Académie de Montpellier

Université de Montpellier II

— Sciences et Techniques du Languedoc —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures et Systèmes**

Représenter des connaissances et raisonner avec des hypergraphes:

de la projection à la dérivation sous contraintes

par

Jean-François Baget

Soutenue le 26 novembre 2001 devant le Jury composé de :

Michel Chein, Professeur, Université de Montpellier II	Co-directeur de Thèse
Roland Ducournau, Professeur, Université de Montpellier II	Examinateur
Michel Habib, Professeur, Université de Montpellier II	Examinateur
Pierre Marquis, Professeur, Université d'Artois	Rapporteur
Marie-Laure Mugnier, Maître de Conférence, Université de Montpellier II	Co-directrice de Thèse
Marie-Christine Rousset, Professeur, Université de Paris-Sud	Rapporteur
Marie-Catherine Vilarem, Professeur, Université de Montpellier II	Examinateur

Académie de Montpellier

Université de Montpellier II

— Sciences et Techniques du Languedoc —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures et Systèmes**

Représenter des connaissances et raisonner avec des hypergraphes:

de la projection à la dérivation sous contraintes

par

Jean-François Baget

Soutenue le 26 novembre 2001 devant le Jury composé de :

Michel Chein, Professeur, Université de Montpellier II	Co-directeur de Thèse
Roland Ducournau, Professeur, Université de Montpellier II	Examinateur
Michel Habib, Professeur, Université de Montpellier II	Examinateur
Pierre Marquis, Professeur, Université d'Artois	Rapporteur
Marie-Laure Mugnier, Maître de Conférence, Université de Montpellier II	Co-directrice de Thèse
Marie-Christine Rousset, Professeur, Université de Paris-Sud	Rapporteur
Marie-Catherine Vilarem, Professeur, Université de Montpellier II	Examinateur

Université de Montpellier II (Université des Sciences et Techniques du Languedoc)
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Référence
BAGET, Jean-François *« Représenter des connaissances et raisonner avec des hypergraphes: de la projection à la dérivation sous contraintes »* Thèse de doctorat, Université de Montpellier II, 26 novembre 2001

À Christine et Colin

Remerciements

a

Table des matières

Table des figures

Liste des définitions

Chapitre 1

Introduction

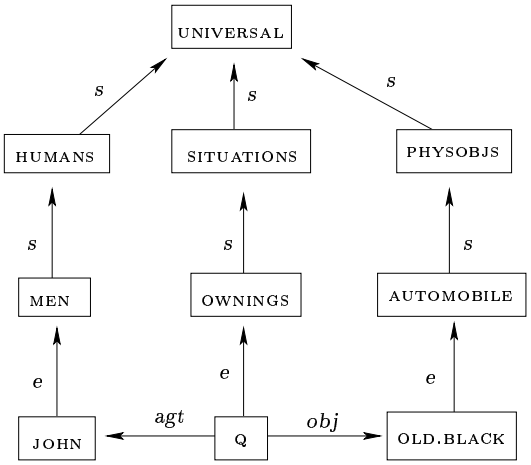
Des notations diagrammatiques ont été de tous temps utilisées pour représenter des connaissances, de manière intuitive, que ce soit en logique, en philosophie, en linguistique, ou aujourd'hui en informatique. Afin de pouvoir définir formellement des raisonnements sur ces connaissances, et de pouvoir les traiter de façon automatisée par un ordinateur, il est nécessaire de munir ces diagrammes d'une structure mathématique formelle. Les graphes, qui sont des objets structurés et admettent une représentation graphique « naturelle » (un diagramme particulier, mais souvent utilisé), peuvent être considérés comme le candidat privilégié pour fonder un modèle de représentation de connaissances dont les objets sont « des dessins ». L'objet de cette thèse est l'étude d'un modèle formel de représentation de connaissances utilisant des graphes, basé sur les graphes conceptuels.

Les graphes conceptuels ont été introduits dans [Sowa, 1976] comme une interface graphique des bases de données relationnelles. L'idée de représenter des connaissances par des graphes n'est alors pas neuve, et on peut trouver une source d'inspiration des graphes conceptuels dans les réseaux sémantiques de [Quillian, 1968] (voir, pour une présentation synthétique de ce domaine, [Lehman, 1992]). Cette première présentation des graphes conceptuels comporte deux idées fortes :

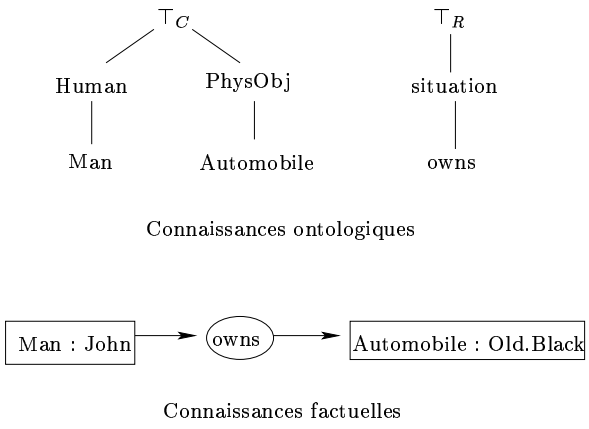
D'un point de vue représentation de connaissances, les graphes de Sowa apportent une structuration des différents types de connaissances qui n'était pas présente dans les réseaux sémantiques. En effet, ce formalisme marque une séparation nette entre les connaissances ontologiques[1] (encodées dans une structure particulière que nous appellerons par la suite support), et les connaissances factuelles qui sont représentées par les graphes. De plus, ce formalisme distingue nettement les entités (représentées dans le graphe par des rectangles) des relations entre ces entités (représentées par des ovales). La FIG. 1.1 illustre la représentation, par un réseau sémantique puis par un graphe conceptuel, de la même information : « John (qui est un homme, sorte d'humain) possède Old Black (qui est une automobile, sorte d'objet physique) ».[2]

[1] Du grec *ontos* (l'être) et *logos* (le monde), le terme d'ontologie est souvent utilisé pour une taxonomie classifiant des catégories/types de concepts dans une base de connaissances.

[2] La représentation des graphes conceptuels simples n'est pas si éloignée de celles utilisées dans les méthodes de conception objet telles qu'UML. En cela, nous pourrions dire que les graphes conceptuels simples



Un réseau sémantique [Hendrix, 1979]



Représentation avec un graphe conceptuel

FIG. 1.1 – Des réseaux sémantiques aux graphes conceptuels

De plus, Sowa définit un ensemble d'opérations élémentaires qui permettent de dériver des graphes à partir d'autres graphes, et montre que ces opérations permettent de répondre à une requête conjonctive en bases de données. Ces opérations de graphes préfigurent le théorème d'homomorphisme de [Chandra and Merlin, 1977].

1.1 Diverses approches des graphes conceptuels

Dans son ouvrage de référence sur les graphes conceptuels, [Sowa, 1984] munit ces graphes d'une sémantique dans le fragment positif, conjonctif et existentiel de la logique du premier ordre, par l'intermédiaire d'une transformation Φ, et montre l'adéquation d'une opération de graphe, la projection, par rapport à la déduction en logique du premier ordre. [Chein and Mugnier, 1992] reformuleront la projection en termes d'homomorphisme de graphe, montreront son équivalence avec les opérations élémentaires de [Sowa, 1976], et prouveront la complétude de la projection par rapport à la déduction. Ces graphes ont par la suite été appelés « graphes conceptuels simples », par opposition à des graphes conceptuels plus complexes, notamment ceux que nous appellerons ici les « graphes conceptuels généraux », introduits dans [Sowa, 1984].

Inspirés des travaux du logicien Peirce (1897–1906) sur les graphes existentiels (on peut voir à ce sujet la présentation qu'en donne [Roberts, 1992]), ces graphes conceptuels généraux sont une généralisation des graphes conceptuels simples à l'ensemble de la logique du premier ordre. En enfermant des parties de graphes dans des boîtes qui indiquent une

sont une représentation objet des réseaux sémantiques. Cependant, pour prolonger cette comparaison, il manque une « couche méta-objet » à cette représentation par des graphes conceptuels (le support n'est pas un graphe, et *is-a* n'est pas une relation). Bien que cette problématique ne soit pas abordée dans cette thèse, le lecteur pourra se référer à [Baget, 2000] pour une ébauche de solution.

négation, et qui peuvent s'imbriquer les unes dans les autres (comme illustré, au chapitre suivant, dans la FIG. 2.4), on obtient une représentation graphique de la totalité de la logique du premier ordre.

Les travaux de la communauté « graphes conceptuels », qui s'est formée autour des graphes de Sowa, ont pris trois directions très différentes, suivant l'intérêt que l'on porte à telle ou telle particularité du modèle de base :

Approche diagrammatique Dans une optique de pure représentation, c'est l'aspect visuel, *diagrammatique*, des graphes qui importe. Ces travaux s'intéressent à la clarté, la lisibilité de la représentation (par exemple dans un contexte de représentation de la langue naturelle, problématique majeure dans [Sowa, 1984]). Les raisonnements, quant à eux, sont considérés comme un problème externe, traité par la traduction des graphes vers un autre formalisme, par exemple la (une) logique. Les graphes sont alors une interface pour cet autre formalisme. De façon extrême, ces objets ne sont parfois même pas traduisibles en logique, comme le montre l'exemple de la FIG. 1.2, tiré de la page web de Sowa (`http ://users.bestweb.net/~sowa/cg/cgstand.htm`).

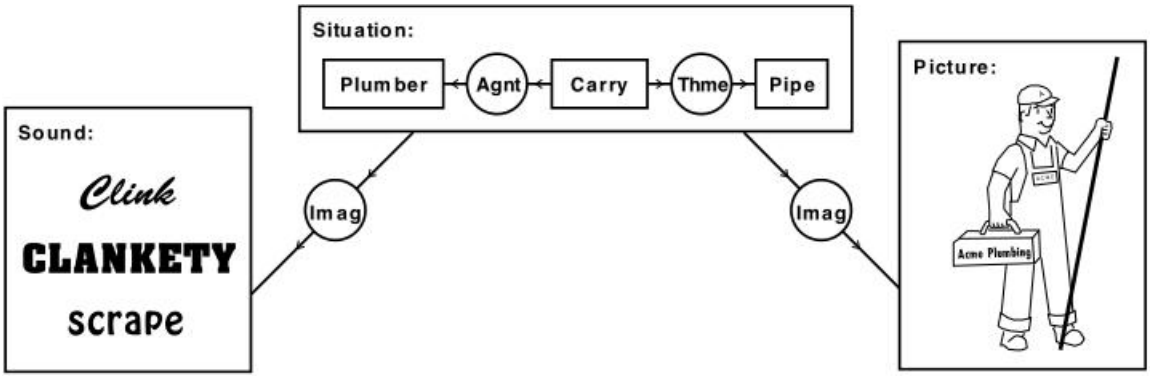


FIG. 1.2 – Les graphes conceptuels vus comme une « pure représentation ».

Approche logique Privilégiant l'aspect logique (les graphes conceptuels sont vus comme une représentation graphique de la logique du premier ordre), un autre axe de recherche s'est constitué dans la continuation des travaux de Peirce. Les calculs sur les graphes conceptuels sont dans ce cas des opérations sur les boîtes indiquant la négation : déplacer ou effacer une boîte, remplacer dans une boîte un graphe par un autre ... Un ensemble de telles règles, adéquat et complet par rapport à la déduction en logique du premier ordre, est proposé par [Wermelinger, 1995], qui corrige les erreurs des règles de [Sowa, 1984]. Une autre version est proposée dans [Dau, 2000]. Les opérations utilisées sont diagrammatiques (ce sont des retranscriptions graphiques, sur les dessins des graphes, de règles logiques), mais ne sont pas des opérations de graphes, au sens usuel de la théorie des graphes. Ce genre d'opérations est illustré dans la FIG. 1.3 (aussi tirée de la page web de Sowa), qui

montre une série de réécritures utilisant les règles de graphes existentiels de Peirce pour prouver le *Praeclarum Theorema* de Leibnitz $(((p \to r) \wedge (q \to s)) \to ((p \wedge q) \to (r \wedge s)))$.

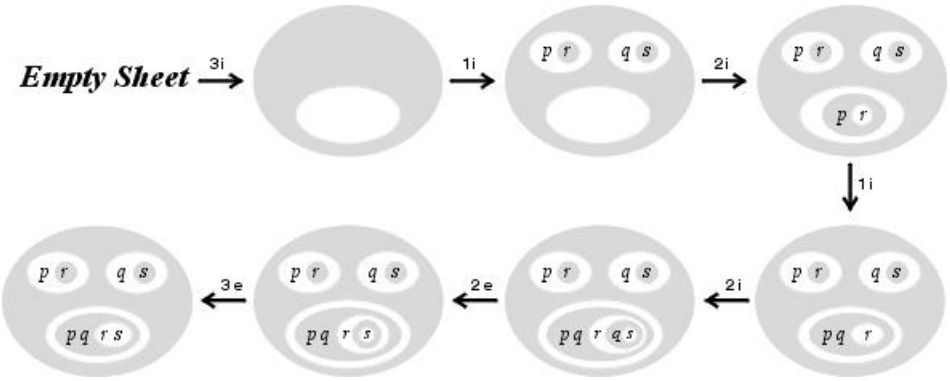


FIG. 1.3 – Preuve du *Praeclarum Theorema* par les règles diagrammatiques de Peirce.

Approche « graphes et opérations de graphes » Enfin, un autre axe de recherche, dans lequel s'inscrit cette thèse, cherche à développer les graphes conceptuels comme un modèle de représentation de connaissances fondé logiquement [3], mais autonome de la logique, basé sur des graphes et des opérations de graphes (voir notamment [Levinson and Ellis, 1991, Chein and Mugnier, 1992]).

La frontière entre ces différentes approches n'est pas toujours très claire. Qu'en est-il, par exemple, de la démarche originale de [Kerdiles, 1997], qui combine la projection des graphes conceptuels simples et la méthode des tableaux de [Smullyan, 1968] pour obtenir une méthode adéquate et complète pour les graphes conceptuels généraux correspondant à la totalité de la logique du premier ordre ?

1.2 L'approche « graphes et opérations de graphes »

Selon notre point de vue, l'intérêt des graphes conceptuels simples réside d'une part dans la représentation par des graphes, qui (au moins s'ils sont de petite taille ou possèdent une structuration particulière) sont facilement lisibles/visualisables par un être humain, et d'autre part dans l'utilisation d'opérations de graphes pour les raisonnements. La projection est une sorte d'appariement de graphes, que l'on peut facilement interpréter et visualiser. Le même langage étant utilisé au niveau interface et au niveau opérationnel, les raisonnements peuvent être expliqués étape par étape à un utilisateur, sur le graphe qui correspond à sa modélisation. Le lecteur pourra se référer aux discussions sur cet aspect

[3] Même si l'on manipule des graphes, il est normal, puisque l'on s'intéresse à la formalisation des raisonnements, de les comparer à ce formalisme de référence qu'est la logique. C'est bien pourquoi nous cherchons à munir nos modèles de sémantiques logiques adéquates et complètes.

des graphes conceptuels, dans le cadre d'une application en ingénierie des connaissances [Bos et al., 1997], ou dans le cadre d'une expérimentation en recherche documentaire dans [Genest, 2000]. De plus, nous jugeons fondamental le résultat d'équivalence avec un fragment de la logique du premier ordre [Sowa, 1984, Chein and Mugnier, 1992], qui permet de relier un problème qui serait purement combinatoire (la recherche d'homomorphismes) aux problématiques de représentation de connaissances.

En résumé, nous considérerons comme essentielles ces trois caractéristiques du modèle de base :

- les connaissances sont représentées par des graphes ;
- les raisonnements sont effectués par des opérations de graphes ;
- ils sont adéquats et complets par rapport à une sémantique logique.

Particularités de notre approche Depuis 1992, l'équipe « Graphes Conceptuels » du LIRMM en particulier étudie des extensions du modèle de base qui conservent ces caractéristiques essentielles. Outre la possibilité de représenter graphiquement les connaissances comme les raisonnements, les membres de cette équipe ont parié sur le fait que « faire de la logique avec des graphes » permettrait d'utiliser, au niveau algorithmique, des notions de théorie des graphes qui ne s'expriment pas de façon naturelle en logique (par exemple un chemin, le voisinage d'un sommet, une composante connexe ou biconnexe, ...). La réussite de ce pari (voir, par exemple, la notion de *pièces* dans les règles de graphes) est la principale justification de cette approche. En effet, l'approche « logique » des graphes conceptuels apporte à la logique les mêmes avantages de lisibilité des connaissances et des raisonnements. Notre approche en diffère sur deux points :

1) Si nous souhaitons donner à nos opérations une sémantique logique, nous ne cherchons pas à représenter la logique. La notion de contrainte utilisée dans ce mémoire, par exemple, qui s'exprime de façon naturelle par des projections de graphes, n'est pas « importée » de la logique du premier ordre (et n'est pas directement traduisible).

2) Considérer des graphes, au sens de la théorie des graphes, permet de développer des algorithmes originaux, traduisibles bien entendu en logique, mais qui n'auraient peut-être pas été envisagés dans ce paradigme.

Extensions du modèle de base Parmi les extensions du modèle de base étudiées au LIRMM, nous pouvons citer :

1. Les liens de co-référence, qui permettent d'indiquer que deux sommets d'un graphe représentent la même entité.
2. Les graphes emboîtés [Chein et al., 1998], qui permettent une structuration hiérarchique de l'information en associant à des sommets d'un graphe une description qui est elle-même un graphe.
3. Les règles de graphes, qui expriment des connaissances de la forme « si ... alors ... », représentent un fragment plus important de la logique du premier ordre. Le mécanisme de chaînage arrière de [Gosh and Wuwongse, 1995] est adéquat et complet

par rapport à la déduction en logique du premier ordre, mais n'utilise pas la structure de graphes des règles (elles sont décomposées en sous-structures assimilables à des atomes en logique). L'exploitation, sous la forme de *pièces*, de la structure du graphe par [Salvat, 1997], montre un réel apport par rapport à la méthode de résolution de Prolog [Coulondre and Salvat, 1998].

4. Les contraintes, utilisées pour porter un jugement sur la validité d'un graphe. Différentes sortes de contraintes ont été étudiées à cet effet [Mineau and Missaoui, 1997, Dibie et al., 1998, Baget et al., 1999]. Ces contraintes utilisent la projection comme mécanisme de base, et représentent des connaissances de la forme « si l'information A est présente dans le graphe, alors l'information B doit aussi/ne doit pas y être ».

Ces différentes extensions ne doivent pas être considérées de la même façon. En effet, bien que toutes aient un intérêt d'un point de vue représentation de connaissances, certaines n'apportent pas un réel enrichissement du modèle. Nous avons montré [Baget, 1998, Baget, 1999] que les deux premières extensions n'augmentent pas l'expressivité du modèle de base (nous les appellerons des formalismes du modèle de base, intitulé, de façon générique $\mathcal{SG}$), tandis que les deux autres engendrent des modèles de raisonnement plus riches, faisant partie de la famille de modèles que nous étudierons.

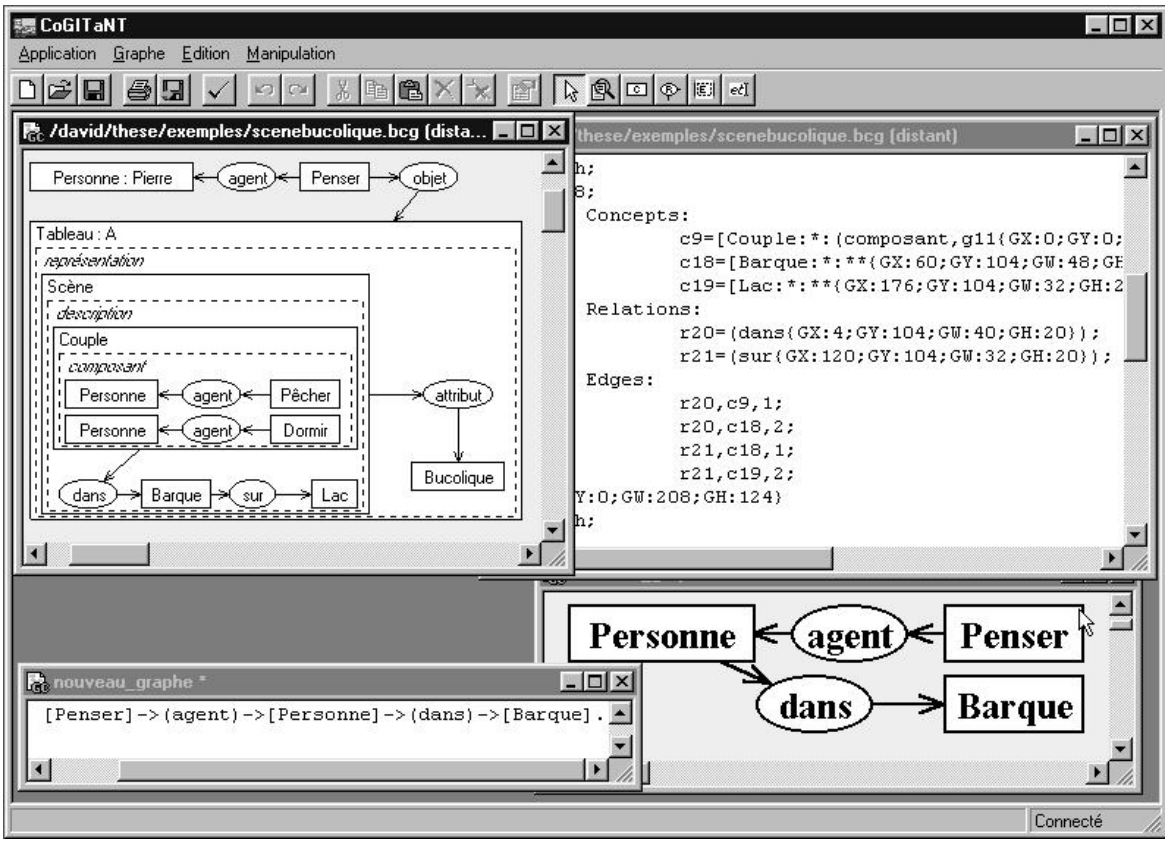


FIG. 1.4 – Interface graphique de CoGITaNT'2000.

Applications de ces travaux L'objectif principal de l'équipe « Graphes Conceptuels » du LIRMM est la construction d'un système général de représentation, d'acquisition et d'interrogations de connaissances utilisant les graphes conceptuels [Chein, 1997]. Pour ce faire, une bibliothèque de classes C++, appelée CoGITo [Haemmerlé, 1995] puis CoGITaNT [Genest and Salvat, 1998], a été développée.

La version actuelle de cette plate-forme (que l'on pourrait appeler CoGITaNT'2000), réalisée par David Genest, a vu une mise à jour des classes C++, la rédaction d'une documentation complète, l'intégration d'une architecture client-serveur et, enfin, d'une interface graphique : nous pouvons en voir une capture d'écran dans la FIG. 1.4. L'intégralité de cette plate-forme est disponible sur le site de l'équipe : `http ://www.lirmm.fr/~cogito/`.

Les outils théoriques aussi bien que logiciels développés au sein de l'équipe ont pu être appliqués et validés par la participation à de nombreux projets. Nous pouvons citer le projet national GRAFIA qui a proposé un système d'interrogation et d'acquisition de données en accidentologie [Alpay et al., 1995] ; une collaboration avec Dassault Électronique pour la modélisation de plans d'urgence sur les sites sensibles [Bos et al., 1997], le travail avec l'ABES sur un système de recherche documentaire [Genest, 2000] et, le projet OPALES, en collaboration avec l'INA, étendant ce système à une indexation de documents audiovisuels.

1.3 Contenu de ce mémoire

Dans ce mémoire, nous considérons un modèle de base, que nous noterons $\mathcal{SG}$, pour lequel l'opération élémentaire de raisonnement est la projection, mais dont les objets pourront être indifféremment des graphes conceptuels simples, avec ou sans liens de co-référence, ou des graphes conceptuels emboîtés. Pour cela, nous définissons un formalisme apparenté aux graphes conceptuels simples, que nous appelons celui des *graphes élémentaires*. Bien qu'il ne soit pas d'un grand intérêt d'un point de vue représentation de connaissances, ce formalisme servira d'*abstraction, de structure de données commune* aux formalismes précédents, et nous verrons que les graphes de ces différents formalismes peuvent être traduits de façon naturelle en graphes élémentaires, sur lesquels nous pourrons calculer les projections.

Nous aurions pu de la même manière choisir les graphes conceptuels simples (le modèle standard) comme modèle de base pour les autres formalismes. Notre choix d'un nouveau formalisme (encore un !) comme formalisme de référence s'explique essentiellement par le fait que les graphes élémentaires nous semblent être, d'un point de vue algorithmique, « le bon niveau d'abstraction » : la vision des relations comme des hyperarcs et non comme des sommets nous permet d'écrire des algorithmes plus efficaces. Nos graphes élémentaires sont d'ailleurs très proches de la structure de donnée simplifiée proposée par [Becker and Hereth, 2001] pour les graphes conceptuels. Il ne s'agit pas de remplacer ces formalismes « graphes conceptuels » par un formalisme « graphes élémentaires ». Nous verrons en effet au long de ce mémoire les intérêts des graphes conceptuels en termes de représentation de connaissances. Cependant, nous pensons qu'il peut être avantageux d'utiliser les graphes élémentaires pour *calculer* les différents raisonnements (généricité et efficacité des algorithmes), alors que les différentes sortes de graphes conceptuels (simples

ou emboîtés) doivent être les objets manipulés par l'utilisateur, à un niveau interface.

Ensuite, nous pourrons partir d'un représentant quelconque de ces formalismes équivalents pour construire une famille de (vraies) extensions du modèle. Suivant les objets que nous considérerons dans une base de connaissances (graphes, règles, contraintes), nous obtiendrons les différents modèles de raisonnement de la famille $\mathcal{SG}$.

Plan

Dans le chapitre 2 de cette thèse, nous proposons, de façon intuitive, un panorama des différents modèles qui composent la famille $\mathcal{SG}$. Ce mémoire est ensuite organisé en deux parties distinctes.

La première partie est consacré au modèle de base, $\mathcal{SG}$.[4] Les objets de ce modèle sont des graphes (que l'on peut définir suivant plusieurs formalismes), et l'opération de raisonnement est la projection. Dans le Chap. 3, nous dressons un inventaire des principaux résultats (structure de la relation de spécialisation, sémantique logique Φ) obtenus sur les graphes conceptuels simples, en les adaptant au formalisme simplifié des graphes élémentaires. Nous apportons à certains de ces résultats de nouvelles preuves (irredondance, sémantique logique), ou étendons des résultats par ceux acquis dans des domaines de recherche voisins (ensemble des projections vu comme une catégorie, treillis dénombrable universel des graphes irredondants). Le Chap. 4 présente d'autres formalismes du modèle $\mathcal{SG}$: les graphes conceptuels simples et les graphes conceptuels emboîtés. Nous y montrons que tous ces formalismes sont équivalents. Enfin, dans le Chap. 5, nous rappelons que le problème de l'existence d'une projection entre deux graphes est un problème NP-complet, et utilisons sa similarité avec le problème de satisfaction de contraintes pour adapter des algorithmes de résolution efficaces. Enfin, nous proposons un algorithme original, BCC, reposant sur les composantes biconnexes du graphe. Cet algorithme a été proposé, dans [Baget and Tognetti, 2001], dans le cadre des réseaux de contraintes binaires. Nous le présentons ici dans le formalisme des graphes élémentaires, prenant ainsi en compte des relations (et donc des contraintes) n-aires.

La deuxième partie est consacrée à une famille d'extensions du modèle de base, que nous avons appelée famille $\mathcal{SG}$ [Baget and Mugnier, 2001b]. Dans le Chap. 6, nous présentons le modèle $\mathcal{SR}$, qui intègre les règles de graphes de [Salvat and Mugnier, 1996]. Nous définissons ces règles en utilisant le formalisme des graphes élémentaires, et les adaptons aux graphes des autres formalismes de $\mathcal{SG}$. Nous rappelons ensuite leur sémantique logique, par une extension de Φ. Ensuite, nous rappelons la semi-décidabilité du problème de déduction dans $\mathcal{SR}$ [Coulondre and Salvat, 1998], et en exhibons une autre démonstration, qui prouve que $\mathcal{SR}$ est un modèle de calcul. Après avoir fixé les limites de cette indécidabilité d'après quelques critères naturels (le nombre d'étiquettes disponibles dans le support, le nombre de règles, et la taille de la frontière), nous proposons une méthode,

[4] Ne pas confondre le modèle de représentation de connaissances $\mathcal{SG}$ avec la famille de modèles $\mathcal{SG}$, dont il est l'élément le plus spécifique. Cette structuration en une famille de modèles est inspirée, dans sa présentation, de la démarche de [Donini et al., 1997] dans la famille $\mathcal{AL}$ de logiques de description (un autre héritier des réseaux sémantiques).

basée sur une compilation de la base de règles, qui améliore l'algorithme de résolution en marche avant et permet de donner des critères plus larges de décidabilité. Enfin, dans le Chap. 7, nous présentons les trois autres modèles de la famille $\mathcal{SG}$, qui utilisent des graphes appelés contraintes : celles-ci apportent la notion de validité/cohérence à nos graphes conceptuels. Le modèle le plus général de cette famille est vraiment indécidable [Baget and Mugnier, 2001b], mais nous exhibons une restriction des règles qui fait retomber la complexité du problème de déduction pour tous les modèles de cette famille dans la hiérarchie polynomiale [Baget and Mugnier, 2001a].

Chapitre 2

Un panorama de la famille $\mathcal{SG}$

Nous souhaitons donner dans ce chapitre une idée générale, intuitive, des différents modèles de représentation de connaissances qui seront étudiés dans ce mémoire. Dans chacun de ces modèles, pour des raisons de clarté, nous utiliserons le formalisme des graphes conceptuels simples (mais nous aurions pu présenter ces modèles, de la même manière, en utilisant un des formalismes équivalents que nous définirons plus précisément au Chap. 4). Dans la suite de ce chapitre et de cette thèse, nous utiliserons souvent le terme « graphe » pour désigner un objet du modèle de base. Le lecteur pourra traduire ce terme par « graphe conceptuel simple », ou considérer un objet construit dans un formalisme équivalent.

Dans un premier temps (Sect. 2.1), nous exposons les objets et les opérations du modèle de base, $\mathcal{SG}$. Le modèle $\mathcal{SR}$, présenté à la Sect. 2.2, permet, grâce à l'utilisation des règles, de représenter des connaissances de la forme « si ... alors ... ». Nous proposons ensuite (Sect. 2.3) plusieurs modèles différents, obtenus en considérant des graphes appelés contraintes. Ces graphes permettent d'exprimer un jugement sur la validité d'autres graphes.

Tout au long de ce chapitre, nous prendrons des exemples issus de la modélisation d'une étude de cas en acquisition de connaissances, SISYPHUS-I. Il s'agit d'un problème d'allocation de ressources, le but étant de placer des personnes dans différents bureaux, tout en satisfaisant un certain nombre de contraintes. L'énoncé de ce problème ainsi que l'intégralité de la modélisation que nous en avons faite [Baget et al., 1999] sont donnés, en anglais, dans l'annexe B.

2.1 Le modèle de base : $\mathcal{SG}$

Dans ce modèle, une base de connaissances est constituée d'un support $\mathcal{S}$, qui encode les connaissances ontologiques, et d'un ensemble de graphes $\mathcal{G}$, étiquetés par les éléments de $\mathcal{S}$, qui représentent des faits. Ces graphes, bipartis, comportent deux classes de sommets : les sommets concepts, qui représentent des entités, et les sommets relations, qui représentent des relations entre ces entités. Une requête H sera aussi un graphe étiqueté par les éléments de $\mathcal{S}$, et nous définissons une opération d'appariement, la projection, qui répond à la

question : « est-ce-que les connaissances représentées dans H sont présentes dans $\mathcal{G}$? ».

Selon les applications, on pourra considérer que le graphe H est un *but* plutôt qu'une requête. Nous employons ici le terme de requête pour indiquer que, si le problème de base est celui de l'existence d'une projection, nous nous intéressons en fait à la recherche de toutes les projections (solutions).

Examinons ces différents objets, tels qu'ils sont définis dans le modèle des graphes conceptuels simples dans [Chein and Mugnier, 1992].

2.1.1 Le support

Le support $\mathcal{S}$ sert à encoder à la fois les types disponibles pour étiqueter les éléments d'un graphe, et une relation d'ordre sur ces types, la relation de sous-typage. Une hiérarchie de types, T_C, encode les types de concepts disponibles pour étiqueter les sommets concepts du graphe, et un ensemble de hiérarchies, $T_R = \{T_R^1, \ldots, T_R^k\}$, encode les types de relations d'arités respectives $1, \ldots, k$. Notons que ces hiérarchies sont des ordres partiels quelconques (possibilité d'héritage multiple), et qu'il s'avère souvent utile (d'un point de vue représentation de connaissances) de considérer pour chacune un élément maximum, que nous noterons respectivement $\top_C, \top_1, \ldots, \top_k$.

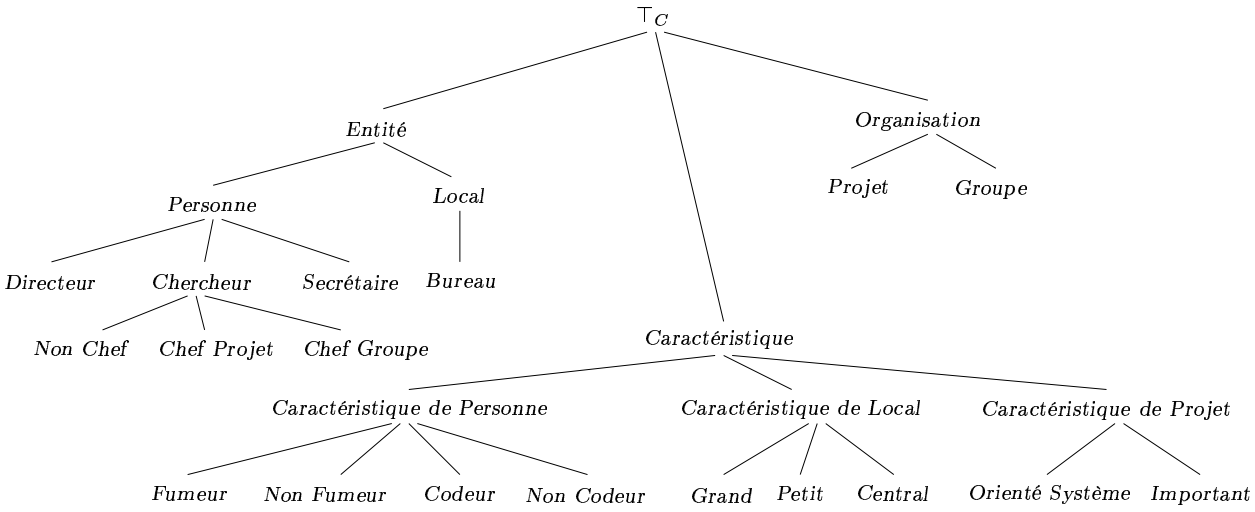


FIG. 2.1 – La hiérarchie de types de concepts utilisée pour SISYPHUS-I

A un type de relation t d'arité p, nous associerons souvent sa signature $\sigma(t) \in (T_C)^p$, qui est un tuple de p éléments de T_C. Bien que cette signature ne soit pas utilisée par les opérations de graphes que nous définissons, il s'agit d'un guide appréciable pour la modélisation. Ainsi, par exemple, la signature du type de relation *membre* est (*Personne, Organisation*), ce qui signifie que le premier argument d'une relation de type *membre* est de type *Personne* (ou un de ses sous-types), et que le second argument est de type *Organisation* (ou un de ses sous-types). Cette signature sert de guide sur la façon d'utiliser les types de relations du support, et pose des contraintes sur les graphes que l'on est en droit d'écrire. Notons

que si r est un sous-type de r', alors le $i$ième élément de $\sigma(r)$ devra être un sous-type du $i$ième argument de $\sigma(r')$. En effet, la signature posant une contrainte sur le type maximal de chacun des arguments d'une relation, cette contrainte peut être renforcée lorsque l'on spécialise le type de relation, mais pas relaxée.

Enfin, nous nous donnons un ensemble $\mathcal{I}$ de marqueurs individuels, qui serviront à étiqueter des sommets concepts représentant une entité distinguée : deux sommets concepts ayant un même marqueur individuel représenteront la même entité. Une relation de conformité, c, indique le type de concept associé à chaque marqueur individuel.

La FIG. 2.1 représente la hiérarchie de types de concepts utilisée dans notre modélisation du problème SISYPHUS-I. Notons que cette hiérarchie est un arbre (héritage simple), mais que ce n'est pas toujours le cas en général.

2.1.2 Les faits

Un graphe conceptuel simple est un multigraphe (il peut y avoir plusieurs arêtes entre deux sommets), biparti (il y a deux classes de sommets telles qu'aucune arête ne relie deux sommets de la même classe), étiqueté par les éléments d'un support $\mathcal{S}$.

Les sommets de la première classe sont appelés sommets concepts. Ils sont étiquetés par un élément de T_C (leur type), et par un marqueur individuel (notons que le type doit alors être celui associé au marqueur individuel par la relation de conformité), ou le marqueur dit générique, $*$, indépendant du support. Un sommet concept dont l'étiquette comprend un marqueur individuel est dit individuel (il représente une entité nommée), sinon, il est dit générique (il représente une certaine entité non identifiée).

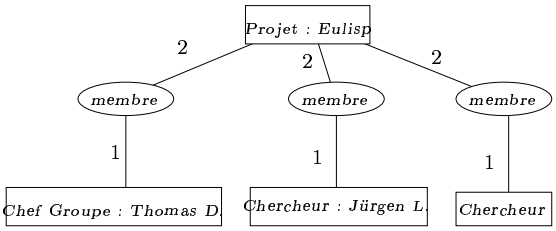
Les sommets de la deuxième classe sont appelés sommets relations. Ils sont étiquetés par un type de relation. Si l'arité du type d'un sommet relation r est p, et sa signature $(t_1, \ldots, t_p)$, alors ce sommet relation est relié à p sommets concepts, non nécessairement distincts, par p arêtes distinctes. Ces arêtes sont numérotées de 1 à p, et, pour respecter la signature, le sommet concept relié à r par l'arête numérotée j, le $j^{\text{ième}}$ argument de la relation r, doit avoir un type plus spécifique que t_j.

Les sommets concepts seront représentés par un rectangle autour de leur étiquette, inscrite sous la forme `Type : marqueur`. Afin d'alléger les notations, nous adopterons souvent la convention consistant à ne pas représenter le marqueur générique. Les relations seront représentées par un ovale entourant l'inscription de leur type. Le numéro associé à une arête sera inscrit à côté de cette arête (voir, par exemple, les graphes représentés dans la FIG. 2.2).

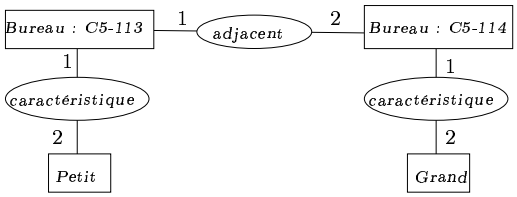
Notons qu'un graphe conceptuel n'est pas nécessairement connexe. Un ensemble de graphes peut ainsi être considéré comme un seul graphe, formé par l'union disjointe (le graphe dont le dessin est la juxtaposition des dessins des graphes initiaux) de ses éléments. Ajoutons dès à présent que, pour des raisons de complétude de l'opération de projection, nous aurons parfois besoin de mettre un graphe sous « forme normale » : celle-ci s'obtient en fusionnant les sommets concepts ayant le même marqueur individuel. Remarquons que, grâce à la relation de conformité, cette fusion ne pose aucun problème : le sommet concept résultant de la fusion de ces sommets a le même type (et le même marqueur) que ceux-ci.

Un ensemble de graphes composant une base de faits $\mathcal{G}$ peut ainsi être considéré comme un seul graphe, sous forme normale.

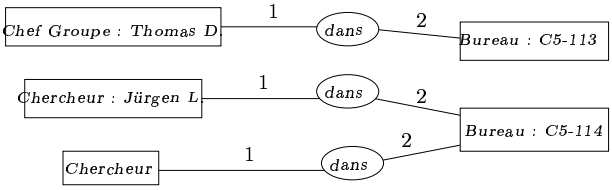
La possibilité d'éditer/ construire un graphe, par morceaux et pas d'un seul bloc, à partir de plusieurs autres graphes est d'une grande importance pour l'utilisateur qui visualise une base de faits. En effet, un avantage des graphes conceptuels réside dans la lisibilité des graphes. Cependant, cet avantage n'est plus aussi clair lorsque les graphes deviennent grands. Aussi, l'utilisateur préférera dessiner des petits graphes, qui seront « joints » automatiquement par leurs sommets concepts individuels.



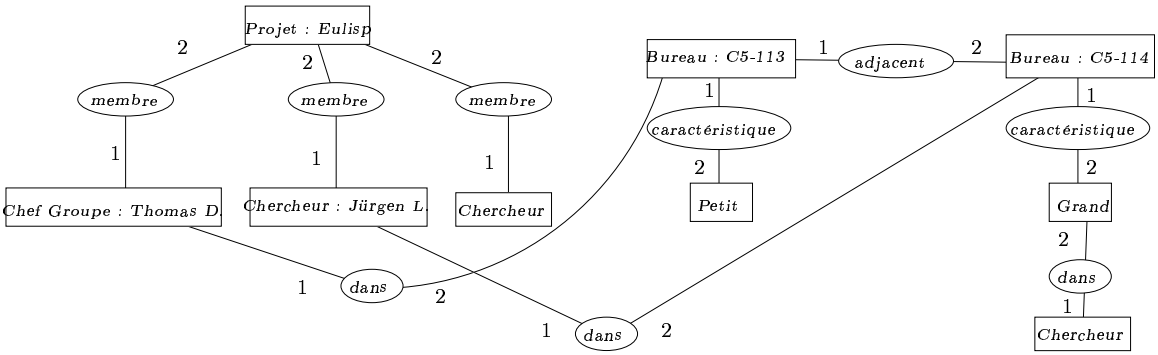
G_1 : structuration des projets



G_2 : informations géographiques



G_3 : affectation des locaux



Construction de la base de faits $\mathcal{G}$: union disjointe et mise sous forme normale

FIG. 2.2 – Construction de la base de faits à partir de plusieurs graphes.

