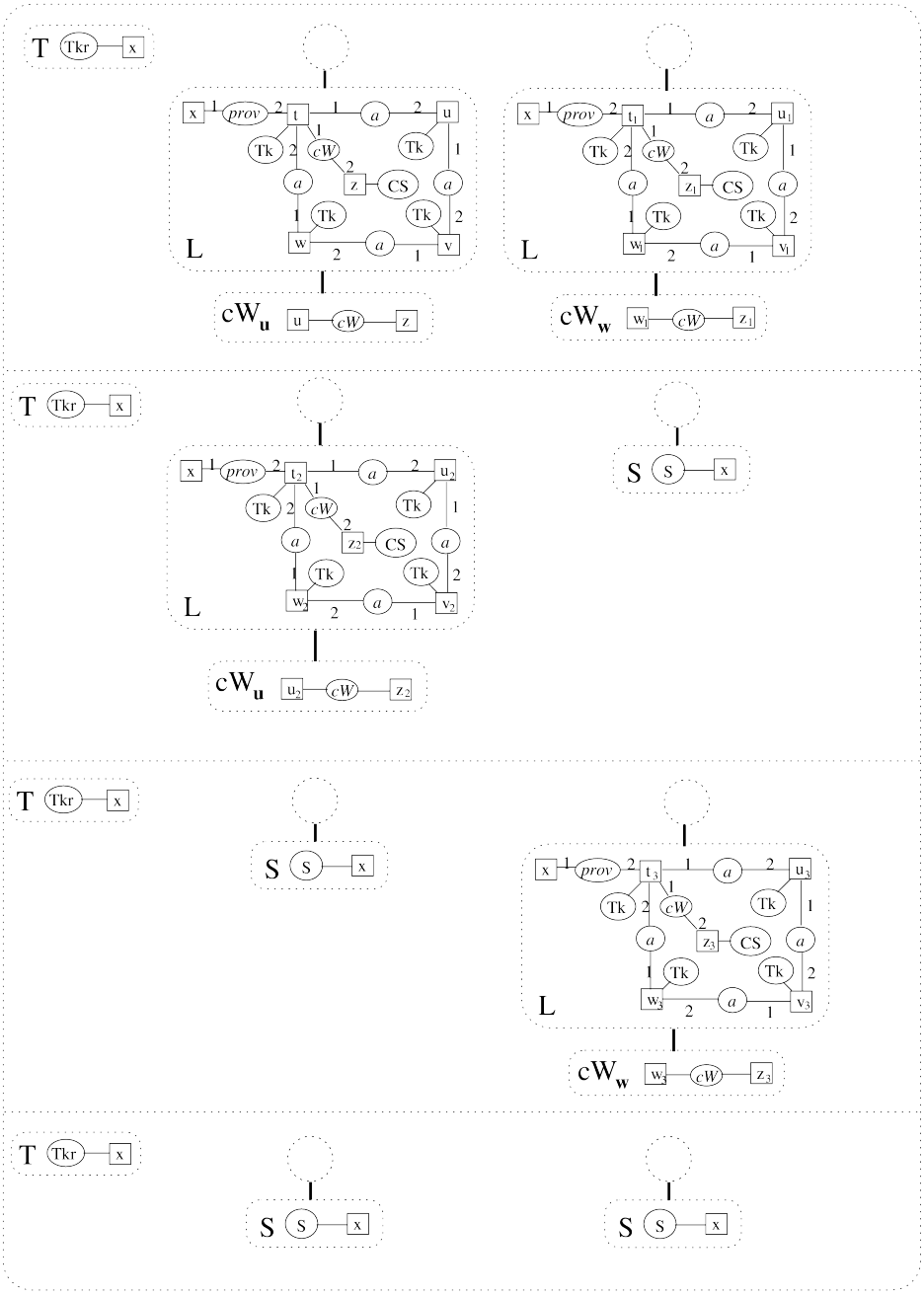


Figure 17: The indivisible clause $C_3[x]$ corresponding to the arborescence $A_3[x]$ and the graph formula $F_3[x]$ is BSR and contains 4 forests.

6.1.2 Translation to logic

Given a BSR clause $C[X]$, the translation of $C[X]$ into the $\exists^*\forall^*$ quantifier prefixed prenex normal form formula $\beta(C[X])$ is obtained by :

- translating $C[X]$ into the graph formula $g = \mathcal{C}2g(C[X])$
- translating g into the logical formula $f = g2f(g)$
- computing the prenex normal form f' of f

The prenex normal form of the logical formula corresponding to the clause is straightforward. Lets denote by G° the quantifier free logical formula resulting from the translation $g2f(G[vars(G)])$ of a SG G, and by $\mathcal{C}(G, A)$ the set of children $children(G, A)$ of SG G in an arborescence A. Given a BSR graph arborescence $A[X]$, the graph formula $g = \mathcal{A}2g(A[X])$ corresponding to $A[X]$ is equivalent to

$$\exists\widetilde{x}\left(G^\circ \bigwedge_{H\in\mathcal{C}(G,A)} \neg\left(\exists\widetilde{y}H^\circ \bigwedge_{r\in\mathcal{C}(H,A)} \neg(r^\circ)\right)\right)$$

where $\widetilde{x} = vars(G)$ and $\widetilde{y} = vars(sub(H, A)) \setminus \widetilde{x}$. By applying De Morgan's laws in order to propagate negations the deeper as possible in the formula, we obtain

$$\exists\widetilde{x}\left(G^\circ \bigwedge_{H\in\mathcal{C}(G,A)} \left(\forall\widetilde{y}\bigvee_{\lambda\in\mathcal{L}}\lambda\right)\right)$$

where $\mathcal{L}$ is the set of literals containing the predicates r° corresponding to each SG $r \in \mathcal{C}(H, A)$ or the negation of those appearing in H°. The previous formula is equivalent to

$$\exists\widetilde{x}\forall\widetilde{Y}\left(G^\circ \bigwedge_{H\in\mathcal{C}(G,A)\lambda\in\mathcal{L}}\bigvee\lambda\right)$$

where $\widetilde{Y} = \bigcup_{A'\in subs(A)} vars(A') \setminus \widetilde{x}$; which is in prenex normal form and whose quantifier prefix is $\exists^*\forall^*$. Since the logical translation of a forest (resp. a clause) is respectively a conjunction (resp. a disjunction) of formulas of the previous form, a prenex normal form with the same quantifier prefix is obtained by distributing disjunction over conjunction in order to migrate all quantifiers at the begining of the formula. Thus the prenex normal form of the logical formula corresponding to a BSR clause belongs to the BSR-fragment.

The following formula is obtained by translating with β the forest at the top of the clause shown in the figure 17. Since the quantifier prefix is $\exists^*\forall^*$, this formula belongs to the BSR-fragment.

$$\begin{aligned}
&(\exists x\forall u\forall v\forall z\forall t\forall w\forall u_1\forall v_1\forall z_1\forall t_1\forall w_1\\
&\quad Tanker(x)\\
&\quad\wedge(\\
&\qquad \neg Tank(t)\vee\neg Tank(u)\vee\neg Tank(v)\vee\neg Tank(w)\vee\neg CoolerSystem(z)\vee\\
&\qquad \neg providedWith(x,t)\vee\neg adjacent(t,u)\vee\neg adjacent(u,v)\vee\neg adjacent(v,w)\vee\\
&\qquad \neg adjacent(w,t)\vee\neg cooledWith(t,z))\vee cooledWith(u,z)
\end{aligned}$$

$$
\begin{array}{l}
) \wedge (\\
\quad \neg Tank(t_1) \vee \neg Tank(u_1) \vee \neg Tank(v_1) \vee \neg Tank(w_1) \vee \neg CoolerSystem(z_1) \vee \\
\quad \neg providedWith(x, t_1) \vee \neg adjacent(t_1, u_1) \vee \neg adjacent(u_1, v_1) \vee \\
\quad \neg adjacent(v_1, w_1) \vee \neg adjacent(w_1, t_1) \vee \neg cooledWith(t_1, z_1)) \vee \\
\quad cooledWith(u_1, z_1) \vee cooledWith(w_1, z_1) \\
) \\
)
\end{array}
$$

6.2 The guarded fragment

The $\mathcal{FOL}$ *guarded* and *loosely guarded* fragments were introduced in [1] as a generalization of modal logic. It appears that $\mathcal{FOL}$ guarded fragments retain a lot of good properties of modal logics: its satisfiability problem is decidable, it has the finite model property and a sort of tree model property (see [12]). A significant number of knowledge representation formalisms (as description logics or frame logics) are related to the guarded fragment in the sense that their logical interpretations belong or are very close to this fragment. For instance, the ALC description logic has a translation in modal logic (see [2]) which has a standard translation in logic whose range is the guarded fragment. Let also cite the suggestion of [15] to define a n-ary description logic whose expressivity might be greater than that of the guarded fragment. In the following, we focus on the guarded fragment and do not consider the loosely guarded fragment (which is a strictly more expressive generalization of the guarded fragment).