Considérons une base de faits $\mathcal{G}$ composée de plusieurs graphes, par exemple les graphes G_1, G_2 et G_3 de la FIG. 2.2. Le graphe G_1 peut être lu : « le chef de groupe, Thomas D., le chercheur Jürgen L. et *un* chercheur sont membres du projet Eulisp » ; le graphe G_2 : « le petit bureau C5-113 est adjacent au grand bureau C5-114 » ; et le graphe G_3 : « le chef de groupe Thomas D. est dans le bureau C5-113, le chercheur Jürgen L. et un chercheur sont dans le bureau C5-114 ». La structuration des connaissances en plusieurs graphes distincts,

représentant différents types d'information, est naturelle. A la base de faits $\mathcal{G}$ composée des graphes G_1, G_2 et G_3, nous associons le graphe obtenu par la mise sous forme normale de l'union disjointe de ces graphes, comme indiqué dans la FIG. 2.2.

2.1.3 Projection d'une requête

Nous souhaitons maintenant répondre à la question « est-ce-que les connaissances représentées dans un graphe (requête) H sont présentes dans $\mathcal{G}$? » L'opération qui nous permet d'y répondre est un mécanisme d'appariement de graphes, une sorte d'homomorphisme de graphes appelé projection. Rappelons qu'un *homomorphisme de graphes* est une application qui associe à tous les sommets d'un graphe H des sommets d'un graphe G, en respectant la relation de voisinage sur H : si deux sommets sont voisins dans H, alors leurs images dans G sont voisines. Dans le cas de la projection, il s'agit d'associer, par une application π, à chaque sommet concept de H un sommet concept de G, et à chaque sommet relation de H un sommet relation de G, de façon à respecter à la fois l'ordre défini sur les étiquettes et la relation de voisinage du graphe.

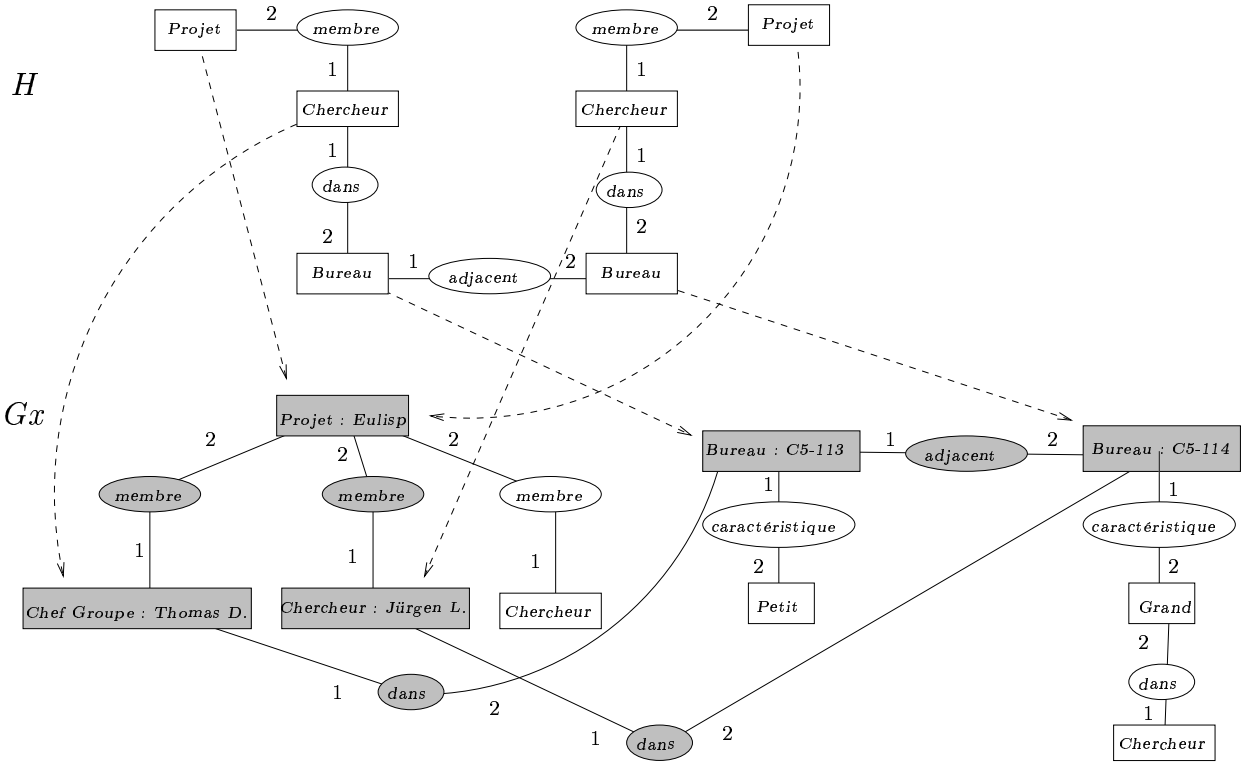


FIG. 2.3 – Projection d'une requête dans la base de faits $\mathcal{G}$.

Plus précisément, si c est un sommet concept de H, alors le type de $\pi(c)$ doit être plus spécifique que celui de c et, si c est un sommet concept individuel, $\pi(c)$ doit être un sommet concept individuel ayant le même marqueur que c. De plus, si r est une relation de H, alors

$\pi(r)$ doit avoir un type plus spécifique que celui de r, et, si c_i est le $i^{\text{ième}}$ argument de r dans H, alors son image $\pi(c_i)$ doit être le $i^{\text{ième}}$ argument de $\pi(r)$.

Nous avons illustré dans la FIG. 2.3 une projection d'une requête H dans la base de faits $\mathcal{G}$. Vu comme une requête, le graphe H peut s'interpréter par : « existe-t-il deux chercheurs[1], chacun membre d'un projet, dans des bureaux adjacents ? » Notons que nous n'avons représenté (par les flèches en pointillés) que la projection des sommets concepts, celle des sommets relations étant déterminée par la première. Vérifions qu'il s'agit bien d'une projection : nous pourrons, par exemple, vérifier que chacun des *[Projet]* a pour image *[Projet : Eulisp]*, dont l'étiquette est plus spécifique que la sienne, qu'un des *[Chercheur]* a pour image *[Chef Groupe : Thomas D.]*, dont l'étiquette est plus spécifique que la sienne, et que le sommet relation *membre* entre ces deux sommets a bien une image possible entre leurs deux images. Remarquons aussi dans cet exemple que la projection n'est pas nécessairement une application injective.

Le sous-graphe (partiel) de $\mathcal{G}$ représenté en grisé est celui engendré par les images par π des sommets de la requête. Il représente donc graphiquement une réponse à la question représentée par le graphe H.

Le problème de déduction dans le modèle $\mathcal{SG}$ se pose de la façon suivante :

$\mathcal{SG}$-DÉDUCTION
Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G})$, composée d'un support $\mathcal{S}$ et d'un ensemble de faits $\mathcal{G}$; et une requête H.
Question : Le graphe H est-il *déductible* (par $\mathcal{S}$) de $\mathcal{G}$, *i.e.* existe-t-il une projection de H dans $\mathcal{G}$?

2.2 Règles de graphes : $\mathcal{SR}$

Les règles sont des graphes permettant d'exprimer des connaissances de la forme « si ... alors ... ». Bien que la représentation que nous en donnons soit celle que nous avons adopté depuis [Baget, 1998], ces objets sont identiques (même mécanisme d'application, même sémantique) à ceux exposés dans [Salvat and Mugnier, 1996, Salvat, 1997].

2.2.1 Règles

Une règle sera représentée par un graphe coloré. La partie composée des sommets ayant la couleur 0 (plus précisément le sous-graphe engendré par ces sommets) sera appelée l'hypothèse de la règle, et l'autre partie, composée des sommets ayant la couleur 1 (qui, notons le, n'est pas un graphe conceptuel, car les voisins d'un sommet relation peuvent ne pas y figurer) sera appelée la conclusion. Ces règles sont un cas particulier des graphes

[1]En fait, nous ne précisons pas dans H que les deux sommets concepts de type *Chercheur* doivent représenter des personnes différentes. Mais, dans le contexte de SISYPHUS-I, un chercheur ne pouvant pas être « dans » deux bureaux différents, ces sommets auront forcément pour image des sommets représentant des entités distinctes.

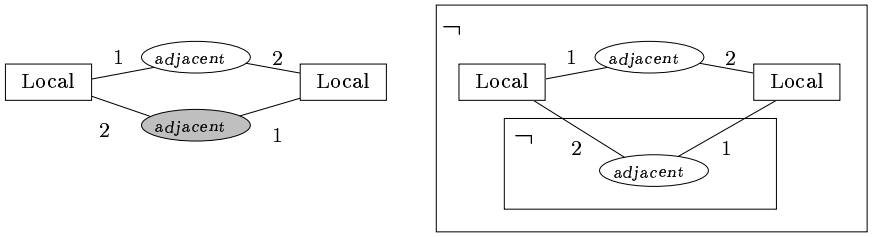


FIG. 2.4 – Un exemple de règle, et vue comme un graphe conceptuel de [Sowa, 1984].

conceptuels « généraux », correspondant à deux contextes négatifs imbriqués. La règle représentée dans la FIG. 2.4 exprime que « si un local x est adjacent à un local y, alors y est adjacent à x ». Les sommets de l'hypothèse ont été colorés en blanc, et ceux de la conclusion en gris. Nous avons dessiné à côté de cette règle un graphe conceptuel (au sens de [Sowa, 1984]) équivalent. Une boîte marquée du symbole ¬ entourant un graphe représente la négation de la formule logique associée à ce graphe : nous ne sommes déjà plus dans le paradigme graphes/opérations de graphes qui nous intéresse.

2.2.2 Application d'une règle

Une règle R sera dite applicable à un graphe G s'il existe une projection π de son hypothèse dans G. Dans ce cas, l'application de R à G suivant π produit un graphe G' (on le note $G' = \lambda(G, R, \pi)$) obtenu de la façon suivante :

- faire l'union disjointe de G et de la conclusion de R (ce qui ne nous donne pas encore un graphe conceptuel simple ;
- pour toute arête reliant un sommet c (nécessairement concept) de l'hypothèse à un sommet r de la conclusion de R, rattacher l'« arête incomplète » correspondante dans G' à $\pi(c)$.

Remarquons qu'il peut être nécessaire de remettre le graphe obtenu sous forme normale.

Nous notons qu'une règle dont l'hypothèse est le graphe vide peut toujours s'appliquer à un graphe (même si ce graphe est lui-même vide). Un graphe peut donc être considéré comme une règle dont l'hypothèse est vide. C'est pourquoi nous avons appelé $\mathcal{SR}$ plutôt que $\mathcal{SGR}$ le modèle de raisonnement obtenu en considérant les règles.

2.2.3 Le mécanisme de dérivation

Considérons maintenant une base de connaissances composée d'un support $\mathcal{S}$, d'un graphe (ou d'un ensemble de graphes) $\mathcal{G}$, et d'un ensemble de règles $\mathcal{R}$. Alors nous disons qu'un graphe G' est une $\mathcal{R}$-dérivation immédiate de $\mathcal{G}$ s'il existe une règle R de $\mathcal{R}$ et une projection π de l'hypothèse de $\mathcal{R}$ dans $\mathcal{G}$ telles que G' est la forme normale de $\lambda(\mathcal{G}, R, \pi)$. Un graphe G' est une $\mathcal{R}$-dérivation de $\mathcal{G}$ s'il existe une séquence de graphes $\mathcal{G} = G_0, G_1, \ldots, G_k$ telle que, pour $1 \leq i \leq k$, G_i est une $\mathcal{R}$-dérivation immédiate de G_{i-1}.

Nous pouvons maintenant définir le problème de déduction dans le modèle $\mathcal{SR}$:

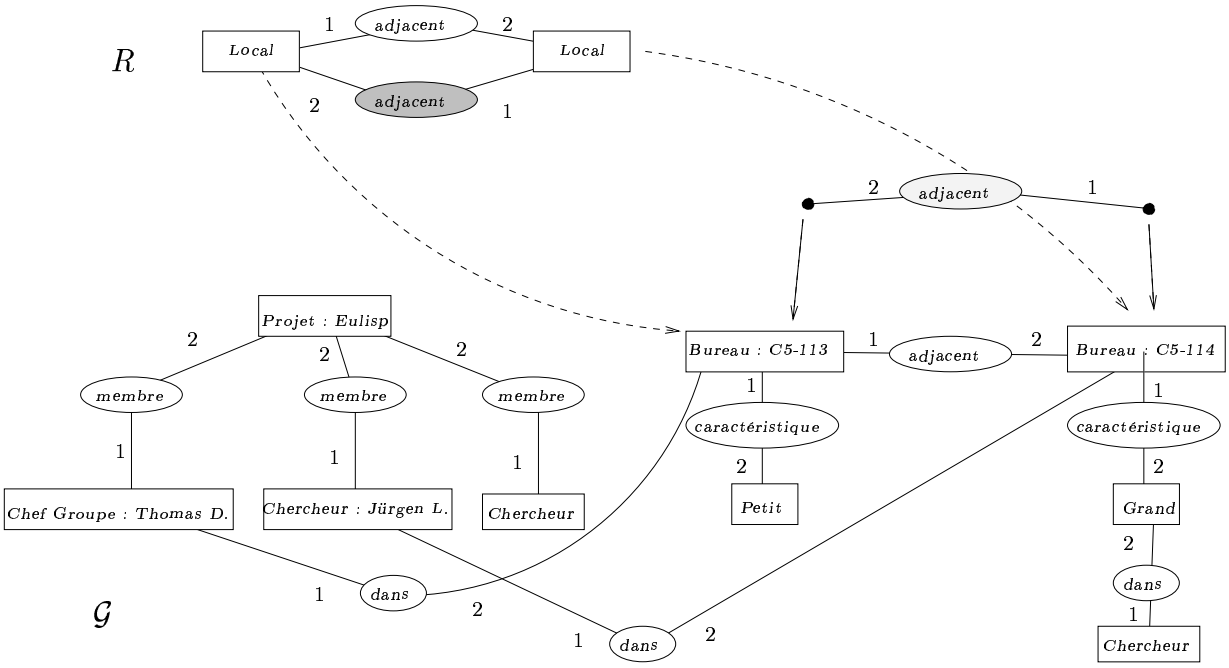


FIG. 2.5 – Application d'une règle à un graphe.

$\mathcal{SR}$-DÉDUCTION
Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G}, \mathcal{R})$, composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, et d'un ensemble de règles $\mathcal{R}$; et une requête H.
Question : Le graphe H est-il *déductible* (par $\mathcal{S}$) de $\mathcal{KB}$, *i.e.* existe-t-il une $\mathcal{R}$-dérivation G' de $\mathcal{G}$ telle que H se projette dans G' ?

Prenons par exemple la requête $H = [Bureau : C5\text{-}113] - 2 - (adjacent) - 1 - [Bureau]$ (« existe-t-il un bureau adjacent au bureau *C5-113* ? »). Cette requête est déductible de la base de faits $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \{R\})$ de la FIG. 2.5, grâce à une dérivation de longueur 1.

2.3 Contraintes et validité d'un graphe

Les contraintes sont des graphes utilisés pour porter un jugement sur la validité d'autres graphes (par exemple les faits du graphe $\mathcal{G}$). Comme les règles, nous les représentons par des graphes colorés. Si le modèle dans lequel nous les utilisons ne contient pas de règles, les contraintes servent alors simplement à effectuer une vérification de la validité d'une base de faits (et peuvent alors être vues comme une généralisation de la notion de signature). Des contraintes similaires sont utilisées à cet effet dans [Dibie et al., 1998]. Si nous les utilisons en conjonction avec des règles, alors la vérification de la validité devient un problème dynamique, à tester après chaque application de règle. Nous verrons deux façons différentes d'envisager ce problème.

2.3.1 Contraintes

Nous considérerons deux types de contraintes, positives et négatives, représentées, comme les règles, par des graphes colorés. La partie colorée par 0 (en blanc) des contraintes est appelée condition, la partie colorée par 1 (en gris) est appelée obligation dans le cas des contraintes positives, interdiction dans le cas des contraintes négatives.

Intuitivement, une contrainte positive peut s'exprimer par : « à chaque fois que *condition*, alors il faut aussi *obligation* », tandis qu'une contrainte négative s'exprime par : « à chaque fois que *condition*, alors il ne faut pas avoir aussi *interdiction* ».

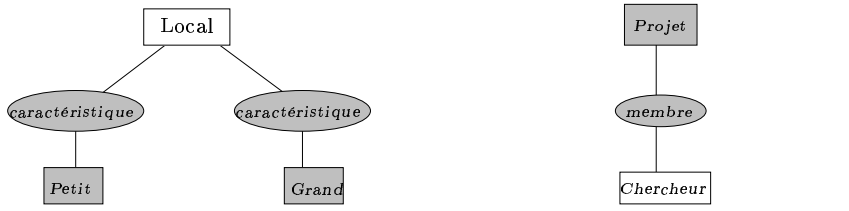
Un graphe $\mathcal{G}$ viole une contrainte positive s'il existe une projection de la condition de cette contrainte qui ne s'étend pas à une projection de la contrainte toute entière. Il viole une contrainte négative s'il existe une projection de la condition qui s'étend à une projection de la contrainte toute entière (autrement dit, s'il existe une projection de tout le graphe représentant la contrainte négative).

Si $\mathcal{C}$ est un ensemble de contraintes, positives ou négatives, alors un graphe $\mathcal{G}$ viole $\mathcal{C}$ s'il viole une des contraintes de $\mathcal{C}$. Dans le cas contraire, il est dit cohérent pour $\mathcal{C}$. Si nous n'avons pas de règles, $\mathcal{C}$ sera utilisé une et une seule fois pour vérifier la cohérence d'une base de faits. Une requête ne pourra être posée qu'à une base de faits cohérente, mais n'a pas besoin d'être elle-même cohérente.

Le problème de déduction se pose ici de la façon suivante :

$\mathcal{SGC}$-DÉDUCTION
Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$, composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, et d'un ensemble de contraintes $\mathcal{C}$; et une requête H.
Question : La base de faits $\mathcal{G}$ est-elle cohérente pour $\mathcal{C}$, et, si oui, H est-il déductible de $\mathcal{G}$?

La FIG. 2.6 représente deux contraintes, une contrainte négative, C^-, qui peut s'exprimer par « pour tout local, il ne faut pas qu'il soit à la fois grand et petit », et une contrainte positive, C^+, qui peut s'exprimer par « tout chercheur doit être membre d'un projet » Le graphe $\mathcal{G}$ de la FIG. 2.5 viole la contrainte C^+ et ne viole pas la contrainte C^-. Il n'est donc pas cohérent pour $\{C^+, C^-\}$. La base de faits devra donc être modifiée/réparée avant qu'une requête puisse lui être posée.



Une contrainte négative C^- Une contrainte positive C^+

FIG. 2.6 – Exemple de contraintes négatives et positives.

Nous verrons dans le Chap. 7 que cette définition pour la violation d'une contrainte nécessite quelques précautions d'emploi, afin d'éviter que deux graphes ayant la même sémantique soient jugés différemment quant à leur cohérence. Nous en resterons cependant á la définition ci-dessus dans le cadre de cette présentation intuitive.

2.3.2 Dérivations sous contraintes

Le problème se complique quelque peu lorsque l'on considère une base de connaissances comportant à la fois des règles et des contraintes. Nous avons envisagé deux façons d'interpréter les règles, qui déterminent trois problèmes très différents.

1. nous pouvons interpréter l'application d'une règle comme une transformation élémentaire, une transition, qui permet de passer d'un monde à un autre monde. Dans ce modèle, la base de faits $\mathcal{G}$ représente un monde initial, et les règles décrivent les évolutions possibles vers d'autres mondes. Nous appellerons ces règles « règles d'évolution », et désignerons un ensemble de telles règles par $\mathcal{E}$. Une requête H est alors dite déductible s'il existe une séquence de dérivations immédiates $\mathcal{G} = G_0, G_1, \ldots, G_k$ telle que tous les G_i soient cohérents pour $\mathcal{C}$, que G_k réponde à la requête H. Autrement dit, nous recherchons une suite de transformations qui font évoluer le monde de façon cohérente, jusqu'à un monde vérifiant la requête, sans s'intéresser à ce qui se passera par la suite ...

   $\mathcal{SEC}$-DÉDUCTION
   Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G}, \mathcal{E}, \mathcal{C})$, composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, d'un ensemble de règles d'évolution $\mathcal{E}$ et d'un ensemble de contraintes $\mathcal{C}$; et une requête H.
   Question : Existe-t-il une séquence de $\mathcal{E}$-dérivations immédiates $\mathcal{G} = G_0, G_1, \ldots, G_k$ telle que tous les G_i soient cohérents pour $\mathcal{C}$, et telle que H soit déductible de G_k ?

2. supposons maintenant que les règles soient des règles d'inférence « standard », qui encodent des connaissances implicites pour un monde unique, résultant de la base de faits $\mathcal{G}$ et de cet ensemble de règles (que nous appellerons règles d'inférence et dénoterons par $\mathcal{R}$). Nous nous étions restreint dans [Baget et al., 1999] au cas où $\mathcal{G}$ et $\mathcal{R}$ déterminent un graphe fini G', tel que pour toute requête H, H est déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$ si et seulement si H se projette dans G'. Dans ce cas, le monde défini par $\mathcal{G}$ et $\mathcal{R}$ est cohérent pour $\mathcal{C}$ si et seulement si ce graphe G' est cohérent pour $\mathcal{C}$. Une requête H est dite déductible de la base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{C})$ si et seulement si ce graphe G' est cohérent pour $\mathcal{C}$, et H se projette dans G'. Cependant, un tel graphe fini n'existe pas nécessairement dans le cas général, où nous devrons adopter une caractérisation plus complexe [Baget and Mugnier, 2001b] : toute violation de contrainte que l'on peut obtenir est restorable.

$\mathcal{SRC}$-DÉDUCTION
Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{C})$, composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, d'un ensemble de règles d'inférence $\mathcal{R}$ et d'un ensemble de contraintes $\mathcal{C}$; et une requête H.
Question : Le monde défini par $\mathcal{G}$ et $\mathcal{R}$ est-il cohérent pour $\mathcal{C}$, et, si oui, H est-il déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$?

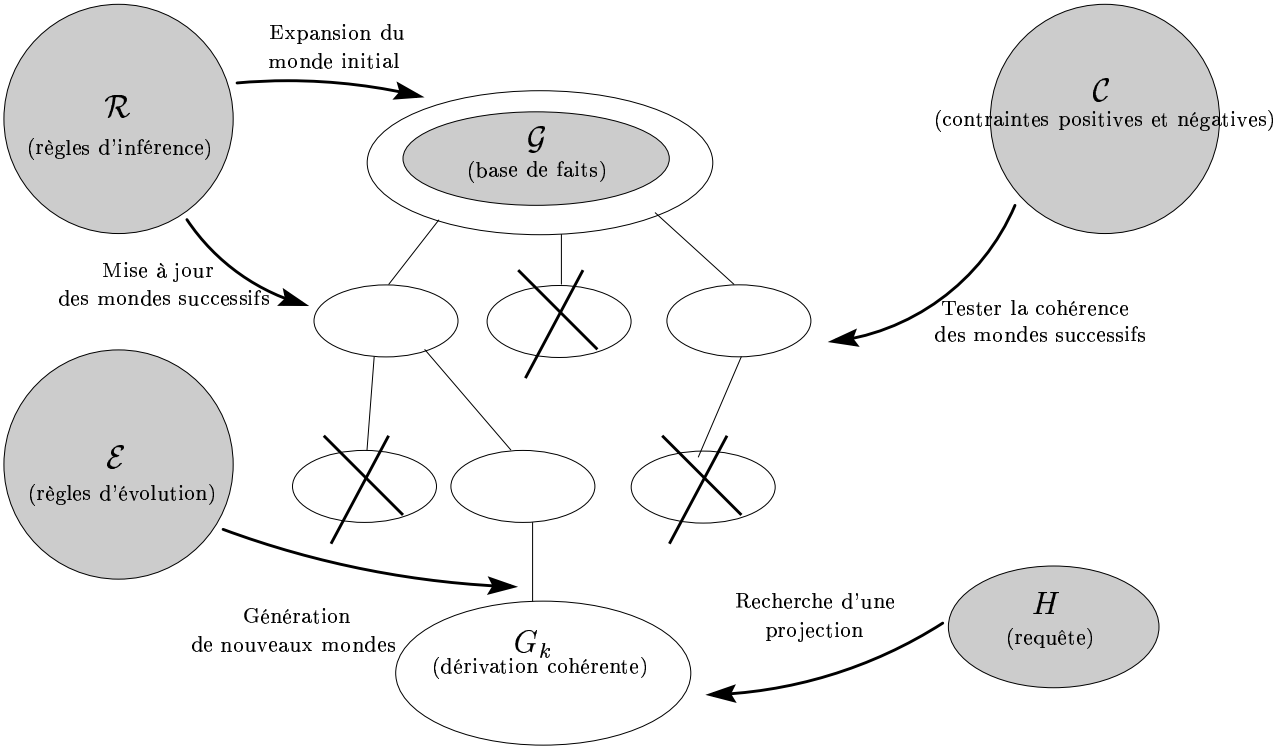


FIG. 2.7 – $\mathcal{SREC}$: le modèle de raisonnement le plus général de la famille $\mathcal{SG}$.

3. Enfin, nous pouvons considérer une base de connaissances qui contient à la fois des règles d'évolution et des règles d'inférence. Nous obtenons alors le modèle le plus général de la famille $\mathcal{SG}$. Dans ce cas $\mathcal{G}$ représente un monde initial. Les règles de $\mathcal{R}$ représentent les connaissances implicites sur le monde $\mathcal{G}$ et ses successeurs obtenus par application des règles d'évolution de $\mathcal{E}$. Les contraintes de $\mathcal{C}$ définissent la cohérence de ces mondes. La FIG. 2.7 illustre le mécanisme de raisonnement dans ce modèle. Le problème de déduction s'exprime de la même manière que dans le cas 1., mais les tests successifs de cohérence des mondes utilisent la caractérisation du cas 2.

$\mathcal{SREC}$-DÉDUCTION
Données : Une base de connaissances $\mathcal{KB} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{E}, \mathcal{C})$, composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, d'un ensemble de règles d'inférence $\mathcal{R}$, d'un ensemble de règles d'évolution $\mathcal{E}$, et d'un ensemble de contraintes $\mathcal{R}$; et une requête H.
Question : Existe-t-il une séquence de $\mathcal{R} \cup \mathcal{E}$-dérivations immédiates $\mathcal{G} = G_0, G_1, \ldots, G_k$ telle que, pour tout G_i obtenu après application d'une règle de $\mathcal{E}$ ($\mathcal{G}$ compris), le monde défini par G_i est $\mathcal{R}$ est cohérent pour $\mathcal{C}$ et et telle que H soit déductible de G_k ?

Dans le cas particulier de la modélisation du problème SISYPHUS-I (App. B), $\mathcal{G}$ est un ensemble de graphes qui décrivent les conditions initiales (la position et la taille des différents bureaux, les particularités des différentes personnes, l'organisation du groupe, ...) ; $\mathcal{R}$ est un ensemble de règles d'inférence (exprimant, par exemple, que la relation d'adjacence est symétrique) ; $\mathcal{C}$ décrit les contraintes d'allocation (par exemple, un fumeur et un non-fumeur ne doivent pas être dans le même bureau) ; et $\mathcal{E}$ contient une unique règle d'évolution, qui essaie de placer les personnes dans des bureaux. La requête H représente une situation dans laquelle toutes les personnes sont placées dans un bureau. Une réponse à cette requête est donc une situation dans laquelle toutes les personnes sont placées dans un bureau, obtenue à partir du monde initial en appliquant la règle d'évolution, tout en satisfaisant les contraintes.

2.3.3 La famille $\mathcal{SG}$

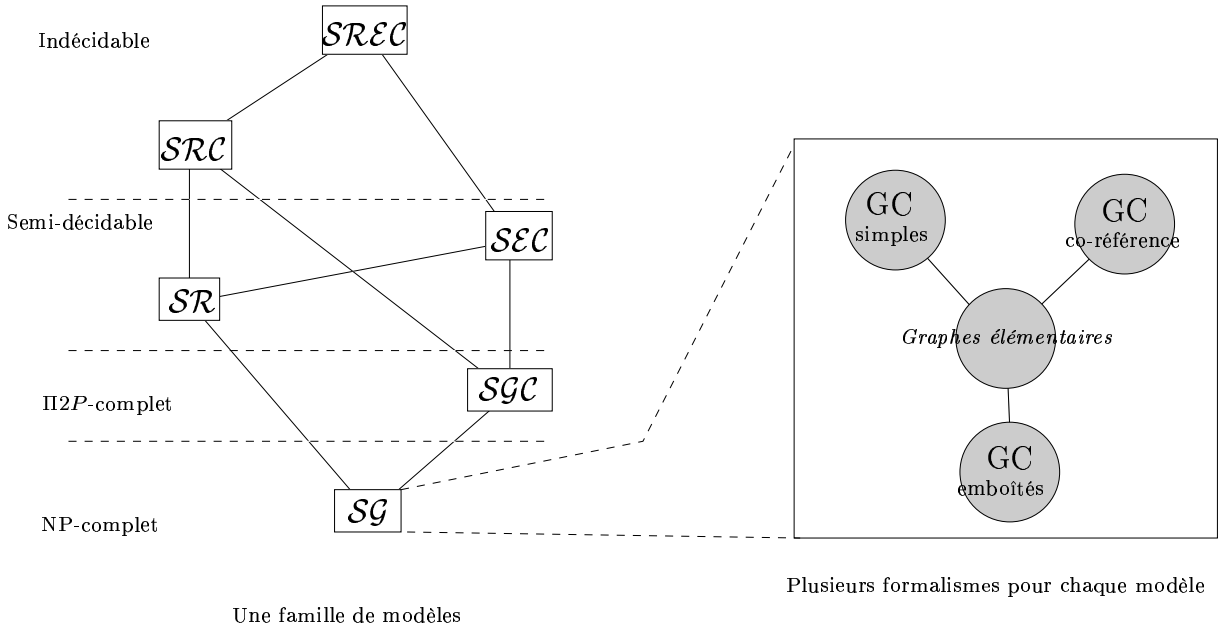


FIG. 2.8 – La famille $\mathcal{SG}$: modèles et complexité du problème de déduction.

Nous avons présenté dans ce chapitre différents ensembles d'objets représentés par des graphes ou des graphes colorés :

- un ensemble $\mathcal{G}$ de graphes constituant une base de faits ;
- un ensemble $\mathcal{C}$ de graphes colorés représentant des contraintes, positives ou négatives ;
- un ensemble $\mathcal{R}$ de graphes colorés représentant des règles d'inférence ;
- un ensemble $\mathcal{E}$ de graphes colorés représentant des règles d'évolution.

Les différents modèles de la famille $\mathcal{SG}$ sont définis par les types de graphes qui composent la base de connaissance $\mathcal{K}$:

- $\mathcal{SG}$ lorsque la base de connaissances ne comporte que des graphes faits ($\mathcal{K} = (\mathcal{S}, \mathcal{G})$) ;
- $\mathcal{SGC}$ lorsque la base de connaissances ne comporte que des graphes faits et des contraintes ($\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$) ;
- $\mathcal{SR}$ lorsque la base de connaissances ne comporte que des graphes faits et des règles ($\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{E})$; notons qu'en l'absence de contraintes, les règles d'inférence et d'évolution ont le même comportement, et qu'un fait est équivalent à une règle dont l'hypothèse est vide, d'où la notation) ;
- $\mathcal{SRC}$, $\mathcal{SEC}$ et $\mathcal{SREC}$ suivant le type des règles qui composent, avec les faits et les contraintes, la base de connaissances.

Ces différents modèles sont représentés dans la FIG. 2.8, par une hiérarchie basée sur la relation de généralisation entre ces modèles : $\mathcal{SG}$ est le modèle le plus spécifique de cette famille, et $\mathcal{SREC}$ le plus général. Nous avons aussi indiqué la classe de complexité ou l'indécidabilité du problème de déduction dans ces différents modèles.

Enfin, bien que nous ayons construit les différents objets qui déterminent les modèles de la famille $\mathcal{SG}$ à partir des graphes conceptuels simples de [Sowa, 1984], rappelons que nous aurions pu, de la même manière, partir d'autres extensions des graphes conceptuels simples qui sont les « graphes conceptuels simples avec liens de co-référence » ou les « graphes conceptuels emboîtés » de [Chein et al., 1998]. Nous montrerons que tous ces formalismes sont équivalents, et qu'ils peuvent être représentés dans un formalisme simplifié (qui peut être vu comme une structure de données permettant de coder ces différents graphes), que nous avons appelé celui des « graphes élémentaires ».

Première partie

Formalismes de $\mathcal{SG}$

Chapitre 3

Graphes élémentaires

Sommaire

Les graphes conceptuels simples constituent le noyau à partir duquel ont été élaborés la plupart des modèles se réclamant des graphes conceptuels. Il s'agit de plus du modèle de base pour la famille $\mathcal{SG}$ que nous étudions dans ce mémoire. De l'expressivité et de l'efficacité des raisonnements que nous pouvons effectuer dans ce modèle dépendent donc celles des raisonnements que nous définirons dans les modèles plus généraux de cette famille.

Nous présentons ici une version simplifiée des graphes conceptuels simples, ne prenant en compte ni les types de concepts, ni les marqueurs individuels. Ces *graphes élémentaires* sont définis comme des hypergraphes : les relations ne sont pas des sommets, comme il est d'usage dans les graphes conceptuels, mais des hyperarcs. Les définitions initiales en seront simplifiées d'autant, et nous insisterons, dans un chapitre ultérieur, sur les algorithmes plus efficaces qui découlent de cette vision n-aire.

Nous définissons d'abord formellement les objets que nous allons manipuler (le *support*, qui encode les connaissances ontologiques, et les graphes élémentaires, qui représentent des connaissances factuelles), et l'opération élémentaire de raisonnement, la *projection*.

La définition du *graphe minimal* (le plus petit graphe conservant *toutes* les projections) permet de s'affranchir des problèmes de notations liés aux hyperarcs multiples, en reléguant cette difficulté dans le support. Le *graphe irredondant*, lui, ne conserve que l'*existence* des projections : nous rappellerons quelques propriétés structurelles importantes de ces graphes. Enfin, nous donnerons à la projection une sémantique (au sens de [Kayser, 1997]), par l'intermédiaire d'un résultat d'équivalence entre la projection entre les graphes élémentaires et la déduction en logique du premier ordre, positive, conjonctive, et existentielle.

Tout au long de ce document, nous essaierons de présenter nos définitions de façon la moins restrictive possible. Cette volonté de généralisation a pour but d'établir de façon plus naturelle des ponts entre différents formalismes, de préciser l'utilité des restrictions données dans le modèle des graphes conceptuels, et facilitera, nous l'espérons, la tâche d'un lecteur qui souhaiterait adapter nos résultats à ses propres travaux.

3.1 Notions de base

Un graphe élémentaire est un hypergraphe multiple, orienté, dont les *hyperarcs* sont étiquetés par les éléments d'une structure appelée *support*. L'opération définie pour comparer deux graphes élémentaires (*i.e.* « est-ce-que l'information codée par le graphe H est contenue dans le graphe G ? ») est appelée *projection*. Il s'agit d'une sorte d'*homomorphisme* de graphe (plus précisément une extension de la notion d'homomorphisme aux hypergraphes multiples orientés que nous utilisons) qui préserve une *relation d'ordre*, imposée par le support, sur les étiquettes.

3.1.1 Le formalisme des graphes élémentaires

Support élémentaire

Le lecteur pourra se référer à l'appendice C pour quelques rappels sur les notions relatives aux ordres utilisés dans ce mémoire.

Un *support élémentaire* (ou *support*) est une structure qui encode à la fois les étiquettes (ou *types*) disponibles pour étiqueter les hyperarcs d'un graphe élémentaire, et une relation d'ordre sur ces étiquettes (ou plus généralement un préordre), qui traduit la relation de sous-typage. La définition de la projection que nous allons adopter rend impossible la comparaison d'hyperarcs d'arité différente (voir cependant la discussion dans [Wermelinger and Lopes, 1994] sur l'intérêt de recourir à une telle opération), nous pouvons donc, sans perte de généralité, établir une partition sur les types suivant leur arité.

Définition 3.1 (Support élémentaire) *Un* support élémentaire *(ou simplement* support*) est un tuple d'ensembles disjoints (dont l'un au moins n'est pas vide) munis d'un préordre* $\mathcal{S} = ((T_1, \leq_1), \ldots, (T_k, \leq_k))$. *Les éléments de* T_i *sont appelés* types de relations d'arité i.

Nous noterons, par abus de notation, $t \in \mathcal{S}$ si $t \in T_i$, et $|t| = i$ l'arité de ce type de

relation. De même, nous noterons $t \leq t'$ ssi $|t| = |t'| = i$ et $t \leq_i t'$; et nous dirons alors que t est un *sous-type* de t' (réciproquement t' est un *supertype* de t).

Cette définition est volontairement très large. En effet, bien que la plupart des auteurs dans la communauté « Graphes Conceptuels » considèrent des ensembles de types finis, certains envisagent des ensembles infinis. Originellement [Sowa, 1984], l'ensemble des types de concepts (ici, les types de relations unaires) était un treillis, et les autres types de relations étaient deux à deux incomparables. Le besoin d'ordonner tous les types de relations est apparu par la suite. L'équipe « Graphes Conceptuels » du LIRMM a d'abord envisagé des treillis [Chein and Mugnier, 1992], puis des relations d'ordre quelconques, mais admettant un maximum et un minimum [Mugnier and Chein, 1996], et enfin des relations d'ordre quelconques, où seul $\leq_1$ doit admettre un maximum [Mugnier, 2000]. Au cours de ce mémoire, nous étudierons en quoi ces différents choix influent sur le modèle. Toutefois, nous verrons rapidement que, pour associer à la projection des propriétés que nous jugeons fondamentale (la composition de deux projections est une projection, l'identité est une projection ...), il est nécessaire et suffisant de considérer des *préordres*. Certaines constructions intermédiaires que nous allons réaliser nous imposent de plus de considérer des préordres dans le cas général.

Les préordres $(T_i, \leq_i)$ pourront éventuellement être *calculés*. Cette façon de les définir, nécessaire dans le cas où T_i est infini, consiste à définir le préordre par la fonction qui sert à décider si $t \leq_i t'$. Par exemple, on peut avoir T_1 l'ensemble des entiers naturels, et $\leq_1$ définie par $n \leq_1 n'$ ssi n est un diviseur de n', et pour T_2 l'ensemble des mots sur un alphabet, $\leq_2$ peut être l'ordre lexicographique.

Lorsqu'ils portent sur des ensembles T_i finis, les préordres $(T_i, \leq_i)$ pourront être *donnés* par la fermeture réflexo-transitive d'une relation R, définie en extension. Lorsque ce sont des ordres, il sera suffisant d'énumérer leur *relation de couverture*.

Graphes élémentaires

Nous définissons ici les *graphes élémentaires*, étiquetés par les types appartenant à un support élémentaire donné $\mathcal{S}$. Nous noterons $\mathcal{G}^e(\mathcal{S})$ l'ensemble des graphes élémentaires définis sur $\mathcal{S}$.

Définition 3.2 (Graphes élémentaires) *Soit $\mathcal{S}$ un support élémentaire. Nous appelons* graphe élémentaire sur $\mathcal{S}$ *un quadruplet $G = (V, U, \tau, \gamma)$, où U et V sont deux ensembles finis disjoints, respectivement de* sommets concepts *et de* relations, *$\tau : U \to \mathcal{S}$ associe à chaque relation son* type, *et $\gamma : U \to V^+$ associe à toute relation dont le type est d'arité p un tuple de sommets concepts de taille p, son* voisinage.

Si $G = (V, U, \tau, \gamma)$ est un graphe élémentaire, nous noterons $V(G)$ et $U(G)$ ses ensembles de sommets concepts et de relations, quel que soit leur nom. Si $r \in U(G)$ est une relation, nous noterons $|r|$ l'arité de cette relation, qui est égale à l'arité $|\tau(r)|$ de son type (et à la taille de son voisinage $\gamma(r)$). Si $\gamma(r) = (x_1, \ldots, x_p)$, nous dirons que x_j est le $j^{\text{ième}}$ *argument* de la relation r, et nous noterons $\gamma_j(r) = x_j$. Si $y_1, \ldots, y_p$ sont des sommets

concepts arguments d'une même relation r, nous dirons qu'ils sont *voisins* par r. Nous noterons $\emptyset$ le *graphe élémentaire vide*.

Un sommet concept de G sera dit *utile* s'il est argument d'au moins une relation de G. Sinon c'est un sommet isolé, et il sera dit *inutile* . Pour ne pas surcharger les démonstrations avec des cas limites présentant peu d'intérêt, nous considérerons dans cette partie qu'un graphe élémentaire ne contient que des sommets concepts utiles. Nous verrons lors de l'étude du modèle $\mathcal{SR}$ que les définitions relatives aux règles peuvent nécessiter la prise en compte des sommets inutiles, mais nous oublions ceux-ci pour l'instant.

Notation prédicative

Considérons le graphe élémentaire G suivant, défini sur le support $\mathcal{S}$ de la FIG. 3.1 :

$$\begin{array}{rcl} U & = & \{x_1, x_2, x_3\} \\ V & = & \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\} \\ \tau & : & \tau(r_1) = t^1_1, \tau(r_2) = t^4_1, \tau(r_3) = t^2_1, \tau(r_4) = t^3_2, \tau(r_5) = \tau(r_6) = t^1_2, \tau(r_7) = t^2_2, \\ & & \tau(r_8) = t^3_1, \tau(r_9) = t^1_3 \\ \gamma & : & \gamma(r_1) = \gamma(r_3) = (x_1), \gamma(r_2) = (x_3), \gamma(r_4) = \gamma(r_5) = \gamma(r_6) = \gamma(r_7) = (x_1, x_2), \\ & & \gamma(r_8) = (x_2), \gamma(r_9) = (x_1, x_3, x_3) \end{array}$$

Comme nous ne nous intéressons qu'aux sommets concepts utiles, nous pouvons représenter G de façon équivalente par un *multiensemble* (une sorte d'ensemble dont les éléments peuvent avoir plusieurs occurrences) de couples $(\tau(r), \gamma(r))$. Ainsi, la relation r_1 pourra être représentée par la paire $(t^1_1, (x_1))$. Pour mettre d'avantage en valeur le type, nous pourrons aussi la noter $t^1_1(x_1)$. Avec cette *notation prédicative*, nous pourrons définir le même graphe élémentaire G, de façon équivalente au nom des relations et à la présence des sommets concepts utiles près, par :

$$G = [t^1_1(x_1), t^4_1(x_4), t^2_1(x_1), t^3_2(x_1, x_2), t^1_2(x_1, x_2), t^1_2(x_1, x_2), t^2_2(x_1, x_2), t^3_1(x_2), t^1_3(x_1, x_4, x_4)]$$

Un graphe élémentaire est donc un *hypergraphe* (les relations sont d'arité quelconque), *orienté* (l'ordre sur le voisinage des relations étant important, ces relations sont donc des *hyperarcs*), *multiple* (plusieurs relations distinctes peuvent avoir le même voisinage).

Projection

La projection est l'opération élémentaire de raisonnement dans les modèles basés sur les graphes conceptuels simples. Il s'agit d'une sorte d'homomorphisme de graphe respectant la relation d'ordre définie dans le support. Chercher une projection d'un graphe élémentaire H dans un graphe élémentaire G revient à poser la question : « est-ce que les connaissances représentées dans le graphe H sont présentes dans le graphe G ? »

Définition 3.3 (Projection de graphes élémentaires) *Soit $\mathcal{S}$ un support élémentaire, et G et H deux graphes élémentaires sur $\mathcal{S}$ (*i.e. $G, H \in \mathcal{G}^e(\mathcal{S})$*). Une application $\pi : V(H) \to V(G)$, qui associe à chaque sommet concept de H un sommet concept de G, est appelée* $\mathcal{S}$-projection *(ou* projection *quand il n'y a pas d'ambiguïté sur le support) de H dans G si et seulement si elle préserve la relation de voisinage (homomorphisme) et respecte la relation $\leq$ définie par $\mathcal{S}$, c'est à dire :*

$$\forall r \in U(H), \gamma(r) = (x_1, \ldots, x_p) \Rightarrow \exists r' \in U(G), \gamma(r') = (\pi(x_1), \ldots, \pi(x_p)) \text{ et } \tau(r') \leq \tau(r)$$

Remarquons que cette définition s'applique sans modification aux graphes comportant des sommets concepts inutiles, et que ceux-ci n'ont pas d'intérêt dans le cadre de la projection : un concept inutile de H pourra se projeter dans n'importe quel concept de G (la seule condition étant que $V(G)$ soit non vide), et un concept inutile de G ne pourra servir à la projection d'aucun concept utile de H.

Déduction dans le modèle $\mathcal{SG}$

Nous pouvons maintenant définir le problème de déduction dans le modèle $\mathcal{SG}$:

Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G})$ une base de connaissances composée d'un support $\mathcal{S}$ et d'un ensemble $\mathcal{G}$ de graphes élémentaires. Un graphe élémentaire H (la requête) est dit déductible de $\mathcal{K}$ s'il existe une $\mathcal{S}$-projection de H dans $\mathcal{G}$ (considéré comme l'union disjointe de ses éléments).

3.1.2 Représentations graphiques

Représentation graphique du support

Dans les travaux sur les graphes conceptuels, les $(T_i, \leq_i)$ sont généralement des ordres sur des ensembles finis donnés par leur relation de couverture. De façon classique, on représente graphiquement la relation de couverture d'un ordre fini par un *diagramme de Hasse* : les éléments de T_i sont représentés par des points, à côté desquels on inscrit le nom de ces éléments. Si t couvre t' (auquel cas on aura représenté t *au dessus* de t'), alors on dessine un trait entre t et t'.

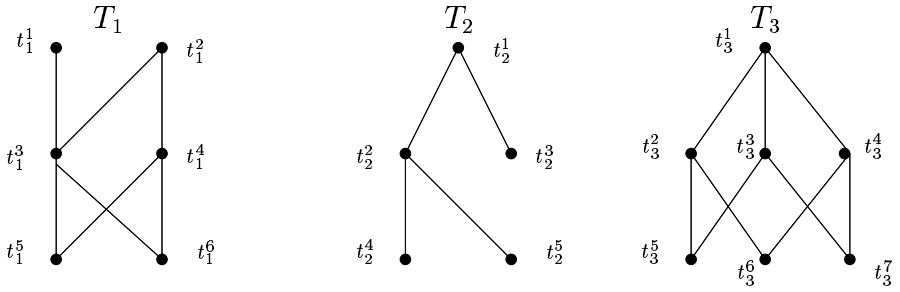


FIG. 3.1 – Diagrammes de Hasse définissant un support.

La relation t couvre t' se lit t' est un *sous-type* immédiat de t (et réciproquement, t est un *supertype* immédiat de t'). Ainsi, la relation d'ordre $\leq_1$ du support défini dans la FIG. 3.1 sera la fermeture réflexo-transitive de la relation $\leq'_1$ que l'on peut aussi donner par :

$$\begin{array}{lll} t_1^1 & \leq'_1 & \{\} \\ t_1^2 & \leq'_1 & \{\} \\ t_1^3 & \leq'_1 & \{t_1^1\} \\ t_1^4 & \leq'_1 & \{t_1^2\} \\ t_1^5 & \leq'_1 & \{t_1^3, t_1^4\} \\ t_1^6 & \leq'_1 & \{t_1^3, t_1^4\} \end{array}$$

Pour savoir si $t \leq_i t'$, il suffit, dans le dessin de T_i, de partir du sommet t et de remonter dans le graphe en suivant les traits, jusqu'à ce qu'on ait trouvé t' (auquel cas $t \leq_i t'$), ou qu'on ait épuisé toutes les possibilités. Il s'agit du principe du mécanisme de *lookup*, bien connu dans les langages à objets. On peut en déduire, par exemple, que $t_1^6 \leq_1 t_1^1$, mais $t_2^5 \not\leq_2 t_2^3$. Autrement dit, t_1^6 est un *sous-type* de t_1^1 (ou t_1^1 est un *supertype* de t_1^6), mais t_2^5 n'est pas un sous-type de $2t_2^3$. Dans le pire des cas, ce mécanisme devra examiner tous les traits de la représentation de la hiérarchie.

Représentation graphique standard d'un graphe élémentaire

À un hypergraphe est naturellement associé un graphe biparti (son *biparti d'incidence*), les hyperarêtes devenant l'une des deux classes de sommets : c'est ainsi que nous passerons, dans le chapitre suivant, des graphes élémentaires aux "graphes très simples". Nous avons choisi de représenter graphiquement un graphe élémentaire de la même façon que le graphe biparti associé, retrouvant le dessin classique d'un graphe conceptuel.

Ainsi, on dessine chaque sommet concept utile par un rectangle, et chaque relation en écrivant son type et en l'encadrant par un ovale. Puis, pour chaque relation r, pour chaque sommet concept x tel que $x = \gamma_i(r)$ (x est le $i^{\text{ième}}$ argument de r), nous dessinons un trait entre le rectangle représentant x et l'ovale représentant r, et inscrivons le nombre i à côté de ce trait. La position des rectangles et des ovales, comme la façon (droite, courbe,...) de dessiner les traits n'ont formellement aucune importance : il ne faut cependant pas négliger leur impact sur la *lisibilité* du graphe.

La FIG. 3.2 est la représentation graphique du graphe élémentaire G défini précédemment. Nous avons inscrit, pour que le lecteur puisse les identifier, les noms des sommets concepts et des relations à côté des ovales, mais il ne s'agit que d'une information extérieure à la représentation. Comme pour la notation prédicative, la représentation graphique de G définit exactement le même graphe élémentaire, au nom des sommets concepts et des relations près. Nous parlerons maintenant indifféremment du graphe élémentaire G, de sa notation prédicative et de sa représentation graphique, qui sont des notations équivalentes pour le même objet. Au cours de ce mémoire, nous privilégierons la notation graphique.

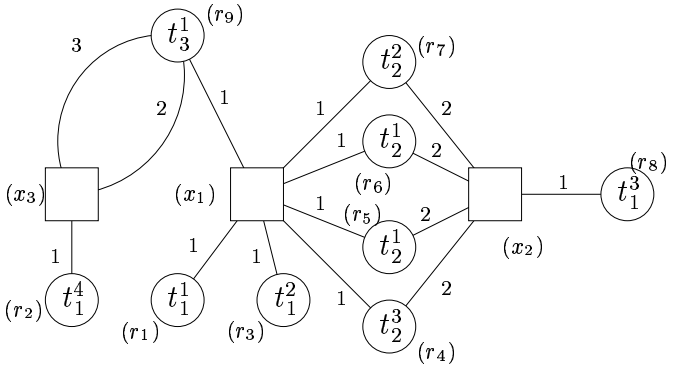


FIG. 3.2 – Une représentation graphique (standard) du graphe élémentaire G.

Représentation graphique compacte d'un graphe élémentaire

Cette représentation graphique (que nous appellerons *représentation standard*) est un peu lourde, et nous pourrons associer à un graphe sa *représentation compacte* . Considérons l'ensemble des relations d'un graphe élémentaire G. Nous pouvons définir une classe d'équivalence $\doteq$ sur $U(G)$, déterminée par $r \doteq r'$ ssi $\gamma(r) = \gamma(r')$. Les relations d'une telle classe d'équivalence sont dites *jumelles*. Soit $R = \{r_1, \ldots, r_p\}$ une de ces classes. Alors nous notons $\tau(R)$ le *multiensemble* composé de $\{\tau(r_1), \ldots, \tau(r_p)\}$, et $\gamma(R) = \gamma(r_1) = \cdots = \gamma(r_p)$. Nous représenterons donc chacun des sommets concepts de G par un rectangle, chacune des classes d'équivalence R par un ovale entourant l'inscription de $\tau(R)$, et pour chaque sommet concept x tel que $x = \gamma_i(R)$, nous rajouterons un trait entre le rectangle représentant x et l'ovale représentant R, à côté duquel nous inscrirons le nombre i.

Cette représentation est illustrée par la FIG. 3.3, où la représentation compacte du graphe élémentaire G de la FIG. 3.2 bénéficie d'un raffinement supplémentaire : si R est une classe d'équivalence de relations d'arité 1, alors nous pouvons « oublier » de dessiner l'ovale représentant R, et inscrire $\tau(R)$ à l'intérieur du rectangle représentant l'argument de R. Le dessin en est ainsi simplifié, mais nous obligeons le lecteur à penser « hyperarc étiqueté d'arité 1 » quand il voit un « sommet étiqueté » .

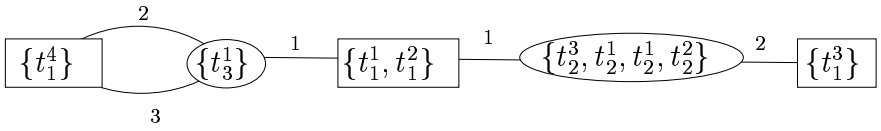


FIG. 3.3 – La représentation compacte du graphe élémentaire G.

Remarquons que cette représentation est aussi celle d'un autre graphe élémentaire, dont les types de relations sont des ensembles de types. Ce codage nous permet de parler de *la* relation R telle que $\gamma(r) = (x_1, \ldots, x_p)$, et rendra notre discours plus clair. Alors que la représentation graphique standard est celle d'un *hypergraphe multiple orienté*, la représentation compacte est celle d'un *hypergraphe orienté*, plus facilement manipulable :

nous y reviendrons lorsque nous parlerons des graphes élémentaires minimaux.

Représentations alternatives, dans le cas binaire

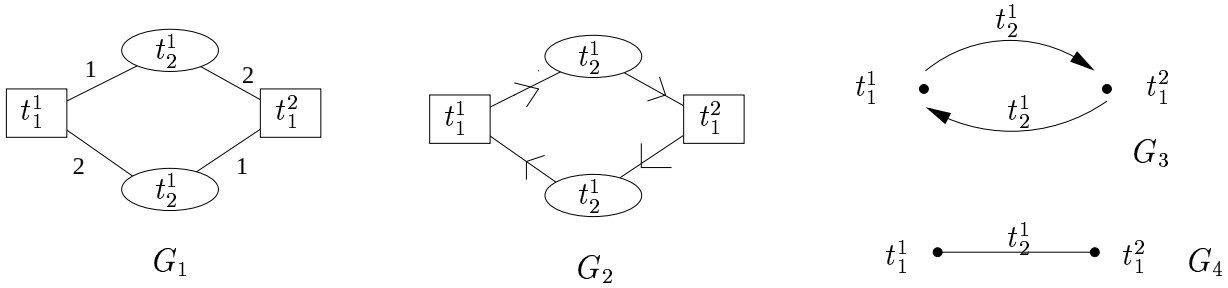


FIG. 3.4 – Quelques représentations équivalentes pour les relations unaires et binaires.

Notons que nous utiliserons parfois des représentations encore plus simplifiées, comme dans la FIG. 3.4, quand les types de relations utilisés sont tous unaires ou binaires. Si une relation est binaire, on peut avantageusement remplacer les traits numérotés indiquant l'ordre de ses arguments par des flèches (voir le graphe G_2, une flèche du sommet concept vers la relation à la place du trait indiqué 1, et une flèche de la relation vers le sommet concept à la place du trait 2). Nous pouvons aller plus loin, en ne dessinant plus que l'arc allant du premier sommet concept au second, l'arc étant étiqueté par le type de la relation. Nous pouvons aussi dans ce cas nous débarrasser des rectangles (graphe G_3). Nous obtenons ainsi une représentation habituelle d'un *graphe orienté*, dont les sommets comme les arcs sont étiquetés. Enfin, deux arcs (x, y) et (y, x) étiquetés par un même type peuvent être remplacés par une unique *arête*, notée xy (graphe G_4). Nous considérerons donc les graphes G_1, G_2, G_3 et G_4 de la FIG. 3.4 comme des représentations équivalentes du même graphe élémentaire.

3.1.3 Une première analyse de la projection

Nous commençons par donner quelques propriétés immédiates de la projection : elle induit une relation de préordre (qui n'est pas un ordre) sur l'ensemble des graphes élémentaires, c'est même un morphisme au sens de la théorie des catégories. Nous donnons ensuite un exemple de projection d'un graphe dans un autre, qui nous servira d'introduction à la notion d'image homomorphe d'un graphe. Enfin, nous définirons, à partir de la projection, la notion d'isomorphisme de graphes élémentaires.

La relation de subsomption est un préordre

Nous noterons $G \preceq_{\mathcal{S}} H$ lorsqu'il existe une $\mathcal{S}$-projection de H dans G (ou simplement $G \preceq H$ lorsqu'il n'y a aucune ambiguïté sur le support). Et nous dirons que H *subsume* G. La notation $\preceq_{\mathcal{S}}$ se justifie par le fait que la relation de subsomption est un *préordre* sur $\mathcal{G}^e(\mathcal{S})$.

Propriété 3.1 ($\preceq$ est un préordre) *La relation $\preceq_{\mathcal{S}}$ est un préordre sur $\mathcal{G}^e(\mathcal{S})$ si et seulement si chaque relation $\leq_i$ est un préordre sur T_i.*

Preuve: Supposons que chaque relation $\leq_i$ est un préordre sur T_i (*i.e.* $\leq_i$ est réflexive est transitive). Alors :

1. $\preceq_{\mathcal{S}}$ est réflexive (il existe au moins une projection, l'identité notée *Id*, de tout graphe dans lui-même, de par la réflexivité des $\leq_i$) ;
2. $\preceq_{\mathcal{S}}$ est transitive (si π est une projection de G_1 dans G_2 et π' est une projection de G_2 dans G_3, alors $\pi' \circ \pi$ est une projection de G_1 dans G_3, par transitivité des $\leq_i$).

L'autre sens de l'équivalence est également immédiat. Supposons qu'un des $\leq_i$ ne soit pas un préordre. Alors :

1. soit $\leq_i$ n'est pas réflexive, donc il existe $t_i^j \in T_i$ tel que $t_i^j \not\leq_i t_i^j$, et alors il n'y a aucune projection de $G = \{t_i^j(x_1, \ldots, x_i)\}$ dans lui-même ;
2. soit $\leq_i$ n'est pas transitive, donc il existe $t_i^{j_1}, t_i^{j_2}, t_i^{j_3} \in T_i$ tels que $t_i^{j_1} \leq_i t_i^{j_2}$, $t_i^{j_2} \leq_i t_i^{j_3}$, et $t_i^{j_1} \not\leq_i t_i^{j_3}$. Considérons alors le graphe $G_1 = \{t_i^{j_1}(x_1, \ldots, x_i)\}$, $G_2 = \{t_i^{j_2}(x_1, \ldots, x_i)\}$ et $G_3 = \{t_i^{j_3}(x_1, \ldots, x_i)\}$. Alors $G_1 \preceq_{\mathcal{S}} G_2$, $G_2 \preceq_{\mathcal{S}} G_3$, mais $G_1 \not\preceq_{\mathcal{S}} G_3$.

Ce qui suffit à prouver que $\preceq_{\mathcal{S}}$ n'est pas un préordre. □

Cette propriété et sa preuve nous montrent que l'identité est une projection, et que la composition de deux projections est une projection, deux propriétés qui seront souvent utilisées dans ce mémoire. La définition que nous avons donnée du support est donc juste suffisante pour assurer ces deux propriétés que nous jugeons essentielles. De façon générale, en représentation de connaissances, elles peuvent s'exprimer par « toute information $\mathcal{I}$ est contenue dans elle-même » et par « si $\mathcal{J}$ contient l'information $\mathcal{I}$, et $\mathcal{K}$ contient l'information $\mathcal{J}$, alors $\mathcal{K}$ contient l'information $\mathcal{I}$ ». La transitivité peut cependant ne pas être jugée souhaitable. Par exemple, en recherche documentaire, [Genest, 2000] définit une « projection approchée » qui n'est pas nécessairement transitive. « Si $\mathcal{J}$ ressemble à $\mathcal{I}$, et $\mathcal{K}$ ressemble à $\mathcal{J}$, alors $\mathcal{K}$ ne ressemble pas nécessairement à $\mathcal{I}$ ». Dans ce formalisme, la non-transitivité de $\preceq_{\mathcal{S}}$ tient autant à la non-transitivité des $\leq_1$ (c'est à dire l'ordre sur les types de concepts dans ce formalisme) qu'aux approximations sur la structure des graphes.

La projection est un morphisme, au sens de la théorie des catégories

De plus, cette propriété s'étend immédiatement à une propriété plus forte, qui associe l'ensemble des projections à une catégorie (voir, par exemple, [Pultr and Trnková, 1980] pour une présentation « combinatoire » des catégories). Ce résultat n'est pas surprenant, les homomorphismes de graphes étant un des exemples les plus connus de catégorie, puisqu'ils les ont inspirées.

Propriété 3.2 (Projection et catégories) *Soit $\mathcal{S}$ un support. Nous notons $\Pi^e(\mathcal{S})$ l'ensemble des projections sur $\mathcal{G}^e(\mathcal{S})$, et si $\pi : H \to G$ est une de ces projections, nous notons $cod(\pi) = H$ et $dom(\pi) = G$. Notons Id l'application qui associe à tout graphe élémentaire G l'identité sur G. Notons $\circ$ l'opérateur usuel de composition.*

Alors $\mathfrak{C}(\mathcal{S}) = (\mathcal{G}^e(\mathcal{S}), \Pi^e(\mathcal{S}), cod, dom, Id, \circ)$ *forme une* catégorie *si et seulement si chaque* $\leq_i$ *est un préordre sur* T_i.

Preuve: Nous avons déjà prouvé qu'on pouvait associer la projection identité à tout graphe et que la composition de deux projections était une projection si et seulement si tous les $(T_i, \leq_i)$ étaient des préordres. Il ne reste plus qu'à prouver que $\circ$ est associative, et que Id est un neutre pour la composition, ce qui est immédiat par utilisation de l'opérateur usuel de composition. □

Graphes élémentaires équivalents

Une projection π de H dans G est *injective* si, pour tout couple x, y de sommets concepts distincts de H, $\pi(x), \pi(y)$ sont des sommets concepts distincts de G. Une projection est dite *surjective* si à tout sommet concept y de G correspond au moins un sommet concept x de H tel que $y = \pi(x)$. Une projection est dite *bijective* si elle est à la fois injective et surjective.

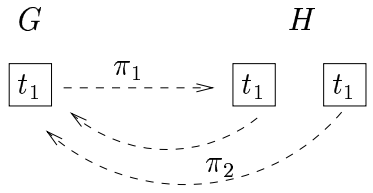


FIG. 3.5 – Un exemple de deux graphes équivalents, non isomorphes.

Nous dirons que G et H sont équivalents, et noterons $G \cong_{\mathcal{S}} H$ (ou $G \cong H$ lorsqu'il n'y a pas d'ambiguïté sur le support) si $G \preceq_{\mathcal{S}} H$ et $H \preceq_{\mathcal{S}} G$. Contrairement à ce qui est affirmé dans [Sowa, 1984], [Jackman, 1988, Chein and Mugnier, 1992] remarquent que $\preceq_{\mathcal{S}}$ n'est pas un ordre. En effet, deux graphes élémentaires peuvent être équivalents sans être isomorphes[1] (prenons par exemple

$$G = [t_1(x)]$$

et

$$H = [t_1(x), t_1(y)]$$

, voir FIG. 3.5, où les projections sont représentées par les arcs en pointillés). Le non respect de l'antisymétrie est donc inhérent à la structure de l'hypergraphe, et n'est pas induit par les propriétés du support. En d'autres termes :

Propriété 3.3 ($\preceq$ n'est pas un ordre) *Quel que soit le support* $\mathcal{S}$, $\preceq_{\mathcal{S}}$ *n'est pas un ordre sur* $\mathcal{G}^e(\mathcal{S})$.

[1]Notion à laquelle nous garderons pour l'instant une signification intuitive : ce sont les mêmes graphes à un renommage près de leurs sommets concepts et de leurs relations.

Un exemple de projection

A titre d'exemple, considérons le support $\mathcal{S}$ illustré dans la figure FIG. 3.1 et les deux graphes élémentaires $G, H \in \mathcal{G}^e(\mathcal{S})$ illustrés dans la FIG. 3.6 suivante :

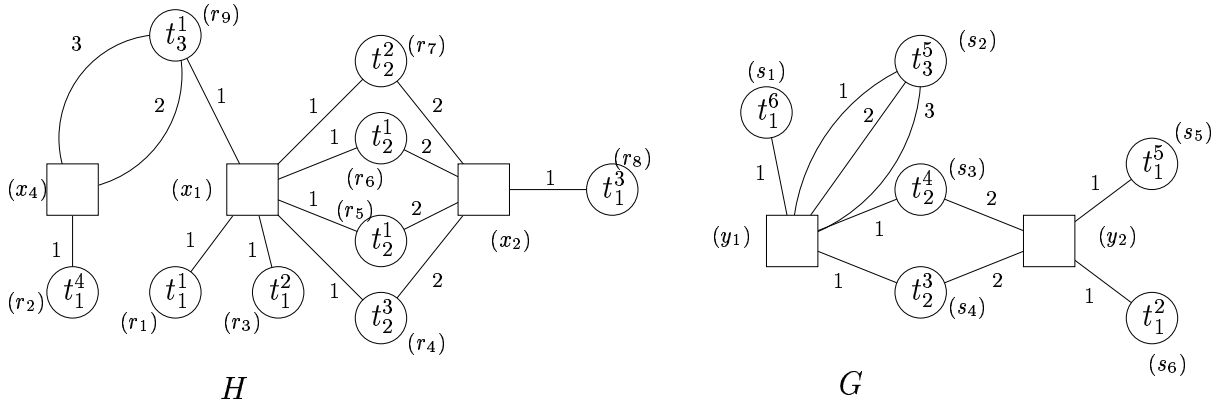


FIG. 3.6 – Recherche d'une projection de H dans G.

Nous allons montrer que l'application π qui associe aux sommets concepts x_4 et x_1 de H le sommet concept y_1 de G et au sommet concept x_2 de H le sommet concept y_2 de G est bien une projection de H dans G. Pour chaque relation r de H, nous devrons donc exhiber *une* relation r' de G (le *certificat* de r pour π) dont les arguments sont les images des arguments de r, et dont le type est un sous-type de celui de r. Le tableau suivant liste l'ensemble des certificats de chaque relation de H pour la projection π.

Relations de H	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
Certificats dans G	$\{s_1\}$	$\{s_1\}$	$\{s_1\}$	$\{s_4\}$	$\{s_3, s_4\}$	$\{s_3, s_4\}$	$\{s_3\}$	$\{s_5\}$	$\{s_2\}$

Nous pouvons vérifier, par exemple, que s_1 est bien un certificat de r_1 pour π (voir que $\tau(s_1) = t_1^6 \leq t_1^1 = \tau(r_1)$, que $\gamma(r_1) = (x_1)$ et que $\gamma(s_1) = (y_1) = (\pi(x_1))$). Cet exemple illustre aussi deux points qu'il est nécessaire de souligner :

1. la projection n'est pas une application injective (x_4 et x_1 ont la même image) ;
2. une relation peut avoir plusieurs certificats distincts (r_5 a deux certificats).

Image homomorphe pour une projection

Définition 3.4 (Image homomorphe) *Si r est une relation de H, et π une projection de H dans G, nous notons $\pi_U(r)$ l'ensemble des relations de G qui sont des certificats de r pour π. Une* image homomorphe *de H dans G pour π est un* sous-graphe partiel [2]

[2] Un *sous-graphe* G' d'un graphe élémentaire G est déterminé par un ensemble $V' \subseteq V(G)$ de sommets concepts. On l'appelle le sous-graphe *engendré* par V'. $U(G')$ est alors égal à l'ensemble des relations de G dont tous les arguments sont dans V'. Un *sous-graphe partiel* est obtenu en ôtant un certain nombre de relations (éventuellement aucune) d'un sous-graphe.

de G dont les sommets concepts sont les images par π des sommets concepts de H et dont les relations sont obtenues en choisissant une *relation de $\pi_U(r)$, pour chaque relation $r \in U(H)$.*

Remarquons qu'il peut y avoir un nombre exponentiel de façons de faire le choix d'un certificat pour chaque relation de H : plus exactement, si $U(H) = \{r_1, \ldots, r_m\}$,il y a $|\pi_U(r_1)| \times \cdots \times |\pi_U(r_m)|$ façons possibles de faire ce choix. Si nous reprenons l'exemple de la FIG. 3.6, les quatre façons possibles de faire notre choix correspondent cependant toutes à la même image homomorphe, qui est le graphe dessiné en grisé dans la FIG. 3.7.

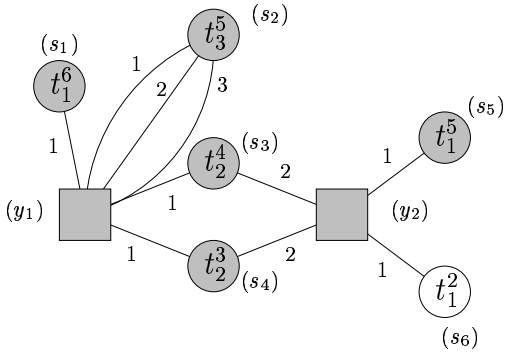


FIG. 3.7 – Image homomorphe de H dans G pour π.

Discussion

L'image homomorphe est d'un grand intérêt d'un point de vue *interface utilisateur* : il s'agit de la représentation graphique de la réponse à une requête. La donnée de l'image homomorphe ne remplace cependant pas la projection, puisqu'une même image homomorphe peut être obtenue à partir de plusieurs projections. Soit G le graphe élémentaire dont la notation prédicative est $[t(x_1, x_2), t(x_2, x_1)]$. Considérons les deux projections de G dans lui-même : $\pi_1 : x_1 \mapsto x_1,\ x_2 \mapsto x_2$, et $\pi_2 : x_1 \mapsto x_2,\ x_2 \mapsto x_1$. Ces deux projections (distinctes) induisent la même image homomorphe, qui est G tout entier. L'image homomorphe sera donc considérée comme un complément, au niveau interface, d'une projection qui l'a induite.

Ceci amène un nouveau problème : quelle image homomorphe présenter à l'utilisateur ? Nous pouvons construire des graphes H et G tels qu'à une projection de H dans G correspond un nombre exponentiel d'images homomorphes distinctes (même à un isomorphisme près). Soient les graphes élémentaires $H = \{t_2(x_0, x_1), t_2(x_1, x_2), \ldots, t_2(x_{p-1}, x_p)\}$ et $G = \{t_2^1(y_0, y_1), \ldots, t_2^k(y_0, y_1), t_2^1(y_1, y_2), \ldots t_2^k(y_1, y_2), \ldots\ldots, t_2^1(y_{p-1}, y_p), \ldots, t_2^k(y_{p-1}, y_p)\}$, contenant respectivement p et $p \times k$ relations, tous deux définis sur un support $\mathcal{S}$ tel que chaque t_2^i est inférieur à t_2. En dessinant ce graphe, on voit qu'il n'y a qu'une seule projection π de H dans G qui associe à chaque x_i le sommet concept y_i. Pour chaque relation r de H, $|\pi_U(r)| = k$, et aux k^p choix possibles correspondent k^p images homomorphes distinctes.

Ce problème du nombre des images homomorphes associées à une projection est cependant un faux problème, engendré par une inflation « artificielle » du nombre de relations. D'une part, dans un contexte de représentation de connaissances, les relations jumelles devraient être assez rares, bien qu'on se doive de les prendre en compte. D'autre part, ce problème peut être éliminé en considérant la représentation compacte des graphes.

Isomorphisme

Considérons π une projection de H dans G, et *choix* une application de $U(H)$ dans $U(G)$ qui à toute relation r associe une relation $r' \in \pi_U(r)$.

Une telle application est dite *fidèle* si, pour toute relation r' appartenant au sous-graphe G' engendré par les sommets concepts de $\pi(V(H))$, il existe $r \in U(H)$ telle que $r' = choix(r)$. Une projection π sera dite *fidèle* s'il existe un choix fidèle pour π. Un choix fidèle est dit *complet* si, de plus, pour toute paire r, r' de relations distinctes de $U(H)$, $choix(r) \neq choix(r')$. [3] Une projection π sera dite *complète* s'il existe un choix complet pour π. Un choix est dit *conservateur* si, pour toute relation r, $\tau(r) \equiv \tau(choix(r))$. Si de plus, $\tau(r) = \tau(choix(r))$ (ce qui sera toujours le cas si $\leq$ est un ordre), ce choix sera dit *fortement conservateur*. Remarquons que, si $\leq$ est un ordre, tout choix conservateur est fortement conservateur. Une projection π de H dans G est un *isomorphisme* si π est bijective et s'il existe un choix complet et fortement conservateur pour π. Deux graphes G et G' seront dit *isomorphes* s'il existe un isomorphisme de G dans G'. Cette définition correspond bien à l'idée intuitive de l'isomorphisme : G et G' sont les mêmes graphes, à un renommage près des sommets concepts et des relations. Nous noterons donc $G = G'$.

3.2 Graphes minimaux et irredondants

Nous avons vu que la multiplicité des hyperarcs dans les graphes élémentaires, quoique nécessaire d'un point de vue représentation de connaissances, induisait des problèmes de notation. Or nous avons proposé une représentation graphique alternative des graphes élémentaires, la *représentation compacte*, où un tuple de sommets concepts ne détermine au maximum qu'un hyperarc (*i.e.* γ^{-1} est injective). Une modification du support nous permettra de définir des graphes élémentaires ayant cette propriété (appelés *graphes compressés*), et nous comparerons les projections dans les différents modèles obtenus. Nous définirons ensuite la notion de *graphe minimal* associé à un graphe élémentaire (ce sera un graphe compressé particulier), qui a la propriété d'être le plus petit graphe équivalent tel que toutes les projections d'un graphe élémentaire à un autre sont conservées lorsque l'on passe à leurs minimaux associés. Nous définirons également la notion de *graphe irredondant* associé à un graphe élémentaire, qui est le plus petit graphe équivalent conservant cette

[3] Un choix fidèle est une application surjective dans les relations du sous-graphe de G engendré par $\pi(V(H))$. Un choix complet est une application bijective dans ce même ensemble. Les termes *fidèle* et *complet* sont les traductions de *faithful* et *full*, traditionnellement utilisés pour les homomorphismes de graphes (voir, par exemple, [Hahn and Tardif, 1997]), et que nous avons étendu à nos hyperarcs multiples.

fois l'existence de projections (et non pas toutes les projections) . Enfin, nous verrons que la relation de subsomption, qui est un préordre sur l'ensemble des graphes élémentaires, devient un treillis lorsque l'on ne considère que les graphes irredondants : il s'agit même du treillis dénombrable universel.

3.2.1 Support conjonctif

La représentation compacte d'un graphe élémentaire $G \in \mathcal{G}^e(\mathcal{S})$ peut être vue comme la *représentation graphique standard* d'un graphe G' étiqueté sur un support $\mathcal{S}^*$ dont les éléments sont des ensembles finis d'éléments de $\mathcal{S}$.

Nous commençons par définir formellement ce support $\mathcal{S}^*$, que nous appellerons *support conjonctif* .

Définition 3.5 (Support conjonctif) *Soit $\mathcal{S} = ((T_1, \leq_1), \ldots, (T_k, \leq_k))$ un support élémentaire quelconque. Alors nous définissons $\mathcal{S}^* = ((T_1^*, \leq_1^*), \ldots, (T_k^*, \leq_k^*))$, son* support conjonctif*, par :*

- *chaque T_i^* est composé de tous les* multiensembles finis non vides *de T_i ;*
- *si $T, T' \in T_i^*$, alors $T \leq_i^* T'$ ssi $\forall t' \in T'\ \exists t \in T,\ t \leq_i t'$.*

Nous remarquons que, si $\leq_i$ est un préordre, alors $\leq_i^*$ est un préordre. Cependant, même si $\leq_i$ est un ordre, $\leq_i^*$ n'est pas un ordre : si $t \leq t'$, alors $\{t\} \leq^* \{t, t'\}$ et $\{t, t'\} \leq^* \{t\}$ mais $\{t, t'\} \neq \{t\}$. Nous pourrons cependant nous ramener à un ordre en ne considérant que des types simplifiés, équivalents.

Propriété 3.4 (Type conjonctif irredondant) *Soit $\leq_i$ un ordre sur T_i. Alors, pour tout type $T \in T_i^*$, il existe un unique T' de taille minimale équivalent à T (qui peut être obtenu en parcourant les éléments de T, et en supprimant chaque t tel qu'il existe $t' \in T$ avec $t' \leq_i t$). Nous appellerons T' le* type irredondant *de T. Alors $\leq_i^*$ est un inf-demi-treillis sur les éléments irredondants de T_i^*. Si, de plus, $\leq_i$ admet un élément maximum, alors $\leq_i^*$ est un treillis (c'est même un sous-treillis de $\mathcal{P}(T_i)$, l'ensemble des parties de T_i).*

Nous ne nous servirons ici que du fait que $\leq_i^*$ est un ordre sur l'ensemble des types irredondants. La démonstration de cette propriété plus forte a cependant son intérêt, puisque nous aurons exactement le même schéma de preuve pour la relation de subsomption sur l'ensemble des graphes élémentaires irredondants.

Preuve: Soient $T = \{t_1, \ldots, t_p\}$ et $T' = \{t'_1, \ldots, t'_q\}$ deux types quelconques irredondants de T_i^*. Alors $T'' = \{t_1, \ldots, t_p, t'_1, \ldots, t'_q\}$ est un minorant de T et T'. Vérifier que, si X est un minorant de T et T', alors $X \leq_i^* T''$. Alors le type irredondant de T'' est la borne inférieure de T et T' dans l'ensemble des types irredondants de T_i^*. Donc $\leq_i^*$ est bien un inf-demi-treillis sur les éléments irredondants de T_i^*. Si, de plus, t est l'élément maximal de T_i, alors $\{t\}$ est l'élément maximal pour $\leq_i^*$. Nous avons donc un treillis (voir Prop. C.1). □

Soit G un graphe élémentaire sur un support $\mathcal{S}$. Alors soit G^* le graphe élémentaire sur $\mathcal{S}^*$ obtenu en remplaçant le type $\tau(r)$ de chaque relation de G par le type $\{\tau(r)\}$. Alors

il est immédiat de démontrer qu'une telle transformation du graphe élémentaire (et du support) conserve toutes les projections :

Propriété 3.5 (Un type élémentaire est un type conjonctif) *Soient $G, H \in \mathcal{G}^e(\mathcal{S})$. Alors une application $\pi : V(H) \to V(G)$ est une $\mathcal{S}$-projection de H dans G si et seulement si π est une $\mathcal{S}^*$-projection de H^* dans G^*.*

Nous noterons donc, par abus de langage, $G = G^*$, et considérerons qu'il s'agit du même graphe. Nous pouvons donc voir $\mathcal{G}^e(\mathcal{S})$ comme un sous-ensemble de $\mathcal{G}^e(\mathcal{S}^*)$, composé des graphes élémentaires sur $\mathcal{S}^*$ dont tous les types sont de taille 1.

Expansion et compression

Soit maintenant G un graphe élémentaire quelconque sur un support conjonctif $\mathcal{S}^*$. Nous appelons *expansion* de G et notons $exp(G)$ le graphe obtenu en remplaçant chaque relation r de G, avec $\tau(r) = \{t_1, \ldots, t_p\}$, par p relations ayant mêmes arguments que r, dont les types sont respectivement $\{t_1\}, \ldots, \{t_p\}$. Si $exp(G) = G$, on dira que le graphe G est *expansé*.

De façon duale, nous définissons la *compression* de G et notons $comp(G)$ le graphe obtenu en fusionnant toutes les relations ayant mêmes arguments. Plus précisément, si $C = \{r_1, \ldots, r_p\}$ est un ensemble non vide composé de *toutes* les relations ayant mêmes arguments ($\gamma(r_1) = \cdots = \gamma(r_p)$), alors les relations $r_1, \ldots, r_p$ sont remplacées par une unique relation R. Le voisinage de R est celui des r_i ($\gamma(R) = \gamma(r_1) = \cdots = \gamma(r_p)$, *i.e.* les relations sont jumelles) et, si $\tau(r_1) = \{t_1^1, \ldots, t_1^{q_1}\}, \ldots, \tau(r_p) = \{t_p^1, \ldots, t_p^{q_p}\}$, alors $\tau(R) = \{t_1^1, \ldots, t_1^{q_1}, \ldots\ldots, t_p^1, \ldots, t_p^{q_p}\}$. Alternativement, on peut voir $comp(G)$ comme le graphe élémentaire sur $\mathcal{S}^*$ déterminé par la *représentation compacte* de $exp(G)$. Si $comp(G) = G$, on dira que le graphe G est *compressé*. Notons qu'un graphe élémentaire peut être à la fois expansé et compressé.

Si un graphe G n'est ni compressé, ni expansé, on dira qu'il est *mixte*. La propriété suivante, immédiate, donne une caractérisation des graphes élémentaires compressés, expansés, et mixtes :

Propriété 3.6 (Caractérisation des graphes expansés et compressés) *Soit G un graphe élémentaire défini sur un support conjonctif $\mathcal{S}^*$. Alors :*

- *G est compressé ssi il n'existe aucune relation jumelle (γ^{-1} est injective) ;*
- *G est expansé ssi $\forall r \in U(G)$, $|\tau(r)| = 1$ ($G \in \mathcal{G}^e(\mathcal{S})$) ;*
- *G est mixte ssi il contient deux relations jumelles r et r' telles que $|\tau(r)| > 1$ ou $|\tau(r')| > 1$.*

Nous n'avons plus qu'à déterminer la façon dont la compression et l'expansion de graphes élémentaires influe sur les projections :

Théorème 3.1 (Projections de graphes expansés et compressés) *Soient H et G deux graphes élémentaires sur un support conjonctif $\mathcal{S}^*$. Soit π une application de $V(H)$ dans $V(G)$.*

(i) si π est une projection de H dans G, alors c'est une projection de $\exp(H)$ *dans G (mais la réciproque n'est pas toujours vraie) ;*
(ii) si π est une projection de H dans G, alors c'est une projection de H dans $comp(G)$ *(mais la réciproque n'est pas toujours vraie) ;*
(iii) π est une projection de H dans $comp(G)$ *si et seulement si c'est une projection de* $\exp(H)$ *dans G.*

Preuve: Nous remarquerons tout d'abord que, pour tout graphe élémentaire G, il existe une projection (assimilée à l'identité, puisqu'elle fait intervenir les mêmes sommets concepts par construction, notée *id*), de $exp(G)$ dans G et de G dans $comp(G)$. La vérification de cette propriété est immédiate, par définition de *comp* et de *exp*. Alors si π est une projection de H dans G, *(i)*, $id \circ \pi = \pi$ est une projection de $exp(H)$ dans G, et, *(ii)*, $\pi \circ id = \pi$ est une projection de H dans $comp(G)$.

Pour montrer que les réciproques ne sont pas toujours vérifiées, nous exhibons les graphes $G = \{\{t_1, t_2\}(x, y), \{t_3, t_4\}(x, y)\}$ et $H = \{\{t_1, t_3\}(x, y), \{t_2, t_4\}(x, y)\}$, tels que les types t_1, t_2, t_3 et t_4 sont incomparables dans $\mathcal{S}$. Ces deux graphes sont incomparables, et pourtant, il existe une projection de $exp(H)$ dans G comme de H dans $comp(G)$. On a même $comp(G) = comp(H)$ et $exp(G) = exp(H)$.

Prouvons maintenant l'équivalence *(iii)*.