6.2.1 The guarded fragment of $\mathcal{FOL}$

The language $\mathcal{GF}$ of guarded formulas of $\mathcal{FOL}$ are defined by induction as follows.

- an atomic formula is a guarded formula
- if ϕ, ψ are guarded formulas then $\phi \wedge \psi$ and $\neg\psi$ are guarded formulas
- let G be an atom, $\widetilde{x} \subseteq free(G)$ a non-empty set of variables and ψ a guarded formula such that $free(\psi) \subseteq free(G)$, then $\exists\widetilde{x} G \wedge \psi$ is a guarded formula.

6.2.2 The guarded fragment and propositions of conceptual graphs

When dealing with possibly negated propositions of conceptual graphs, [3] suggests to restrict SGs such that their logical translation lead to a formula from the $\mathcal{FOL}$ *guarded* and *loosely guarded* fragments. However, the definition of *guarded* graph propositions sets strong restrictions on the syntax of the SGs used in propositions. Roughly speaking, in order for a proposition to be guarded, all concept nodes of a SG must be neighbours of one unic relation (when the graph contains variables not appearing elsewhere in the proposition). It turns out that SGs in propositions are very compact and small in practice. In the following, we present a characterisation of a guarded graph formula in which the previous restriction is relaxed such that SGs may have a much more complex structure which is similar to that of a tree.

6.2.3 The $\mathcal{SGBF}$-model guarded fragment

The next paragraph defines the notion of guard and subordinate in the $\mathcal{SGBF}$-model. It is followed by the definitions of guarded SGs, arborescences, forests and clauses.

Guards and subordinates
Given a SG G, let $guard(G[X])$ and $subords(G[X])$ return respectively (1) the sub-SG G' of G induced by X and (2) the connected components $H_1[X_1]$, $H_2[X_2]$, ..., $H_n[X_n]$ of $G \ominus G'$ where $X_i = vars(H_i) \cap vars(G')$. The subgraph $G' = guard(G)$ is called the *guard* of the subgraphs in $subords(G)$ which are referred as its *subordinates*.

Guarded simple graphs
For a given relational support $S = (T_R, \mathcal{I})$ and an *indivisible* SG $G[X]$ defined over S, $G[X]$ is said to be *guarded* iff (1) there exists *exactly one* relation node r in $G' = guard(G[X])$ such that each variable symbol not in X appears in a concept linked to r and (2) for any $H[Y] \in subords(G[X])$:

- $H[Y]$ is guarded
- there exists *exactly one* relation node r in G' such that each variable symbol in Y appears in a concept linked to r.

The set of all guarded SGs defined over a relational support S is denoted by $\mathcal{SG}_G(S)$.

The basic idea standing behind the previous definition is that a guarded SG may be decomposed in several subgraphs that can be organised in a tree-like structure according to their shared variables. The subgraph $guard(G)$ may be considered as the *parent* of each subgraph H in $subords(G)$ provided that all the variables shared by $guard(G)$ and H appear in the concepts linked to a single relation in $guard(G)$.

In the figure 5, since the SGs are not indivisible it can not be decided whether ther are guarded or not. However, the figure 18 shows the hierarchy reproducing the dependencies between the guards and subordinates of one indivisible connected component of the SG G_2. In this hierarchy, each SG Q has a single relation containing all the variables not appearing in its parent and for each of its subordinates R, Q has a single relation containing all the variables it shares with R. For instance, the SG with the relation $doubleShovedBy(x, y, z, t)$ contains the variables involved with one of its subordinates : the SG with the relation $cooledWith(y, z)$.

On the contrary, the figure 19 shows the hierarchy of guards and subordinates corresponding to one indivisible connected component G'_4 of the SG G_4. It appears that, although the SG K_4 is a subordinate of the SG containing the relation $providedWith(x, y)$, the variables u and w of the K_4 guard sub-SG do not appear in a single relation. Thus, K_4 is not guarded and neither is G'_4.

Guarded arborescence
Let $A[X]$ be a graph arborescence whose root is R, $A[X]$ is said to be *guarded* iff (1) $R[X]$ is guarded and (2) for each top-subarborescence $A'[X_{A'}]$ where $A' \in subs(A)$ and $X_{A'} = \mathcal{B}(A', A[X])$:

- $A'[X_{A'}]$ is guarded
- there exists *exactly one* relation node r in R such that each variable symbol in $X_{A'}$ is a marker of a concept linked to r.

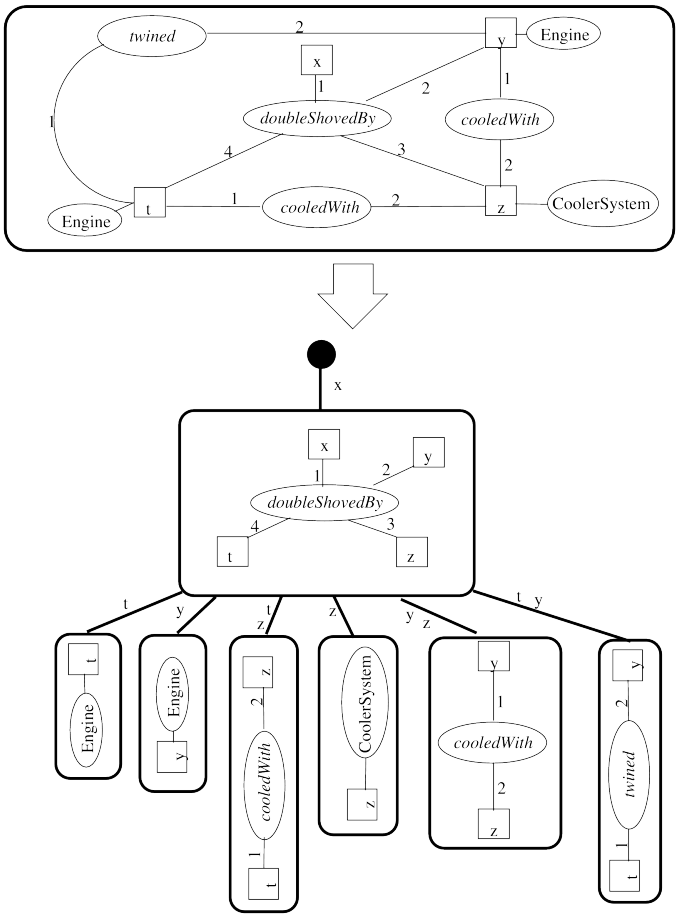


Figure 18: The hierarchy of dependencies between *guards* and *subordinates* of a guarded SG.

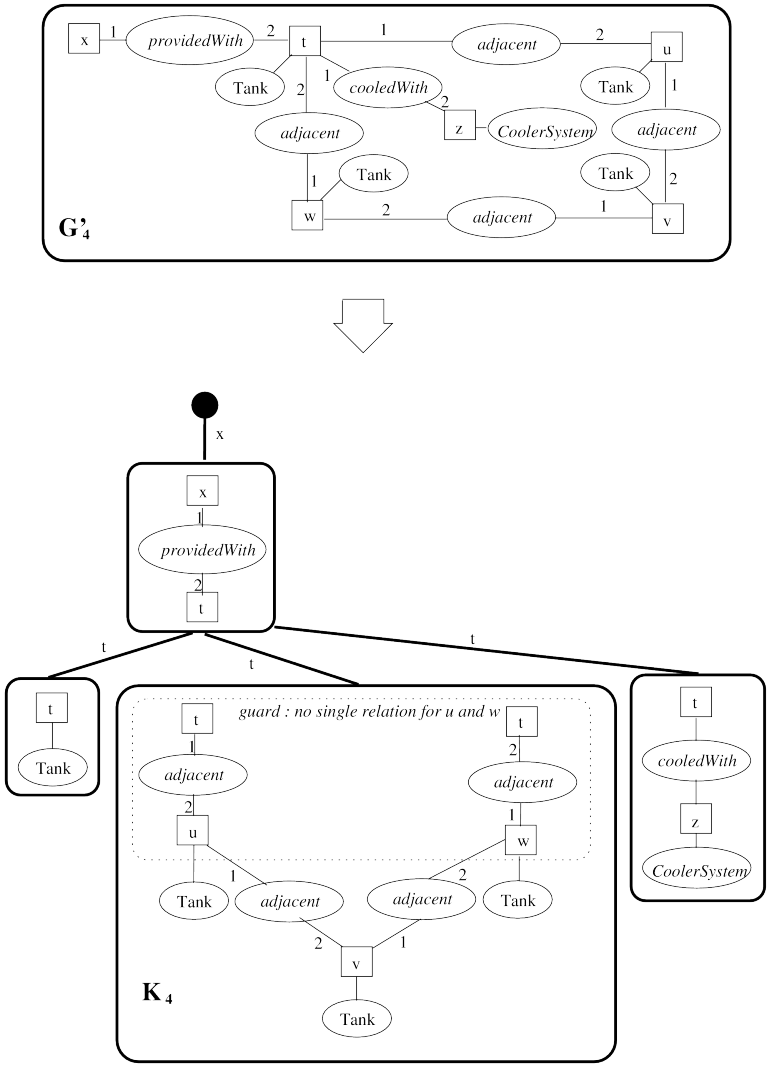


Figure 19: The hierarchy of dependencies between *guards* and *subordinates* of a SG not guarded.