($\Rightarrow$) Supposons π une projection de H dans $comp(G)$. Alors π est aussi une projection de $exp(H)$ dans $comp(G)$ (d'après *(i)*). Alors pour chaque relation $r \in U(exp(H))$, il existe une unique relation R telle que $\pi_U(r) = R$. Le type de R est plus spécifique que celui de chacune des relations jumelles de r, *i.e.* pour chaque relation r' jumelle de r, il existe un type $t \in \tau(R)$ tel que $t \leq \tau(r')$. Dans le graphe G, il y a au moins une relation R' ayant les mêmes arguments que R dont le type contient t. Donc $R' \in \pi_U(r')$, et π est bien une projection de $exp(H)$ dans G.

($\Leftarrow$) Réciproquement, supposons π une projection de $exp(H)$ dans G. Alors π est aussi une projection de $exp(H)$ dans $comp(G)$ (d'après *(ii)*). De la même façon que pour ($\Rightarrow$), pour chaque relation $r \in U(exp(H))$, il existe une unique relation R telle que $\pi_U(r) = R$. Le type de R est plus spécifique que celui de chacune des relations jumelles de r, *i.e.* pour chaque relation r' jumelle de r, il existe un type $t \in \tau(R)$ tel que $t \leq \tau(r')$. Ceci reste donc vrai pour un sous-ensemble quelconque de relations jumelles de r, et donc pour le type de la relation de H qui s'est décomposé en celles-ci. Alors π est une projection de H dans $comp(G)$. □

Grâce à ce théorème, nous avons (enfin) un outil qui nous permet de rechercher les projections sur la représentation compacte d'un graphe élémentaire.

Corollaire 3.1 (Équivalence de la projection de graphes compressés) *Soient G et H deux graphes élémentaires sur $\mathcal{S}$. Alors une application $\pi : V(H) \to V(G)$ est une $\mathcal{S}$-projection si et seulement si π est une $\mathcal{S}^*$-projection de* $comp(H)$ *dans* $comp(G)$.

Preuve: La démonstration est immédiate : π est une $\mathcal{S}$-projection de H dans $G \Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de H dans G (Prop. 3.5) $\Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de $exp(H)$ dans G

(car $exp(H) = H$) $\Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de H dans $comp(G)$ (Th. 3.1, *(iii)*) $\Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de $exp(comp(H))$ dans $comp(G)$ (car $exp(comp(H)) = H$) $\Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de $comp(H)$ dans $comp(comp(G))$ (Th. 3.1, *(iii)*) $\Leftrightarrow \pi$ est une $\mathcal{S}^*$-projection de $comp(H)$ dans $comp(G)$ (car $comp(comp(G)) = G$). □

Remarquons que notre démonstration n'utilise que la troisième partie du Th. 3.1. Les deux premiers résultats (et surtout leur absence de réciproque) nous serviront de justificatifs lorsque nous établirons des limites quant à la façon dont on peut représenter la conjonction de types.

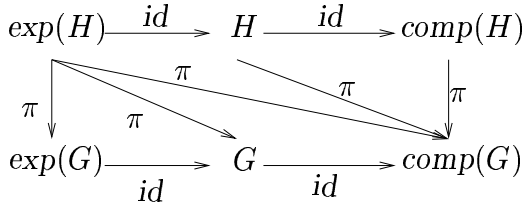


FIG. 3.8 – Équivalence des projections entre graphes compressés, expansés et mixtes

Les projections équivalentes sont représentées dans la FIG. 3.8. Comme il est d'usage en théorie des catégories, les objets (graphes élémentaires) sont représentés par des sommets, et les morphismes (projections) par des arcs. Nous pourrons donc indifféremment utiliser la $\mathcal{S}$-projection sur des graphes élémentaires expansés, ou la $\mathcal{S}^*$-projection sur les graphes compressés. En travaillant avec les graphes compressés, nous ne créons ni ne perdons aucune projection. Il s'agit d'un modèle équivalent, où la multiplicité des hyperarcs est codée par la multiplicité des types. L'intérêt algorithmique est évident, puisqu'une partie de la multiplicité des choix pour projeter une relation est reléguée dans le support, qui pourra être compilé (voir Sec. 5). D'un point de vue représentation de connaissances, nous verrons qu'un ensemble de types peut être considéré comme une conjonction de types, besoin fréquemment rencontré dans la communauté « graphes conceptuels » (par exemple dans [Cao, 1999] ou pour les constructions de [Baader et al., 1999]).

Graphes élémentaires minimaux

Nous avons vu que nous pouvions coder les graphes élémentaires sur $\mathcal{S}$ par des graphes élémentaires sur $\mathcal{S}^*$, ayant moins de relations. Ce codage conserve l'ensemble des projections. Cependant, ce graphe n'est bien entendu pas le plus petit ayant cette propriété, puisque ses types peuvent ne pas être irredondants (voir Prop. 3.4).

Nous supposerons donc maintenant que $\leq$ définie sur $\mathcal{S}$ est un ordre, ce qui est nécessaire pour considérer les types irredondants sur $\mathcal{S}^*$.

Définition 3.6 (Graphe élémentaire minimal) *Soit G un graphe élémentaire sur un support $\mathcal{S}$. Nous appelons* graphe minimal *de G et notons $min(G)$ le graphe élémentaire sur $\mathcal{S}^*$ (le support conjonctif associé à $\mathcal{S}$ obtenu à partir de $comp(G)$ (le graphe compressé*

obtenu à partir de G en fusionnant ses relations jumelles) en remplaçant chacun de ses types par son type irredondant équivalent.

Par l'équivalence entre les projections de graphes expansés et de graphes compressés (Cor. 3.1) et l'équivalence des étiquettes, nous obtenons l'équivalence entre les projections de graphes expansés et les projections de graphes minimaux. Les graphes minimaux sont même les plus petits graphes ayant cette propriété.

Propriété 3.7 (Équivalence de la projection de graphes minimaux) *Soit G un graphe élémentaire sur $\mathcal{S}$. Alors $min(G)$ est le plus petit graphe sur $\mathcal{S}^*$ tel que, pour tout graphe élémentaire H sur $\mathcal{S}$, pour toute application $\pi : V(H) \to V(G)$, π est une $\mathcal{S}$-projection de H dans G si et seulement si π est une $\mathcal{S}^*$-projection de $min(H)$ dans $min(G)$.*

Preuve: Le sens ($\Rightarrow$) de cette équivalence est une conséquence directe du Cor. 3.1 et de l'équivalence entre un type de relation sur $\mathcal{S}^*$ et son type irredondant. Pour prouver, ($\Leftarrow$), il suffit de remarquer que *id* est une projection de G dans $min(G)$, mais que si on ôte un type $t \in \mathcal{S}$ du type $T \in \mathcal{S}^*$ d'une relation de $min(G)$ (*i.e.* T devient un de ses supertypes), si on ôte une relation r de $min(G)$, ou si on ôte un sommet concept x de $min(G)$, alors *id* n'est plus une projection de G dans $min(G)$. □

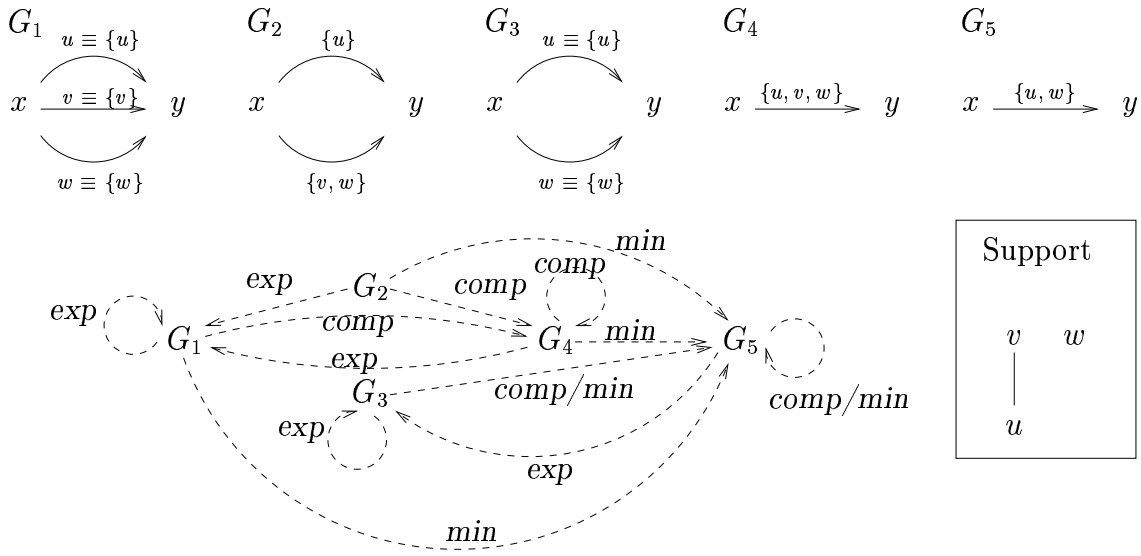


FIG. 3.9 – Illustration des transformations *comp*, *exp* et *min*.

Le dessin de la FIG. 3.9 résume et illustre les différentes transformations *comp*, *exp* et *min*. Nous avons supposé ici que la seule comparaison possible entre u, v et w dans $\mathcal{S}$ était $u \leq v$. La présence d'un arc pointillé étiqueté *exp* de G_2 vers G_1 se lit $exp(G_2) = G_1$.

Utiliser des graphes minimaux simplifiera bien des démonstrations, par exemple, la notion d'isomorphisme peut s'exprimer de façon plus simple, grâce à une propriété bien connue dans la communauté « homomorphismes » (voir, par exemple [Hahn and Tardif, 1997]) :

Propriété 3.8 (Isomorphismes de graphes minimaux) *Si π est une projection bijective entre deux graphes minimaux, dont* la *fonction de choix associée est fidèle, alors elle est complète. De plus, tout choix conservateur est fortement conservateur. Il s'ensuit que toute projection π bijective, fidèle et conservatrice est un isomorphisme.*

3.2.2 Graphes élémentaires irredondants

Nous avons vu que les projections de H dans G étaient exactement les projections de $min(H)$ dans $min(G)$. Que se passe-t-il quand on ne s'intéresse qu'à *l'existence* d'une projection ?

L'ensemble $(\mathcal{G}^e(\mathcal{S}), \text{muni de la relation } \preceq_{\mathcal{S}})$ est un *préordre* si et seulement si $\mathcal{S}$ encode un tuple de préordres (Prop. 3.1). Par contre, quel que soit le support $\mathcal{S}$, $(\mathcal{G}^e(\mathcal{S}), \preceq_{\mathcal{S}})$ n'est pas un ordre (Prop. 3.3). Le contre-exemple que nous avons donné ($G = \{t(x)\}$, $H = \{t(x), t(y)\}$) portant sur des graphes minimaux, il s'ensuit que la relation $\preceq^*$ sur l'ensemble des graphes minimaux de $\mathcal{S}^*$ n'est pas non plus un ordre. Pour obtenir des ordres, il nous faudra choisir un représentant dans chacune des classes d'équivalence pour $\cong$.

Existence et unicité

De façon intuitive, un graphe G est dit irredondant s'il n'existe pas de projection de G dans l'un de ses sous-graphes stricts. D'un point de vue représentation de connaissances, ceci exprime que G ne contient pas d'information redondante.

Définition 3.7 (Graphe élémentaire irredondant) *Un graphe élémentaire G sur un support $\mathcal{S}$ est dit* irredondant *si, pour tout endomorphisme π de G (projection de G dans lui-même), pour toute image homomorphe G' de G dans lui même pour π, G' est isomorphe à G.*

Les choix que nous avons fait pour les graphes élémentaires (les relations sont des hyperarcs multiples dont on ne calcule pas la projection) imposent cette définition complexe. Elle est cependant nécessaire dans le cas général : prenons le graphe élémentaire $G = [t(x), t'(y), r(x, y), r'(x, y)]$, sur un support $\mathcal{S}$ tel que t et t' sont incomparables, et $r \leq r'$. Il n'existe qu'un endomorphisme de G, l'identité, mais pour lequel les deux choix possibles (r dans r et r' dans r', ou r et r' dans r) déterminent deux images homomorphes distinctes. La première est isomorphe à G, alors que la seconde ne l'est pas. G n'est donc pas irredondant (par contre, son graphe partiel $G = [t(x), t'(y), r(x, y)]$ l'est. La notion d'irredondance doit donc prendre en compte toutes les images homomorphes possibles pour une projection.

Théorème 3.2 (Existence et unicité du graphe irredondant) *Soit C une classe d'équivalence de $(\mathcal{G}^e(\mathcal{S}), \preceq_{\mathcal{S}})$. Alors C contient un unique graphe irredondant (noté, pour un représentant quelconque G de C, irr(G)) : c'est le plus petit graphe de cette classe quant au nombre de sommets concepts et de relations.*

Ce théorème est dû à [Chein and Mugnier, 1992] dans le cadre des graphes conceptuels simples ; dans le cadre des homomorphismes de graphes, une version équivalente est donnée par [Hell and Nešetřil, 1979] (le graphe irredondant est alors appelé *core*). Nous donnons ici une preuve différente de celle de [Chein and Mugnier, 1992], en particulier, il s'agit d'une preuve constructive, qui repose sur un algorithme permettant de calculer $irr(G)$. Afin de ne pas surcharger cette démonstration par la considération de toutes les images homomorphes engendrées par un endomorphisme, nous prouverons le Th. 3.2 en deux temps : nous ne considérerons tout d'abord que des graphes élémentaires minimaux sur $\mathcal{S}^*$, puis utiliserons ce résultat dans le cas général. Remarquons déjà que notre démonstration reste valide dans le cas d'un support infini.

Cas des graphes élémentaires minimaux

Remarquons immédiatement que la caractérisation de l'irredondance devient plus simple dans le cas des graphes élémentaires minimaux.

Propriété 3.9 (Graphe irredondant minimal) *Un graphe élémentaire minimal G sur $\mathcal{S}^*$ est irredondant si et seulement si tout endomorphisme de G dans lui-même est un automorphisme.*

Preuve: Immédiat, en remarquant que la donnée d'une projection d'un graphe élémentaire dans un graphe élémentaire minimal détermine de façon unique la fonction de choix pour les relations, et donc l'image homomorphe pour cette projection. □

Propriété 3.10 (Graphe irredondant d'un graphe minimal) *Soit C une classe d'équivalence de l'ensemble des graphes élémentaires minimaux sur $\mathcal{S}^*$. Alors C contient un unique graphe irredondant : c'est le plus petit graphe de cette classe quant au nombre de sommets concepts et de relations.*

Preuve: Nous prouverons 1), que s'il existe un graphe irredondant G dans C (*i.e.* un graphe G dont tous les endomorphismes sont des automorphismes), alors tout graphe H équivalent à G a plus de sommets et plus de relations que G ($\geq$), 2) que s'ils ont la même taille, alors ils sont isomorphes (unicité), et 3) que pour tout graphe H, il existe un graphe G équivalent à H qui est irredondant (existence).

1) Supposons G dont tous les endomorphismes sont des automorphismes, et H un graphe équivalent à G. Alors pour toutes les projections (elles existent) $\pi : V(H) \to V(G)$ et $\pi' : V(G) \to V(H)$, $\pi \circ \pi'$ est un endomorphisme de G, c'est donc un isomorphisme. Comme $\pi \circ \pi'$ est bijective, alors π est surjective, donc $|V(H)| \geq |V(G)|$. De même, la donnée de π et de π' détermine *de façon unique* (G et H sont compressés) les applications $\pi_U : U(H) \to U(G)$ et $\pi'_U : U(H) \to U(G)$. Voir tout d'abord que $(\pi \circ \pi')_U = \pi_U \circ \pi'_U$. Comme $\pi \circ \pi'$ est un isomorphisme, alors $(\pi \circ \pi')_U$ est un choix complet (*i.e.* c'est une bijection de $U(G)$ dans lui-même). Donc π'_U est surjective (fidèle) et $|U(H)| \geq |U(G)|$.

2) Supposons de plus que $|V(H)| = |V(G)|$ et $|U(H)| = |U(G)|$. Alors π' est une bijection de $V(G)$ dans $V(H)$ (car π étant surjective entre deux ensembles finis [4] de même taille, c'est donc une bijection ; et il faut, pour que $\pi \circ \pi'$ soit une bijection, que π' soit aussi une bijection). Pour les mêmes raisons, π'_U est une bijection, c'est donc un choix complet. Voir que, pour toute relation $r \in U(G)$, $t = \tau(r) = \tau(\pi_U(r)) = t'$ (car $t \leq t'$, $t' \leq t$, et G et H sont étiquetés par des types irredondants). Alors π' est un isomorphisme de G dans H.

3) Soit G un graphe élémentaire compressé quelconque sur $\mathcal{S}^*$. Nous allons construire *irr*(G), grâce à un algorithme (Alg. 1) semblable à celui de [Mugnier, 1995].

Algorithme 1: *irredondant*(G)

Données : Un graphe élémentaire minimal G sur un support conjonctif $\mathcal{S}^*$.

Résultat : Le graphe élémentaire irredondant *irr*(G) équivalent à G.

```
E ← endomorphismes(G);
pour (π ∈ E) faire
    si (¬ automorphisme ?(π|_G)) alors
        // où π|_G est la restriction de π au graphe courant G
        G ← image-homomorphe (G, π);
retourner G;
```

Cet algorithme commence par calculer toutes les projections $(\pi_1, \ldots, \pi_k)$ de G dans lui-même, puis les parcourt dans cet ordre. A chaque étape, on remplace éventuellement G par un autre graphe élémentaire, son image homomorphe par la projection courante. Notons $G \stackrel{\pi_1}{\rightarrow} G_1 \cdots \stackrel{\pi_k}{\rightarrow} G_k$. Nous commençons par prouver que G_k est équivalent à G, ce qui est évident puisque $\pi_k \circ \cdots \circ \pi_1$ est une projection de G dans G_k, et que, comme G_k est un sous-graphe partiel de G (une propriété de l'image homomorphe), il existe une projection de G_k dans G. L'algorithme s'arrête donc sur un graphe G_k équivalent à G.

Reste à voir que le graphe G_k obtenu est irredondant. Soit π un endomorphisme quelconque de G_k. Nous prouvons tout d'abord que π est bijective, et, pour cela, il nous suffira de prouver que π est surjective (une surjection d'un ensemble fini dans un ensemble fini de même taille est une bijection). Par l'absurde, supposons que π n'est pas surjective. Alors il existe un sommet concept x de G_k qui n'a pas d'antécédent par π. Voir que $\pi' = \pi \circ \pi_k \circ \cdots \circ \pi_1$ est une projection de G dans G_k (un sous-graphe partiel de G) qui n'est pas surjective (le même sommet x n'a pas d'image par π'), on peut donc considérer π' comme un endomorphisme de G qui n'est pas un automorphisme. Donc π' a été calculé par *endomorphismes* (et posons π_p le premier de ces endomorphismes pour lesquels x n'a pas d'antécédent), et donc a été envisagé par l'appel de *automorphisme ?*, qui a bien entendu

[4] La Def. 3.7 est donc une bonne caractérisation des graphes irredondants tant qu'on reste dans le cas des graphes finis. Dans le cas des graphes infinis, on peut voir, par exemple dans [Hahn and MacGillivray, 2001], qu'on peut passer par la notion de *retract* d'un graphe (mais nous n'en aurons pas besoin dans ce mémoire).

répondu faux. Donc le graphe G_p est obtenu en calculant l'image homomorphe de G_{p-1} suivant π_p. Alors x, qui n'a pas d'antécédent par π_p, ne fait pas partie de G_p. Il ne fait donc pas partie de G_k, ce qui est absurde. Donc π est bijective.

Comme G est un graphe compressé, tout sous-graphe partiel de G est un graphe compressé. Alors les projections π d'un G_i dans un G_{i+1} déterminent de façon unique une fonction de choix associée π_U. Soit π un endomorphisme (nécessairement bijectif) de G_k. Alors nous prouverons que π_U est bijectif (*i.e.* complet), comme précédemment. Il nous suffira de considérer de la même manière une relation r non atteinte par π_U.

Enfin, nous montrons que si π est un endomorphisme (nécessairement bijectif et complet) de G_k, alors il est fortement conservateur. Puisque π est complet, π_U est une bijection de $U(G_k)$ dans $U(G_k)$. C'est donc une permutation de $U(H)$ ($U(H)$ est fini). Dessinons le graphe de cette permutation. A chaque relation correspond un sommet, et si $\pi_U(r) = r'$, nous dessinons un arc allant de r vers r'. Puisque π_U est une permutation, alors les degrés entrant et sortant de chacun des sommets sont égaux à 1, et ce graphe est un ensemble de circuits disjoints (les orbites de la permutation, aussi utilisées dans la démonstration de [Cogis and Guinaldo, 1995]). Donc, soit r une relation quelconque de G_k, et $r' = \pi_U(r)$. L'arc (r, r') fait partie d'un circuit $(r, r', r_1, \ldots, r_q, r)$. On a donc $\tau(r) \leq \tau(r_q) \leq \cdots \leq \tau(r_1) \leq \tau(r') \leq \tau(r)$ (sinon, le $(j+1)^{\text{ième}}$ élément de ce circuit ne serait pas un certificat pour le $j^{\text{ième}}$). Donc $\tau(r) \equiv \tau(r')$, et comme les relations de G_k sont typées par des types irredondants, $\tau(r) = \tau(r')$. Donc π_U est un choix fortement conservateur.

Alors tout endomorphisme de G_k est un automorphisme. □

Retour au cas général

Nous savons donc calculer le graphe irredondant d'un graphe élémentaire minimal sur $\mathcal{S}^*$. Ceci nous permet de calculer facilement le graphe irredondant sur $\mathcal{S}$ d'un graphe élémentaire sur $\mathcal{S}$.

Propriété 3.11 (Calcul du graphe irredondant) *Soit C une classe d'équivalence de $(\mathcal{G}^e(\mathcal{S}), \preceq_{\mathcal{S}})$. Soit H un représentant quelconque de C, et $G = exp(irr(min(H)))$. Alors :*

1. *G est le plus petit graphe de C quant au nombre de sommets concepts et de relations.*
2. *G est l'unique graphe élémentaire irredondant de C ;*

Nous pourrons alors noter $G = irr(H)$.

Cette propriété suffit à prouver le Th. 3.2, tout en donnant un algorithme de construction pour le graphe irredondant, utilisant l'Alg. 1.

Preuve: Commençons par prouver le point 1. de cette propriété :

Tout d'abord, nous prouvons que $H \in \mathcal{G}^e(\mathcal{S})$ et G ainsi défini sont équivalents (d'où $G \in C$). Nous remarquons tout d'abord que $H = exp(comp(H))$ (par construction), puis procédons par équivalences successives : $H \cong exp(comp(H)) \Leftrightarrow H \cong exp(min(H))$ (car $min(H) \cong comp(H)$) $\Leftrightarrow H \cong exp(irr(min(H)))$ (car, $\forall X,\ X \cong irr(X)$) $\Leftrightarrow H \cong G$.

Supposons maintenant un graphe élémentaire H' équivalent à H, et prouvons que G est plus petit que H'. Procédons par équivalences successives (mais seules les implications sont nécessaires) : $H' \cong H \Leftrightarrow comp(H') \cong comp(H)$ (Cor. 3.1) $\Leftrightarrow comp(H') \cong min(H)$ (types remplacés par des types équivalents) $\Leftrightarrow comp(H') \cong irr(min(H))$ (car, $\forall H,\ H \cong irr(H)$).

Le graphe $irr(min(H))$ est donc le graphe irredondant de $comp(H')$. Donc $irr(min(H))$ est un sous-graphe partiel de $comp(H')$ (voir la preuve de la Prop. 3.10). Alors $exp(irr(min(H)))$ est un sous-graphe partiel de $exp(comp(H')) = H'$ (par construction). Donc G est un sous-graphe partiel de H', il est donc plus petit.

Montrons maintenant le point 2. de cette propriété.

Le graphe élémentaire $H' = irr(min(H))$ est irredondant. Donc tout endomorphisme de H' est un automorphisme. Or, comme H' est minimal, les endomorphismes de H' sont exactement ceux de $G = exp(H')$ (Th. 3.1, *(iii)*). Supposons maintenant que G ne soit pas irredondant. Alors il existe une image homomorphe G' de G par un endomorphisme π qui n'est pas isomorphe à G. Alors $min(G')$ est un sous-graphe *strict* de H' (ou ayant une étiquette plus spécifique), dans lequel il se projette (Th. 3.1, *(ii)*), ce qui est absurde puisque H' est irredondant.

Enfin, l'unicité de G provient de celle de H'. □

3.2.3 Structure de la relation de subsomption

Nous avons vu que la relation $\preceq$ était un préordre sur l'ensemble des graphes élémentaires. Cependant, si on ne considère que l'ensemble des graphes élémentaires irredondants $\mathcal{G}^e_{\text{irr}}(\mathcal{S})$ (et pour pouvoir le faire il nous faut un *ordre* $\leq$ sur le support), la structure obtenue est beaucoup plus intéressante.

Le treillis des graphes élémentaires irredondants ...

Théorème 3.3 (Treillis des irredondants) *La relation $\preceq_{\mathcal{S}}$ est un treillis sur $\mathcal{G}^e_{irr}(\mathcal{S})$.*

Ce théorème est dû à [Mugnier and Chein, 1996] dans le cadre des graphes conceptuels, et ce treillis (*coloring poset*), dans le cas de graphes binaires, non orientés, sans étiquettes, a fait l'objet de nombreux travaux dans la communauté « Homomorphismes de Graphes » (voir, par exemple, [Hell and Nešetřil, 1992]).

Preuve: Nous rappelons ici la démonstration de [Mugnier and Chein, 1996]. Tout d'abord, puisque deux graphes équivalents admettent le même graphe irredondant, il s'agit bien d'un ordre. Ensuite, soient G_1 et G_2 deux graphes de $\mathcal{G}^e_{\text{irr}}(\mathcal{S})$. Alors soit $G_1 \oplus G_2$ l'union disjointe de G_1 et G_2 (*i.e.* le graphe dont le dessin est la juxtaposition du dessin de G_1 et de celui de G_2). Comme $G_1 \oplus G_2 \preceq G_1$ et $G_1 \oplus G_2 \preceq G_2$, $G = irr(G_1 \oplus G_2)$ est un minorant de G_1 et G_2 appartenant à $\mathcal{G}^e_{\text{irr}}(\mathcal{S})$. Pour tout minorant H de G_1 et G_2, on a $H \preceq G_1 \oplus G_2$ (car si π_1 est une projection de G_1 dans H et π_2 est une projection de G_2 dans H, alors π_{1+2}, qui associe $\pi_j(x)$ à un sommet concept x correspondant à un sommet concept de G_j, est une projection de $G_1 \oplus G_2$ dans H). Donc $H \preceq irr(G_1 \oplus G_2) = G$. Donc toute paire de graphes irredondants admet une borne inférieure, et $\preceq_{\mathcal{S}}$ est un inf-demi-treillis sur $\mathcal{G}^e_{\text{irr}}(\mathcal{S})$.

Il ne nous reste plus qu'à trouver un élément maximum pour $\preceq_{\mathcal{S}}$, ce sera le *graphe élémentaire vide* $\emptyset$. Alors $\preceq_{\mathcal{S}}$ est un treillis sur $\mathcal{G}^e_{\text{irr}}(\mathcal{S})$. Dans [Mugnier and Chein, 1996], la possibilité d'avoir un graphe vide n'est pas envisagée, et c'est pourquoi T_1 doit admettre un maximum. Nous voyons ici que ce n'est pas une nécessité théorique. □

Ce théorème est important puisqu'il assure, pour deux graphes élémentaires G_1 et G_2 donnés, qu'ils admettent une borne supérieure (une plus petite généralisation commune) et une borne inférieure (une plus grande spécialisation commune). Les démonstrations des propriétés 3.10 et 3.11 nous donnent un algorithme (exponentiel) du calcul de la borne inférieure. Nous verrons dans le Chap. 5 qu'il s'agit d'un problème NP-complet. Le calcul de la borne supérieure de deux graphes peut se faire par une opération de produit de graphes, appelée « weak product » dans [Welzl, 1984]. Le graphe obtenu n'est généralement pas irredondant, il reste donc à la mettre sous forme irredondante.

... est le treillis dénombrable universel

Théorème 3.4 (Treillis dénombrable universel) *Si* $\exists i \geq 2$ *t.q.* $T_i \neq \emptyset$, *alors* $\preceq$ *sur* $\mathcal{G}^e_{irr}(\mathcal{S})$ *admet comme sous-ordre le treillis dénombrable universel.*

$\mathcal{C}$ est le *treillis universel dénombrable* veut dire que $\mathcal{C}$ est un treillis, et que tout treillis dénombrable est un sous-ordre de $\mathcal{C}$. Le théorème de Hedrlìn et Kucěra affirme que l'ensemble $\mathcal{C}$ des *cores*,[5] ordonné par « $G \leq G'$ ssi il existe un homomorphisme de G' dans G », est le treillis universel dénombrable.

Le problème de l'universalité du *coloring poset*, posé par [Hedrlín, 1969], a longtemps été un problème ouvert ; sa démonstration (25 pages) par Hedrlìn et Kucěra est disponible dans [Pultr and Trnková, 1980]. La démonstration de [Nešetřil, 2000], plus directe, ne fait que 10 pages. Ces deux démonstrations utilisent cependant un grand nombre d'outils théoriques non triviaux, que nous ne jugeons pas nécessaire d'inclure dans ce mémoire.

Preuve: Voir que $\mathcal{C}^* = \mathcal{C} \setminus \{\emptyset\}$ est encore le treillis universel dénombrable (il reste encore le *core* à un sommet comme maximum).

Vérifions maintenant que, s'il existe un T_i non vide dans $\mathcal{S}$, avec $i \geq 2$, alors $\mathcal{C}^*$ est un sous-ordre de $\preceq_{\mathcal{S}}$. Soit t un type quelconque de T_i, et f l'application (injective) qui a tout *core* G associe le graphe élémentaire sur $\mathcal{S}$ dont les sommets concepts sont les sommets de G et tel que toute arête x, y de G est remplacée par deux relations typées par t. La première a pour voisinage $(x, y, \ldots, y)$ et la seconde $(y, \ldots, y, x)$ (une généralisation du plus naturel (x, y) et (y, x) utilisés dans le cas binaire). Alors tout homomorphisme d'un *core* H dans un *core* G est une projection de $f(H)$ dans $f(G)$, et $\mathcal{C}$ est un sous-ordre de $\preceq_{\mathcal{S}}$. □

Si $\mathcal{S}$ est dénombrable, $\preceq$ sera donc le treillis dénombrable universel. Il s'agit d'une propriété très forte : tout modèle de représentation de connaissances, manipulant un ensemble dénombrable d'objets, et pour lequel la relation de subsomption est un treillis peut être vu

[5] Les *cores* sont l'équivalent des graphes irredondants lorsque l'on considère des graphes binaires, non orientés, sans étiquettes.

comme un cas particulier du modèle des graphes élémentaires (ou, de façon équivalente par l'intermédiaire de la sémantique logique Φ, du fragment positif, conjonctif, existentiel de la logique du premier ordre). Cependant, d'un point de vue opérationnel, nous ne savons rien de la fonction f qui doit associer des graphes élémentaires aux objets de ce modèle.

3.3 Sémantique logique des graphes élémentaires

Selon [Kayser, 1997], un modèle formel de représentation de connaissances repose sur une syntaxe (langage), un système de déduction, et des règles de valuation. Nous avons donné la syntaxe des objets que nous manipulons (les graphes élémentaires définis sur un support), et le système de déduction, la projection. La valuation (parfois appelée sémantique) sera donnée par l'intermédiaire d'une traduction vers la logique du premier ordre, positive, conjonctive, existentielle, sans constante. Le lecteur pourra être surpris de voir des quantificateurs universels dans ces formules. Nous verrons à la Sect. 4.2.3 qu'il est possible de supprimer ces quantificateurs universels.

Nous commencerons par *traduire* un support $\mathcal{S}$ et un graphe élémentaire G par des formules logiques $\Phi(\mathcal{S})$ et $\Phi(G)$, puis montrerons que la relation de subsomption que nous avons définie par l'existence d'une projection correspond à la déduction logique sur ces formules. Il s'agit d'une version graphes élémentaires du résultat d'adéquation et de complétude de la projection des graphes conceptuels par rapport à leur *sémantique logique* Φ [Sowa, 1984, Chein and Mugnier, 1992]. Nous en proposons ici une démonstration originale, qui ne repose pas sur la méthode de résolution mais directement sur les modèles d'une formule. Enfin, remarquons que Φ traduit un ensemble de types par une conjonction, ce qui justifie le nom de *support conjonctif* attribué à $\mathcal{S}^*$.

3.3.1 Interprétation d'un support et d'un graphe

Nous présentons ici la manière de construire les formules logiques $\Phi(\mathcal{S})$ et $\Phi(G)$ respectivement associées à un support *donné* $\mathcal{S}$ et à un graphe élémentaire G sur $\mathcal{S}$.

Formule logique associée à un support

Soit $\mathcal{S}$ un support *donné*. C'est-à-dire que pour tout $(T_i, \leq_i) \in \mathcal{S}$, il existe une relation *finie* $\leq'_i$ sur T_i telle que $\leq_i$ est la fermeture réflexo-transitive de $\leq'_i$. Si $\leq_i$ est un ordre, alors un $\leq'_i$ minimal, unique, existe, il s'agit de la *relation de couverture* de $\leq_i$.

C'est la relation $\leq'_i$ qui nous servira à construire *une* traduction $\Phi(\mathcal{S})$. Puisque nous considérons, dans le cas général, des préordres, nous ne pouvons pas parler de *la* traduction $\Phi(\mathcal{S})$, mais d'une de ses traductions possibles. Les formules obtenues sont cependant toutes équivalentes. Nous pouvons construire une traduction $\Phi(\mathcal{S})$ de la façon suivante :

1. à chaque type de relation t d'arité n dans le support nous associons un symbole de prédicat d'arité n, que nous noterons de la même façon t ;
2. nous disposons d'un ensemble dénombrable de variables $Var = \{x_1, \ldots, x_p, \ldots, \}$;

3. pour chaque T_i de $\mathcal{S}$, à chacun des couples $(t, t') \in \leq'_i$ (*i.e.* $t \leq'_i t'$) nous associons la formule $\phi(t, t') = \forall x_1 \ldots \forall x_i (t(x_1, \ldots, x_i) \rightarrow t'(x_1, \ldots, x_i))$;
4. une formule $\Phi(\mathcal{S})$ qui traduit le support est la conjonction, pour $(t, t') \in \leq'$, des $\phi(t, t')$.

Prenons la restriction à T_2 du support représenté dans la FIG. 3.1 (*i.e.* $\mathcal{S} = ((T_2, \leq_2))$. Alors sa traduction est :

$$\begin{aligned}\Phi(\mathcal{S}) = \quad & (\forall x_1 \forall x_2 \ (t_2^2(x_1, x_2) \rightarrow t_2^1(x_1, x_2))) \\ \wedge \quad & (\forall x_1 \forall x_2 \ (t_2^3(x_1, x_2) \rightarrow t_2^1(x_1, x_2))) \\ \wedge \quad & (\forall x_1 \forall x_2 \ (t_2^4(x_1, x_2) \rightarrow t_2^2(x_1, x_2))) \\ \wedge \quad & (\forall x_1 \forall x_2 \ (t_2^5(x_1, x_2) \rightarrow t_2^2(x_1, x_2)))\end{aligned}$$

Précisons que c'est par précaution que nous considérons ici un support *fini* : nous voulons obtenir une formule $\Phi(\mathcal{S})$ finie. En effet, nous n'avons pas étudié ce que deviennent nos résultats d'adéquation et de complétude, dans le cas de formules infinies.

Formule logique associée à un graphe

Soit G un graphe élémentaire sur un support $\mathcal{S}$. La traduction $\Phi(G)$ d'un graphe élémentaire n'est pas non plus unique, puisqu'elle dépend des variables que nous associerons à chaque sommet concept. Cependant, les formules que nous pouvons obtenir sont toutes équivalentes, à un renommage près des variables.

1. on se donne une application *injective*, *terme* : $V(G) \rightarrow \mathit{Var}$, mais, pour alléger les notations, nous pourrons noter de la même manière x, un sommet concept de G, et $x = \mathit{terme}(x)$, la variable qui lui est associée ;
2. chaque relation r de G, avec $\gamma(r) = (x_1, \ldots, x_p)$ et $\tau(r) = t$, est traduite par la formule $\phi(r) = t(x_1, \ldots, x_p)$;
3. nous notons $\phi(G)$ la conjonction des $\phi(r)$, pour $r \in U(G)$;
4. une formule $\Phi(G)$ qui traduit le graphe G est la fermeture existentielle de $\phi(G)$.

Prenons par exemple le graphe élémentaire G de la FIG. 3.2, dont nous rappelons ici la *notation prédicative* :

$$G = [t_1^1(x_1), t_1^4(x_4), t_1^2(x_1), t_2^3(x_1, x_2), t_2^1(x_1, x_2), t_2^1(x_1, x_2), t_2^2(x_1, x_2), t_1^3(x_2), t_3^1(x_1, x_4, x_4)]$$

Alors une formule logique qui traduit ce graphe élémentaire est :

$$\begin{aligned}\Phi(G) = \quad \exists x_1 \ \exists x_2 \exists x_4 \quad & (t_1^1(x_1) \ \wedge \ t_1^4(x_4) \ \wedge \ t_1^2(x_1) \ \wedge \ t_2^3(x_1, x_2) \ \wedge \ t_2^1(x_1, x_2) \\ & \wedge \ t_2^1(x_1, x_2) \ \wedge \ t_2^2(x_1, x_2) \ \wedge \ t_1^3(x_2) \ \wedge \ t_3^1(x_1, x_4, x_4))\end{aligned}$$

Remarquons que, pour écrire cette formule, il suffit de fermer existentiellement la formule obtenue en remplaçant les virgules par des $\wedge$ dans la notation prédicative. Cette traduction du graphe est une description naturelle, en logique du premier ordre, de la structure du graphe élémentaire.

Notons de plus que l'interprétation de l'union disjointe de plusieurs graphes est équivalente à la conjonction de leurs interprétations : si $\mathcal{G} = \{G_1, \ldots, G_k\}$ est une base de faits, alors $\Phi(\mathcal{G}) \equiv \Phi(G_1) \wedge \cdots \wedge \Phi(G_k)$.

3.3.2 Adéquation et complétude

Le théorème suivant est un des fondements majeurs du modèle formel des graphes conceptuels. Par l'intermédiaire de la traduction vers un fragment de la logique du premier ordre, notre modèle est alors doté d'une sémantique.

Théorème 3.5 (Adéquation et complétude de la projection pour Φ) *Si G et H sont deux graphes élémentaires sur un support énumérable $\mathcal{S}$. Alors $G \preceq_{\mathcal{S}} H$ si et seulement si $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$.*

Grâce à la remarque précédente, ce théorème s'exprime de façon équivalente par : la requête H se déduit d'une base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G} = (G_1, \ldots, G_k))$ si et seulement si $\Phi(\mathcal{S}), \ldots, \Phi(G_k) \models \Phi(H)$.

[Sowa, 1984] prouve l'adéquation, c'est à dire que si $G \preceq_{\mathcal{S}} H$, alors $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$. La démonstration de la complétude est faite dans [Mugnier, 1992, Chein and Mugnier, 1992], et utilise une équivalence entre la projection et la méthode de résolution. Notre méthode utilise directement l'évaluation dans un modèle. L'idée principale est de montrer l'équivalence entre la valuation à VRAI de $\Phi(G)$ dans un modèle et une projection (l'assignation) de G dans un graphe (élémentaire) qui encode le domaine et l'interprétation.

Valuation d'une formule dans un modèle

Pour valuer, dans un modèle $\mathcal{M}$, une formule F en logique du premier ordre (ici, sans symboles de fonctions), nous avons besoin d'un *domaine* $\mathcal{D}$, d'une *interprétation* $\mathcal{I}$ qui, à chaque symbole de prédicat t_n d'arité n fait correspondre une application $\mathcal{I}(t_n) : \mathcal{D}^n \rightarrow \{\text{VRAI, FAUX}\}$, et d'une *assignation* $\alpha : Var \rightarrow \mathcal{D}$.

Des règles de valuation nous permettent ensuite d'utiliser ce modèle pour donner une valeur de vérité à une formule quelconque. Les seules qui nous intéresseront, dans le fragment utilisé de la logique du premier ordre, sont :

1. si F et F' sont deux formules, $Val_{\mathcal{M}}(F \wedge F') = \text{VRAI}$ ssi $Val_{\mathcal{M}}(F) = \text{VRAI}$ et $Val_{\mathcal{M}}(F') = \text{VRAI}$;
2. si F est une formule et x une variable, $Val_{\mathcal{M}}(\exists x\ F) = \text{VRAI}$ si et seulement s'il existe une formule $F_{x \rightarrow x'}$, obtenue à partir de la formule F en remplaçant toutes les occurrences de x par la variable x', telle que $Val_{\mathcal{M}}(F_{x \rightarrow x'}) = \text{VRAI}$;
3. $Val_{\mathcal{M}}(t_i(x_1, \ldots, x_i)) = \text{VRAI}$ ssi $(\mathcal{I}(t_i))(\alpha(x_1), \ldots, \alpha(x_i)) = \text{VRAI}$;

4. $Val_{\mathcal{M}}(\forall x_1 \ldots \forall x_i(t_i(x_1, \ldots, x_i) \rightarrow t'_i(x_1, \ldots, x_i))) =$ VRAI ssi, pour tout tuple de taille i d'éléments du domaine $(d_1, \ldots, d_i) \in \mathcal{D}^i$, si $(\mathcal{I}(t_i))(d_1, \ldots, d_i) =$ VRAI, alors $(\mathcal{I}(t'_i))(d_1, \ldots, d_i) =$ VRAI.

Ces règles ne permettent pas de valuer n'importe quelle formule en logique du premier ordre, mais seront suffisantes dans le fragment que nous considérons. En effet, 1. et 4. suffisent à valuer une formule associée à un support, tandis que 1., 2. et 3. suffisent à valuer la formule associée à un graphe.

Construction du graphe d'interprétation

Considérons un *domaine* $\mathcal{D}$, et une *interprétation* $\mathcal{I}$ des symboles de prédicats associés à un support énumérable $\mathcal{S}$. Alors nous pouvons construire un graphe élémentaire[6] $\mathcal{G}(\mathcal{D}, \mathcal{I})$ sur $\mathcal{S}$, appelé *graphe d'interprétation* de la façon suivante :

- à chaque élément d du domaine, nous associons un sommet concept d ;
- pour chaque symbole de prédicat t_i, pour chaque tuple $(d_1, \ldots, d_i)$, nous ajoutons une relation r telle que $\tau(r) = t_i$ et $\gamma(r) = (d_1, \ldots, d_i)$ si et seulement si $(\mathcal{I}(t_i))(d_1, \ldots, d_i) =$ VRAI.

La présence d'une relation de type t_i entre $(d_1, \ldots, d_i)$ code donc l'interprétation à VRAI de $t_i(d_1, \ldots, d_i)$, et son absence l'interprétation à FAUX.

Graphe d'interprétation cohérent pour un support

Nous disons qu'un graphe d'interprétation $\mathcal{G}$ est *cohérent* pour un support $\mathcal{S}$ si et seulement si, pour chaque couple de types de relations (t, t') tel que $t \leq t'$, toute relation de type t dans $\mathcal{G}$ est jumelle d'une relation de type t'.

Lemme 3.1 (Graphe d'interprétation cohérent pour un support) *Soient $\mathcal{S}$ un support, $\mathcal{M} = (\mathcal{D}, \mathcal{I}, \alpha)$ un modèle interprétant les symboles de prédicat associés au support, et $\mathcal{G}(\mathcal{D}, \mathcal{I})$ un graphe d'interprétation. Alors $Val_{\mathcal{M}}(\Phi(\mathcal{S})) =$ VRAI si et seulement si $\mathcal{G}(\mathcal{D}, \mathcal{I})$ est cohérent pour $\mathcal{S}$.*

Preuve: Cette démonstration est immédiate : la valuation dans $\mathcal{M}$ de $\Phi(\mathcal{S})$ est vraie $\Leftrightarrow$ la valuation dans $\mathcal{M}$ de chacun des $\phi(t, t')$, pour $t \leq' t'$, est vraie (règle 1.) $\Leftrightarrow$ la valuation dans $\mathcal{M}$ de chacun des $\phi(t, t')$, pour $t \leq t'$, est vraie (transitivité) $\Leftrightarrow$ pour chaque $t \leq t'$, si $t(d_1, \ldots, d_i)$ s'interprète à VRAI, alors $t'(d_1, \ldots, d_i)$ s'interprète à VRAI (règle 4.) $\Leftrightarrow$ pour chaque $t \leq t'$, toute relation de type t dans $\mathcal{G}(\mathcal{D}, \mathcal{I})$ est jumelle d'une relation de type t' (construction du graphe d'interprétation) $\Leftrightarrow$ $\mathcal{G}(\mathcal{D}, \mathcal{I})$ est cohérent pour $\mathcal{S}$ (définition). □

[6] Ce graphe élémentaire est *infini*, et donc il ne rentre pas dans le cadre de travail que nous nous sommes fixé. Cependant, nous ne nous intéresserons qu'à des projections de graphes *finis* dans ce graphe. Nous pourrions donc considérer, à chaque fois, le graphe *fini* qui est l'image homomorphe de la projection courante. Nous ne nous intéressons donc qu'à des parties finies de ce graphe. Une autre méthode aurait été de prouver, *en logique*, que nous pouvons ne considérer que des domaines finis pour le fragment de la logique du premier ordre que nous utilisons. Nous avons cependant préféré montrer notre résultat en n'utilisant que la théorie des graphes.

Projection et valuation dans un modèle

Il ne nous reste maintenant qu'à préciser le rapport entre l'assignation des variables de $\Phi(G)$ et la projection de G dans un graphe d'interprétation.

Lemme 3.2 (Projection et valuation dans un modèle) *Soient $\mathcal{S}$ un support, $\mathcal{M} = (\mathcal{D}, \mathcal{I}, \alpha)$ un modèle interprétant les symboles de prédicat associés au support, $\mathcal{G}(\mathcal{D}, \mathcal{I})$ un graphe d'interprétation cohérent pour $\mathcal{S}$, G un graphe élémentaire sur $\mathcal{S}$ et $\Phi(G)$ la traduction de G obtenue par le biais d'une application terme : $V(G) \rightarrow Var$. Alors $Val_{\mathcal{M}}(\Phi(G)) =$ VRAI si et seulement si il existe une application σ : $Var(\Phi(G)) \rightarrow Var$ telle que $\alpha \circ \sigma \circ terme$ est une projection de G dans $\mathcal{G}(\mathcal{D}, \mathcal{I})$.*

Preuve: Ici aussi, nous procéderons par équivalences successives : $Val_{\mathcal{M}}(\Phi(G)) =$ VRAI
$\Leftrightarrow$ il existe une application σ des variables de $\Phi(G)$ dans Var telle que $Val_{\mathcal{M}}(\phi_{x \rightarrow \sigma(x)}(G)) =$ VRAI (cette application σ est la résultante des substitutions de variables de la règle 2., et $\phi_{x \rightarrow \sigma(x)}(G)$ est la formule obtenue à partir de $\phi(G)$ en remplaçant chaque occurrence d'une variable x par $\sigma(x)$)
$\Leftrightarrow$ il existe σ : $Var(\Phi(G)) \rightarrow Var$ telle que, pour chaque relation r de $U(G)$, avec $\tau(r) = t_i$ et $\gamma(r) = (x_1, \ldots, x_i)$, $Val_{\mathcal{M}}(t_i(\sigma(terme(x_1)), \ldots, \sigma(terme(x_i)))) =$ VRAI (règle 1., et association d'une arête avec sa traduction)
$\Leftrightarrow$ il existe $\mu = \sigma \circ terme : V(G) \rightarrow Var$ telle que, pour chaque relation r de $U(G)$, avec $\tau(r) = t_i$ et $\gamma(r) = (x_1, \ldots, x_i)$, $(\mathcal{I}(t_i))(\mu(x_1), \ldots, \mu(x_i)) =$ VRAI (règle 3., et composition)
$\Leftrightarrow$ il existe $\mu = \sigma \circ terme : V(G) \rightarrow Var$ telle que, pour chaque r de $U(G)$, avec $\gamma(r) = (x_1, \ldots, x_i)$, il existe $r' \in U(\mathcal{G}(\mathcal{D}, \mathcal{I}))$ avec $\tau(r') = \tau(r)$ et $\gamma(r') = (\alpha(\mu(x_1)), \ldots, \alpha(\mu(x_i)))$ (par construction du graphe d'interprétation)
$\Leftrightarrow$ il existe $\mu = \sigma \circ terme : V(G) \rightarrow Var$ telle que $\alpha \circ \mu$ est une projection de G dans $\mathcal{G}(\mathcal{D}, \mathcal{I})$ (cohérence du graphe d'interprétation, et définition). □

Remarquons immédiatement (nous en aurons besoin quand nous étudierons les graphes conceptuels simples), que la condition d'injectivité de *terme* n'est absolument pas nécessaire dans cette démonstration.

Démonstration du théorème

Nous utilisons maintenant les lemmes 3.1 et 3.2 pour démontrer le Th. 3.5. Bien que nous donnions la démonstration directe, nous donnons aussi une démonstration utilisant des conditions plus faibles, qui restera valide lorsque nous introduirons les marqueurs individuels dans les graphes (qui correspondront à des constantes en logique).

Preuve: Supposons maintenant G et H deux graphes élémentaires sur un support $\mathcal{S}$, et montrons que $G \preceq_{\mathcal{S}} H$ si et seulement si $\Phi(\mathcal{S}), \Phi(\mathcal{G}) \models \Phi(H)$.

($\Rightarrow$) Supposons $G \preceq_{\mathcal{S}} H$, alors montrons que, pour tout modèle $\mathcal{M}$ si $Val_{\mathcal{M}}(\Phi(\mathcal{S})) =$ VRAI et $Val_{\mathcal{M}}(\Phi(G)) =$ VRAI, alors $Val_{\mathcal{M}}(\Phi(H)) =$ VRAI. Comme $Val_{\mathcal{M}}(\Phi(\mathcal{S})) =$ VRAI, alors le graphe d'interprétation $\mathcal{G}(\mathcal{D}, \mathcal{I})$ est cohérent pour $\mathcal{S}$ (Lem. 3.1), et, en appliquant le Lem. 3.2, il existe une application σ : $Var(\Phi(G)) \rightarrow Var$ telle que $\alpha \circ \sigma \circ \text{terme}_G$ est

une projection de G dans $\mathcal{G}(\mathcal{D},\mathcal{I})$. Soit π une projection de H dans G (elle existe, car $G \preceq_{\mathcal{S}} H$), alors $\alpha \circ \sigma \circ \mathit{terme}_G \circ \pi$ est une projection de H dans $\mathcal{G}(\mathcal{D},\mathcal{I})$.

Nous pourrions conclure immédiatement en remarquant que, puisque terme_H est injective, $\alpha \circ \sigma \circ \mathit{terme}_G \circ \pi \circ \mathit{terme}_H^{-1} \circ \mathit{terme}_H$ est une projection et il existe bien $\sigma' = \sigma \circ \mathit{terme}_G \circ \pi \circ \mathit{terme}_H^{-1}$ telle que $\alpha \circ \sigma' \circ \mathit{terme}_H$ est une projection de H dans $\mathcal{G}(\mathcal{D},\mathcal{I})$, ce qui nous permet d'appliquer le Lem. 3.2.

Cependant, nous préférons utiliser une condition plus faible, qui restera encore valable lorsque nous intégrerons marqueurs individuels et liens de co-référence au modèle des graphes conceptuels simples :

Supposons que la projection π conserve les classes d'équivalence induites par la traduction terme_H, i.e.*, pour toute paire x, y de sommets de H, si $\mathit{terme}_H(x) = \mathit{terme}_H(y)$, on a nécessairement $\mathit{terme}_G(\pi(x)) = \mathit{terme}_G(\pi(y))$.*

Remarquons que l'injectivité de terme_H est un cas particulier de cette condition : si terme_H est injective, alors cette propriété est vérifiée par toute projection π. Cette condition plus faible nous permettra également de conclure. En effet, à toute variable x de $\Phi(H)$, nous pouvons associer une variable $\sigma'(x) \in \mathit{Var}$ de façon à ce que $\sigma' \circ \mathit{terme}_H = \sigma \circ \mathit{terme}_G \circ \pi$. Cette application $\sigma' : \mathit{Var}(\Phi(H)) \to \mathit{Var}$ est construite de la façon suivante : si $x \in \mathit{Var}(\Phi(H)$, alors soient $x_1, \ldots, x_k$ les sommets concepts de H tels que $\mathit{terme}_H(x_i) = x$. Choisissons un sommet concept dans cet ensemble, par exemple x_1. Alors $\sigma'(x) = \sigma(\mathit{terme}_G(\pi(x_1)))$. De par la condition que nous nous sommes donnée, $\sigma'(x)$ sera le même quel que soit le x_i choisi. Alors on a bien $\sigma' \circ \mathit{terme}_H = \sigma \circ \mathit{terme}_G \circ \pi$, et $\alpha \circ \sigma' \circ \mathrm{terme}_H$ est une projection de H dans $\mathcal{G}(\mathcal{D},\mathcal{I})$, ce qui nous permet de conclure grâce au Lem. 3.2. $\diamond$

($\Leftarrow$) Supposons maintenant que, pour tout modèle $\mathcal{M}$, si $\mathit{Val}_{\mathcal{M}}(\Phi(\mathcal{S})) = \text{VRAI}$ et $\mathit{Val}_{\mathcal{M}}(\Phi(G)) = \text{VRAI}$, alors $\mathit{Val}_{\mathcal{M}}(\Phi(H)) = \text{VRAI}$. Considérons maintenant un modèle particulier, que nous noterons $\mathcal{M}_G$, obtenu de la façon suivante :

- le domaine $\mathcal{D}$ est constitué de $|V(G)| + 1$ éléments, et on se donne une application injective $\delta : V(G) \to \mathcal{D}$, l'élément de $\mathcal{D}$ non atteint par δ est nommé ϵ ;
- pour chaque symbole de prédicat d'arité i t_i, pour chaque tuple $(d_1, \ldots, d_i) \in \mathcal{D}^i$, $(\mathcal{I}(t_i))(d_1, \ldots, d_i) = \text{VRAI}$ si et seulement si il existe $r \in U(G)$, avec $\gamma(r) = (x_1, \ldots, x_i)$ telle que $\tau(r) \leq_i t_i$ et $(\delta(x_1), \ldots, \delta(x_i)) = (d_1, \ldots, d_i)$;
- α est l'application qui, à chaque variable $v = \mathit{terme}_G(x)$ fait correspondre $\delta(x)$, et qui associe l'élément ϵ à tous les autres (nous supposons ici que terme_G est injective).

Alors le graphe $\mathcal{G}(\mathcal{D},\mathcal{I})$ est un graphe d'interprétation cohérent pour $\mathcal{S}$, et $\pi = \alpha \circ \mathit{terme}_G$ est une projection de G dans $\mathcal{G}(\mathcal{D},\mathcal{I})$. Donc, d'après le Lem. 3.1, on a $\mathit{Val}_{\mathcal{M}_G}(\Phi(\mathcal{S})) = \text{VRAI}$ et, d'après le Lem. 3.2, $\mathit{Val}_{\mathcal{M}_G}(\Phi(G)) = \text{VRAI}$. Donc, par hypothèse, $\mathit{Val}_{\mathcal{M}_G}(\Phi(H)) = \text{VRAI}$. Alors, encore d'après le Lem. 3.2, il existe une application $\sigma : \mathit{Var}(\Phi(H)) \to \mathit{Var}$ telle que $\alpha \circ \sigma \circ \mathit{terme}_H$ est une projection de H dans $\mathcal{G}(\mathcal{D},\mathcal{I})$.

Ici aussi, nous pourrions immédiatement conclure en affirmant que, comme $\mathit{terme}^{-1} \circ \alpha^{-1}$ est une projection de $\mathcal{G}(\mathcal{D},\mathcal{I})$ dans G, alors $\mathit{terme}_G^{-1} \circ \alpha^{-1} \circ \alpha \circ \sigma \circ \mathit{terme}_H = \mathit{terme}_G^{-1} \circ \sigma \circ \mathit{terme}_H$ est une projection de H dans G. Donc $G \preceq_{\mathcal{S}} H$.

Cependant, les graphes G et $\mathcal{G}(\mathcal{D},\mathcal{I})$ ne seront plus équivalents lorsque nous introduirons les marqueurs individuels (constantes) dans le modèle. Considérons plutôt l'application $\pi = terme_G^{-1} \circ \sigma \circ terme_H$ de H dans G, et la projection $\pi' = \alpha \circ \sigma \circ terme_H$ de H dans $\mathcal{G}(\mathcal{D},\mathcal{I})$. Alors pour toute relation $r \in U(H)$, avec $\gamma(r) = (x_1, \ldots, x_i)$, il existe $r' \in U(\mathcal{G}(\mathcal{D},\mathcal{I}))$ telle que $\tau(r') \leq \tau(r)$ et $\gamma(r') = (\pi'(x_1), \ldots, \pi'(x_i))$. Par construction, cette relation a été construite car il existe $r'' \in U(G)$, avec $\tau(r'') \leq \tau(r')$, et $\gamma(r'') = (terme_G^{-1}(\alpha^{-1}(\pi'(x_1))), \ldots, terme_G^{-1}(\alpha^{-1}(\pi'(x_i))))$. Donc $terme_G^{-1} \circ \alpha^{-1} \circ \pi' = \pi$ est bien une projection de H dans G. □

3.3.3 Discussion : $\mathcal{SG}$ et logique du premier ordre

Nous avons préféré inclure cette démonstration dans ce mémoire, plutôt que de référencer simplement la preuve existante de [Chein and Mugnier, 1992], pour plusieurs raisons.

Tout d'abord, nous avons trouvé intéressante cette façon de voir d'une même manière, par des projections, le mécanisme de déduction sur les graphes élémentaires et le plongement d'un graphe élémentaire dans un modèle (qui intègre à la fois la traduction des sommets concepts, la substitution des variables, et l'assignation de ces variables aux constantes du domaine). Le Lemme 3.2 identifie très clairement les relations entre une valuation à vrai dans un modèle et l'existence d'une projection.

De plus, nous avons également précisé dans cette démonstration en quoi l'injectivité de la traduction est importante, et quelles conditions plus faibles sont suffisantes. Les résultats d'équivalence et de complétude des modèles de graphes conceptuels simples obtenus en rajoutant marqueurs individuels et/ou liens de co-référence seront alors immédiats.

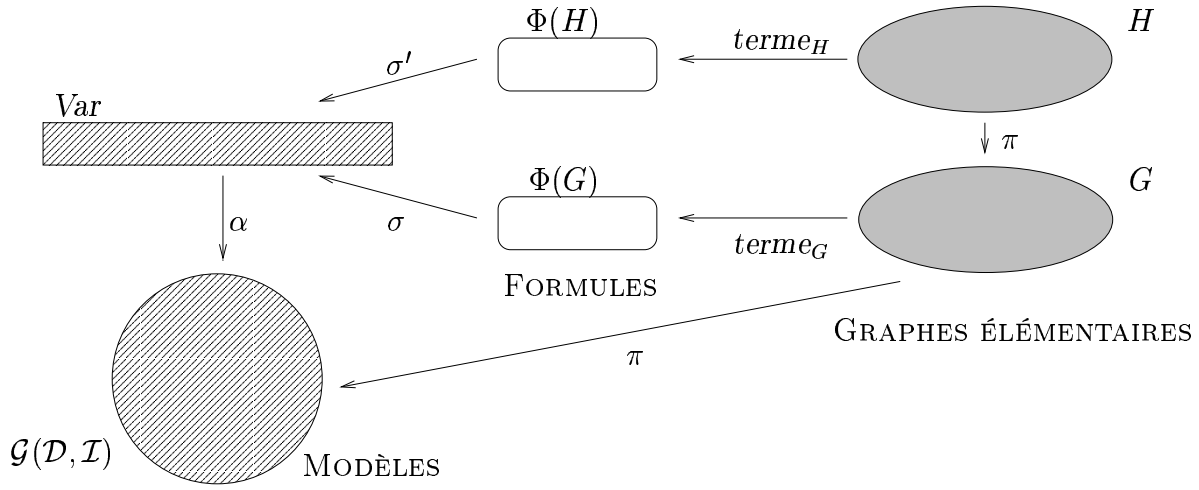


FIG. 3.10 – Graphes élémentaires, traduction logique, et modèles.

Nous souhaitons enfin mettre en lumière plusieurs points abordés dans cette section.

Projection et bases de données

Cette démonstration unifie aussi différents résultats : l'équivalence entre la projection de graphes élémentaires et la déduction logique (dans le fragment positif, conjonctif, existentiel de la logique du premier ordre), mais aussi la satisfaction d'une requête positive conjonctive en bases de données (en considérant le graphe d'interprétation comme une instance d'une base de données et le graphe G comme une requête conjonctive – sans constantes, mais nous les intégrerons à la section suivante, voir [Abiteboul et al., 1995]), ce qui nous ramène au théorème d'homomorphisme en bases de données [Chandra and Merlin, 1977]. De plus, en voyant H et G comme deux requêtes conjonctives, une projection de H dans G signifie que tous les modèles de H sont des modèles de G : nous avons donc une autre version de l'équivalence entre projection et inclusion de requêtes conjonctives (prouvée indirectement par l'équivalence entre inclusion de requêtes conjonctives et satisfaction d'un réseau de contraintes [Kolaitis and Vardi, 1998]).

Support conjonctif et conjonction de types

Une brève remarque à propos des graphes élémentaires compressés sur un support conjonctif $\mathcal{S}^*$: comme la $\mathcal{S}^*$-projection sur ces graphes est équivalente à la $\mathcal{S}$-projection sur leur expansion (Cor. 3.1), il est naturel de leur associer la même traduction logique. La traduction d'une relation r dont le type est $\{t_1, \ldots, t_p\}$ et dont les arguments sont $(x_1, \ldots, x_n)$ sera donc $\phi(r) = t_1(x_1, \ldots, x_n) \wedge \cdots \wedge t_p(x_1, \ldots, x_n)$.

La traduction d'une relation typée par un ensemble étant équivalente à une conjonction de traductions, nous noterons donc :

$$\{t_1, \ldots, t_n\} \equiv t_1 \sqcap \cdots \sqcap t_n$$

Chapitre 4

Autres formalismes de $\mathcal{SG}$

Sommaire

Nous avons pour l'instant présenté un modèle formel de raisonnement comme un formalisme de $\mathcal{SG}$: les graphes élémentaires. Ce formalisme est une version simplifiée de celui des graphes conceptuels simples [Sowa, 1984, Chein and Mugnier, 1992], qui en conserve les propriétés essentielles. Nous présentons maintenant plusieurs formalismes étendant les graphes conceptuels simples, et les comparons aux graphes élémentaires que nous avons définis et étudiés. Nous nous attacherons à montrer d'un côté, d'un point de vue représentation de connaissances, l'intérêt (abstraction, structuration, lisibilité) de ces différentes constructions, et d'un autre côté, d'un point de vue théorique, que tous ces formalismes sont équivalents. A la lumière des résultats que nous venons de prouver sur les graphes élémentaires, nous étudierons l'utilité des différentes restrictions imposées habituellement dans ces modèles.

Dans les graphes élémentaires, seules les relations sont typées. Avec le formalisme des *graphes conceptuels très simples* (utilisé comme formalisme de base dans [Baget, 2000]), nous rajouterons des types aux sommets concepts. Nous rajouterons ensuite d'un même élan les marqueurs individuels (ce qui aurait suffit à obtenir les *graphes conceptuels simples*),

et les liens de co-référence, puisqu'ils présentent exactement les mêmes problèmes (une traduction par Φ des sommets concepts en variables/constantes qui n'est pas injective), et les mêmes solutions (la considération d'une *forme normale*). Nous parlerons ensuite des *graphes conceptuels emboîtés* [Chein et al., 1998], où un sommet concept peut être décrit par un graphe.

Pour chacun de ces différents formalismes, nous adopterons dans notre présentation une démarche similaire :

1. rappeler les définitions usuelles des graphes et de leur projection dans ce formalisme ;
2. montrer l'équivalence de ce formalisme avec celui des graphes élémentaires ;
3. étendre la sémantique Φ aux objets de ce formalisme.

Nous préciserons à cette occasion ce que nous entendons par équivalence entre $\mathcal{SG}$ et le fragment positif, conjonctif, existentiel de la logique du premier ordre.

Notons que, dans chacun de ces formalismes, nous appellerons « projection » l'opération de raisonnement, même si elle est définie chaque fois de façon différente. Afin d'éviter une multiplication abusive des dénominations, nous demanderons au lecteur, à chaque occurrence de ce terme, de lui substituer la définition donnée pour les objets sur lesquels cette opération s'applique.

Formalisme	**Structure**	**Support**	**Intérêt**
Graphes élémentaires	hypergraphes	(T_R)	Algorithmes
Graphes conceptuels très simples	multigraphes bipartis	(T_C, T_R)	Formalisme intermédiaire
Graphes conceptuels simples	multigraphes bipartis + liens de co-référence	$(T_C, T_R, \mathcal{I})$	Représentation de connaissances
Graphes conceptuels emboîtés	multigraphes bipartis hiérarchisés + liens de co-référence	$(T_C, T_R, \mathcal{I})$	Représentation de connaissances

TAB. 4.1 – Structure des graphes employés dans les différents formalismes de $\mathcal{SG}$

La table 4.1 récapitule les différents formalismes de $\mathcal{SG}$, en indiquant la structure des objets utilisés, la composition du support, et les raisons que l'on a de s'y intéresser.

4.1 Graphes conceptuels très simples

Ce formalisme, employé comme modèle de base dans [Baget, 2000], est le plus proche des graphes élémentaires. Cas particulier, sans marqueurs individuels, du formalisme des graphes conceptuels simples de [Sowa, 1984], il ne diffère syntaxiquement des graphes élémentaires que par l'identification de certains types de relations d'arité 1 à des types de concepts. Nous mettrons cependant en valeur les différences entre la projection des graphes élémentaires (qui ne prend en compte que des sommets concepts) et la projection des graphes conceptuels (qui prend en compte aussi bien les sommets concepts que les sommets relations).

4.1.1 Définitions usuelles

Comme pour les graphes élémentaires, nous définirons le support, qui encode la connaissance ontologique, les graphes, qui encodent les connaissances factuelles, et la projection, opération de raisonnement.

Le support

Définition 4.1 (Support) *Un* support *de (graphes conceptuels très simples) est une structure* $\mathcal{S} = ((T_C, \leq_C), T_R = ((T_R^1, \leq_R^1), \ldots, (T_R^k, \leq_R^k)), \sigma)$ *telle que :*

- T_C *est l'ensemble de* types de concepts, *et* $\leq_C$ *un ordre (partiel) sur* T_C *;*
- T_R^i *est un ensemble de* types de relations *d'arité* i*, et* $\leq_R^i$ *un ordre (partiel) sur* T_R^i *;*
- *la* signature σ *associe à chaque relation d'arité* i *un élément de* $(T_C)^i$*.*

Notons que T_C comme les différents T_R^i admettent souvent chacun un élément maximum, qui sont notés respectivement $\top_C, \top_R^1, \ldots, \top_R^k$. [Mugnier, 2000] n'impose un maximum que pour l'ensemble T_C des types de concepts. Nous verrons, par l'équivalence de ce modèle avec celui des graphes élémentaires, qu'aucune de ces restrictions n'est nécessaire aux résultats théoriques. Cependant, l'existence d'un élément maximum est caractéristique des ontologies utilisées en représentation de connaissances. Bien qu'elle ne soit pas théoriquement nécessaire, la définition d'un élément maximum $\top$ pour chacune de ces hiérarchies doit être vue comme une « bonne habitude » de modélisation.

La *signature* (voir Chap. 2) est présente dans [Sowa, 1984]. Bien qu'intéressante d'un point de vue représentation de connaissances, elle n'a aucune influence sur les raisonnements. La signature σ associe à chaque type de relation d'arité i un tuple de taille i de types de concepts. Il s'agit d'une *contrainte* (au sens que nous définirons au Chap. 7) sur les graphes conceptuels que l'on est en droit de définir. En effet, un graphe conceptuel sera dit *bien formé* par rapport à σ si, pour toute relation r dont le type est t, avec $\sigma(t) = (t_1, \ldots, t_p)$, le type du $i^{\text{ième}}$ voisin de r est plus spécifique que t_i (il s'agit d'une propriété de *covariance* couramment utilisée dans les langages à objets). Un prétraitement linéaire des graphes conceptuels suffit à vérifier cette contrainte. Nous oublierons désormais cette signature, qui n'a aucune incidence sur les raisonnements (ou considérerons, d'une manière équivalente, que la signature de tout type de relation d'arité i est le tuple de taille i $(\top_C, \ldots, \top_C)$).

Les graphes

Les graphes conceptuels très simples ressemblent étrangement aux graphes élémentaires, mis à part que les sommets concepts sont typés et que les relations sont des sommets.

Définition 4.2 (Graphes conceptuels très simples) *Un* graphe conceptuel très simple *sur un support* $\mathcal{S}$ *est un multigraphe biparti étiqueté* $G = (V = (V_C, V_R), E, \epsilon)$*, où :*

- V_C *et* V_R *sont deux ensembles finis distincts, respectivement de* sommets concepts *et de* sommets relations *;*

- *E est un multiensemble d'arêtes entre les sommets de V_C et ceux de V_R, les arêtes incidentes à un sommet relation de degré k sont numérotées par ϵ de 1 à k ;*
- *ϵ associe à tout sommet x son type, un élément de T_C si x est un sommet concept, un élément de T_R^i si x est un sommet relation de degré i (ϵ obéit, si besoin est, aux contraintes définies par la signature : le graphe doit être bien formé).*

Les sommets concepts sont représentés par des rectangles, dans lesquels on inscrit leur type, et les sommets relations par des ovales, dans lesquels on inscrit aussi leur type. Les arêtes sont représentées par des traits reliant leurs deux extrémités, et on inscrit à côté d'elles leur numéro.

Projection de graphes conceptuels très simples

Le fait que les relations soient des sommets a une conséquence directe sur la projection, dont les objets seront aussi bien les sommets concepts que les sommets relations :

Définition 4.3 (Projection) *Soient H et G deux graphes conceptuels très simples sur un support $\mathcal{S}$. Une projection de H dans G est une application $\pi = (\pi_C, \pi_R)$ de $V(H)$ dans $V(G)$, où π_C est une application de $V_C(H)$ dans $V_C(G)$ et π_R est une application de $V_R(H)$ dans $V_R(G)$, qui préserve l'ordre sur les types et conserve les arêtes et leur numérotation,* i.e. *:*

- *pour tout sommet concept c de $V_C(H)$, $\epsilon(\pi_C(c)) \leq \epsilon(c)$;*
- *pour tout sommet relation r de $V_R(H)$, $\epsilon(\pi_R(r)) \leq \epsilon(r)$;*
- *pour toute arête $e \in E(H)$, notons c son extrémité dans $V_C(H)$, r son extrémité dans $V_R(H)$, et i son numéro. Alors il existe une arête numérotée i dans G dont les extrémités sont $\pi_C(c)$ et $\pi_R(r)$.*

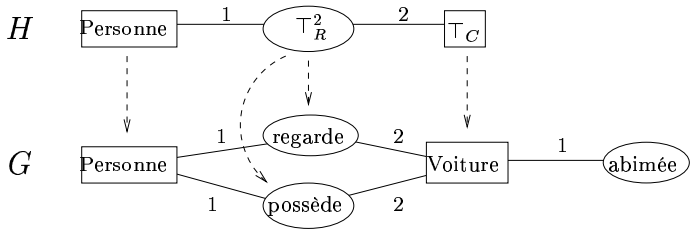


FIG. 4.1 – Les deux projections du graphe conceptuel très simple H dans G.

La FIG. 4.1 montre un exemple (très simple) de projection d'un graphe conceptuel très simple dans un autre. Le graphe H, considéré comme une requête, s'interprète par « est-ce-qu'une personne est en relation avec quelque-chose ? ». Le graphe G, quant à lui, s'interprète par : « une personne regarde sa voiture abimée. ». Il y a deux projections de H dans G.

4.1.2 Graphes conceptuels très simples et graphes élémentaires.

Considérons les graphes de la FIG. 4.1. On pourrait les voir ou bien comme des graphes conceptuels très simples ou bien comme des graphes élémentaires pour lesquels nous avons dessiné *certaines* des relations d'arité 1 à l'intérieur du sommet concept dont ils sont arguments.

Des problèmes équivalents

Un graphe élémentaire G peut être vu comme un graphe conceptuel très simple $G' = E2TS(G)$ particulier : il n'y a qu'un seul type de concept, $\top_C$, et les types de relations restent inchangés. Les sommets concepts de G sont les sommets concepts de G', et les relations de G sont les sommets relations de G'. Les arguments des relations de G déterminent les arêtes de G' : pour chaque relation r de G, pour chaque sommet concept x tel que $x = \gamma_i(r)$, il existe une arête numérotée i entre x et r dans G'. Il est immédiat de vérifier (ce n'est qu'une reformulation de la définition) qu'il existe une projection d'un graphe élémentaire H dans un graphe élémentaire G si et seulement si il existe une projection du graphe conceptuel très simple $H' = E2TS(H)$ dans le graphe conceptuel très simple $G' = E2TS(G)$. Voir, en effet, que si π est une projection de H dans G, il suffit de prendre une des applications *choix* associant à chaque relation de H un de ses certificats dans G, et $(\pi, choix)$ est bien une projection (de graphes conceptuels très simples) de H' dans G'. Réciproquement, si (π_C, π_R) est une projection de H' dans G', alors π_C est une projection de H dans G (dont π_R est une fonction de choix). Cette transformation est illustrée dans la FIG. 4.2.

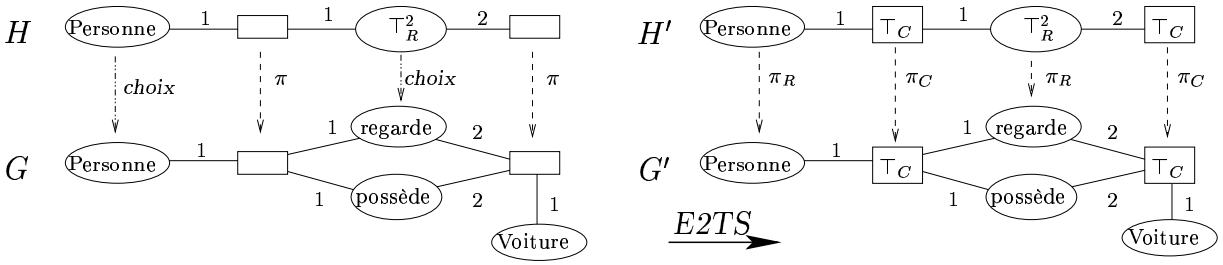


FIG. 4.2 – Transformation de graphes élémentaires en graphes conceptuels très simples.

De la même façon, on peut traduire un graphe conceptuel très simple G en un graphe élémentaire $G' = TS2E(G)$. Les types de relations d'arité 1 du nouveau support sont l'union des types de concepts et des types de relations d'arité 1 de l'ancien support. Les sommets concepts de G sont les sommets concepts de G', et, si un sommet concept x est typé par t, alors on rajoute dans G' une relation d'arité 1, de type t, incidente à x. Les sommets relations deviennent des relations de même type et de même arité, et les arêtes de G déterminent leurs arguments : si un sommet concept c est relié par une arête numérotée i à un sommet relation r, alors, dans G', $\gamma_i(r) = c$. Nous pouvons vérifier, comme ci-dessus, que cette transformation conserve l'existence des projections. Si (π_C, π_R) est une projection

de H dans G, alors π_C est une projection de H' dans G'. π_C comme π_R servent à construire la fonction *choix* qui justifie cette propriété : π_C détermine implicitement le choix pour les relations associées à un type de concept, et π_R pour les autres. Cette transformation est illustrée dans la FIG. 4.3.

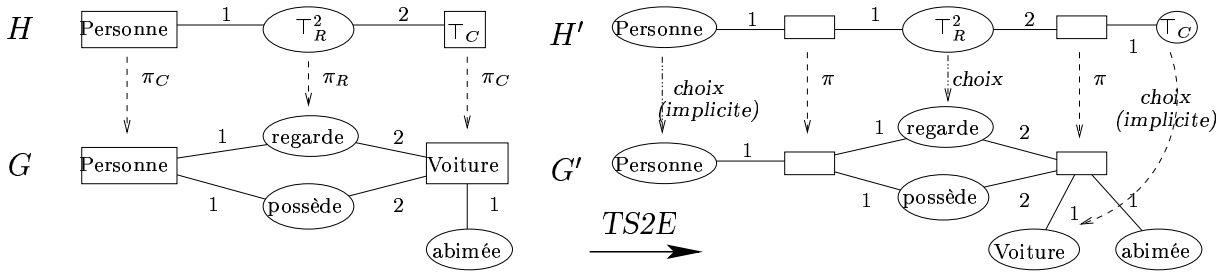


FIG. 4.3 – Transformation de graphes conceptuels très simples en graphes élémentaires.

Enfin, nous pouvons remarquer que $E2TS(TS2E(G)) \neq G$ (nous ne savons pas comment retrouver un type de concept à partir d'une relation d'arité 1).

Ces transformations posent deux problèmes, le premier concernant la non conservation du nombre de projections (car, dans un contexte de réponse à une requête, nous nous intéressons à toutes les solutions), et le deuxième concernant cette « perte de types de concepts ».

Nombre de projections

Nous avons vu que les transformations *E2TS* et *TS2E* conservent l'existence des projections, mais pas toutes les projections. Précisons un peu cette notion. Que ce soit par la transformation *E2TS* ou par la transformation *TS2E*, à toute projection π d'un graphe élémentaire H_e dans un graphe élémentaire G_e correspond un ensemble de projections de la forme (π, π_R). Comme π_R est déterminée par la fonction *choix* associée à π, il s'ensuit qu'à chaque projection de graphes élémentaires peut correspondre un nombre exponentiel de projections de graphes conceptuels très simples (voir Sect. 3.1.1).

D'un point de vue algorithmique, nous ne pouvons que bénéficier de cette réduction du nombre de projections : en traduisant les graphes conceptuels très simples dans le formalisme des graphes élémentaires, chaque projection que nous obtenons par un algorithme d'énumération représente une classe d'équivalence (potentiellement de taille exponentielle) des projections de graphes conceptuels très simples que nous aurions eu à énumérer. De plus, nous verrons à la Sect. 5.2.2 (inspirée des travaux de [Bessière et al., 1999]) que, même lorsque l'on ne s'intéresse pas à l'énumération des projections mais à la recherche d'une projection, la vision hypergraphe des graphes élémentaires permet d'implémenter des algorithmes plus efficaces.

Cependant, d'un point de vue représentation de connaissances, la perte de ces projections peut ne pas être souhaitable. Reprenons l'exemple de la FIG. 4.3. A la question « existe-t-il une personne en relation avec quelque-chose ? », la projection de graphes

conceptuels très simples apporte deux réponses « une personne regarde une voiture », et « une personne possède une voiture ». La projection de graphes élémentaires ne considère qu'une seule de ces réponses, qui sera déterminée par la fonction *choix* utilisée comme certificat de la projection. Il est à craindre que ceci ne satisfasse pas un utilisateur.

La première solution est immédiate : pour chaque projection π de graphe élémentaire obtenue, nous pouvons générer l'ensemble des fonctions *choix* pour π, et ainsi obtenir toutes les projections de graphes conceptuels très simples. La deuxième solution, que nous allons aborder maintenant, est d'utiliser la conjonction de types et les graphes minimaux (Sect. 3.2.1), afin de répondre « une personne (possède et regarde) une voiture », réponse encore plus satisfaisante puisqu'elle fait le lien entre les deux réponses précédentes.

Traduction des types de concepts et conjonction de types

Il se pose un autre problème concernant les transformations *E2TS* et *TS2E* que nous avons présentées. En effet, pour un graphe conceptuel très simple donné, $E2TS(TS2E(G)) \neq G$. Ce qui veut dire que, si nous éditons un graphe conceptuel très simple, calculons les projections dans le formalisme des graphes élémentaires, puis retraduisons le résultat (les images homomorphes) dans le formalisme des graphes conceptuels très simples pour les visualiser, nous n'avons plus les mêmes graphes...

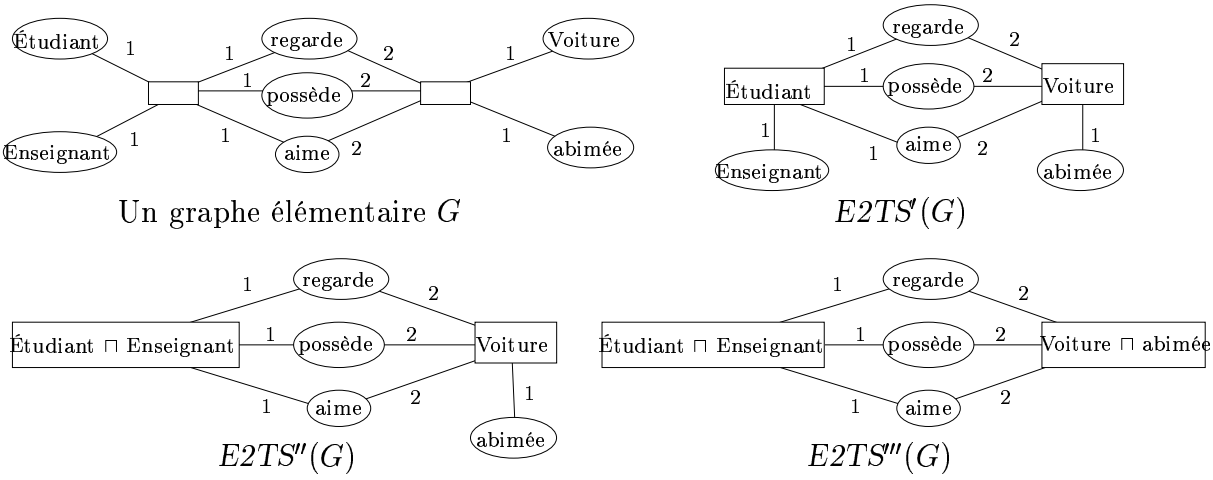


FIG. 4.4 – Transformations utilisant les données du support et/ou les types conjonctifs.

La FIG. 4.4 illustre trois transformations alternatives des graphes élémentaires en graphes conceptuels très simples qui, si elles conservent aussi toutes les projections (à une classe d'équivalence près), vérifient cette fois les propriétés $TS2E \circ E2TS = Id$, et $E2TS \circ TS2E = Id$.

Dans la première, nous utilisons les connaissances encodées dans le support des graphes conceptuels très simples : nous y lisons quels types de relations d'arité 1 doivent être considérés comme des types de concepts. Ainsi, si chaque sommet concept de G est incident à une et une seule relation d'arité 1 étiquetée par un type correspondant à un type de T_C,

nous pouvons donner au sommet concept le type de cette relation (transformation *E2TS'*). Dans l'exemple, seul *Étudiant* est un type de concept de T_C, et si *Enseignant* l'était aussi, il n'y aurait aucun moyen de traduire le graphe élémentaire G.

Avec la seconde transformation, *E2TS"*, nous introduisons dans le formalisme des graphes conceptuels très simples la conjonction de types. Si *Étudiant* et *Enseignant* sont deux types de T_C, alors le type du sommet concept obtenu est *Étudiant* $\sqcap$ *Enseignant*. La comparaison des types de concepts conjonctifs se fait alors comme indiqué dans la Sect. 3.2.1. Remarquons qu'il y a encore une limitation aux graphes élémentaires que l'on est en droit de traduire : il faut que tous les sommets concepts d'un graphe élémentaire G soient incidents à au moins une relation typée par un type de T_C. Cependant, nous pouvons éliminer cette restriction en considérant le type vide (obtenu par une conjonction de 0 types) et $\top_C$ comme équivalents. L'équivalence entre les graphes élémentaires et les graphes conceptuels très simples montre que l'introduction des types conjonctifs n'est pas un réel enrichissement du modèle, et qu'il est implicitement présent de par la possibilité d'utiliser des relations unaires jumelles.