Guarded forest
A forest $F[X]$ is guarded iff (1) each of its arborescences $A[X \cap vars(A)]$ is guarded and (2) for each pair $A, A' \in F$ if $vars(A) \cap vars(A') \neq \emptyset$ then $vars(A) \cap vars(A') \subseteq X$.

Guarded clause
A clause $C[X]$ is guarded iff each of its forests $F[X \cap vars(F)]$ is guarded.

Example
For example, the figure 20 shows a clause with a large number of forests each one containing guarded arborescences. Since the graphs in the arborescences have been split, the number of combinations raises and this lead to a high number of forests in the clause.

6.2.4 Translation to the guarded fragment

Support
Given a support S, the standard translation $\phi(S)$ already returns a set of logical guarded formulas. Indeed, each formula in $\phi(S)$ is of form $\forall \widetilde{x} p(\widetilde{x}) \rightarrow q(\widetilde{x})$ and may be written as the equivalent formula $\neg \exists \widetilde{x} p(\widetilde{x}) \wedge \neg q(\widetilde{x})$ which belongs to the guarded fragment.

Guarded graph arborescence
Given a guarded graph arborescence $A[X]$, let $\mu(A[X])$ be the logical formula corresponding to $A[X]$ obtained by the following steps.

Let $R = root(A)$, $G = guard(R[X])$, $f = g2f(G[vars(G)])$ and $\mathcal{F}^-$, $\mathcal{F}^+$ be respectively the two following sets of logical formulas :

- $\mathcal{F}^-$ contains the formulas $\neg\mu(B[Y])$ where $B[Y]$ is a top-subarborescence of A sharing with G a non-empty set $Y = vars(B) \cap vars(G)$ of variables,
- $\mathcal{F}^+$ contains the formulas $\mu(C[Z])$ where $C[Z]$ is such that $R_C[Z] \in subords(R[X])$, $root(C) = R_C$ and $subs(C)$ contains the top-subarborescences of A sharing at least one variable with R_C,

then $\mu(A[X]) = \exists \widetilde{x} f \wedge \bigwedge_{f' \in \mathcal{F}^- \cup \mathcal{F}^+} f'$ where $\widetilde{x} = free(f) \setminus X$

The basic idea behind the previous translation procedure is that at each step a SG-guard G is extracted from the root R of the arborescence $A[X]$ currently in process. G is translated into an unquantified conjunction F of atoms while two sets $\mathcal{F}^-$ and $\mathcal{F}^+$ of formulas are built such that (1) $\mathcal{F}^-$ contains the *negation* of each formula obtained by recursively translating a top-subarborescence $B[Y]$ of $A[X]$ provided that $Y = vars(B) \cap vars(G)$ is not empty and (2) $\mathcal{F}^+$ contains each formula obtained by recursively translating any arborescence $C[Z]$ whose root is a subordinate $R_C[Z]$ of G in $R[X]$ and the top-subarborescences are those from $A[X]$ provided that they share at least one variable with R_C. The final logical formula $\mu(A[X])$ is obtained by existentially quantifying the variables $\widetilde{x} = free(f) \setminus X$ in the conjunction of f with the formulas in $\mathcal{F}^-$ and $\mathcal{F}^+$.

Then, $\mu(A[X])$ is refactored into a formula $\gamma(A[X])$ meeting the syntactic requirements of the guarded fragment by replacing until stabilization each expression of the

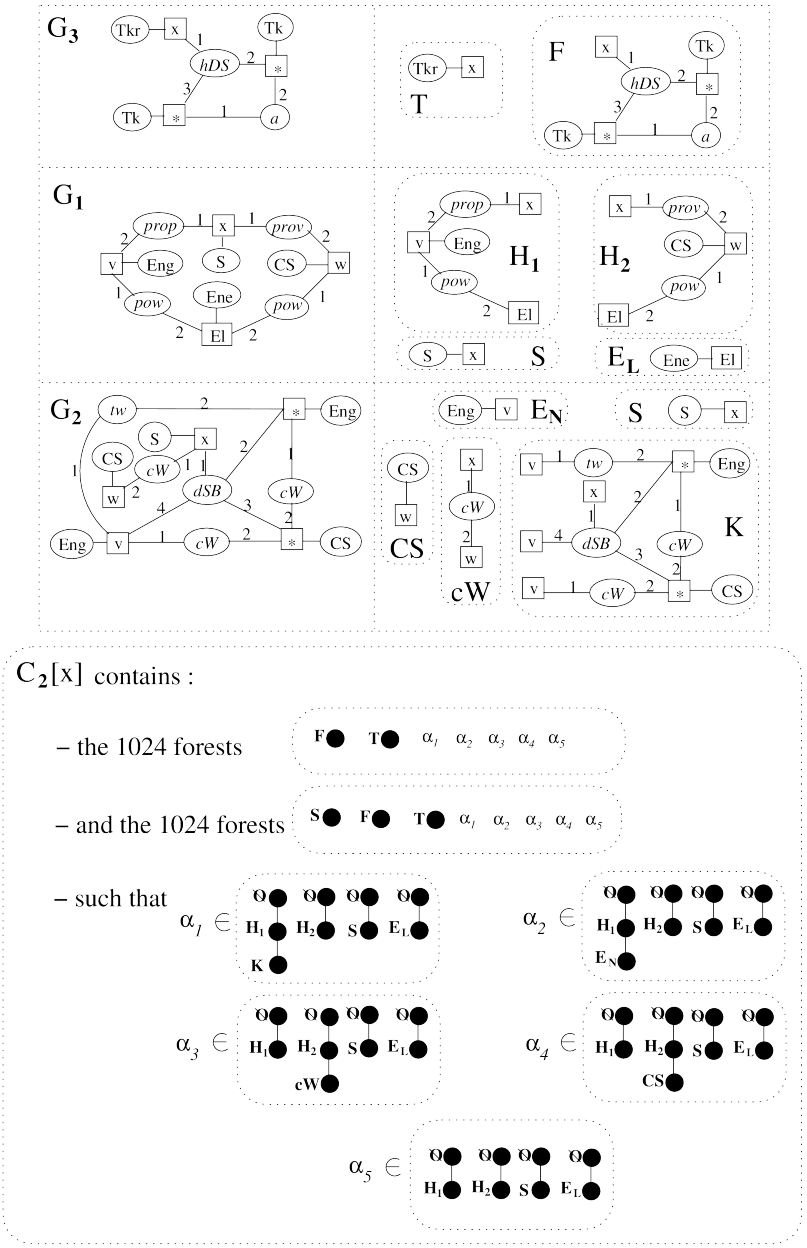


Figure 20: The indivisible clause $C_2[x]$ corresponding to the graph formula $F_2[x]$ is guarded and contains 2048 forests.

form $\exists\widetilde{x} f \wedge \mathcal{B}$ where $\mathcal{B} = \bigwedge_{f' \in \mathcal{F}^- \cup \mathcal{F}^+} f'$ by the conjunction $\bigwedge_{G \in f} F(G)$ whose each formula $F(G)$ is a guarded formula of the form $\exists\widetilde{x}_G\, G \wedge \mathcal{B}_G$ where (1) G is an atom from f, (2) $\widetilde{x}_G = \widetilde{x} \cap free(G)$ and (3) $\mathcal{B}_G$ is the conjunction of all the formulas $B \in \mathcal{B}$ such that $free(B) \subseteq free(G)$. Note that G is unique for B, since by construction of $\mu(A[X])$ there exists only one atom G in f for which $free(B) \subseteq free(G)$.

Guarded forest

Given a guarded forest $F[X]$, the guarded logical formula $\gamma(F[X])$ corresponding to $F[X]$ is the conjunction of the guarded logical fomulas $\gamma(A[X_A])$ corresponding to each of its graph arborescences $A[X_A]$ where $X_A = X \cap vars(A)$.

Guarded clause

Given a guarded clause $C[X]$, the guarded logical formula $\gamma(C[X])$ corresponding to $C[X]$ is the disjunction of the guarded logical fomulas $\gamma(F[X_F])$ corresponding to each of its forests $F[X_F]$ where $X_F = X \cap vars(F)$.

Example

The forest of the figure 21 is guarded since it is also the case for each of its arborescences. Indeed, each SG in the arborescences is guarded. For instance in the SG F, every variables belongs to the same relation hDS. Furthermore, at each level the subarborescences are guarded. For instance, since the variable v and x in the SG K appear in the selection $[x]$ or elsewhere in the arborescence, the selection of K is $[x, v]$ in this arborescence ; and since x and v both appear in a single relation of the parent SG H_1 then the subarborescence starting at H_1 is guarded. Note that the arborescence $A_2[x]$ shown in the figure 10 is equivalent to the clause $C_2[x]$ but it is not guarded since the SG G_2 has a selection $[x, v, w]$ in $A_2[x]$ but the variables x, v, w do not appear as arguments of one single relation in the parent SG G_1. By putting the arborescence $A_2[x]$ in indivisible normal form the SGs G_1 and G_2 are split and their selection are distributed such that the result becomes guarded. This shows the advantage of computing the indivisible normal form of a graph formula or a clause before checking if it is guarded or not.

The following logical formula $\mu(A_K^{H_1}[x])$ is obtained by translating with μ the three-level leftmost arborescence $A_K^{H_1}[x]$ of the forest in the figure 21.

$$
\begin{aligned}
&\mu(A_K^{H_1}[x]) = \top \wedge \neg(\\
&\quad \exists v(\\
&\qquad propelledBy(x, v) \wedge Engine(v) \wedge poweredBy(v, Electricity) \\
&\qquad \wedge(\exists t \exists u \\
&\qquad\quad twined(v, t) \wedge doubleShovedBy(x, t, u, v) \wedge cooledWith(v, u) \\
&\qquad\quad \wedge(cooledWith(t, u) \wedge Engine(t) \wedge CoolerSystem(u)) \\
&\qquad) \\
&\quad) \\
&)
\end{aligned}
$$

The previous formula may be turned into the following guarded formula $\gamma(A_K^{H_1}[x])$ by applying on $\mu(A_K^{H_1}[x])$ the refactoring process previously defined in order to exhibit guards and place them beside existential quantifiers.

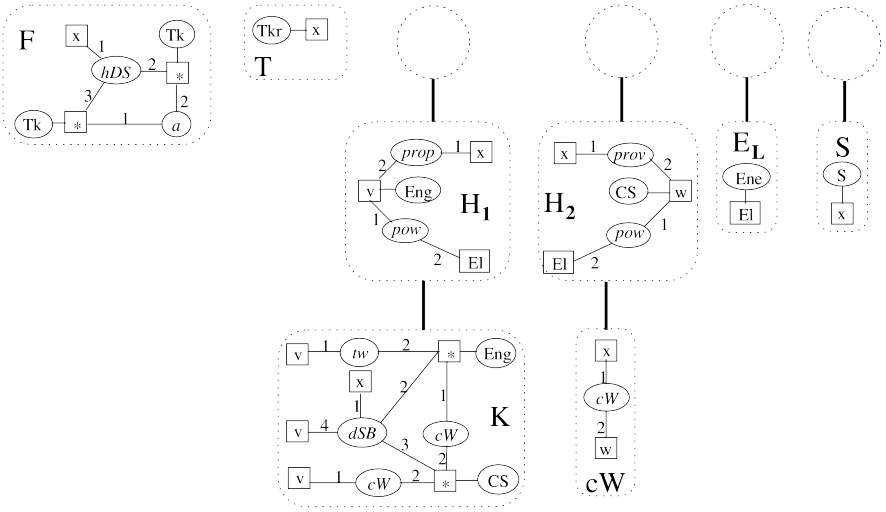


Figure 21: One of the 2048 forests of the clause $C_2[x]$ of the figure 20.

$$\gamma(A_K^{H_1}[x]) = \top \wedge \neg($$
$$\exists v($$
$$propelledBy(x, v) \text{ (here is one guard)}$$
$$\wedge(\exists t \exists u$$
$$doubleShovedBy(x, t, u, v) \text{ (here is another guard)}$$
$$\wedge(twined(v, t) \wedge cooledWith(v, u))$$
$$\wedge(cooledWith(t, u) \wedge Engine(t) \wedge CoolerSystem(u))$$
$$)$$
$$\wedge(Engine(v) \wedge poweredBy(v, Electricity))$$
$$)$$
$$)$$

6.3 A comparison of the two fragments

The main differences between the two fragments may be expressed in terms of *cycles*. Since, in a BSR arborescence, SGs located at the root or under the root are not restricted at all, they may contain any kind of cycles. However, the definition of cycles in the SGs of a guarded arborescence is subject to strong restrictions. Indeed, variables and relation nodes may form cycles of the form $v_1 r_1 v_2 r_2 ... v_{n-1} r_{n-1} v_n = v_1$ where each variables v_i and v_{i+1} appear in the relation node r_i *only if* all the v_i appear in a single relation node.

For instance, in the figure 20, the guarded SG K contains a cycle $v\ cW * cW * tw\ v$ with four variables and three relation nodes. For this SG to be guarded, all variable of the cycle has to appear in the concepts linked to the same relation node. And that is the case, since each variable of the cycle appear in a concept linked to the dSB relation node. Such a relation node is called the *guard* of the cycle and the cycle is said to be *guarded* (otherwise it is *unguarded*). On the contrary, in the figure 17 the SG L

contains an unguarded cycle $t\ a\ u\ a\ v\ a\ w\ a\ t$ (i.e. there does not exist any relation node in L where $t\ u\ v\ w$ appear). This SG is not guarded (as shown in 19) but since it belongs to a BSR graph arborescence it can contains such *unguarded* cycles.

6.4 Deciding standard logical properties

Each standard logical property presented in 3.3.1 (unsatisfiability, entailment, equivalence) between two graph formulas f and g may be associated to a graph formula $u(f)$ or $u(f, g)$ whose unsatisfiability proves this property. More formally, given two graph formulas $F[X]$ and $G[Y]$:

- (unsatisfiability) $F[X]$ is *logically unsatisfiable* iff $u(F[X]) = F[]$ is unsatisfiable
- (entailment) $F[X]$ *logically entails* $G[Y]$ iff $u(F[X], G[Y]) = (F \wedge \neg G)[]$ is unsatisfiable
- (equivalence) $F[X]$ and $G[Y]$ are stated *logically equivalent* iff $u(F[X], G[Y]) = ((F \wedge \neg G) \vee (G \wedge \neg F))[]$ is unsatisfiable

To do so, for each of the three previous logical properties, the corresponding graph formula $u(F[X])$ or $u(F[X], G[Y])$ is translated into a indivisible clause $C[]$. If $C[]$ is neither a *BSR* nor a *guarded* clause, it can not be decided if the corresponding property holds or not. Otherwise, when $C[]$ belongs to one of these fragments, it is translated into a logical formula $\beta(C[])$ or $\gamma(C[])$ and a reasoning engine for the suitable logical fragment is used to check its unsatisfiability.

7 The classification service

As shown in figure 22, the ITM classification service is used to define categories of instances and organize them into hierarchies. *Instances* are factual knowledge stored by end-users in the lowest levels of the ITM knowledge base. For example, an instance may be the representation of a ship with its properties (its length, its tonnage, its date of construction, etc). A category is defined by combining criteria that each of which can be regarded as a constraint on an property (for example, a constraint binding the length of the ship to a certain value). When all properties of an instance meet the criteria combination of a category, the instance is considered to belong to the category. This section presents the key features provided by the classification service in order to create and maintain such hierarchies of categories. The next subsection define formally the notion of category hierarchy and the possible operations on this hierarchy. The second subsection deals with reasoning problems related to category hierarchies. Finally, the last subsection shows how to set the classification service to meet legal content management requirements.

7.1 The category hierarchy

A hierarchy of categories is provided with an inheritance mechanism that allows each category to benefit from the criteria defined in the categories located above. A category can, however, be excluded from the inheritance mechanism in order not to inherit the criteria above and not to transfer its contents downward. The category hierarchy is built

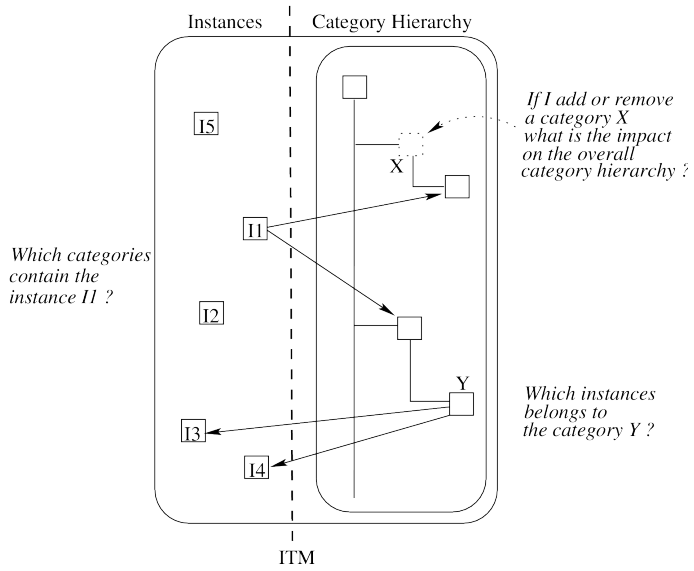


Figure 22: The classification service of ITM.

by means of two elementary operations of insertion and deletion. The following define more formally the hierarchies structure and the possible operations.