Cela veut-il dire que, grâce à la possibilité de représenter la conjonction de types de concepts, les relations unaires sont désormais superflues ? En effet, il est possible d'intégrer toutes les relations unaires au type du sommet concept qui leur est incident, ce que fait la transformation *E2TS"'*. En termes de représentation de connaissances, nous ne pensons pas que ceci soit désirable. Le type conjonctif *Voiture* $\sqcap$ *abimé* de l'exemple ne nous semble pas particulièrement naturel. Bien que certains utilisateurs préfèrent un type conjonctif *Voiture* $\sqcap$ *Objet abimé*, d'autres préfèrent un sommet concept de type *Voiture*, sur lequel porte une relation d'arité 1. Pour répondre aux différents besoins de modélisation, il apparaît donc nécessaire de garder la possibilité d'avoir à la fois des types de concepts et des relations d'arité 1, représentés de façon distincte. Cependant, d'un point de vue algorithmique, le graphe minimal d'un graphe élémentaire sur lequel nous calculerons les projections (Chap. 5) va amalgamer, sans considération pour leur statut épistémologique, types de concepts et types de relations d'arité 1.

Enfin, il serait possible de la même manière de représenter chaque classe de relations jumelles par une seule relation, dont le type est la conjonction des types des relations de la classe. Si on considère le critère de lisibilité du dessin d'un graphe, ceci ne semble pas être une bonne idée. Cependant, ceci permet de diminuer le nombre de projections de graphes conceptuels très simples (il sera identique au nombre de projections de graphes élémentaires), et de faire le lien entre les réponses différentes engendrées par des relations jumelles, comme nous en avons indiqué le besoin. Ne souhaitant pas trancher entre ces avantages/inconvénients (lisibilité des données et lisibilité des réponses aux requêtes), nous maintiendrons le *status quo ante* et ne considérerons la conjonction de types que pour les types de concepts. D'un point de vue algorithmique (Chap. 5), nous utiliserons cette conjonction de types de relations puisque nous considérerons les graphes minimaux.

4.1.3 Une même sémantique logique

La formule logique $\Phi(\mathcal{S})$ associée à un support et la formule logique $\Phi(G)$ associée à un graphe conceptuel très simple G sont définies de la façon suivante :

Formule logique associée au support

La définition de cette formule est identique à celle qui était donnée dans le cas des graphes élémentaires (Sect. 3.3). Nous en donnons toutefois l'adaptation (immédiate) aux graphes conceptuels très simples afin que ce chapitre puisse se lire de façon autonome.

Soit $\mathcal{S} = ((T_C, \leq_C), T_R = ((T_R^1, \leq_R^1), \ldots, (T_R^k, \leq_R^k)), \sigma)$ un support donné par les relations de couverture $\leq_C{}', \leq_R^1{}', \ldots, \leq_R^k{}'$. Nous pouvons construire une traduction $\Phi(\mathcal{S})$ de la façon suivante :

1. à chaque type de concept t nous associons un symbole de prédicat d'arité 1, que nous noterons de la même façon t ;
2. à chaque type de relation u d'arité n dans le support nous associons un symbole de prédicat d'arité n, que nous noterons de la même façon u ;
3. nous disposons d'un ensemble dénombrable de variables $Var = \{x_1, \ldots, x_p, \ldots, \}$;
4. à chaque couple (t, t') de types de concepts tel que $t \leq_C' t'$, nous associons la formule $\phi(t, t') = \forall x(t(x) \rightarrow t'(x))$;
5. pour chaque T_R^i de $\mathcal{S}$, à chacun des couples (t, t') tel que $t \leq_R^i{}' t'$ nous associons la formule $\phi(t, t') = \forall x_1 \ldots \forall x_i(t(x_1, \ldots, x_i) \rightarrow t'(x_1, \ldots, x_i))$;
6. une formule $\Phi(\mathcal{S})$ qui traduit le support est la conjonction de ces formules $\phi(t, t')$.

Formule logique associée à un graphe conceptuel très simple

De la même façon, l'adaptation aux graphes conceptuels très simples de la sémantique logique des graphes élémentaires est immédiate. Soit G un graphe élémentaire sur un support $\mathcal{S}$.

1. on se donne une application *injective*, *terme* : $V_C(G) \rightarrow Var$, mais, pour alléger les notations, nous pourrons noter de la même manière x, un sommet concept de G, et $x = terme(x)$, la variable qui lui est associée ;
2. chaque sommet concept c de G, avec $\epsilon(x) = t$ est traduit par la formule $\phi_C(c) = t(c)$ (si le type de c est un type conjonctif $T = t_1 \sqcap \cdots \sqcap t_q$, sa traduction est $\phi_C(c) = t_1(c) \wedge \cdots \wedge t_q(c)$) ;
3. chaque sommet relation r de G, dont les arguments sont, dans cet ordre, $c_1, \ldots, c_p$ et dont le type est u est traduit par la formule $\phi_R(r) = u(c_1, \ldots, c_p)$;
4. nous notons $\phi(G)$ la conjonction des $\phi_C(c)$, pour $c \in V_C(G)$ et des $\phi_R(r)$ pour $r \in V_R(G)$;
5. une formule $\Phi(G)$ qui traduit le graphe G est la fermeture existentielle de la formule $\phi(G)$.

Nous pouvons par exemple associer au graphe G de la FIG. 4.3 la formule :

$$Personne(x) \wedge Voiture(y) \wedge regarde(x, y) \wedge possède(x, y) \wedge abimé(y)$$

Adéquation et complétude

Nous retrouvons à l'identique, dans le modèle des graphes conceptuels très simples, le théorème d'adéquation et complétude de [Sowa, 1984, Chein and Mugnier, 1992] :

Corollaire 4.1 *Soit $\mathcal{S}$ un support et H et G deux graphes conceptuels très simples sur $\mathcal{S}$. Alors il existe une projection de H dans G si et seulement si $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$.*

Preuve: Soit G un graphe conceptuel très simple sur un support $\mathcal{S}$. Considérer le support $\mathcal{S}' = TS2E(\mathcal{S})$ et le graphe élémentaire $G' = TS2E(G)$. Vérifier que $\Phi(\mathcal{S}) = \Phi(\mathcal{S}')$, et que $\Phi(G) = \Phi(G')$. Comme nous avons vu que les projections de H dans G sont identiques (à une classe d'équivalence près) aux projections de $TS2E(H)$ dans $TS2E(G)$, nous n'avons plus qu'à appliquer le théorème d'adéquation et de complétude démontré dans le cadre des graphes élémentaires (Th. 3.5) pour conclure. □

4.2 Graphes conceptuels simples

Un des intérêts de la représentation par des graphes conceptuels réside dans la lisibilité des graphes. Or cet avantage disparaît rapidement dès lors que les graphes deviennent grands. Un grand graphe ne peut être visualisé que par une succession de graphes plus petits, ce qui est impossible, à moins qu'il ne soit composé de plusieurs composantes connexes, dans le modèle des graphes conceptuels élémentaires.

Afin de pouvoir éditer un graphe « par morceaux », il est donc nécessaire de pouvoir indiquer que deux sommets concepts représentent la même entité. Ceci pourra se faire en nommant l'entité par un marqueur individuel (que l'on peut voir comme une constante en logique), ou en reliant deux sommets concepts par un lien de co-référence.

L'intérêt et l'expressivité des liens de co-référence ne réside pas seulement dans la lisibilité des graphes. Comme on le verra à la Sect. 4.3, avec les graphes conceptuels emboîtés, les liens de co-référence permettent de représenter des situations qui ne seraient pas exprimables dans ce formalisme sinon.

Nous verrons que marqueurs individuels et liens de co-référence présentent exactement les mêmes problèmes et nécessitent exactement les mêmes solutions. Nous les rajoutons donc en même temps au formalisme des graphes conceptuels très simples. Le formalisme ainsi obtenu est celui des graphes conceptuels simples.

4.2.1 Mise à jour des définitions

Nous voyons maintenant comment la prise en compte des marqueurs individuels et des liens de co-référence modifie les définitions que nous avons données pour les graphes

conceptuels très simples. Nous insistons aussi sur les intérêts additionnels de la conjonction de types dans ce formalisme.

Support

Le support $\mathcal{S} = (T_C, T_R, \mathcal{I})$ comprend maintenant un ensemble $\mathcal{I}$ de marqueurs individuels, et nous utilisons un marqueur particulier, dit générique, noté $*$, indépendant du support. Nous considérons une relation d'ordre sur $\mathcal{I} \cup \{*\}$, telle que le marqueur générique $*$ est plus général que tous les marqueurs individuels de $\mathcal{I}$, et les marqueurs individuels de $\mathcal{I}$ sont deux à deux incomparables.

Habituellement, une relation de conformité associe à chaque marqueur individuel un type de T_C. Cette relation de conformité a deux objectifs. D'une part, cette relation « explique » le marqueur individuel, en explicitant quel doit être le type d'un sommet concept étiqueté par ce marqueur. D'autre part, cette relation de conformité assure que l'on pourra fusionner deux sommets concepts ayant même marqueur individuel (cependant, cette nécessité disparaît lorsque l'on utilise des types conjonctifs).

Graphes

Nous ne donnons dans la définitions suivante des graphes conceptuels simples aucune restriction concernant les sommets concepts que l'on peut étiqueter par un même marqueur individuel ou déclarer appartenir à une même classe de co-référence. Des restrictions seront cependant nécessaires, et nous discuterons des différentes possibilités au cours de cette section.

Définition 4.4 (Graphes conceptuels simples) *Un* graphe conceptuel simple *sur un support $\mathcal{S}$ est un multigraphe biparti étiqueté $G = (V = (V_C, V_R), E, \epsilon, \text{co-ref})$, où :*

- *V_C et V_R sont deux ensembles finis distincts, respectivement de* sommets concepts *et de* sommets relations *;*
- *E est un multiensemble d'arêtes entre les sommets de V_C et ceux de V_R, les arêtes incidentes à un sommet relation de degré k sont numérotées par ϵ de 1 à k ;*
- *ϵ associe à tout sommet concept c un type de T_C et un marqueur de $\mathcal{I} \cup \{*\}$; et à tout sommet relation r un type de T_R^i, où i est le degré de r.*
- *co-ref est une classe d'équivalence sur les sommets concepts de G.*

Les sommets concepts sont maintenant étiquetés par un type de concept, et par un marqueur, qui peut être individuel ou générique. Dans le premier cas, on dit que c'est un sommet concept individuel, sinon, que c'est un sommet concept générique. Si le sommet concept est individuel, alors son type doit être celui associé à son marqueur par la relation de conformité. Dans le dessin du graphe, l'étiquette d'un sommet concept est représentée sous la forme `type : marqueur`. Pour alléger les notations, nous pourrons représenter l'étiquette d'un sommet concept générique sous la forme `type`. Plusieurs sommets concepts individuels ayant le même marqueur représentent la même entité.

De la même façon, pour représenter que plusieurs sommets concepts génériques représentent la même entité, nous pouvons indiquer (de façon externe aux sommets, il ne s'agit plus d'une étiquette) que ces sommets concepts appartiennent à une même classe d'équivalence, qui est appelée classe de co-référence. Deux méthodes sont habituellement utilisées pour représenter que deux sommets sont co-référents : tout d'abord, nous pouvons utiliser des marqueurs génériques « nommés », comme « $*123$ ». Deux sommets ayant le même marqueur générique nommé appartiennent à la même classe de co-référence. Un tel mode de représentation est nécessaire lorsqu'un grand graphe est représenté par plusieurs graphes plus petits, qui ne sont pas représentables en même temps sur une même feuille de papier ou un même écran d'ordinateur.

Un autre mode de représentation, lorsque les graphes sont représentables sur un même support (ce qui sera toujours le cas pour les petits graphes présentés dans cette thèse), est de représenter la relation de co-référence par des traits en pointillés reliant les sommets co-référents. Notons que, pour une classe de co-référence de taille k, il n'est pas nécessaire de représenter $k(k-1)/2$ traits entre chaque paire de sommets co-référents. Nous pouvons dessiner seulement k-1 traits, qui suffisent (par l'intermédiaire de la fermeture réflexo-transitive de la relation représentée) à définir la relation de co-référence.

Enfin, de façon à éviter tout problème de fusion de sommets, on impose habituellement à deux sommets concepts génériques co-référents d'avoir le même type. Notons que cette restriction est aussi imposée aux sommets concepts individuels ayant même marqueur, par l'intermédiaire de la relation de conformité. Nous verrons au cours de cette section que ce critère peut être relâché, et étudierons différentes possibilités de fusion de classes plus larges de sommets concepts.

[Chein et al., 1998] introduisent la notion de classe de co-identité, qui unifie les notions de co-référence et de sommets concepts individuels ayant même marqueur :

Définition 4.5 (Classes de co-identité) *Deux sommets concepts d'un graphe conceptuel simple G appartiennent à la même classe de co-identité (sont co-identiques) si et seulement si ce sont deux sommets concepts individuels ayant même marqueur, ou si ce sont deux sommets concepts génériques co-référents.*

Considérons le graphe conceptuel simple de la FIG. 4.5. Les sommets concepts individuels x et y appartiennent à la même classe de co-identité (car ils ont même marqueur individuel), et représentent donc la même entité (*Paul*). Les sommets concepts génériques z et t sont co-référents, car ils ont le même marqueur générique nommé, et les sommets t et u sont co-référents, car ils sont explicitement reliés par un lien de co-référence. Par transitivité, les sommets concepts génériques z, t et u appartiennent à la même classe de co-référence, et donc la même classe de co-identité : ils représentent la même *Voiture*. Les zones grisées matérialisent les classes de co-identité.

Projection

La définition de la projection est, elle aussi, mise à jour pour prendre en compte les sommets concepts individuels. Il s'agit d'une projection au sens des graphes conceptuels

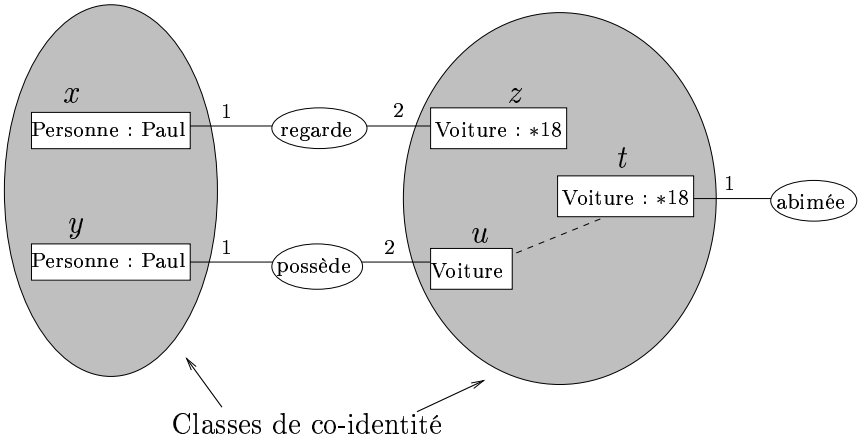


FIG. 4.5 – Graphe conceptuel simple et représentations de la co-référence.

très simples, qui doit de plus respecter l'ordre sur les marqueurs et préserver les classes de co-référence.

Définition 4.6 (Projection) *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. Une projection de H dans G est une application $\pi = (\pi_C, \pi_R)$ de $V(H)$ dans $V(G)$, telle que :*

- *pour tout sommet concept c de $V_C(H)$, $\epsilon(\pi_C(c)) \leq \epsilon(c)$ (ou $\leq$ est l'ordre produit obtenu à partir de celui sur les types de concepts et de celui sur les marqueurs) ;*
- *pour toute paire (c, c') de sommets concepts génériques de H appartenant à la même classe de co-référence, $\pi_C(c)$ et $\pi_C(c')$ appartiennent à la même classe de co-identité dans G ;*
- *pour tout sommet relation r de $V_R(H)$, $\epsilon(\pi_R(r)) \leq \epsilon(r)$;*
- *pour toute arête $e \in E(H)$, notons c son extrémité dans $V_C(H)$, r son extrémité dans $V_R(H)$, et i son numéro. Alors il existe une arête numérotée i dans G dont les extrémités sont $\pi_C(c)$ et $\pi_R(r)$.*

La FIG. 4.6 illustre une projection d'un graphe conceptuel H dans un graphe conceptuel G. Nous y voyons que la conservation des classes de co-identité restreint le nombre de projections : ainsi, le sommet concept a ne peut se projeter dans le sommet concept v, sinon, pour les deux sommets co-référents a et b, $\pi_C(a) = v$ et $\pi_C(b)$, qui est nécessairement y, ne seraient pas dans la même classe de co-identité. Notons aussi que le sommet concept générique c, dont le marqueur est un marqueur individuel nommé, se projette dans un sommet concept générique ayant un marqueur générique nommé différent : en effet, le nom d'un marqueur générique est local à un graphe, tandis qu'on peut voir un marqueur individuel comme un nom global, valable pour tous les graphes définis sur un même support.

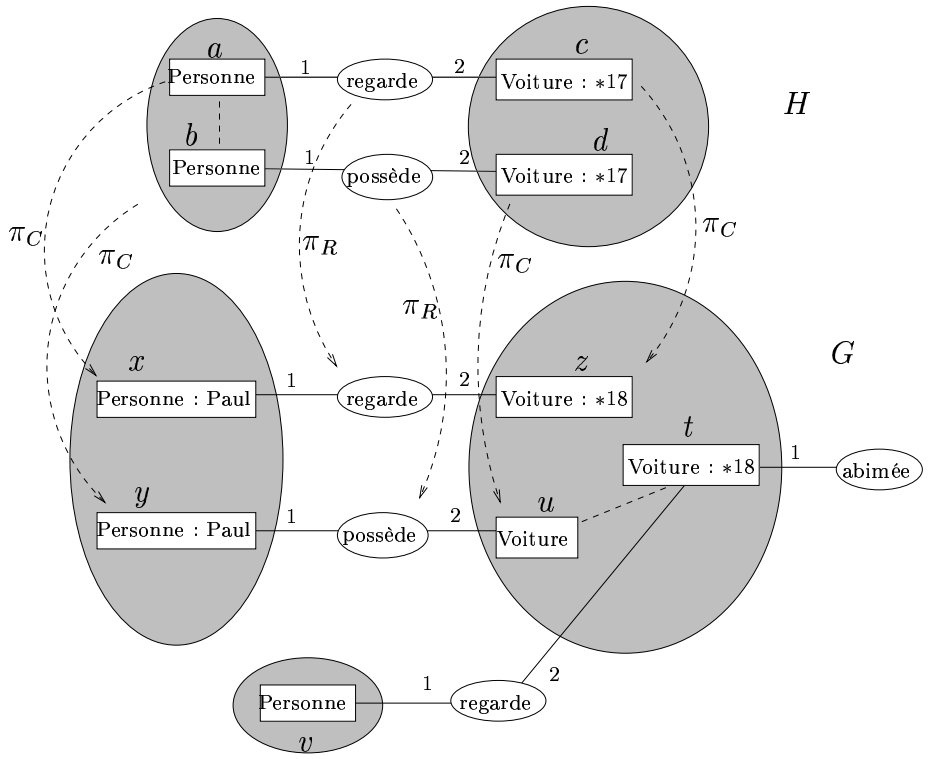


FIG. 4.6 – Projection de graphes conceptuels simples et conservation de la co-identité.

4.2.2 Forme normale et graphes élémentaires

Nous avons présenté l'ajout de la co-identité (terme regroupant à la fois l'introduction des marqueurs individuels et celle des liens de co-référence) comme une méthode permettant de représenter « par morceaux » un grand graphe, dont les sommets sont obtenus en fusionnant tous les sommets appartenant à une même classe de co-identité. Cette opération est appelée la mise sous forme normale d'un graphe conceptuel simple. Remarquons immédiatement que la fusion de sommets concepts quelconques ne peut se faire que sous certaines conditions.

Problèmes de fusionnabilité d'une classe de co-identité

Définition 4.7 *Soit $\mathcal{S}$ un support, et G un graphe. Alors un ensemble de sommets concepts $\{c_1, \ldots, c_p\}$ est dit* fusionnable *si on peut construire sur $\mathcal{S}$ un sommet C dont l'étiquette est la borne inférieure des étiquettes de $c_1, \ldots, c_p$.*

Intuitivement, il faut pouvoir déterminer de façon unique un sommet concept C tel que $c_1, \ldots, c_p$ peuvent se projeter dans $\mathcal{C}$.

Les restrictions que nous avons données à la relation de co-identité (relation de conformité, et même type pour tous les sommets concepts génériques appartenant à une même classe d'équivalence) sont suffisantes pour que toute classe de sommets co-identiques soit

fusionnable : le sommet concept obtenu a le même type et le même marqueur que tous les sommets de la classe. Nous pouvons même relâcher ces restrictions, et obtenir une condition nécessaire et suffisante :

Propriété 4.1 *Soit $\mathcal{S}$ un support, et G un graphe. Alors un ensemble de sommets concepts $\{c_1, \ldots, c_p\}$ est fusionnable si et seulement si l'ensemble des marqueurs individuels contient au plus un marqueur, et si l'ensemble des types de concepts présents dans cette classe contient un type inférieur à tous les autres.*

Preuve: Chacun des sens de cette équivalence est immédiat :
($\Leftarrow$) Le marqueur du sommet concept C que nous construisons est l'unique marqueur individuel, s'il existe, présent dans l'ensemble de sommets concepts, et est le marqueur générique dans le cas contraire. Le type de C est le type minimum pour les types de cet ensemble. Vérifier que ce sommet C vérifie le critère de la Def. 4.7 est immédiat.
($\Rightarrow$) Évident si l'ensemble de marqueurs individuels de cet ensemble est de taille supérieure ou égale à 2 : soient a et b ces marqueurs, on ne peut pas trouver de terme x ni de prédicats tels que $t(x) \equiv t'(a) \wedge t'(b)$. Voir de même que le plus petit type équivalent aux types de l'ensemble est la conjonction des éléments minimaux de cet ensemble. Cette conjonction n'est équivalente à un type unique que si elle est de taille 1, *i.e.* contient un minimum. □

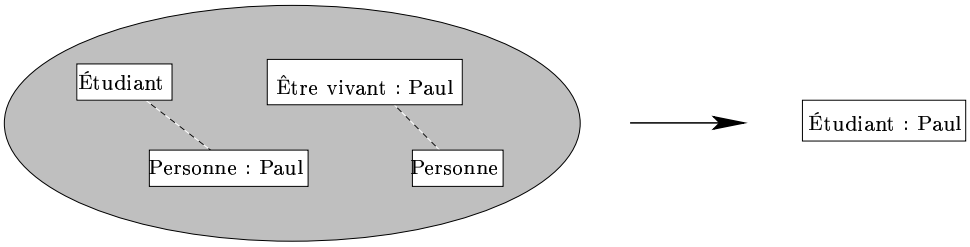


FIG. 4.7 – Co-identité utilisant un critère relâché de fusion des sommets.

Ce critère de fusionnabilité, plus large que celui proposé habituellement, permet de représenter des liens de co-référence qui seraient interdits par la définition usuelle (voir FIG. 4.7).

Afin de relâcher le critère de fusionnabilité, d'autres auteurs considèrent que les types de concepts du support $\mathcal{S}$ doivent être ordonnés par un treillis (un inf-demi-treillis serait cependant suffisant). Dans ce cas, le type donné au sommet concept obtenu par la fusion d'un ensemble de sommets concepts est la borne inférieure des types présents dans cet ensemble (qui n'appartient pas forcément à cet ensemble). Acceptable d'un point de vue théorique (en toute rigueur, il faudrait cependant modifier la sémantique Φ du support et y ajouter des formules du type $\forall x\ ((\mathit{Enseignant}(x) \wedge \mathit{Étudiant}(x)) \rightarrow \mathit{Enseignant\text{-}Étudiant}(x))$), cette interprétation d'une hiérarchie de types par un treillis (*lattice-theoretic interpretation*) est souvent critiquée dans un contexte de représentation de connaissances [Beierle et al., 1992,

Wermelinger and Lopes, 1994, Cao, 1999].[1] Parmi les arguments avancés contre cette interprétation (qui n'est d'ailleurs pas celle de la sémantique Φ de [Sowa, 1984]), nous pouvons citer :

- la difficulté, pour le concepteur du support, d'organiser les types en un treillis ;
- la nécessité de définir une borne inférieure, même lorsqu'elle n'a « aucun sens » ;

Le deuxième point est souvent résolu par l'utilisation du type de concept absurde, $\bot_C$, qui ne doit être présent dans aucun graphe. Cette solution nous semble dangereuse, puisqu'elle associe une signification particulière à un type, et introduit dans le formalisme des graphes simples la notion de validité. Bien qu'on puisse voir cette notion de validité comme un critère syntaxique tant qu'on reste dans le modèle $\mathcal{SG}$ (de la même manière que, par exemple, la relation de conformité), ce ne sera plus le cas, par exemple, avec les règles de $\mathcal{SR}$, où la validité devra être vérifiée à chaque étape de la résolution. Nous souhaitons éviter cette introduction d'une notion de validité dans un modèle qui ne la nécessite pas, et estimons que ce problème est du ressort des contraintes (dans $\mathcal{SGC}$ et ses modèles plus généraux).

Comme [Cao, 1999], nous préférerons garder une interprétation de la hiérarchie des types par un ordre (*order-theoretic interpretation*), ce qui est possible en utilisant les types conjonctifs. En effet, un ensemble de sommets concepts $\{c_1, \ldots, c_p\}$ contenant au plus un marqueur individuel et les types de concepts $\{t_1, \ldots, t_p\}$ peut être fusionné en un sommet concept dont le type est $t_1 \sqcap \cdots \sqcap t_p$ (ou le type irredondant associé à ce type, voir Sect. 3.2.1). Grâce aux types conjonctifs, nous relâchons complètement le critère de fusionnabilité [2], tout en préservant l'interprétation par ordre du support. Nous avons vu à la section précédente que l'ajout de la conjonction de types, si elle apporte des facilités quant à la modélisation (et nous en avons ici une autre confirmation), n'augmente pas l'expressivité du modèle, et nous verrons dans le Chap. 5 que cet ajout n'augmente pas le coût de la recherche des projections. Cependant, nous verrons à la Sect. 6.3 que notre optimisation du mécanisme d'application de règles est sérieusement handicapée par la considération d'un support conjonctif.

Forme normale

Définition 4.8 (Graphe conceptuel simple bien formé) *Un graphe conceptuel simple sur un support $\mathcal{S}$ est dit* bien formé *si et seulement s'il est muni d'une relation de co-référence telle que toutes les classes de co-identité soient fusionnables.*

Grâce à cette définition, nous n'avons plus à considérer la façon dont la fusion doit être faite, ni quel critère de fusionnabilité est envisagé. Plusieurs solutions ont été proposées et discutées dans la section précédente, et nous préférons laisser la liberté de choix au lecteur.

[1] Notons que, contrairement à ce qui est affirmé dans [Cao, 1999], l'interprétation de la hiérarchie de types de concepts dans [Chein and Mugnier, 1992, Salvat, 1997] n'est pas celle du treillis : les restrictions qu'ils se donnent (relation de conformité, mêmes types dans chaque classe de co-référence) permettent la fusion, comme nous l'avons vu, sans recourir à cette interprétation.

[2] Qui n'impose que la présence d'au plus un marqueur individuel dans une classe de co-identité. Même cette condition est supprimée dans [Baget, 2000], qui considère les marqueurs individuels comme des *alias*.

Nous pouvons maintenant utiliser cette abstraction pour définir la forme normale, et la mise sous forme normale d'un graphe.

Définition 4.9 (Forme normale) *Un graphe conceptuel simple sur un support $\mathcal{S}$ est dit sous* forme normale *si chacune de ses classes de co-identité est de taille 1.*

Soit G un graphe conceptuel simple bien formé sur un support $\mathcal{S}$. Alors on appelle forme normale *de G et on note $nf(G)$ le graphe obtenu en fusionnant chaque classe de co-identité en un seul sommet (dont le type et le marqueur sont déterminés par le critère de fusionnabilité choisi). Nous obtenons ainsi un graphe sous forme normale.*

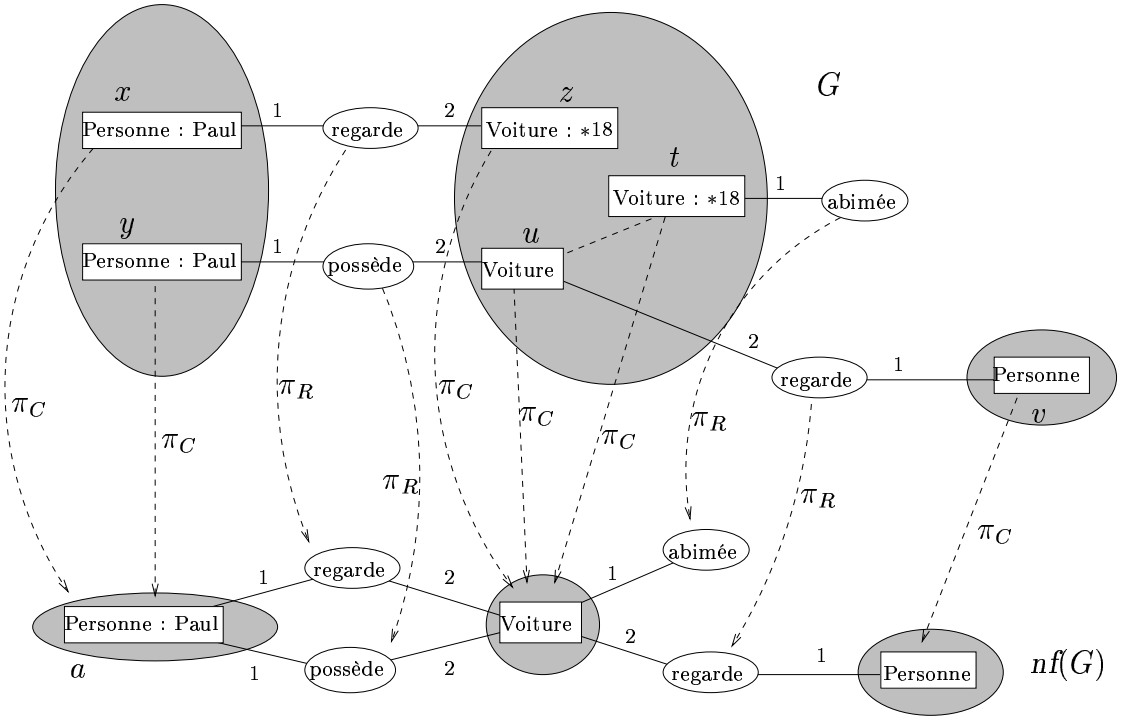


FIG. 4.8 – Projection d'un graphe conceptuel simple dans sa forme normale.

La mise sous forme normale d'un graphe est illustrée dans la FIG. 4.8. Cette mise sous forme normale correspond bien à l'idée intuitive de la co-identité des sommets concepts : les sommets co-identiques sont des représentations partielles de la même entité. Cependant, nous allons voir qu'un graphe et sa forme normale ne sont pas équivalents au sens de la projection.

Propriété 4.2 *Soit G un graphe conceptuel simple bien formé sur un support $\mathcal{S}$. Alors $nf(G) \preceq G$, mais ces deux graphes ne sont pas équivalents.*

Preuve: L'application (π_C, π_R) qui associe à tout sommet concept c de G le sommet de nf(G) obtenu par la fusion de la classe de co-identité de c et à tout sommet relation de G son sommet correspondant dans $nf(G)$ est bien une projection (voir FIG. 4.8).

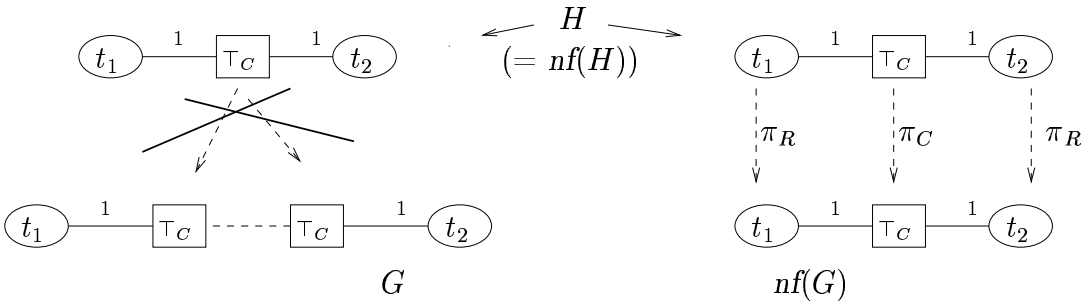
Remarquer, sur la même figure, que nf(G) ne se projette pas dans G (en effet, si on projette le sommet concept a dans x, x n'est pas voisin de *possède*, et si on projette a dans y, y n'est pas voisin de *regarde*). □

Précisons un peu plus la façon dont la mise sous forme normale influe sur les projections :

Propriété 4.3 *Soient H et G deux graphes conceptuels simples bien formés sur un support $\mathcal{S}$. Alors :*
(i) si $G \preceq H$, alors $nf(G) \preceq H$ (mais la réciproque n'est pas vraie) ;
(ii) si $G \preceq H$, alors $nf(G) \preceq nf(H)$ (mais la réciproque n'est pas vraie) ;
(iii) $nf(G) \preceq H$ si et seulement si $nf(G) \preceq nf(H)$.

L'absence de réciproque pour *(ii)* est très importante, puisqu'elle est la source du contre-exemple à la complétude de la projection par rapport à la sémantique logique Φ de [Sowa, 1984]. Ce contre-exemple a été remarqué simultanément par [Chein and Mugnier, 1995] et par [Gosh and Wuwongse, 1995], qui y ont apporté des solutions différentes : la forme normale dans le premier cas, une forme dite antinormale dans le second. Nous n'insisterons pas sur cette forme antinormale dans ce mémoire. En effet, celle-ci nécessite la décomposition du graphe requête en ses entités atomiques, en éclatant ce graphe en sous-graphes comportant au plus un sommet relation, reliés par des liens de co-référence. Cette forme antinormale perd ainsi la structure du graphe, sur laquelle nous nous basons pour écrire nos algorithmes.[3]

Preuve: Vérifions successivement ces trois propriétés :
(i) Si π est une projection de H dans G, alors considérons π' une projection de G dans $nf(G)$ (elle existe, voir Prop. 4.2) : $\pi' \circ \pi$ est une projection de H dans $nf(G)$. Un contre-exemple à la réciproque est proposé dans la FIG. 4.9, où les types de relations unaires t_1 et t_2 sont incomparables.



Pas de projection de H (ni de $nf(H)$) dans G — Projection de H (ou de $nf(H)$) dans $nf(G)$

FIG. 4.9 – Contre-exemple aux deux réciproques de la Prop. 4.3.

[3]Cette forme antinormale a cependant un avantage, puisqu'elle ne nécessite pas de prétraitement sur le graphe de faits G, qui est souvent un « gros graphe ». [Guinaldo and Haemmerlé, 1998], en changeant la définition de la projection, proposent un algorithme qui ne nécessite pas de prétraitement, ni de la requête, ni des faits, pour arriver au même résultat : l'adéquation et complétude par rapport à la sémantique Φ.

(ii) Si (π_C, π_R) est une projection de H dans G, alors il existe une projection (π'_C, π'_R) de $nf(H)$ dans $nf(G)$. Comme tous les sommets concepts d'une même classe de co-identité de H sont projetés dans une même classe de co-identité de G, alors π_C détermine une application π'_C des sommets concepts de $nf(H)$ dans les sommets concepts de $nf(G)$: si c'_H est un sommet concept de $nf(H)$ obtenu par la fusion d'une classe de sommets concepts de H dont c_H est un représentant, et si c'_G est un sommet concept de $nf(G)$ obtenu par la fusion d'une classe de sommets concepts de G dont c_G est un représentant, alors $\pi_C(c_H) = c_G \Rightarrow \pi'_C(c'_H) = c'_G$. Il reste à vérifier que le type de c'_G est plus spécifique que le type de c'_H. Comme chacun des types des sommets concepts de la classe de co-identité de c_G est plus spécifique que celui d'un sommet concept de la classe de c_H (sinon, (π_C, π_R) ne serait pas une projection), alors le type de c'_G est plus spécifique que le type de c'_H (sinon, le type de c'_H ne serait pas une borne inférieure pour les types de la classe de co-identité de c_H, condition donnée par la Def. 4.7). Le contre-exemple pour la réciproque est le même que pour la propriété *(i)*, et est donné par la FIG. 4.9.
(iii) Le sens $(\Rightarrow)$ de l'équivalence est une conséquence de *(ii)* (prendre $nf(G)$ pour G, et vérifier que $nf(nf(G)) = nf(G)$), tandis que le sens $(\Leftarrow)$ est obtenu par composition de la projection de $nf(H)$ dans $nf(G)$ avec celle de H dans $nf(H)$. □

Nous allons maintenant montrer que la projection de graphes conceptuels simples *sous forme normale* peut être calculée par la projection de graphes élémentaires.

Équivalence avec les graphes élémentaires

Les transformations que nous utiliserons, *S2E* et les différentes versions de *E2S*, sont extrêmement similaires à celles que nous avons présentées à la Sect. 4.1 dans le cadre des graphes conceptuels très simples. La seule différence est que nous allons devoir considérer les marqueurs individuels, qui seront traduits de la même façon qu'un type.

La transformation *S2E*, qui transforme un graphe conceptuel simple, sous forme normale, G sur un support $\mathcal{S}$ en graphe élémentaire, est définie de la façon suivante :

- les types de relations d'arité 1 du support sur lequel est défini $S2E(G)$ sont ceux de T_C, de T_R^1, auxquels nous rajoutons les marqueurs individuels de $\mathcal{S}$, tous incomparables à n'importe quel autre type ;
- chaque sommet concept c de G est transformé en un sommet concept de $S2E(G)$, incident à une relation d'arité 1 dont le type est celui de c (il y en a plusieurs si le type est conjonctif) et, dans le cas où c est un sommet concept individuel, le sommet concept obtenu est incident à une relation d'arité 1 typée par le marqueur individuel de c ;
- les sommets relations et les arêtes deviennent des relations (hyperarcs), comme indiqué pour la transformation *TS2E* de la Sect. 4.1.

Cette transformation est illustrée par la transformation du graphe G de la FIG. 4.10. Le lecteur pourra vérifier que, comme la transformation *TS2E*, *S2E* conserve, à une classe d'équivalence près, toutes les projections.

Il est immédiat de transformer un graphe élémentaire en graphe conceptuel simple sous forme normale : il suffit, ce que nous savons déjà faire, de le transformer en graphe concep-

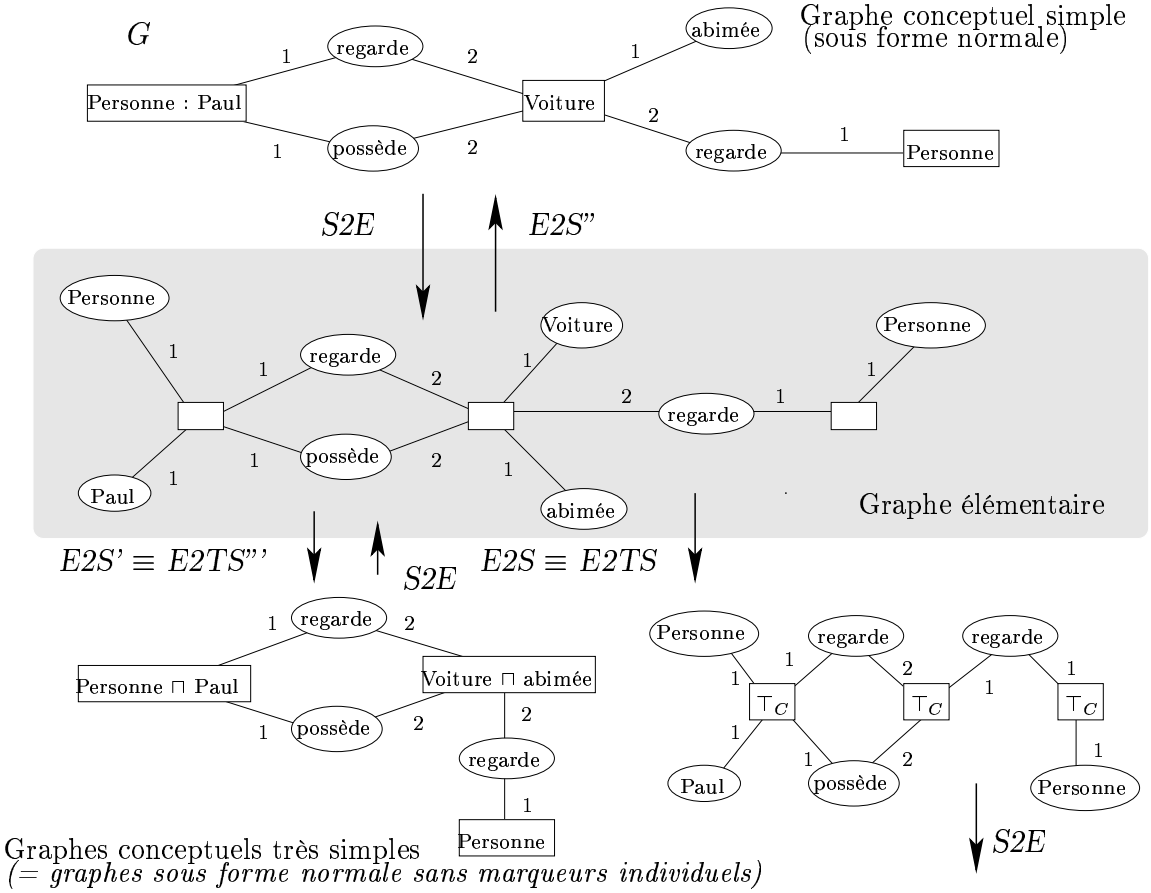


FIG. 4.10 – Transformations de graphes sous forme normale en graphe élémentaire.

tuel très simple. Si nous n'avons aucune connaissance relative au support sur lequel sera défini la traduction (*i.e.* nous ne savons pas partitionner les relations d'arité 1 du graphe élémentaire en types de concepts, marqueurs individuels et types de relation), nous pouvons utiliser les transformations *E2TS* et *E2TS"'* d'un graphe élémentaire en graphe conceptuel très simple (Sect. 4.1). Ces transformations déterminent les transformations *E2S* et *E2S'* qui transforment un graphe élémentaire en un graphe conceptuel simple sans marqueur individuel (la deuxième utilisant un support conjonctif). Ces deux transformations sont illustrées dans la FIG. 4.10. Il est intéressant de remarquer, avec la transformation *E2S'*, qu'après son rôle dans la mise sous forme normale, un marqueur individuel se comporte exactement comme un type dans la projection de graphes sous forme normale.