7.1.1 Category

A category is an object stored in the workspace dedicated to category hierarchies in the ITM knowledge base. Each category C is associated to a *query* $Q(C)$ which is a *graph formula* $F[x]$ whose definition is F and selection is x. The x variable is shared by all category queries and represents the instance to classify.

7.1.2 Hierarchy

A category hierarchy $\mathcal{H}$ is an *arborescence* whose each node is a category. If needed, a category C may be flagged with an *excluded status*. The semantics of the category hierarchy is as follows: for each *non-excluded* category C_c located below another *non-excluded* category C_p, if an instance *belongs to* C_c, this instance must also *belong to* C_p. The category hierarchy provides an inheritance mechanism which respects this semantics and generates for each non-excluded category C, a query $Q_H(C)$ called *inheritance query* combining all criteria defined in $Q(C)$ and the queries of the non-excluded categories located above in the hierarchy. More precisely, the inheritance query of a non-excluded category C is defined as the conjunction of its query $Q(C)$ and the inheritance query $Q_H(C_p)$ of its least non-excluded ancestor C_p. If needed, a category C_E may be flagged as *excluded* in order to consider it as being out of the hierarchy. In such a case, C_E is ignored by the inheritance mechanism : it does not inherit from its parent and does not transfer its query content to its children (i.e. $Q_H(C_E) = Q(C_E)$).

For example, when the inheritance mechanism is applied to a category hierarchy consisting of a root category C_p with query $Q(C_p)$ being the parent of a category C_c

with query $Q(C_c)$, it generates for C_c the *inheritance query* $Q_H(C_c) = Q(C_p) \wedge Q(C_c)$.

7.1.3 Operations

Category hierarchies are created or updated using two elementary operations which preserves their tree structure.

- The category insertion consists in adding a category to the hierarchy, setting it as the root or linking it to a parent and a subset of its children (if any). Given a category X and a category hierarchy $\mathcal{H}$ the insertion of X in $\mathcal{H}$ consists in adding X in $\mathcal{H}$ and - if $\mathcal{H}$ is not empty - (1) linking X as the parent of the current root or (2) releasing the filiation links between a category P and a subset $\mathcal{C}$ of its children (if any) and linking X as a new child of P and the new parent of the $\mathcal{C}$ categories (if any).
- The category deletion consists in removing a category from the hierarchy while preserving its tree structure. Given a category X and a category hierarchy $\mathcal{H}$ the deletion of X in $\mathcal{H}$ consists in (1) linking the children of X (if any) to its parent (if X is not root) and (2) removing X from $\mathcal{H}$ provided that if X is the root then it has only one child.

Those primitive operations can be combined to define higher level operations. For example, the *move* operation of a category is defined by deleting the category before inserting it elsewhere in the hierarchy.

7.2 Reasoning problems

The classification service is able to solve two main kinds of reasoning problems. The first is about instances/categories memberships and the second about categories containments.

7.2.1 Membership problems

The membership of an instance into a category is defined as follows : a category C *contains* an instance I (or I *belongs to* C) iff I answers the query $Q_H(C)$. There is two kinds of problems about memberships :

(1) The *categorization problem* which consists in finding the categories containing a given instance I. This problem is solved by standard querying methods. The basic idea consists in returning each category C whose inheritance query $Q_H(C)$ accepts I as one of its results.

(2) The *instanciation problem* which consists in finding all instances in the knowledge base belonging to a given category C. Here again, the problem is solved by usual query techniques : the content of the category C is computed by a run of the $Q_H(C)$ query on the knowledge base.

7.2.2 Containment problems

The classification service is able to highlight significant mathematical properties of the category hierarchy which can be used for example to determine useless and redundant

categories in such a way that these properties always hold (i.e. whatever the instances in the base). In contrast to the previous paragraph, these problems can not be solved anymore by standard querying. In this case, the system uses the decision method presented in 6.4 to determine which mathematical properties hold for the queries associated to categories. More formally, for two given categories A and B :

(1) A and B are said to be *tantamount* iff $Q_H(A)$ and $Q_H(B)$ are *logically equivalent*. This means that for any knowledge in the base, $Q_H(A)$ and $Q_H(B)$ have same results.

(2) A is *empty* iff $Q_H(A)$ is *logically unsatisfiable*. This indicates that for any knowledge in the base, $Q_H(A)$ has no result.

(3) A *subsumes* B iff $Q_H(B)$ *logically entails* $Q_H(A)$. This involves that for any knowledge in the base, a result of $Q_H(B)$ is a result of $Q_H(A)$.

7.2.3 Semantic audit

At any time, users can request the *semantic audit* of a hierarchy $\mathcal{H}$ in order, for instance, to check its overall consistency. Such a request leads the system to return a *report* which is a tuple $((\mathcal{T},\mathscr{T}),(\mathcal{E},\mathscr{E}),(\mathcal{S},\mathscr{S}))$ of three couples of collections :

(1) the collection $\mathcal{T}(\mathcal{H})$ of all *tantamount* classes : categories belonging to such a class means that each one is tantamount to the other

(2) the collection $\mathcal{E}(\mathcal{H})$ of all empty categories

(3) the collection $\mathcal{S}(\mathcal{H})$ of *subsumptions* : that is to say all couples of categories such that the first member subsumes the second.

Each previous collection $\mathcal{T}(\mathcal{H})$, $\mathcal{E}(\mathcal{H})$ or $\mathcal{S}(\mathcal{H})$ is respectively associated to an *undecidable* set $\mathscr{T}(\mathcal{H})$, $\mathscr{E}(\mathcal{H})$ or $\mathscr{S}(\mathcal{H})$. If it can not be decided if *yes* or *no* a category C_U belongs to one of the previous collections, then C_U is stored in the corresponding *undecidable* set. This means that the system could not analyse the properties shared by C_U and the other categories of the hierarchy. Thus the user has to check itself which of these properties depends on the existence of C_U in the hierarchy.

7.2.4 Semantic feedback after a change

The semantics of categories is directly or undirectly affected by the changes made on the overall structure of the category hierarchy by *insertions* and *deletions*. Indeed, after such a change the new inheritance query corresponding to a category may differ from the old because one of its ancestors has been inserted, removed or moved elsewhere in the hierarchy.

When a new category C' with query $Q(C')$ is *inserted* under an existing category C in the hierarchy, C becomes the parent of C' and some of C children may be transferred as children of C'. For each category X located below C', X inherits of the criteria defined in the query $Q(C')$. Thus the inheritance query $Q_H(X)$ becomes the conjunction of its old content with $Q(C')$.

On the contrary, when a category C' is *deleted* below an existing category C in the hierarchy, C becomes the parent of all the C' children. For each category X located below C, X does not inherit anymore of the criteria defined in the query $Q(C')$. Thus the inheritance query $Q_H(X)$ does not contain anymore the criteria defined in $Q(C')$.

After a change operated on the hierarchy, the classification service returns a *semantic feedback* which can be regarded as the semantic audit of the modification impacts. This feedback returns a report $((\mathcal{T},\mathcal{T}'),(\mathcal{E},\mathcal{E}'),(\mathcal{S},\mathcal{S}'))$ restricted to the categories affected by the change. More formally, the restrictions on the report are as follows.

(1) the content of the collection $\mathcal{T}(\mathcal{H})$ of all tantamount classes is restricted such that each class contains at least one category located below the added or removed C' category.

(2) the collection $\mathcal{E}(\mathcal{H})$ of all empty categories contains only categories located below the C' category

(3) the collection $\mathcal{S}(\mathcal{H})$ of subsumptions contains only couples whose one member is a category located below the C' category

Such a *semantic feedback* is necessary to understand how and in what extent the semantics of the hierarchy changed after an insertion or a deletion.

7.3 Use-case in legal content management

The next part of this paper presents four of the most significant actors of the legal domain, their motivations and their interactions. Then it shows how to use the previous classification service in the legal domain.

7.3.1 The publisher

The main activity of a law publisher is to publish the last up to date version of various regulations into reference books. Throughout its life, a regulation is continuously modified by insertion of new rules having recently entry into force or deletion of existing rules which have been repealed. Each new state resulting of a change in the regulation is stored by the publisher for further access in complex versioning systems providing the ability to search the contents by time criteria or restore old versions of a regulation by going back through the history of changes. Since shortly, law publishers intend to diversify their offer by providing legal expertise services to three other key actors of the legal domain : the *lawmaker*, the *individual* and the *compliant-seller*.

7.3.2 The individual

In the following, the term *individual* refers to a physical or moral person required to respect a given law. That is to say people or companies who need to know for personal or professional reason if their private situations and/or activities (related to job, business, profession, trading, etc) are concerned by a change in the law. The problem here is not to check accurately if an individual situation is compliant with a rule but rather to determine if a rule concerns somehow its situation (the conformity having to be checked afterward). In such a case, we say that the rule *applies* or *is applicable* to the individual situation ; or conversely that the individual situation is *affected* by the rule. For example, in the naval field *individuals* might be the shippers whose fleet must be compliant with the naval law. Shippers need to know which new naval rules are *applicable* to a given ship (and in a next step, they will check themselves if this given ship is compliant with these new rules). The usual way for such a person/company to know if he/it is affected by an update of the law is to seek advice from a lawyer or a law specialist. Several law publishers aim to provide automated or semi-automated legal

services in order to bring a low-cost answer to these questions. This kind of services are not designed to replace the lawyer advices but may be seen as a low-cost preliminary step before going ahead for seeking from experts higher-cost services.

7.3.3 The compliant-seller

The *compliant-seller* actor of the legal domain is motivated by two issues related to the law. A compliant-seller sells products that must comply to a regulation. A change in the regulation is a strong marketing asset for the seller : he can inform his customers that the products they purchased are deprecated and need to be upgraded to the next compliant version. Thus, his primary motivation is to upgrade former customer products and identify new customers who have non-compliant products and might be interested by buying his own. His second motivation is to check that he himself complies with the law (that is to say the company and the products or services sold). This second need is equivalent to that of the *individual*. Thus the compliant-seller has a dual motivation : on the one hand the business development and on the other the seek for law compliance (so has the individual). If we consider for example, that a rule concerning the chemicals requested in the keel of oil tankers has been added to the naval regulation : as an *individual*, a keel manufacturer needs to know on which extent he is impacted by this change (in order to be compliant with the law, he must add the requested chemicals to the keel during its building process) ; but as a *seller* he wants to know which shippers need to change the keel of their ships to become compliant with the law and take this opportunity to sell them its new product.

7.3.4 The lawmaker

The *lawmaker* is the person which produces or updates the law. Among other duties, he has to ensure the overall coherence of a law whose rules have been changed. Such a work is usually done by comparing all updated/impacted rules one-each-other and may often results into an intensive search that can be time consuming. He needs a tool to manage, search among rules of a regulation and compare them together. If a rule R has to be added to or removed from a regulation, he has to evaluate the general impact on the full law content. More precisely, he has to find among the existing regulations :

(1) the rules *redundant* with R: these rules and R are always applicable to the same situations,

(2) the *useless* rules if R is added: these rules are not applicable anymore to any situations,

(3) the *functional* rules if R is removed: these rules become again possibly applicable to some situations,

(4) the rules *subservient* to R : if R is applicable to a situation then so do these rules to the same situation,

(5) and the rules to which R is *subservient* : if these rules are applicable to a situation then so does R to the same situation

Mondeca designed a tool for legal content management fitting the requirements mentionned above and giving to the law publisher a strategic position in the legal domain. This tool provides :

(1) versioning features to store and browse the different versions of a regulation ;

(2) a law triggering service to alert the individual when a new rule in a regulation is applicable to his situation ;

(3) a search feature for the compliant-seller to know which individual situations are concerned by a given rule in a regulation ;

and (4) a regulation management service to the lawmaker.

The figure 23 shows the functional architecture of the service which is very close to that of the ITM classification service. The knowledge base represented as a box in the center is divided in two parts. The right side stands for the publisher private environment and the left side is dedicated to the services provided by the publisher to the three other actors of the legal domain. Each actor has a private environment (named *workspace*) which is represented as a box. The restrictions on the workspaces vary according to the nature of the user.

7.3.5 The publisher workspace

The publisher workspace contains the *law hierarchy*. However, the textual content of the law is not stored in the base. Only chapters and their organisation are represented in a *tree*-structured hierarchy of *rule objects* providing a link to the real content stored outside ITM. This workspace grants the publisher a full access to the law hierarchy. He can navigate or update the law as it feels like without any restriction. Each action on the rules of the law hierarchy is recorded in a history of changes. This gives the ability to browse old versions of a regulation, to search rules by date, or to rewind a sequence of actions in order to retrieve an old state of the regulation. Each rule R is defined as the *category* of all individual situations *affected by* the legal content of R. If needed, R may be flagged with an *excluded status* which exludes it from the inheritance mechanism. The semantics of the regulation hierarchy is identical to that of the category hierarchy. That is to say : for each *non-excluded* rule R_c located below another *non-excluded* rule R_p, R_c must only apply to situations *affected by* R_p. The inheritance mechanism of the category hierarchy is mainly used to mutualize redundant criteria appearing in different rules by creating an upper-rule defined with a query Q and located above the rules whose queries share the criteria of Q. This feature is useful because it is very common in the legal domain for a criterion to appear in different rules of a regulation.

7.3.6 The individual workspace

An individual workspace provides a read-only access to the law hierarchy and the ability for a user to define his private situations. Such a workspace is not aware of the lawmaker nor compliant-seller workspaces existence. Situations are stored in ITM and may be defined on the fly by individuals. The classification service is configured to launch the *categorization* and trigger the individual when a new rule R is applicable to his situation (i.e. if the new category R contains his situation).

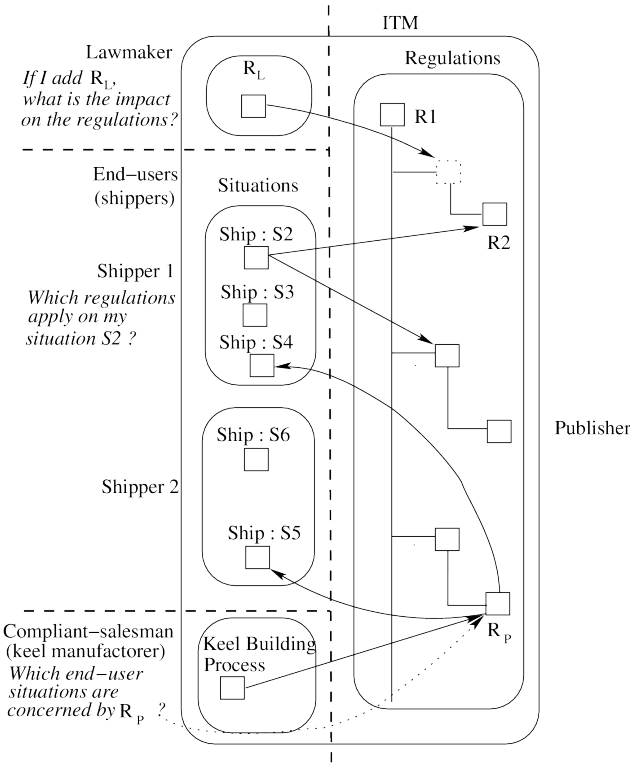


Figure 23: The standard classification architecture for legal content management.

7.3.7 The compliant-seller workspace

As a compliant-seller is driven by some of the individual motivations, its dedicated workspace inherits of the rights and restrictions of that of the individual with the exception that it provides a read-only access to the private situations stored in individual workspaces. Furthermore, it is unaware of the lawmaker workspace. When the compliant-seller needs to know which individual situations are concerned by a given rule R, the system performs the *instanciation* of the category R and returns its members.

7.3.8 The lawmaker workspace

A lawmaker workspace provides a full access to the law hierarchy but is unaware of the other workspaces existence. As said before, the lawmaker needs to check if some specific properties hold between rules of a given hierarchy. The classification service may be used in this purpose provided that the semantics of these properties is expressed in terms of categories. For two given rules R_1 and R_2 :

(1) R_1 and R_2 are said to be in *redundancy* iff the categories R_1 and R_2 are *tantamount*. On the contrary a rule R is said to be *single* iff it is not involved in any redundancies.

(2 and 3) R_1 is *useless* iff the corresponding category is *empty* ; and otherwise it is known as *functional*

(4 and 5) R_1 is *subservient* to R_2 iff the category R_1 *subsumes* the category R_2. On the contrary, two functional rules are said to be *incomparable*, if none of them is *subservient* to the other.