Enfin, si nous connaissons le support sur lequel nous voulons traduire un graphe élémentaire G (les types de relations d'arité 1 sont organisés en deux hiérarchies distinctes, que nous associons à T_C et à T_R^1, et en un ensemble de types incomparables, que nous associons à l'ensemble $\mathcal{M}$ des marqueurs individuels), nous pouvons définir la transformation *E2S"* (aussi illustrée dans la FIG. 4.10) de la façon suivante :

– à tout sommet concept c de G, nous associons un sommet concept c' de *E2S"*(G),

puis examinons chaque relation r d'arité 1 incidente à c :
- si le type de r correspond à celui d'un type de relation, r est transformé en un sommet relation de degré 1 relié à c' ;
- si le type de r correspond à celui d'un type de concept, alors ce type devient celui de c' (dans le cas d'un support conjonctif, il est rajouté au type conjonctif) ;
- si le type de r correspond à un marqueur individuel, alors ce marqueur devient celui de c'.

– les relations d'arité supérieure ou égale à 2 sont transformées de la même manière que pour la transformation *E2TS*.

Cette transformation conserve elle aussi toutes les projections. Son intérêt est que, cette fois-ci, il s'agit bien de la réciproque de *S2E*. Nous pouvons donc retraduire les résultats d'un raisonnement, effectué dans le formalisme des graphes élémentaires, vers le graphe conceptuel simple initial. Cette transformation met donc en bijection les graphes conceptuels simples sous forme normale avec un sous-ensemble des graphes élémentaires, immédiat à caractériser :
- tout sommet concept est incident à au plus une relation d'arité 1 assimilable à un marqueur individuel ;
- il ne peut y avoir deux relations d'arité 1 assimilables à un même marqueur individuel ;
- tout sommet concept est incident à au moins une relation d'arité 1 assimilable à un type de concept ;
- les relations d'arité 1, assimilables à des types de concepts, incidentes à un même sommet concept doivent être fusionnables (Def. 4.7).

4.2.3 Introduction de constantes dans la sémantique logique

Nous allons maintenant étendre la sémantique Φ des graphes élémentaires et des graphes conceptuels très simples afin de prendre en compte les marqueurs individuels. Avant cela, nous examinerons une autre sémantique logique, qui s'obtient de façon immédiate grâce aux résultats précédents, Υ. Enfin, nous évoquerons brièvement la sémantique Ψ proposée, comme une alternative à Φ, par [Simonet, 1998].

Une sémantique logique immédiate : Υ

Ces transformations nous permettent d'obtenir immédiatement une sémantique logique pour les graphes conceptuels simples, que nous appellerons Υ. Soit G un graphe conceptuel simple sur un support $\mathcal{S}$, et soient $\mathcal{S}'$ et G' le support de graphe élémentaire et le graphe élémentaire obtenus en appliquant la transformation $S2E$ à $\mathcal{S}$ et à $nf(G)$. Alors nous pouvons définir une sémantique Υ par l'intermédiaire de la sémantique Φ des graphes élémentaires : $\Upsilon(\mathcal{S}) = \Phi(\mathcal{S}')$ et $\Upsilon(G) = \Phi(G')$.

Du Th. 3.5 (adéquation et complétude de la projection de graphes élémentaires par rapport à la sémantique Φ) et de la Prop. 4.3 (étudiant les effets de la mise sous forme normale pour l'existence de projections), nous pouvons déduire immédiatement les corollaires suivants :

Corollaire 4.2 *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. S'il existe une projection de H dans G, alors $\Upsilon(\mathcal{S}), \Upsilon(G) \models \Upsilon(H)$. La réciproque n'est pas vraie en général.*

Preuve: S'il existe une projection de H dans G, alors il existe une projection de *nf*(H) dans *nf*(G) (Prop. 4.3, *(ii)*), donc une projection de *S2E*(*nf*(H)) dans *S2E*(*nf*(G)) (voir Sect. 4.2.2), et il s'ensuit que $\Upsilon(\mathcal{S}), \Upsilon(G) \models \Upsilon(H)$ (Th. 3.5 et définition de Υ).

Pour la réciproque, considérer de nouveau l'exemple de la FIG. 4.9. Il n'y a pas de projection de H dans G, et pourtant $\Upsilon(H) = \exists x(\top_C(x) \wedge t_1(x) \wedge t_2(x))$ est équivalente à $\Upsilon(G) = \exists x(\top_C(x) \wedge \top_C(x) \wedge t_1(x) \wedge t_2(x))$. □

Corollaire 4.3 *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. Les assertions suivantes sont équivalentes :*

(i) il existe une projection de H dans nf(G)
(ii) il existe une projection de nf(H) dans nf(G)
(iii) $\Upsilon(\mathcal{S}), \Upsilon(\mathcal{G}) \models \Phi(H)$

Remarquons que l'équivalence *(i)* $\Leftrightarrow$ *(ii)* est déjà donnée par la Prop. 4.3. Ce corollaire est souvent exprimé (pour la sémantique Φ des graphes conceptuels) sous la forme *(i)* $\Leftrightarrow$ *(iii)*, mettant en valeur le fait que la projection est calculée de H dans la forme normale de G. Or nous sommes plus intéressés par sa version *(ii)* $\Leftrightarrow$ *(iii)* (car nous calculons les projections sur les graphes élémentaires obtenus à partir des formes normales de H et de G, supprimant ainsi le besoin de prendre en compte les classes de co-identité au cours de la projection). Nous présentons ce corollaire sous cette forme afin de concilier ces deux visions du problème.

Preuve: Nous n'avons donc à prouver que *(ii)* $\Leftrightarrow$ *(iii)*, ce que nous ferons par équivalences successives. Il existe une projection de *nf*(H) dans *nf*(G) $\Leftrightarrow$ il existe une projection de *S2E*(*nf*(H)) dans *S2E*(*nf*(G)) (voir Sect. 4.2.2) $\Leftrightarrow$ $\Upsilon(\mathcal{S}), \Upsilon(\mathcal{G}) \models \Phi(H)$ (Th. 3.5 et définition de Υ). □

Cette sémantique est très similaire à la sémantique Φ, qui nécessite aussi l'utilisation de graphes sous forme normale. Elle est moins intéressante que Φ, qui considère les marqueurs individuels comme des constantes, ce qui est plus naturel et génère de plus petites formules. Il faut cependant retenir l'intérêt de la démarche effectuée pour définir Υ : une transformation d'un formalisme de graphes en graphes élémentaires est suffisante pour définir une sémantique logique adéquate et complète.

Les marqueurs vus comme constantes : la sémantique Φ

Il est bien plus naturel, ce qui est fait dans la sémantique Φ de [Sowa, 1984], de traduire les marqueurs individuels par des constantes. Nous rappelons ici cette sémantique, et prouvons les résultats d'adéquation et de complétude en utilisant la même démonstration (mais cette fois-ci en version considérablement abrégée) que dans le cadre des graphes élémentaires.

La traduction par Φ du support se fait de la même manière que dans le cas des graphes conceptuels très simples (Sect. 4.1.3) ou des graphes élémentaires (Sect. 3.3). Ceci veut dire que nous choisissons ici une « interprétation par ordre » (*order-theoretic interpretation*) du support (le lecteur pourra se référer à la discussion sur les différents critères de fusionnabilité de la Sect. 4.2.2).

Nous associons à chaque marqueur individuel m du support une constante distincte, et dénotons de la même façon le marqueur individuel et la constante qui lui est associée.

Traduisons maintenant par Φ un graphe conceptuel simple bien formé G (qui, notons le, n'a pas besoin d'être sous forme normale).

1. on se donne une application injective *classe*, qui associe à toute classe C de sommets concepts co-identiques une variable si tous les sommets sont génériques, et la constante associée à l'unique marqueur individuel distinct étiquetant les sommets de cette classe dans le cas contraire ;
2. on se donne une application (notons qu'elle n'est pas injective), *terme*, qui associe à chaque sommet concept c le terme $\mathit{classe}(C)$ (variable ou constante) déterminé par sa classe de co-identité C ;
3. chaque sommet concept c de G, dont le type est t, est traduit par la formule $\phi_C(c) = t(\mathit{terme}(c))$ (si le type de c est un type conjonctif $T = t_1 \sqcap \cdots \sqcap t_q$, sa traduction est $\phi_C(c) = t_1(\mathit{terme}(c)) \wedge \cdots \wedge t_q(\mathit{terme}(c))$) ;
4. chaque sommet relation r de G, dont les arguments sont, dans cet ordre, $c_1, \ldots, c_p$ et dont le type est u est traduit par la formule $\phi_R(r) = u(\mathit{terme}(c_1), \ldots, \mathit{terme}(c_p))$;
5. nous notons $\phi(G)$ la conjonction des $\phi_C(c)$, pour $c \in V_C(G)$ et des $\phi_R(r)$ pour $r \in V_R(G)$;
6. une formule $\Phi(G)$ qui traduit le graphe G est la fermeture existentielle de la formule $\phi(G)$.

Prenons par exemple le graphe G de la FIG. 4.8. Sa traduction par Φ est :

$$\exists x\ \exists y\ (\mathit{Personne}(\mathit{Paul}) \wedge \mathit{Personne}(\mathit{Paul}) \wedge \mathit{Voiture}(x) \wedge \mathit{Voiture}(x) \wedge \mathit{Voiture}(x) \wedge \mathit{Personne}(y) \wedge \mathit{regarde}(\mathit{Paul}, x) \wedge \mathit{possède}(\mathit{Paul}, x) \wedge \mathit{abimée}(x) \wedge \mathit{regarde}(y, x))$$

Cette formule se simplifie immédiatement (en enlevant les occurrences multiples d'un même atome) en une formule qui est l'interprétation par Φ de $\mathit{nf}(G)$:

$$\exists x\ \exists y\ (\mathit{Personne}(\mathit{Paul}) \wedge \mathit{Voiture}(x) \wedge \mathit{Personne}(y) \wedge \mathit{regarde}(\mathit{Paul}, x) \wedge \mathit{possède}(\mathit{Paul}, x) \wedge \mathit{abimée}(x) \wedge \mathit{regarde}(y, x))$$

Ce résultat ($\Phi(G) \equiv \Phi(\mathit{nf}(G))$) est toujours vérifié si nous utilisons une des trois méthodes de fusion des sommets basées sur une interprétation par ordre du support (voir Sect. 4.2.2). Cependant, si nous utilisons une interprétation par treillis (avec la mise à jour de la traduction du support que nous avons évoquée), nous avons un résultat d'équivalence plus faible : $\Phi(\mathcal{S}), \Phi(G) \equiv \Phi(\mathcal{S}), \Phi(\mathit{nf}(G))$. Notons que, si G est un graphe conceptuel simple sans marqueurs individuels, alors $\Phi(\mathit{nf}(G)) = \Upsilon(G)$.

Le résultat d'adéquation de la projection de graphes conceptuels simples par rapport à la sémantique logique Φ est dû à [Sowa, 1984] :

Corollaire 4.4 *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. S'il existe une projection de H dans G, alors $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$.*

Preuve: Si il existe une projection de H dans G, alors il existe une projection de $nf(H)$ dans $nf(G)$ (Prop. 4.3, *(ii)*). Il reste à prouver que si il existe une projection de $nf(H)$ dans $nf(G)$, alors $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$. De par la remarque précédente ($\Phi(G) \equiv \Phi(nf(G))$), il suffit de prouver la propriété suivante : si H et G sont deux graphes conceptuels simples sous forme normale, alors $\Phi(\mathcal{S}), \Phi(G) \models \Phi(H)$.

Pour prouver ceci, nous devons reprendre le sens ($\Rightarrow$) de la démonstration du Th. 3.5 sur l'adéquation et la complétude de la projection de graphes élémentaires par rapport à la sémantique Φ. Nous invitons le lecteur à reprendre cette démonstration, en y apportant les modifications suivantes (la FIG. 4.11 reprend le schéma de principe de cette démonstration, en incluant les constantes) :

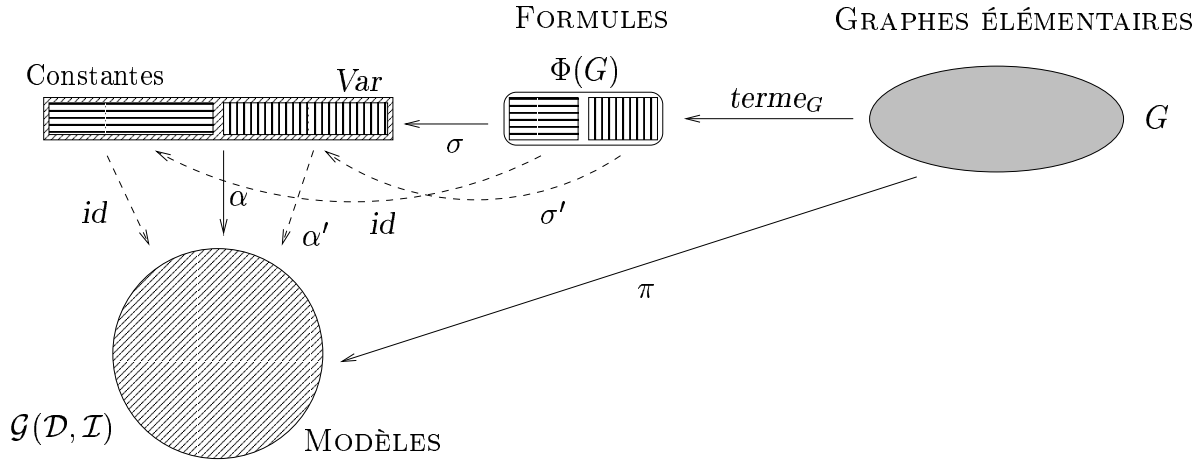


FIG. 4.11 – Graphes conceptuels simples, traduction logique, et modèles.

- *Valuation d'une formule dans un modèle :* sans changer les règles de valuation, nous pouvons les considérer de façon quelque peu différente l'application d'assignation α. Nous définissons $\alpha : Var \cup \mathcal{D} \to \mathcal{D}$, qui à toute variable x associe une constante $\alpha'(x)$ (α' est donc l'assignation « habituelle »), et à toute constante c associe $id(c)$. Nous appelons aussi assignation cette application $\alpha = (\alpha', id)$.
- *Construction du graphe d'interprétation :* inchangée.
- *Graphe d'interprétation cohérent pour un support :* la définition comme le lemme 3.1 restent inchangés.
- *Projection et valuation dans un modèle :* grâce à notre modification de la définition de α (qui conserve cependant les mêmes valuations), le lemme 3.2 comme sa démonstration restent inchangés. Notons que l'application σ considérée est cette fois ci composée de deux applications. La première, σ', utilisée pour les variables, est la résultante des substitutions de variables de la règle 2., tandis que la seconde, définie pour les constantes, est l'identité *id*.

– *Démonstration du théorème :* nous n'avons besoin que du sens ($\Rightarrow$) de cette démonstration, qui reste valide si la traduction et la projection utilisées respectent ce que nous avons appelé condition faible : la projection π conserve les classes d'équivalence induites par la traduction *terme*. Cette condition est trivialement vérifiée, puisque les classes d'équivalence pour la traduction par *terme* sont les classes de co-identité du graphe, conservées par la projection.

□

La réciproque du Cor. 4.4 n'est pas nécessairement vraie, comme l'ont remarqué simultanément [Chein and Mugnier, 1995] et [Gosh and Wuwongse, 1995]. Le même contre-exemple peut une nouvelle fois être utilisé : dans la FIG. 4.9, il n'y a pas de projection de H dans G, et pourtant $\Phi(H) = (\exists x(\top_C(x) \wedge t_1(x) \wedge t_2(x))) \equiv (\exists x(\top_C(x) \wedge \top_C(x) \wedge t_1(x) \wedge t_2(x))) = \Phi(G)$. Comme le montre le résultat suivant, la mise sous forme normale du graphe représentant les faits, proposée par [Chein and Mugnier, 1995] (comme la mise sous forme antinormale de la requête proposée par [Gosh and Wuwongse, 1995]) est une manière d'obtenir la complétude par rapport à la sémantique Φ.

Corollaire 4.5 *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. Les assertions suivantes sont équivalentes :*

(i) il existe une projection de H dans nf(G)
(ii) il existe une projection de nf(H) dans nf(G)
(iii) $\Phi(\mathcal{S}), \Phi(\mathcal{G}) \models \Phi(H)$

Preuve: Comme nous l'avons déjà remarqué (Prop. 4.3), *(i)* et *(ii)* sont équivalents. Nous n'avons plus qu'à démontrer, par exemple, que *(ii)* et *(iii)* sont équivalents. Le sens ($\Rightarrow$) de cette équivalence est donné dans la démonstration du Cor. 4.4. Continuons l'adaptation de la preuve du Th. 3.5 utilisée pour prouver le Cor. 4.4. Il nous reste maintenant à vérifier que le sens ($\Leftarrow$) de l'équivalence est encore valide, ce qui découle de l'injectivité de l'application *terme* utilisée pour traduire un graphe sous forme normale. □

Une sémantique qui ne nécessite pas la mise sous forme normale : Ψ

[Simonet, 1996, Simonet, 1998] propose une autre sémantique, Ψ, pour laquelle l'adéquation et la complétude de la projection ne dépendent pas d'un critère de normalité [4]. En cela, Ψ est une sémantique qui traduit exactement la projection. Si on considère cette sémantique, un ensemble de sommets concepts co-identiques ne sont plus des représentations partielles d'une même entité, mais des points de vue différents sur cette entité.

La traduction par Ψ d'un support $\mathcal{S}$ est légèrement modifiée. En effet, à tout type de concept de T_C nous associons maintenant un symbole de prédicat *binaire*. Ceci se répercute sur la traduction du support, puisque chaque paire (t, t') de types de concepts telle que t' couvre t est traduite par $\psi(t, t') = (\forall x\ \forall y(t(x, y) \rightarrow t'(x, y)))$. La traduction des types de relations est la même que pour la sémantique Φ.

[4] Dans le même but, [Preller et al., 1998] proposent de traduire les graphe en « formules colorées ».

La traduction par Ψ d'un graphe conceptuel simple (pas nécessairement sous forme normale) G sur un support $\mathcal{S}$ est obtenue de la façon suivante :

1. l'application injective *classe* associe à toute classe C de sommets concepts co-identiques une constante ou une variable, comme indiqué pour la sémantique Φ ;
2. l'application injective *terme* associe à chaque sommet concept une variable distincte ;
3. chaque sommet concept c de G, dont le type est t, est traduit par la formule $\psi_C(c) = t(\mathit{classe}(c), \mathit{terme}(c))$ (si le type de c est un type conjonctif $T = t_1 \sqcap \cdots \sqcap t_q$, sa traduction est $\psi_C(c) = t_1(\mathit{classe}(c), \mathit{terme}(c)) \wedge \cdots \wedge t_q(\mathit{classe}(c), \mathit{terme}(c))$) ;
4. chaque sommet relation r de G, dont les arguments sont, dans cet ordre, $c_1, \ldots, c_p$ et dont le type est u est traduit par la formule $\psi_R(r) = u(\mathit{terme}(c_1), \ldots, \mathit{terme}(c_p))$;
5. nous notons $\psi(G)$ la conjonction des $\psi_C(c)$, pour $c \in V_C(G)$ et des $\psi_R(r)$ pour $r \in V_R(G)$;
6. une formule $\Psi(G)$ qui traduit le graphe G est la fermeture existentielle de la formule $\psi(G)$.

La traduction par Ψ du graphe G de la FIG. 4.12 (une copie du graphe G de la FIG. 4.8, où nous avons indiqué les termes associés aux classes de co-identité et les variables associées aux sommets concepts) est donc :

$$\exists c_1\ \exists c_2\ \exists x_1\ \exists x_2\ \exists x_3\ \exists x_4\ \exists x_5\ \exists x_6\ (\mathit{Personne}(\mathit{Paul}, x_1) \wedge \mathit{Personne}(\mathit{Paul}, x_2) \wedge \mathit{Voiture}(c_1, x_3) \wedge \mathit{Voiture}(c_1, x_4) \wedge \mathit{Voiture}(c_1, x_5) \wedge \mathit{Personne}(c_2, x_6) \wedge \mathit{regarde}(x_1, x_3) \wedge \mathit{possède}(x_2, x_5) \wedge \mathit{abimée}(x_4) \wedge \mathit{regarde}(x_6, x_5))$$

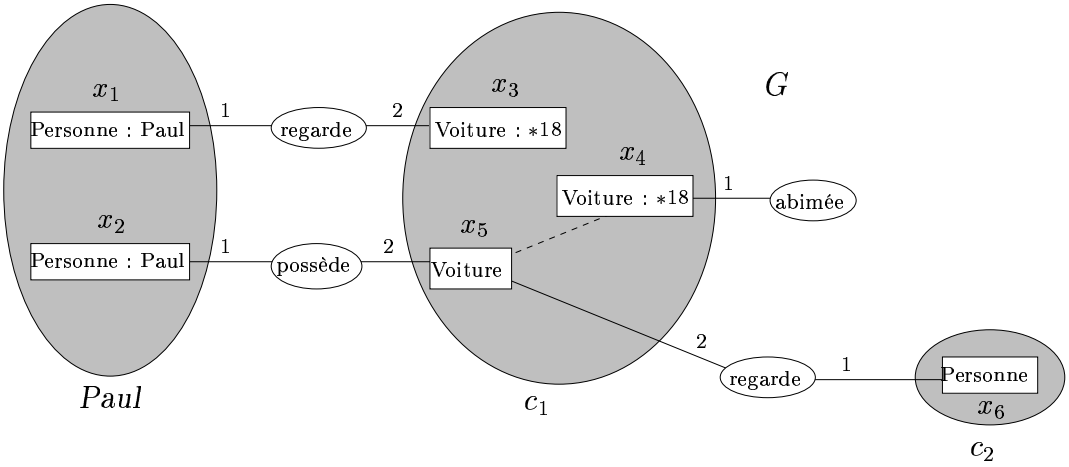


FIG. 4.12 – Traduction logique d'un graphe par le biais de la sémantique Ψ.

Enfin, la projection de graphes conceptuels simples est adéquate et complète pour la sémantique Ψ, sans condition de normalité [Simonet, 1996, Simonet, 1998] :

Corollaire 4.6 *Soient H et G deux graphes conceptuels simples sur un support $\mathcal{S}$. Alors il existe une projection de H dans G si et seulement si $\Psi(\mathcal{S}), \Psi(\mathcal{G}) \models \Psi(H)$.*

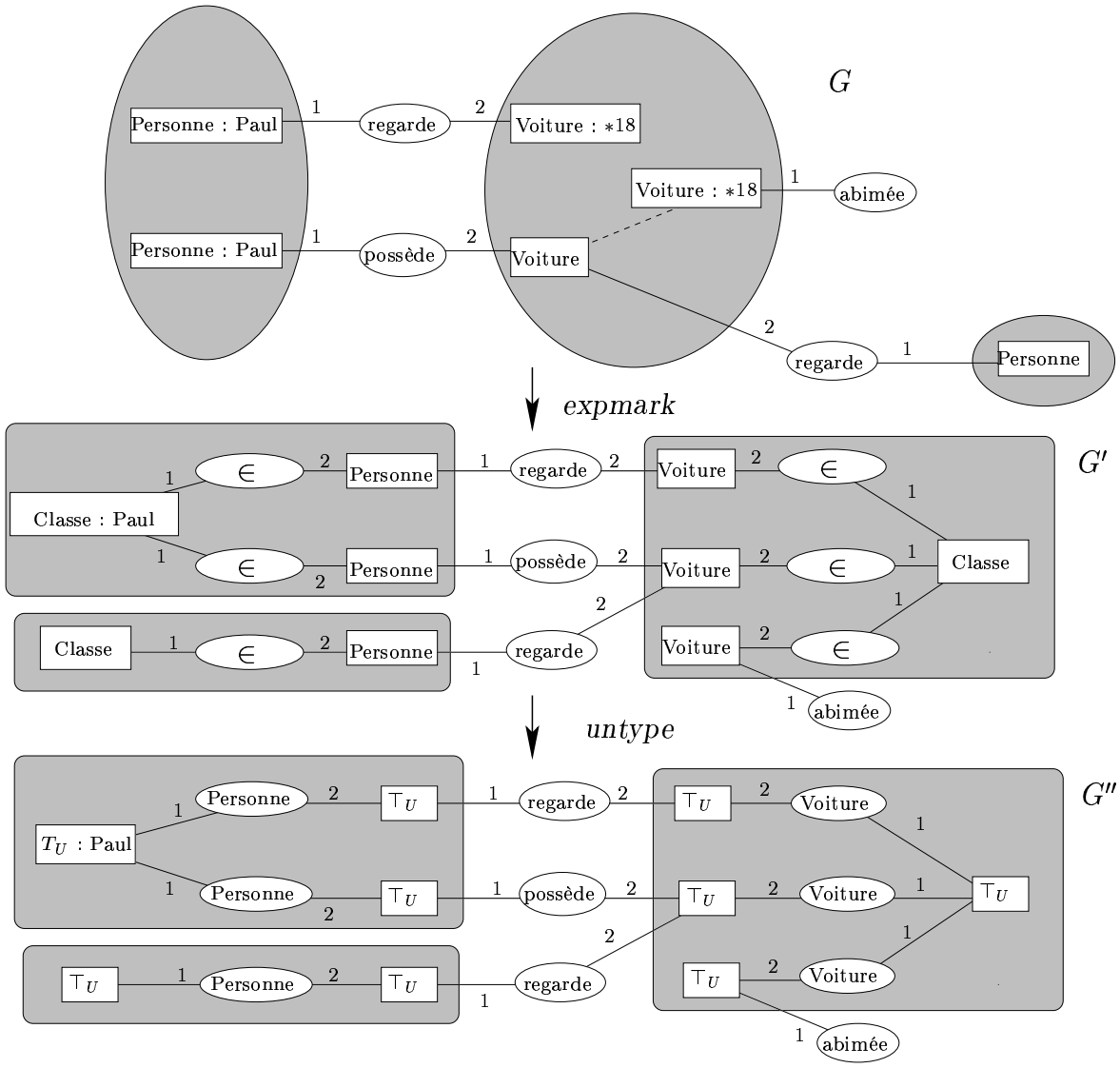


FIG. 4.13 – Transformations utilisées pour prouver l'adéquation et la complétude pour Ψ.

La communauté « Graphes Conceptuels » n'a pas accordé à cette sémantique l'intérêt qu'elle mérite. En effet, Ψ décrit exactement la projection, et ne nécessite pas de passer par un mécanisme de raisonnement externe qui est la mise sous forme normale. Une critique de cette sémantique Ψ est qu'elle n'accorde à la co-identité qu'une signification minimaliste : des sommets co-identiques appartiennent à une même classe d'équivalence, qui ne peut être perdue par la projection. La sémantique Φ, elle, traite la co-identité comme l'égalité : toute information portant sur un sommet concept est valable pour tous les sommets concepts de sa classe de co-identité. Nous avons montré, dans [Baget, 1998, Baget, 1999], que ces « connaissances ajoutées » de la sémantique Φ pouvaient être implémentées par des règles de graphes. Ceci permet d'obtenir l'expressivité de la sémantique Φ, et de garder la sémantique

Ψ qui ne nécessite pas de mise sous forme normale. L'inconvénient de cette méthode est le coût algorithmique de cette interprétation du mécanisme de mise sous forme normale, son intérêt est de pouvoir prendre en compte des sémantiques intermédiaires de la co-identité. En considérant que seules certaines informations portant sur un sommet concept sont partagées par les sommets de sa classe de co-identité, nous pouvons implémenter, par exemple, des mécanismes de protection (données privées/publiques).

Preuve: Nous donnons ici une démonstration originale de ce résultat, pour laquelle nous nous servirons de l'adéquation et de la complétude de la projection de graphes sous forme normale par rapport à Φ (Cor. 4.5), par l'intermédiaire des transformations de graphes illustrées dans la FIG. 4.13.

La première transformation (voir FIG.4.13) que nous allons utiliser est *expmark*. Si G est un graphe conceptuel simple, nous obtenons le graphe $G' = expmark(G)$ en rajoutant au graphe G un sommet concept de type *Classe* pour chaque classe de co-identité de G. Ce sommet aura un marqueur générique si tous les sommets de la classe représentée sont génériques, et aura l'unique marqueur individuel présent dans cette classe sinon. Chaque sommet concept de G est ensuite relié, par l'intermédiaire d'une relation binaire de type $\in$, au sommet concept qui représente sa classe de co-identité. Le marqueur individuel d'un sommet concept étant représenté dans sa classe, nous pouvons le supprimer. Notons que nous devons rajouter au support ces nouveaux types *Classe* et $\in$, qui ne sont comparables à aucun autre type du support.

Il est immédiat de vérifier que, si H et G sont deux graphes conceptuels simples, alors π est une projection de H dans $G \Rightarrow \pi'$ est une projection de $expmark(H)$ dans $expmark(H)$ (où π' associe à un sommet concept de $expmark(H)$ issu d'un sommet concept c de H le sommet concept de $expmark(G)$ issu de $\pi(c)$; et à un sommet concept représentant une classe le sommet concept représentant la classe de l'image par π de l'un de ses éléments). Réciproquement, si π' est une projection de $expmark(H)$ dans $expmark(H)$, alors la restriction de π' aux sommets concepts issus de sommets concepts de H détermine une projection de H dans G.

La seconde transformation (voir FIG.4.13) est appelée *untype*. Elle remplace le type de chaque relation de type $\in$ par le type de son deuxième argument (si ce type est conjonctif, il faudra la remplacer par des relations jumelles, une par type primitif). Le type de chaque sommet concept étant maintenant déterminé par celui d'une relation dont le type était $\in$ (le type *Classe* s'il est le premier argument d'une telle relation, le type de cette relation si il est le second argument), on peut sans perte d'information remplacer tous les types de concepts par un type unique, disons $\top_U$. Le support doit aussi être modifié de façon adéquate, puisque T_R^2 devient l'union de T_C et de T_R^2, tandis que T_C ne contient plus qu'un nouveau type de concept, $\top_U$.

Là encore, cette transformation ne change rien aux projections. Si H et G sont deux graphes conceptuels, notons $H' = expmark(H)$ et $G' = expmark(G)$. Alors une application π est une projection de H' dans H si et seulement si c'est une projection de $H'' = untype(H')$ dans $G'' = untype(G')$.

Remarquons que G', G'', H' et H'' sont des graphes sous forme normale. Alors, de par

le Cor. 4.5, il existe une projection de H'' dans G'' si et seulement si $\Phi(\mathcal{S}''), \Phi(G'') \models \Phi(H'')$. Donc (voir ci-dessus), il existe une projection de H dans G si et seulement si $\Phi(\mathcal{S}''), \Phi(G'') \models \Phi(H'')$. Remarquons aussi que $\Phi(\mathcal{S}'') = \Psi(\mathcal{S})$, car les types de concepts sont devenus des types de relations binaires, et sont donc traduits par des symboles de prédicats binaires.

La dernière étape nécessite de considérer les formules. Considérons la transformation *simp* qui supprime d'une formule tous les atomes dont le prédicat est $\top_U$. Alors $\Psi(\mathcal{S}), \Phi(G'') \models \Phi(H'')$ si et seulement si $\Psi(\mathcal{S}), simp(\Phi(G'')) \models simp(\Phi(H''))$ (ces atomes ne codent que de l'information redondante, déjà encodée dans la traduction par ϕ des relations binaires issues de relations *marqueur*).

Enfin, il ne nous reste plus qu'à vérifier que, pour tout graphe conceptuel simple G, $simp(\Phi(untype(expmark(G)))) = \Psi(G)$ pour conclure. □

Rapports entre $\mathcal{SG}$ et le fragment FOL$(\wedge, \exists)$

Nous avons affirmé précédemment que le modèle $\mathcal{SG}$ était équivalent au fragment positif, conjonctif, existentiel de la logique du premier ordre (sans fonctions), que nous noterons FOL$(\wedge, \exists)$. Or les formules associées au support par Φ sont quantifiées universellement, et elles interviennent dans le processus de déduction.

Cependant, on peut associer à un graphe une formule, toujours quantifiée existentiellement, qui intègre les connaissances du support intervenant dans la déduction, et rend inutile l'utilisation de $\Phi(\mathcal{S})$. Soit Φ' cette nouvelle sémantique. À un graphe élémentaire G sur un support $\mathcal{S}$, on associe un graphe $G' = desup(G, \mathcal{S})$ obtenu à partir de G de la façon suivante : pour toute relation r de type t_r de G, pour chaque type de relation t'_r du support telle que $t_r \leq t'_r$, on ajoute à G' une relation jumelle à r dont le type est t'_r. Nous notons $desup(\mathcal{S})$ le support obtenu à partir de $\mathcal{S}$ en supprimant toute comparabilité entre les types ($\leq$ est l'identité *Id*).

Cette transformation vérifie la propriété suivante :

Propriété 4.4 *Soient H et G deux graphes élémentaires sur $\mathcal{S}$. Notons $\mathcal{S}' = desup(\mathcal{S})$, $H' = desup(H, \mathcal{S})$ et $G' = desup(G, \mathcal{S})$. Alors les assertions suivantes sont équivalentes :*
(i) π est une $\mathcal{S}$-projection de H dans G
(ii) π est une $\mathcal{S}'$-projection de H dans G'
(iii) π est une $\mathcal{S}'$-projection de H' dans G'

Preuve: L'équivalence $(i) \Leftrightarrow (ii)$ est utilisée dans la démonstration du Th. 3.5. En effet, le graphe G' est construit de la même manière que le graphe $\mathcal{M}_G$ que nous construisons pour prouver la partie ($\Leftarrow$) du théorème. La partie $(ii) \Leftrightarrow (iii)$ est tout aussi immédiate. □

On peut également adapter la transformation précédente pour ne pas considérer tous les types t'_r supérieurs à un type t_r, mais seulement ceux qui sont susceptibles d'être utiles pour le calcul des projections de H dans G. On pose alors $H' = H$, quant à G', il est obtenu à partir de G en ne considérant cette fois pour une relation de type t_r que les

types t'_r apparaissant dans H tels que $t_r \leq t'_r$. Cette seconde transformation a l'avantage de pouvoir être utilisée même dans le cas où le support est de taille infinie.

Si nous notons $\Phi'(G, \mathcal{S}) = \Phi(desup(G, \mathcal{S}))$, alors nous avons les équivalences suivantes :
il existe une $\mathcal{S}$-projection de H dans G
$\Leftrightarrow$ il existe une $\mathcal{S}'$-projection de H' dans G' (Prop. 4.4)
$\Leftrightarrow$ $\Phi(\mathcal{S}'), \Phi(G') \models \Phi(H')$ (Th. 3.5)
$\Leftrightarrow$ $\Phi'(G', \mathcal{S}) \models \Phi'(H, \mathcal{S})$ (par def. de Φ', et voir que $\Phi(\mathcal{S}')$ est un ensemble vide de formules)

Précisons maintenant ce que nous entendons par équivalence entre les graphes élémentaires et le fragment FOL$(\wedge, \exists)$:

Propriété 4.5 (Plongement de FOL$(\wedge, \exists)$ dans $\mathcal{SG}$) *Soient g et h deux formules de FOL$(\wedge, \exists)$. Soient G et H leurs graphes conceptuels naturellement associés sur un support trivial ($G = \Phi^{-1}(g)$ et $H = \Phi^{-1}(h)$). Alors $g \models h$ si et seulement s'il existe une projection de H dans G.*

Propriété 4.6 (Plongement de $\mathcal{SG}$ dans FOL$(\wedge, \exists)$) *Soient H et G deux graphes élémentaires sur un support $\mathcal{S}$. Soit $g = \Phi'(G, \mathcal{S})$ et $h = \Phi'(H, \mathcal{S})$ deux formules de FOL$(\wedge, \exists)$. Alors il existe une projection de H dans G si et seulement si $g \models h$.*

4.3 Graphes conceptuels emboîtés

Les graphes conceptuels emboîtés sont une autre extension du modèle de base. L'emboîtement permet d'ajouter de l'information à l'intérieur d'un sommet concept. La présentation qui en est faite ici est basée sur celle de [Mugnier and Chein, 1996, Chein et al., 1998]. Nous évoquons l'extension qui en est faite aux « graphes de boîtes » par [Baget, 1998]. Les résultats d'équivalence avec les graphes élémentaires sont basés sur ceux de [Baget, 1998], et montrent que ces formalismes sont équivalents aux autres formalismes de $\mathcal{SG}$: il existe une projection d'un graphe emboîté H dans un graphe emboîté G si et seulement si il existe une projection de $f(H)$ dans $f(G)$.

4.3.1 Graphes emboîtés et graphes de boîtes

Les graphes emboîtés permettent d'associer aux sommets concepts une description présentée sous la forme d'un graphe emboîté. Le support sur lequel nous construisons ces graphes est défini de la même façon que pour les graphes conceptuels simples. Nous définissons ces objets et la projection utilisée pour raisonner sur ceux-ci, puis proposons une extension utilisée dans [Baget, 1998, Baget, 1999], les graphes de boîtes.

Graphes conceptuels emboîtés

La FIG. 4.14 est un exemple de graphe conceptuel emboîté. Ce graphe peut s'interpréter par : « une personne regarde une photographie, et cette photographie représente cette

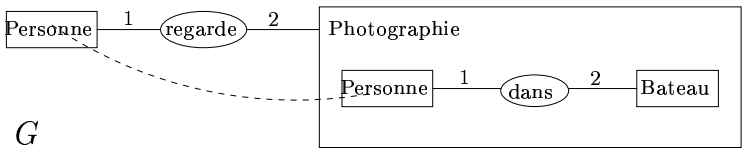


FIG. 4.14 – Un exemple de graphe conceptuel emboîté.

même personne (le lien de co-référence) sur un bateau ». Les graphes conceptuels emboîtés permettent de structurer hiérarchiquement les connaissances. On peut imaginer qu'une application bénéficie de cette structuration pour aider le lecteur à visualiser, à parcourir un graphe. Par exemple, en lisant le graphe *[Personne]→(regarde)→[Photographie]*, cliquer sur le sommet concept *[Photographie]* permet d'accéder à sa description.

Définition 4.10 *Soit G un graphe conceptuel simple sur un support $\mathcal{S}$. On construit un* graphe conceptuel emboîté *dit* basique *G' à partir de G en rajoutant à chaque sommet concept de G un troisième champ, la* description *de ce sommet concept, initialisé à $**$. La* profondeur *de ce graphe emboîté est 0, et nous notons $V_C^{\cup}(G') = V_C(G)$ et $V_R^{\cup}(G') = V_R(G)$ les ensembles de sommets (concepts et relations) intervenant dans ce graphe.*

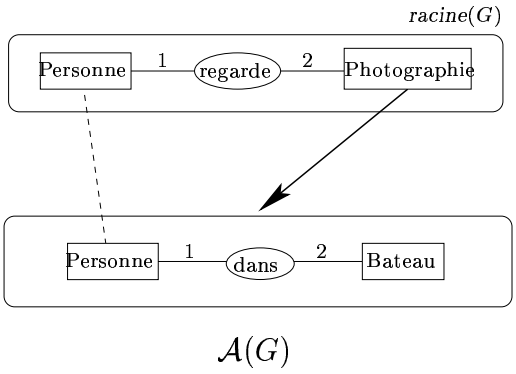
Si G est un graphe conceptuel emboîté basique, $c_1, \ldots, c_k$ des sommets concepts de G, et $G_1, \ldots, G_k$ des graphes conceptuels emboîtés non vides, alors le graphe G' obtenu à partir de G en substituant, pour $1 \leq i \leq k$, le graphe G_i à la description de c_i est un graphe emboîté. Sa profondeur est définie par $\mathit{profondeur}(G') = 1 + max_{1 \leq i \leq k}(\mathit{profondeur}(G_i))$. Les ensembles de sommets (concepts et relations) intervenant dans G' sont respectivement $V_C^{\cup}(G') = V_C(G) \cup (\cup_{1 \leq i \leq k} V_C^U(G_i))$ et $V_R^{\cup}(G') = V_R(G) \cup (\cup_{1 \leq i \leq k} V_R^U(G_i))$.

Il est important de remarquer que si un graphe conceptuel simple (ou un graphe conceptuel emboîté) H est utilisé plusieurs fois dans la construction d'un graphe conceptuel emboîté G, alors nous utilisons à chaque fois une copie distincte de H (et pas le graphe H lui-même). Notons aussi que cette définition prend en compte la co-référence : si G est un graphe conceptuel emboîté, des sommets concepts génériques de $V_C^{\cup}(G)$ ayant même marqueur générique nommé (voir Sect. 4.2.1) seront considérés co-référents (et donc co-identiques), même s'ils n'appartiennent pas au même graphe conceptuel simple. Nous représenterons cette relation de co-référence par un trait en pointillé (qui peut donc « traverser les boîtes ») reliant ces sommets (voir FIG. 4.14). Les classes de co-identité doivent respecter un des critères de fusionnabilité donnés dans la Sect. 4.2.2.

Un graphe conceptuel emboîté peut être représenté comme sur la FIG. 4.14, en dessinant chaque graphe emboîté à l'intérieur du rectangle représentant le sommet concept qu'il décrit. Comme pour le marqueur générique $*$, nous ne représentons pas dans le graphe la description vide $**$.