7.3.9 Starting with a regulation

Before starting to work on a regulation, the lawmaker ask for a *semantic audit* of the law hierarchy (as explained in section 7.2.3). The classification service returns the requested *report* $((\mathcal{T},\not{\mathcal{T}}),(\mathcal{E},\not{\mathcal{E}}),(\mathcal{S},\not{\mathcal{S}}))$ which describe the initial state of the law hierarchy. More precisely,

(1) $\mathcal{T}$ is the collection of tantamount classes of categories (in each set all categories are *tantamount* one another). As categories may be regarded as rules, for each set $T \in \mathcal{T}$, a rule belongs to T iff it is *redundant* with each other rule of T. Then, $\mathcal{T}$ may be considered as the collection of the largest classes of rule redundancies. $\not{\mathcal{T}}$ contains all couples of rules for which it can not be decided if they are in redundancy or not. Finally, rules not appearing in a set of $\mathcal{T}$ neither in a couple of $\not{\mathcal{T}}$ may be stated as *single*.

(2 and 3) $\mathcal{E}$ is the set of all empty categories. These categories are rules no longer applicable to any situation. Thus, $\mathcal{E}$ may be regarded as the set of all *useless* rules while $\not{\mathcal{E}}$ is the set of all rules for which the uselessness can not be decided. Finally, rules not belonging to $\mathcal{E}$ neither to $\not{\mathcal{E}}$ are stated *functional*.

(4 and 5) The last set $\mathcal{S}$ contains couples of categories whose first member subsumes the second. In a rule point of view, $\mathcal{S}$ contains all couples of rules whose first rule is *subservient* to the second rule. As before, $\not{\mathcal{S}}$ contains all couples of rules for which the question can not be solved. Finally, two rules are *incomparable* if they do not appear in a same couple of either $\mathcal{S}$ and $\not{\mathcal{S}}$.

7.3.10 Regulation rectification and maintenance

The regulation has to be corrected if the lawmaker feels that too many redundancies or useless rules were found by the semantic audit. The rectification of a regulation consists in :

(1) moving rules in the hierarchy

(2) changing the category definition of a rule

(3) or removing rules

such that the number of the redundant and/or useless rules become acceptable according to the criteria of the lawmaker. For this purpose, the lawmaker can request to receive a semantic feedback each time a change is done on the law hierarchy. As explained in section 7.2.4, such a feedback may be delivered to him after an insertion or a deletion in the category hierarchy. This feedback highlights the parts of the law hierarchy that have been impacted by the insertion or deletion of a rule. For example, if one rule seems erroneous to the lawmaker and it appears that its removal might decrease the number of useless rules, then a feedback on its deletion may help him to validate his intuition.

If the lawmaker consider that the law hierarchy is clean enough, the maintenance process can begin. During the maintenance of a regulation, the work of the lawmaker consists in :

(1) adding new rules having come into force recently

(2) removing old rules having be repealed

such that the number of the redundancy classes and useless rules stay the lowest possible (according to the criteria of the lawmaker). In the same way as before, the lawmaker may rely on the insertion/deletion semantic feedbacks to improve the consistency of the hierarchy.

8 Conclusion

The work presented in this paper is divided in three main parts : (1) the definition of a *graphical* knowledge representation formalism called $\mathcal{SGBF}$-model and a characterisation of some decidable fragments in terms of graph theory, (2) the design of a classification system for the ITM tool and (3) the description of an industrial application in legal content management. The $\mathcal{SGBF}$-model permits the definition of *graph formulas* by combining conceptual graphs with boolean connectors. A normal form called *indivisible normal form* has been defined for this formalism. When put in normal form a graph formula is turned into a clause of forests of indivisible graph arborescences. Two decidable fragments are defined for the $\mathcal{SGBF}$-model each of which relying on the first order logic BSR-fragment and guarded fragment. This formalism is used to define categories in the classification service of ITM which gives to the user the ability to manage and organise them in tree structured hierarchies provided with an inheritance mechanism. This generic classification service can be set to meet some requirements of the legal domain. To do so, personal situations of individuals are represented as instances in the lowest level of the ITM knowledge base and rules of a regulation are formalized into categories. The classification of a situation-instance in a

category means that the owner of this instance is affected by the rule corresponding to the category. A workspace is dedicated to each of the main actors of the legal domain : the lawmaker, the compliant-seller and the individual. According to the workspace, the services provided by the classification service depends on the workspace. While standard querying methods may be used to satisfy the requirements of the compliant-seller and individual, higher level reasoning mechanisms are needed to meet that of the lawmaker. In that case, the classification service translates - when needed and possible - graph formulas in their corresponding guarded or BSR logical formulas, in order to check their unsatisfiability with a logical reasoning engine. Our future work is to transfer existing decision procedures defined for the guarded and BSR fragments into the conceptual graph formalism, by making use of the projection elementary operation to build a corresponding "graphical" decision procedure.

References

[1] H. Andréka, I. Nemeti, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, pages 27:217–274, 1998.

[2] F. Baader, I. Horrocks, and U. Sattler. Description logics. *Handbook of Knowledge Representation. Frank van Harmelen and Vladimir Lifschitz and Bruce Porter.*, 2007.

[3] Franz Baader, Ralf Molitor, and Stephan Tobies. The guarded fragment of conceptual graphs. Technical report, RWTH Aachen, LuFg Theoretische Informatik, 1998.

[4] J.-F. Baget and M.-L. Mugnier. Extensions of Simple Conceptual Graphs: The Complexity of Rules and Constraints. *JAIR*, 16:425–465, 2002.

[5] P. Baumgartner. Fdpll: A first-order davis-putnam-logeman-loveland procedure. *CADE-17: Proceedings of the 17th International Conference on Automated Deduction, Springer*, pages 200–219, 2000.

[6] P. Bernays and M. Schnfinkel. Zum entscheidungsproblem der mathematischen logik. Math. Annalen, pages 342–372, 1928.

[7] Olivier Carloni, Michel Leclère, and Marie-Laure Mugnier. Introducing graph-based reasoning into an industrial knowledge management tool. *Applied Intelligence*, 2007.

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

[9] K. Claessen and N. Srensson. New techniques that improve mace-style model finding. *MODEL'03: Proceedings of the Workshop on Model Computation.*, 2003.

[10] A. Fuchs. Darwin: A theorem prover for the model evolution calculus. *Master's thesis, University of Koblenz-Landau*, 2004.

[11] H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. *LICS'03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 55, 2003.

[12] Erich Grdel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1998.

[13] J. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first order logic. *Journal of Automated Reasoning.*, pages 371–396, 2002.

[14] K. Korovin. Implementing an instantiation-based theorem prover for first-order logic. *IWIL'06: Proceedings of the 6th International Workshop on the Implementation of Logics*, 2006.

[15] C. Lutz, U. Sattler, and S. Tobies. A suggestion for an n-ary description logic. *Proceedings of the 1999 International Workshop on Description Logics (DL'99)*, 1999.

[16] F. Ramsey. On a problem of formal logic. Proc. of the London Math. Soc. 2^{nd} series, pages 264–286, 1930.

[17] 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.

[18] S. Schulz. A comparison of different techniques for grounding near propositional cnf formulae. FLAIRS'02: Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference. AAAI Press., pages 72–76, 2002.

[19] J. F. Sowa. Conceptual graphs for a data base interface. IBM Journal of Research and Development, pages 336–357, 1976.

[20] J. F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. 1984.

[21] J. F. Sowa and E. C. Way. Implementing a semantic interpreter using conceptual graphs. IBM Journal of Research and Development, pages 57–69, 1986.

A Relational simple graph

Any simple graph G defined according to a support S has a semantically equivalent *relational simple graph* $rf(G)$ defined according to a *relational support* $rf(S)$ such that :

- $rf(S)$ is identical to S excepted that the concept-type hierarchy (under $\top$) has been duplicated under the $\top^1$ type of the unary relation-type hierarchy,
- $rf(G)$ is identical to G excepted that for any concept c in G there exists a concept c' in $rf(G)$ such that (1) $type(c') = \top$ and (2) c' is linked to a unary relation whose type is $type(c)$.

However, the pair (G', S') where G' is a *relational simple graph* defined according to a *relational support* S' may lead to several pairs (G, S) whose concepts in the simple graph G and the concept-type hierarchy in the support S may not only be restricted to the type $\top$. Indeed, the number of pairs (G, S) depends on the number of unary relation with incomparable types linked to a same concept node. More precisely, given a couple (G', S') where G' is a *relational simple graph* defined according to a *relational support* S' the set $ru(G', S')$ of all semantically equivalent couples (G, S) of *relational unrestricted* simple graphs and supports is defined such that :

- given a subset X of $T_R^1 \cup T_R^2 \cup ... \cup T_R^n$, $mosit(X)$ returns the subset of X containing the most specific incomparable types of X
- given a concept c from a SG G, $urt(c, G)$ is the set containing the types of all unary relations linked to c in G
- $\mathcal{G}^S$ is the set of all couples (G, S) such that :
  - G is identical to G' excepted that for each c in G, $type_G(c) \in mosit(urt(c, G'))$ and c is not linked in G to any unary relation of type $type_G(c)$
  - S is identical to S' excepted that the concept-type hierarchy in S contains each concept-type used in G such that for any concept-types t and t' in S', $t \leq t'$ iff the same hold for unary-relation types t and t' in S'

Let G' be a relational simple graph defined according to a relational support S', if (G, S) is a couple from $ru(G', S')$ then $rf(G) = G'$ and $rf(S) = S'$. Conversely, given a simple graph G defined according to a support S, the couple (G, S) belongs to the set $ru(rf(G), rf(S))$. Thus, given a relational support S the language $\mathcal{SG}(rf(S)) \subseteq \mathcal{SG}(S)$ of all relational simple graphs defined over S has the same expressivity than the language $\mathcal{SG}(S)$ of all simple graphs defined over S.

B Arborescence merging

The semantics of the merge $A''[X' \cup X] = A' \oplus A$ of two arborescences A' and A is equivalent to the conjunction of the graph formulas corresponding to $A'[X']$ and $A[X]$. Indeed, the semantics of the arborescence $A''[X' \cup X]$ is $((root(A') \oplus root(A) \bigwedge_{\alpha \in subs(A)} \neg\mathcal{A}2g(\alpha) \bigwedge_{\alpha' \in subs(A')} \neg\mathcal{A}2g(\alpha'))[X' \cup X]$. According to the algebraic laws of the figure 9, the semantics of the merge of two graphs is the conjunction of these graphs. Thus, the semantics of A'' may be rewritten $(root(A) \bigwedge_{\alpha \in subs(A)} \neg\mathcal{A}2g(\alpha) \wedge root(A') \bigwedge_{\alpha' \in subs(A')} \neg\mathcal{A}2g(\alpha'))[X' \cup X]$ which is the conjunction of $\mathcal{A}2g(A[X])$ and $\mathcal{A}2g(A'[X'])$.

C Rewriting rules

This subsection shows that each rewriting rule from the section **??** preserve the semantics of the arborescence on which they are applied. In the following, $A^\star$ stands for $\mathcal{A}2g(A)$.

C.0.11 The top splitting rule TS

Given an arborescence $A[X]$, a graph G in $A[X]$ and $A_G = sub(G, A)$ if A_G is not top-indivisible, there exists two subarborescences A^1_G and A^2_G such that :

- A^1_G is top-indivisible such that $root(A^1_G) \neq G_\emptyset$,
- $vars(A^1_G) \cap vars(A^2_G) \subseteq X$
- A_G is equal to $A^1_G \oplus A^2_G$

In that case, the application $TS(G, A[X])$ returns a clause C whose content varies with the following cases :

G `is root`
If $G = root(A)$ then $C = \{\{A^1_G, A^2_G\}\}$.

The graph formula corresponding to the merge of two arborescences is equivalent to the conjunction of the graph formulas corresponding to them (see previous subsection). Thus, the graph formula $\mathcal{A}2g(A[X])$ is equivalent to the graph formula $(\mathcal{A}2g(A^1_G) \wedge \mathcal{A}2g(A^2_G))[X]$ which is the semantics $\mathcal{C}2g(C)$ of the clause C resulting from the application of TS for the current case.

G `is a root child`
If $G \in children(root(A))$ then the clause $C = \{\{\rho(B_1[X])\}, \{\rho(B_2[X])\}\}$ such that $root(B_i) = root(A)$ and $subs(B_i) = (subs(A) \setminus \{A_G\}) \cup \{A^i_G\}$.

- The following shows that $\mathcal{A}2g(A)$ is equivalent to $\mathcal{C}2g(C)$
- Let $R_A = root(A)$, $\mathcal{A} = subs(A) \setminus \{A_G\}$ and $\mathcal{A}^\star = \bigwedge_{A' \in \mathcal{A}} \neg A'^\star$),
- then $\mathcal{A}2g(A) = (R_A \wedge \neg A^\star_G \wedge \mathcal{A}^\star$.
- Since $A_G = A^1_G \oplus A^2_G$ and the merge is equivalent to the conjunction of the merged arborescences then $\mathcal{A}2g(A) = (R_A \wedge \neg(A^1_G)^\star \wedge (A^2_G)^\star \wedge \mathcal{A}^\star$.
- According to the *restricted De Morgan's* law of the figure 9, $\neg((A^1_G)^\star \wedge (A^2_G)^\star)$ can be rewritten $(\neg(A^1_G)^\star \vee \neg(A^2_G)^\star)$.
- Then, the distribution of $\wedge$ on $\vee$ lead to $\mathcal{A}2g(A) = (\neg(A^1_G)^\star \wedge \rho((R_A \wedge \mathcal{A}^\star)[X])) \bigvee (\neg(A^2_G)^\star \wedge \rho((R_A \wedge \mathcal{A}^\star)[X]))$
- which is equivalent to the graph formula $\mathcal{C}2g(C) = \rho((R_A \wedge \neg(A^1_G)^\star \wedge \mathcal{A}^\star)[X]) \bigvee \rho((R_A \wedge \neg(A^2_G)^\star \wedge \mathcal{A}^\star)[X])$ corresponding to our goal : the clause C.

`Otherwise :` G `is any other node`
Let F and E be two graphs in A, $G \in children(F)$ and $F \in children(E)$ then the clause $C = \{\{B[X]\}\}$ such that $subs(E, B) = subs(E, A) \setminus \{sub(F, A)\} \cup \{\rho(C_1[Y]), \rho(C_2[Y])\}$ where $Y = \mathcal{B}(sub(F, A), A[X]), root(C_i) = F$ and $subs(C_i) = (subs(F, A) \setminus \{A_G\}) \cup \{A^i_G\}$.

- The following shows that $\mathcal{A}2g(sub(E, A))$ is equivalent to $\mathcal{A}2g(sub(E, B))$
- Let $A_G = sub(G, A)$, $A_F = sub(F, A)$, $A_E = sub(E, A)$,

- $\mathcal{A}_F = subs(A_F) \setminus \{A_G\}$,
- $\mathcal{A}_E = subs(A_E) \setminus \{A_F\}$,
- $\mathcal{A}_F^\star = \bigwedge_{A' \in \mathcal{A}_F} \neg A'^\star)$
- and $\mathcal{A}_E^\star = \bigwedge_{A' \in \mathcal{A}_E} \neg A'^\star)$,
- then $\mathcal{A}2g(A_E) \equiv (E \wedge \neg(F \wedge \neg A_G^\star \wedge \mathcal{A}_F^\star) \wedge \mathcal{A}_E^\star$.
- Since $A_G = A_G^1 \oplus A_G^2$ and the merge is equivalent to the conjunction of the merged arborescences then $\mathcal{A}2g(A_E) \equiv (E \wedge \neg(F \wedge (A_G^1)^\star \wedge (A_G^2)^\star \wedge \mathcal{A}_F^\star) \wedge \mathcal{A}_E^\star$.
- According to the *restricted De Morgan's* law of the figure 9, $\neg((A_G^1)^\star \wedge (A_G^2)^\star)$ can be rewritten $(\neg(A_G^1)^\star \vee \neg(A_G^2)^\star)$.
- Then, the distribution of $\wedge$ on $\vee$ turns $(F \wedge (\neg(A_G^1)^\star \vee \neg(A_G^2)^\star) \wedge \mathcal{A}_F^\star)$ into the equivalent formula $\rho((F \wedge \neg(A_G^1)^\star \wedge \mathcal{A}_F^\star)[Y]) \bigvee \rho((F \wedge \neg(A_G^2)^\star \wedge \mathcal{A}_F^\star)[Y])$ where $Y = \mathcal{B}(sub(F, A), A)$. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the two argument of the previous disjunction.
- $\mathcal{A}2g(A_E)$ is now equivalent to $(E \wedge \neg(\mathcal{F}_1 \vee \mathcal{F}_2) \wedge \mathcal{A}_E^\star$
- Since $vars(\mathcal{F}_1) \cap vars(\mathcal{F}_2) = Y$, restricted De Morgan's law application on $\neg(\mathcal{F}_1 \vee \mathcal{F}_2)[Y]$ lead to $(E \wedge \neg\mathcal{F}_1 \wedge \neg\mathcal{F}_2 \wedge \mathcal{A}_E^\star$ which is equivalent to our goal $\mathcal{A}2g(sub(E, B))$

which is equivalent to the graph formula $\mathcal{C}2g(C) = \rho((R_A \wedge \neg(A_G^1)^\star \wedge \mathcal{A}^\star)[X]) \bigvee \rho((R_A \wedge \neg(A_G^2)^\star \wedge \mathcal{A}^\star)[X])$ corresponding to the clause C.