Alternativement, nous pouvons utiliser une *représentation arborée*. En effet, à tout graphe conceptuel emboîté G, nous pouvons associer un arbre enraciné $\mathcal{A}(G)$, dont les sommets sont les graphes conceptuels simples utilisés dans la définition. Cet arbre peut être construit par induction structurelle, en notant $\mathit{racine}(G)$ le graphe conceptuel simple

qui est la racine de $\mathcal{A}(G)$. Si G est un graphe conceptuel emboîté basique, obtenu à partir d'un graphe conceptuel simple G', dont les descriptions des sommets concepts $c_1, \ldots, c_k$ ont été remplacées par les graphes emboîtés $G_1, \ldots, G_k$, alors $\mathcal{A}(G)$ est construit à partir des arbres enracinés $\mathcal{A}(G_1), \ldots, \mathcal{A}(G_k)$ en ajoutant le graphe $racine(G) = G'$ et k arêtes $((G', c_i), G_i)$ reliant le sommet concept c_i de G' à la racine de $\mathcal{A}(G_i)$. La FIG. 4.15 est la représentation arborée du graphe G de la FIG. 4.14.



$\mathcal{A}(G)$

FIG. 4.15 – Représentation arborée du graphe G de la FIG. 4.14.

Projection de graphes conceptuels emboîtés

Nous pouvons maintenant définir récursivement la projection de graphes conceptuels emboîtés :

Définition 4.11 *Soient H et G deux graphes conceptuels emboîtés sur un support $\mathcal{S}$. Alors une projection Π de H dans G est une application de $V_C^{\cup}(H) \cup V_R^{\cup}(H)$ dans $V_C^{\cup}(G) \cup V_R^{\cup}(G)$ définie par des applications $\pi, \Pi_0, \ldots, \Pi_k$ telles que :*

- *π est une projection (au sens des graphes conceptuels simples) de racine(H) dans racine(G) ;*
- *pour chaque sommet concept c_i de racine(H) dont la description n'est pas vide, Π_i est une projection (au sens des graphes conceptuels emboîtés) de la description de c_i dans la description de $\pi(c_i)$;*
- *Π préserve les classes de co-identité,* i.e. *si c et c' sont deux sommets co-référents de $V_C^{\cup}(H)$, $\Pi(c)$ et $\Pi(c')$ doivent être deux sommets co-identiques de $V_C^{\cup}(G)$.*

De façon plus intuitive, on ne peut projeter un sommet concept c de H dans un sommet concept c' de G que si la description de c se projette dans la description de c' (la description vide $**$ étant assimilable au graphe conceptuel vide, que l'on peut projeter dans n'importe quel graphe). La FIG. 4.16 illustre une telle projection. La racine du graphe requête H (que l'on peut interpréter par « une personne regarde-t-elle une image qui représente un bateau ? ») se projette dans la racine du graphe G, et la description du sommet concept dont le type est *Image* se projette bien (par Π_1) dans la description de l'image par π de ce sommet.

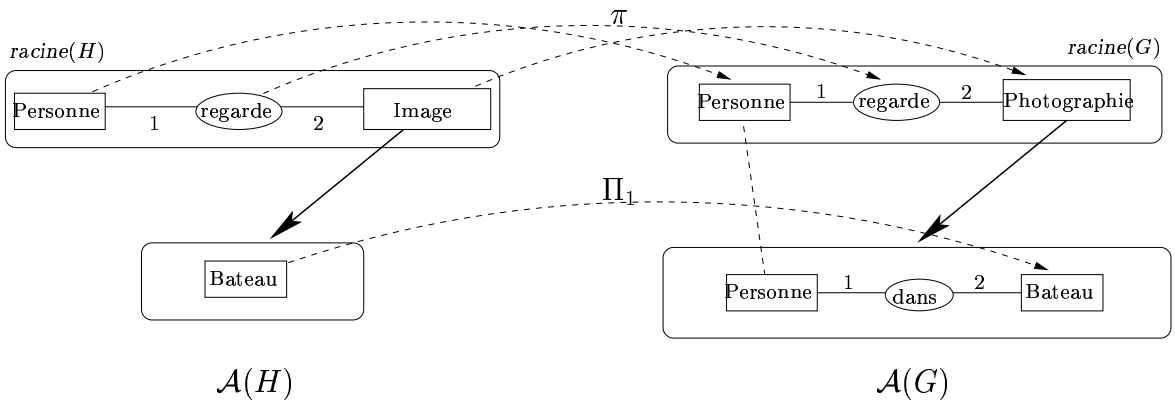


FIG. 4.16 – Exemple de projection de graphes conceptuels emboîtés.

Graphes de boîtes

Nous avons proposé, dans [Baget, 1998, Baget, 1999], une extension des graphes conceptuels emboîtés à des objets que nous avons appelé « graphes de boîtes ». Nous allons, très brièvement, en rappeler les principes et les intérêts. Les différences avec les graphes conceptuels emboîtés sont :

- nous ne considérons pas une racine unique pour un graphe de boîtes G ;
- les arguments d'un sommet relation peuvent être des sommets concepts appartenant à des graphes conceptuels simples différents de $\mathcal{A}(G)$, auquel cas ce sommet relation est dit décontextualisé ;
- la projection d'une racine n'est pas obligatoirement dans une racine ;
- si r est un sommet relation décontextualisé de H, dont les arguments sont $c_1, \ldots, c_p$, alors son image par Π dans G doit être un sommet relation (pas nécessairement décontextualisé), dont le type est plus spécifique et dont les arguments sont $\Pi(c_1), \ldots, \Pi(c_k)$.

Un graphe de boîtes est donc défini par un ensemble de graphes emboîtés $G_1, \ldots, G_k$, et par un ensemble de sommets relations décontextualisés dont les arguments appartiennent à différents graphes élémentaires. Notons que les graphes emboîtés sont un cas particulier de graphes de boîtes, qui n'ont pas de sommets relations décontextualisés, qui n'ont qu'un seul graphe racine, et dont on force, d'une manière ou d'une autre, la projection à se calculer de racine à racine.

La FIG. 4.17 montre les représentations arborées de deux graphes de boîtes H et G, et une projection Π de H dans G qui, notons le, n'envoie pas la racine de H dans une des deux racines de G.

Cette extension des graphes conceptuels emboîtés aux graphes de boîtes répond à plusieurs besoins [Baget, 1998, Baget, 1999]. En particulier, la possibilité d'avoir plusieurs racines, jointe à une projection qui n'est pas obligatoirement de racine à racine, permet de poser des requêtes sans connaître la « méthodologie de hiérarchisation » utilisée pour construire le graphe G. Par exemple, si nous enfermons chaque sommet concept d'un graphe H dans sa propre racine, et si tous ses sommets relations sont décontextualisés, alors la

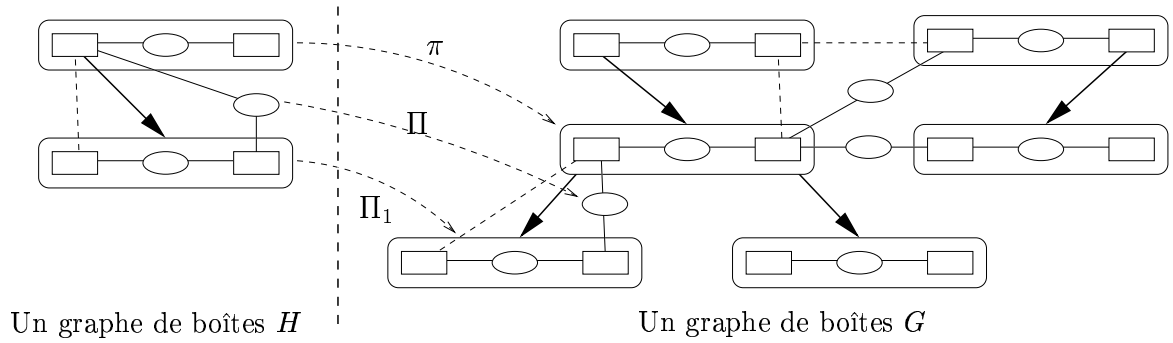


FIG. 4.17 – Représentations arborées de graphes de boîtes et projection.

projection que nous calculons ne dépend pas de la hiérarchisation en boîtes du graphe G. Cette possibilité a une grande importance pour pour pouvoir écrire des bibliothèques de règles de graphes emboîtés génériques.

4.3.2 Transformations en graphes conceptuels simples

Nous allons maintenant montrer que la projection de graphes de boîtes comme celle de graphes conceptuels emboîtés peut s'exprimer par la projection de graphes conceptuels simples (et donc la projection de graphes élémentaires). Pour cela, nous allons exhiber deux transformations similaires, *B2S* et *CE2S*, qui conservent toutes les projections. Le problème est que la forme normale des graphes conceptuels simples obtenus ne correspond pas à un graphe obtenu par la transformation d'un graphe conceptuel emboîté (resp. de boîtes).

Des graphes conceptuels emboîtés aux graphes conceptuels simples

Dans chacune des transformations que nous allons utiliser, le support d'un graphe conceptuel emboîté (ou d'un graphe de boîtes) est enrichi par l'ajout d'un type de concept, *Box*, et de deux types de relations binaires, *desc* (qui indiquera qu'une boîte est la description d'un sommet concept) et $\in$ (qui indiquera qu'un sommet concept appartient à une boîte). Ces nouveaux types sont incomparables avec tous les types présents dans le support. Enfin, pour traduire les graphes conceptuels emboîtés, nous avons besoin d'un autre type de concept, *Root*, et d'un marqueur individuel, Ω, qui servira à identifier cette racine.

Décrivons tout d'abord la transformation *B2S* qui transforme un graphe de boîtes G en un graphe conceptuel simple G'. Ce graphe G' est constitué de l'union disjointe des graphes conceptuels simples composant les sommets de $\mathcal{A}(G)$, auquel on rajoute les sommets relations décontextualisés, dont les arguments sont les sommets concepts correspondant à leurs arguments dans G. Pour chaque sommet de $\mathcal{A}(G)$, nous rajoutons un sommet concept typé par *Box*. Chaque sommet concept issu d'un sommet de G est alors relié par une relation typée par $\in$ au sommet concept représentant sa boîte. Chaque sommet concept typé par

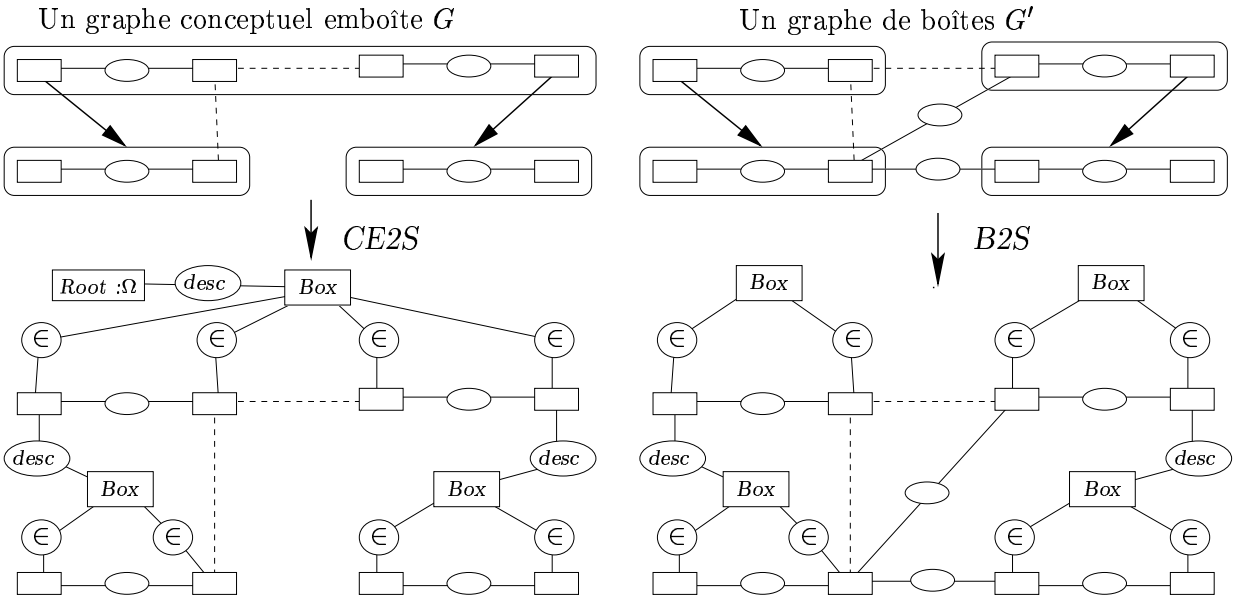


FIG. 4.18 – Des graphes emboîtés (ou de boîtes) aux graphes conceptuels simples.

Box (sauf s'il représente une boîte racine) est relié par une relation typée par *desc* au sommet concept dont cette boîte est la description.

La transformation *CE2S*, qui transforme un graphe conceptuel emboîté en un graphe conceptuel simple est identique, mis à part qu'elle n'a pas à considérer de sommets relations décontextualisés, est que l'unique sommet concept représentant une boîte racine est relié par *desc* à un sommet concept de type *Root* et de marqueur individuel Ω.

Ces deux transformations sont illustrées dans la FIG. 4.18. Notons que ces transformations permettent facilement de retrouver le graphe conceptuel emboîté (ou de boîtes) à partir duquel on a obtenu le graphe élémentaire. Nous ne donnerons pas ici, exercice qui serait fastidieux, la caractérisation des graphes conceptuels simples que l'on peut retraduire en graphes conceptuels emboîtés (ou de boîtes).

Il ne reste plus qu'à vérifier, ce qui est immédiat, que si Π est une projection d'un graphe conceptuel emboîté H dans un graphe conceptuel emboîté G, alors Π peut s'étendre de manière unique à une projection π de $CE2S(H)$ dans $CE2S(G)$. Réciproquement, si π est une projection de $CE2S(H)$ dans $CE2S(G)$, alors la restriction de π aux sommets qui sont issus de H est une projection de H dans G. La même propriété est vérifiable pour la transformation *B2S* des graphes de boîtes.

Mise du graphe conceptuel simple obtenu sous forme normale

Le graphe conceptuel simple que nous obtenons, que ce soit par la transformation *CE2S* ou par la transformation *B2S*, n'est pas nécessairement sous forme normale. Cependant, si nous le mettons sous forme normale, nous obtenons un graphe conceptuel simple qui peut ne plus se retraduire en un graphe conceptuel emboîté (resp. de boîtes). Ceci est illustré

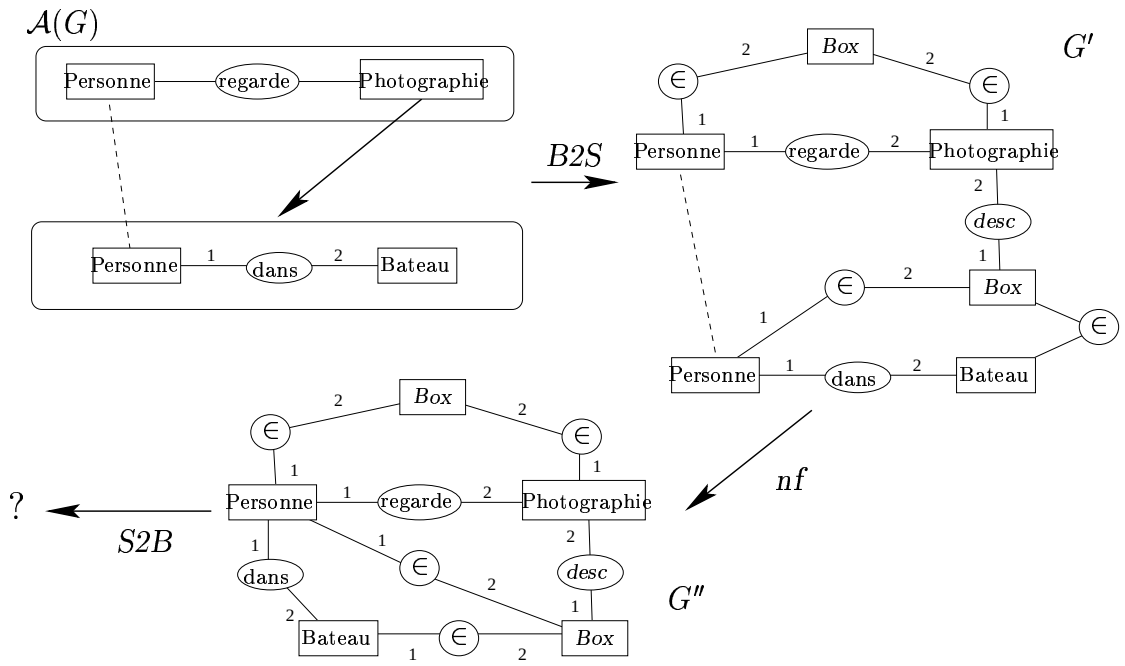


FIG. 4.19 – Transformation en graphe conceptuel simple, et mise sous forme normale.

dans la FIG. 4.19. Un graphe de boîtes G est traduit, par l'intermédiaire de *B2S*, en un graphe conceptuel simple G' qui est ensuite mis sous forme normale. Le graphe G'' ainsi obtenu ne correspond plus à la traduction d'un graphe de boîtes, puisque le sommet de type *Personne* appartient à deux boîtes distinctes.

Note à l'attention des rapporteurs : Je me suis aperçu au tout dernier moment que les deux transformations *CE2S* et *B2S*, bien que conservant bien toutes les projections, ne satisfont pas à deux propriétés que je trouve désirables, *i.e.* pour un graphe emboîté G donné :

1. $\Phi(G) = \Phi(CE2S(G))$ où Φ est l'extension usuelle de la sémantique Φ aux graphes emboîtés [Simonet, 1998]
2. $\Psi(G) = \Psi(CE2S(G))$ où Ψ est l'extension Ψ aux graphes emboîtés *ibid.*

Je supprime donc cette partie erronée, et essaierai de trouver une telle transformation respectant cette propriété. L'équivalence entre la projection de graphes emboîtés et la projection de graphes élémentaires reste valide, ce qui est l'objectif principal de cette section.

Chapitre 5

Algorithmes pour la projection

Sommaire

Nous nous intéressons maintenant aux algorithmes permettant, étant donnés un support $\mathcal{S}$ et deux graphes (élémentaires, simples, emboîtés...) G et H, de répondre aux questions suivantes :

1. Existe-t-il une projection de H dans G ? (problème de décision)
2. Exhiber, si elle existe, une projection de H dans G (problème de recherche)
3. Exhiber toutes les projections de H dans G (problème d'énumération)

Nous avons vu qu'il était possible de résoudre ces problèmes dans les différents modèles de graphes conceptuels que nous avons évoqués en se ramenant aux *graphes élémentaires minimaux*. Le choix de ce modèle particulier comme abstraction de la structure de données utilisée dans tous ces modèles sera justifié au long de cette partie, mais nous en résumons ici les principales raisons :

- la multiplicité des modèles de graphes pour lesquels la recherche de projection(s) est l'opération élémentaire de raisonnement rend nécessaire l'écriture d'algorithmes *génériques*, et donc le choix d'une abstraction commune à chacun de ces modèles ;

- la minimalité des graphes, obtenue en fusionnant les relations jumelles (nous aurions pu définir cette opération dans n'importe lequel de ces modèles), peut réduire exponentiellement le nombre de projections comme le nombre d'échecs dans l'arbre de recherche d'une solution ;
- la différence entre types de concepts et types de relations unaires n'intervient à aucun moment dans les algorithmes, utiliser les graphes élémentaires comme abstraction simplifiera donc leur écriture ;
- enfin, nous verrons que la vision n-aire du graphe, considérant les relations comme des hyperarcs plutôt que comme des sommets, permet (en prenant certaines précautions), d'écrire des algorithmes plus efficaces.

Nous nous intéresserons donc aux problèmes PROJECTION ? (existence d'une projection), PROJECTION (recherche d'une projection), et PROJECTIONS (recherche de toutes les projections) d'un graphe élémentaire minimal dans un autre. Nous commencerons par rappeler la complexité théorique de ces problèmes, et exhiberons deux transformations, inspirées de [Mugnier and Chein, 1996], qui prouvent l'équivalence entre le problème PROJECTION ? et la *satisfaction d'un réseau de contraintes* (CSP, pour CONSTRAINT SATISFACTION PROBLEM). L'équivalence très forte entre ces deux problèmes (conservation de *toutes* les solutions, conservation de la structure du graphe requête H) nous permettra d'adapter de nombreux algorithmes étudiés dans la communauté CSP aux graphes conceptuels. Enfin, nous proposerons un algorithme original, BCC, dont l'efficacité repose sur la structure du graphe H, et plus précisément sur le nombre de ses composantes biconnexes. Alors que nous avons développé cet algorithme dans le formalisme des réseaux de contraintes binaires [Baget and Tognetti, 2001], nous l'adaptons ici aux graphes élémentaires n-aires.

5.1 Complexité

Nous rappelons ici la complexité théorique des différents problèmes que nous avons évoqués : projection, projections, et irredondance. L'équivalence entre projection et satisfaction de contraintes nous donnera un outil efficace pour adapter les algorithmes développés dans la communauté CSP.

5.1.1 Complexité, rappels et notations

Pour évaluer l'efficacité d'un algorithme, nous calculerons, en fonction de la taille d'une instance d'un problème, le nombre d'opérations élémentaires nécessaires à la résolution de ce problème. Cette évaluation pourrait se faire dans le pire des cas, dans le cas moyen, ou dans le meilleur des cas. L'analyse dans le meilleur des cas ne présente pas un grand intérêt. L'analyse dans un cas moyen nécessite soit un modèle aléatoire de génération des instances, soit des choix aléatoires dans l'exécution de l'algorithme (algorithmes dits « randomisés »). Nous nous intéresserons ici essentiellement à l'analyse dans le pire des cas.

Le problème (de décision) qui nous intéresse est :

$\mathcal{S}^*$-PROJECTION ?
Données : Deux graphes élémentaires minimaux G et H définis sur le support $\mathcal{S}^*$.
Résultat : VRAI si et seulement s'il existe une $\mathcal{S}^*$-projection de H dans G.

Et, pour résoudre ce problème, il nous faudra faire appel à la résolution du sous-problème suivant, afin de pouvoir tester si deux types sont compatibles :

$\mathcal{S}$-SOUS-TYPE ?
Données : Deux types t et t' définis sur le support $\mathcal{S}$.
Résultat : VRAI si et seulement si t et t' ont même arité i et $t \leq_i t'$.

Mesure de la complexité

Nous utiliserons, pour mesurer la taille du problème de projection, les paramètres suivants :

- n_G, le nombre de sommets concepts du graphe G (resp. n_H, H) ;
- m_G, le nombre de relations du graphe G (resp. m_H, H) ;
- k, l'arité maximale des relations (la même dans G et H) ;
- $|\tau|$, la taille maximale des ensembles de types utilisés comme types des relations des graphes élémentaires minimaux G et H ;
- d_G, le nombre maximal de relations incidentes à un sommet concept de G (resp. d_H, H) ;
- t la taille du plus grand des T_i dans $\mathcal{S}$;
- s la taille de la plus grande des relations de couverture $\leq_i'$ sur $\mathcal{S}$.

Soit $f(n)$ une fonction qui calcule, dans le pire des cas, le nombre d'opérations nécessaires à l'exécution de l'algorithme en fonction de la taille n du problème. Une approximation de cette fonction sera souvent donnée par la notation asymptotique (voir, par exemple, le Chap. 2 de [Cormen et al., 1990]). Nous noterons :

- $f(n) = O(g(n))$ s'il existe c et n_0 telles que $0 \leq f(n) \leq cg(n)$ pour tout $n \geq n_0$. On dit que $g(n)$ est une *borne supérieure asymptotique* de $f(n)$.
- $f(n) = \Omega(g(n))$ s'il existe c et n_0 telles que $0 \leq cg(n) \leq f(n)$ pour tout $n \geq n_0$. On dit que $g(n)$ est une *borne inférieure asymptotique* de $f(n)$.
- $f(n) = \Theta(g(n))$ si $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$. On dit que $g(n)$ est une *borne asymptotique approchée* de $f(n)$.

Les mêmes bornes peuvent se définir pour une évaluation dans le cas moyen ou le meilleur cas. Elles seront aussi utilisées pour approximer *l'espace mémoire* utilisé au cours de l'exécution d'un algorithme.

Classes de complexité

Les algorithmes que nous proposons pour résoudre le problème PROJECTION ? ont tous une complexité exponentielle dans le pire des cas. En prouvant qu'un problème est NP-complet [Cook, 1971][Garey and Johnson, 1979], nous ne démontrons pas qu'il n'existe pas

d'algorithme polynomial permettant de résoudre ce problème. Cependant, nous montrons qu'il est aussi compliqué que tous ceux d'une grande classe de problèmes. Si un seul de ces problèmes admettait un algorithme polynomial de résolution, alors un tel algorithme existerait pour tous les problèmes de cette classe.

Un problème de décision est dit appartenir à la classe NP s'il existe une machine de Turing (voir Sect. 6.2.1) (non déterministe) résolvant effectivement ce problème telle que, pour toute instance du problème à laquelle la réponse est OUI, il existe une dérivation de longueur polynomiale par rapport à la taille du mot initial codant ce problème telle que la machine s'arrête.

En d'autres termes, on prouve qu'un problème appartient à NP en montrant que, pour toute instance du problème à laquelle la réponse est OUI, on peut fournir un indice (de taille polynomiale) permettant de vérifier cette réponse en un temps polynomial. Un tel indice est appelé un certificat polynomial.

Enfin, pour prouver qu'un problème de décision Π appartenant à NP est NP-complet, il nous faudra prouver qu'il est au moins aussi compliqué que tous les autres problèmes de la classe NP. Plus simplement, nous pouvons aussi prouver qu'il est au moins aussi compliqué qu'un problème Π' connu comme étant lui-même NP-complet. Pour cela, nous utiliserons une transformation polynomiale (une réduction au sens de [Karp, 1972]). Il s'agit d'une application $f : \Pi' \rightarrow \Pi$, calculable polynomialement, qui associe à toute instance I de Π' une instance $f(I)$ de Π telle que la réponse à I est OUI si et seulement si la réponse à $f(I)$ est OUI.

5.1.2 PROJECTION ? : un problème NP-complet

Nous montrons ici, par plusieurs réductions différentes, que PROJECTION ? est un problème NP-complet. La première démonstration est immédiate, puisqu'elle repose sur le fait que la projection est une généralisation de l'homomorphisme de graphes. La deuxième transformation que nous utiliserons montre que toute instance de 3-SAT (satisfiabilité d'une conjonction de clauses à trois variables) peut s'exprimer par une instance de PROJECTION ?. Cette transformation sera la base des transformations plus complexes que nous verrons au Chap. 7. Enfin, nous montrons l'équivalence entre PROJECTION ? et CSP, le problème de satisfaction d'un réseau de contraintes. Cette équivalence est à la base des algorithmes proposés dans la Sect. 5.2.

Complexité de la projection

Nous montrons que PROJECTION ? est un problème NP-complet. Dans le cadre des graphes conceptuels simples, ce théorème a été prouvé par [Chein and Mugnier, 1992], par l'intermédiaire d'une transformation depuis CLIQUE.

Théorème 5.1 *Le problème* $\mathcal{S}^*$-PROJECTION ? *appartient à la classe NP si et seulement si le problème* $\mathcal{S}$-SOUS-TYPE ? *appartient à la classe NP.*

Si, en outre, il existe un T_i *est non vide, avec* $i \geq 2$*, alors* $\mathcal{S}^*$-PROJECTION ? *est un problème NP-complet.*

Preuve: Supposons que $\mathcal{S}$-SOUS-TYPE ? soit un problème de la classe NP. Alors pour chaque paire (t, t') de types de $\mathcal{S}$, $t \leq t'$ admet un certificat polynomial. Un certificat polynomial pour « il existe une projection de H dans G » est fourni par une application $\pi : V(H) \to V(G)$, un certificat, *choix*, pour les relations de H, et pour chaque relation r de H, un certificat polynomial pour $\tau(choix(r)) \leq \tau(r)$. On peut vérifier en temps polynomial que, pour chaque relation r de H, les arguments de $choix(r)$ sont les images par π des arguments de r, et que $\tau(choix(r)) \leq \tau(r)$ (puisque nous en avons un certificat polynomial).

Réciproquement, si $\mathcal{S}$-SOUS-TYPE ? n'appartient pas à NP (*i.e.* il n'est pas possible, dans le cas général, de fournir un certificat polynomial), alors il n'est pas non plus possible de fournir un certificat polynomial pour la projection d'un graphe contenant une seule relation d'arité 1 dans un graphe ne contenant lui aussi qu'une seule relation d'arité 1.

S'il existe $i \geq 2$ tel que T_i est non vide, alors tout graphe G (non orienté, sans boucle, sans étiquette) peut être transformé par une transformation polynomiale f en un graphe élémentaire de $\mathcal{G}^e(\mathcal{S})$, de telle façon que H est homomorphe à G si et seulement si il existe une projection de $f(H)$ dans $f(G)$ (voir preuve du Th. 3.4). $\mathcal{S}^*$-PROJECTION ? est alors au moins aussi compliqué que le problème HOMOMORPHISME DE GRAPHES, qui est un problème NP-complet [Levin, 1973]. [1] La NP-complétude de HOMOMORPHISME DE GRAPHES dérive immédiatement de la remarque suivante : un graphe H est homomorphe à un K_n (le graphe complet à n sommets) si et seulement si H est n-colorable. HOMOMORPHISME DE GRAPHES est donc un cas particulier du problème k-COLORATION, prouvé NP-complet dans [Karp, 1972] (transformation depuis 3-SAT).

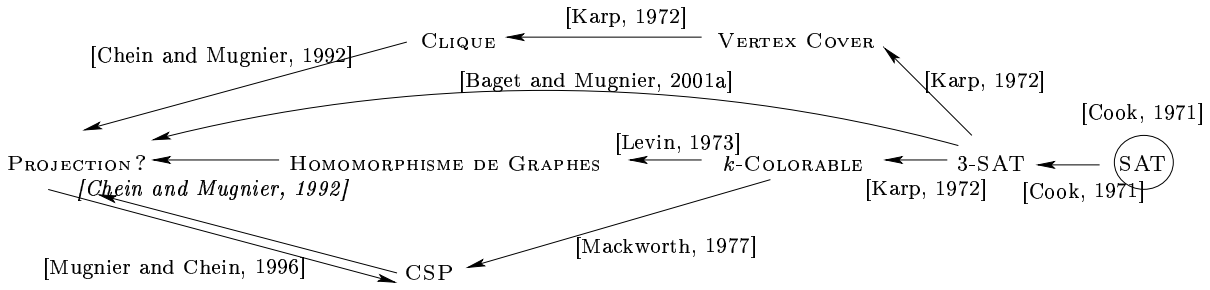


FIG. 5.1 – Quelques réductions prouvant la NP-complétude de PROJECTION ?.

Enfin, si les seuls types de relations du support sont unaires, le problème devient : vérifier que, pour chaque relation de H, il existe une relation de type plus spécifique dans G. Ce problème est polynomial si le test sur les types est polynomial. □

[1] Notons que le problème GRAPH HOMOMORPHISM qui en rend compte dans [Garey and Johnson, 1979] est en fait un cas particulier d'homomorphisme, répondant à la question « existe-t-il un homomorphisme surjectif complet de H dans G ? ».

PROJECTION ? **et** 3-SAT

Une autre preuve de la NP-complétude de PROJECTION ? peut être donnée par une transformation du problème 3-SAT.

3-SAT
Données : Une formule $\mathcal{F}$ sous la forme d'une conjonction de disjonctions (clauses), chaque clause ayant au plus 3 littéraux.
Question : $\mathcal{F}$ est-elle satisfiable ?

Nous donnons cette transformation [Baget and Mugnier, 2001a] car elle forme le noyau de plusieurs transformations qui seront utilisées dans le Chap. 7.

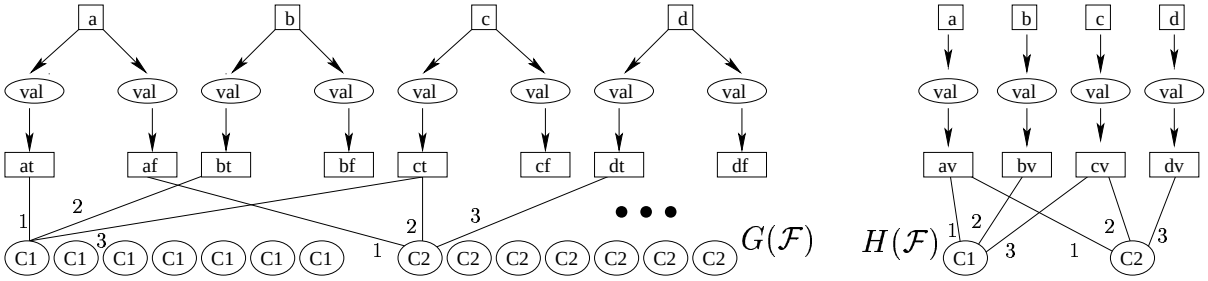


FIG. 5.2 – Transformation d'une instance de 3-SAT en instance de PROJECTION ?.

Soit $\mathcal{F} = C_1 \wedge \ldots C_k$ une instance de 3-SAT. Nous construisons le graphe $G(\mathcal{F})$ de la façon suivante :

- pour chaque variable x de $\mathcal{F}$, nous construisons trois sommets concepts, respectivement typés par *x*, *xt*, et *xf*, et deux relations binaires de type *val* reliant le premier de ces sommets aux deux autres (intuitivement, ceci s'interprète par « x peut être valué à VRAI ou à FAUX ») ;
- pour chaque clause $C_i = (x \vee y \vee z)$ de $\mathcal{F}$ (où x, y et z sont des littéraux), nous ajoutons 7 relations de type C_i représentant les 7 évaluations possibles de cette clause à VRAI (chacune a pour premier argument *xt* ou *xf*, pour deuxième argument *yt* ou *yf*, et pour troisième argument *zt* ou *zf*) ;
- pour les clauses de taille 2 ou 1, nous procédons de façon similaire, en ajoutant 3 relations binaires dans le premier cas, et une relation unaire dans le second ;

Remarquons qu'imposer des clauses de taille maximale fixée par une constante k est d'une grande importance pour obtenir une transformation polynomiale, puisque cette transformation génère $2^k - 1$ relations pour chaque clause de taille k.

Dans le graphe $H(\mathcal{F})$, chaque variable x est représentée par un sommet concept typé par *x*, relié par une relation binaire typée par *val* à un sommet concept typé par *xv*. Notons que les types de la forme *xv* sont plus généraux que les types *xt* et *xf*, et que ce sont les seules comparaisons possibles dans le support. Pour chaque clause C_i, nous rajoutons une relation dont l'arité est la taille de C_i dont les arguments sont les sommets concepts de la forme *xv* associés aux littéraux de la clause. Ce graphe, considéré comme une requête,

signifie « existe-t-il une valuation des variables telle que toutes les clauses soient valuées à VRAI ? »

Cette transformation, associée à la formule de 3-SAT $\mathcal{F} = (a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d)$ est illustrée dans la FIG. 5.2. Afin de sauvegarder la lisibilité du dessin, nous n'avons pas représenté dans le graphe $G(\mathcal{F})$ tous les traits reliant les clauses aux valuations des variables. Il est immédiat de vérifier que $\mathcal{F}$ est satisfiable si et seulement s'il existe une projection de $H(\mathcal{F})$ dans $G(\mathcal{F})$.

Projection et Satisfaction de contraintes

Nous allons ici nous intéresser à la grande similarité qui existe entre les problèmes PROJECTION ? et CSP (satisfaction d'un réseau de contraintes). Par l'intermédiaire de deux réductions, nous allons non seulement montrer que ces deux problèmes sont équivalents (ce que l'on sait déjà, puisqu'ils sont tous deux NP-complets), mais que ces transformations conservent à la fois la structure du graphe et l'ensemble de toutes les solutions. Ceci nous permettra de traduire les algorithmes d'un formalisme à l'autre.

Les réseaux de contraintes ont été utilisés dès le début des années 60, mais leur premier traitement algébrique est dû à [Montanari, 1974]. Depuis [Mackworth, 1977], ces réseaux de contraintes se sont répandus dans de nombreux domaines de l'Intelligence Artificielle, et sont maintenant un outil utilisé dans de nombreuses applications.

Définition 5.1 (Réseau de contraintes) *Un réseau de contraintes sur un domaine fini $\mathcal{D}$ est un triplet $N = \{V, U, \delta\}$ où V est un ensemble de sommets appelés* variables, *$U \subseteq V^*$ est un ensemble d'hyperarcs, et δ est une application qui à tout hyperarc c d'arité k associe un ensemble de tuples de $\mathcal{D}^k$. Un couple $(c, \delta(c))$, où $c \in U$, est appelé une* contrainte, *et $\delta(c)$ contient les tuples de* valeurs autorisées *pour les arguments de c.*

On appelle *instanciation* d'une variable x de $V(\mathcal{N})$ l'assignation d'une valeur $i(x) = d \in \mathcal{D}$ à cette variable. Un réseau est dit *instancié* si toutes ses variables sont instanciées. Une instanciation du réseau est dite *consistante* si, pour chaque contrainte $(c, \delta(c))$ du réseau, dont le voisinage est $\gamma(c) = \{x_1, \ldots, x_p\}$, on a $(i(x_1), \ldots, i(x_p)) \in \delta(c)$. Dans le cas contraire, on dit que cette contrainte est *violée*. S'il existe une instanciation qui ne viole aucune contrainte, le réseau est dit *consistant*. Le problème CSP est donc :

CSP ?
Données : Un réseau de contraintes $\mathcal{N}$ sur un domaine fini $\mathcal{D}$.
Question : Existe-t-il une instanciation consistante de $\mathcal{N}$?

L'ensemble des instanciations consistantes d'un réseau $\mathcal{N}$ sera appelé l'ensemble des *solutions* de $\mathcal{N}$.

Nous représenterons un réseau de contraintes par un hypergraphe, de façon similaire à la représentation des graphes élémentaires. Nous dessinerons les variables par des rectangles, et les contraintes par des ovales.

Prenons par exemple le domaine $\mathcal{D} = \{a, b, c\}$, et le réseau $\mathcal{N}$ défini par :

$$\begin{array}{rcl} U(\mathcal{N}) & = & \{x_1, x_2, x_3\} \\ V(\mathcal{N}) & = & \{c_1 = (x_1, x_2, x_3), c_2 = (x_3, x_2), c_3 = (x_2), c_4 = (x_3)\} \\ \delta & : & \delta(c_1) = \{(a,b,b), (c,b,a), (c,a,b)\}; \delta(c_2) = \{(a,b), (b,a)\}; \\ & & \delta(c_3) = \delta(c_4) = \{(a), (b)\} \end{array}$$

La FIG. 5.3 illustre la représentation graphique de ce réseau. Notons que les contraintes d'arité 1 sont souvent « intégrées » à la variable : si une variable x n'est incidente à aucune contrainte d'arité 1, on dit que son domaine $dom(x)$ est $\mathcal{D}$, sinon son domaine sera celui qui est autorisé par cette contrainte : soit c telle que $\gamma(c) = (x)$, on pose $dom(x) = \{d \in \mathcal{D}, (d) \in \delta(c)\}$. La représentation est plus simple, et on peut alors ignorer les contraintes unaires en posant qu'une instanciation associe à une variable un élément de son domaine. Les deux graphes de la FIG. 5.3 sont donc les représentations de deux formulations équivalentes du même problème. Notons que cette transformation est identique à celle qui remplace les relations unaires par des types de concepts dans les graphes élémentaires.

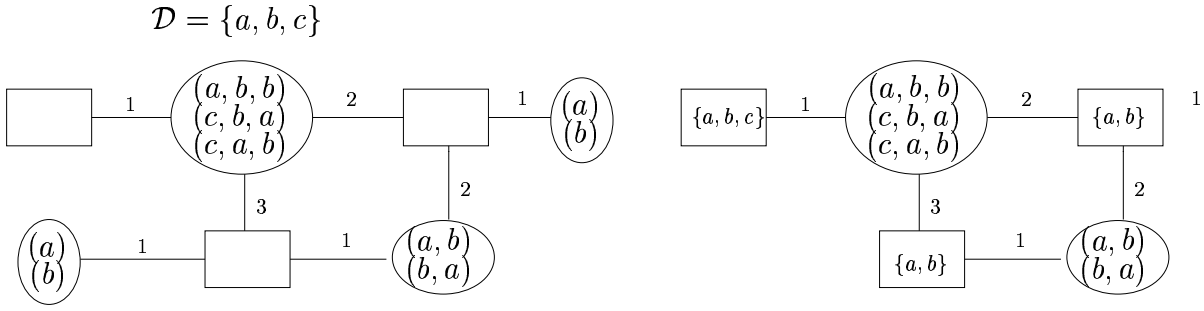


FIG. 5.3 – Deux représentations équivalentes d'un réseau de contraintes.

Ce réseau a deux instanciations consistantes : $i_1 : x_1 \mapsto c,\ x_2 \mapsto a,\ x_3 \mapsto b$ et $i_2 : x_1 \mapsto c,\ x_2 \mapsto b,\ x_3 \mapsto a$. On peut vérifier que toute autre instanciation des variables de ce réseau viole au moins une contrainte.

De CSP à PROJECTION

L'équivalence très forte entre projection et satisfaction de contraintes a été remarquée par [Mugnier and Chein, 1996] et, de manière indépendante dans un formalisme très similaire, par [Rudolf, 1998]. Les transformations que nous présentons ici sont une variante de celles utilisées dans [Mugnier and Chein, 1996].

Soit $\mathcal{S}$ un réseau de contraintes. Alors nous définissons la transformation *C2P* (réseau de *C*ontraintes à *P*rojection) de la façon suivante :

- construction du support $\mathcal{S}$:
  - à chaque variable x_i du réseau, nous associons un type d'arité 1, t_i ;

- à chaque valeur $d_j \in \mathcal{D}$, nous associons un type d'arité 1, d_j ;
- pour chaque arité k de contrainte présente dans le réseau, nous avons un type de relation d'arité k, r_k ;
- tous les types de $\mathcal{S}$ sont incomparables.

– construction du graphe H :
- à chaque variable x_i de $\mathcal{N}$ correspond un sommet concept x_i de type t_i (le type associé à la variable) ;
- à chaque contrainte $c = (x_1, \ldots, x_k)$ correspond une relation $r = (x_1, \ldots, x_k)$, dont le type est r_k.

– construction du graphe G :
- pour chaque variable x_i, pour chaque valeur $d_j \in dom(x_i)$, nous créons un sommet concept de type $t_i \sqcap d_j$;
- pour chaque contrainte $c = (x_1, \ldots, x_k)$ d'arité k, pour chaque tuple autorisé $(d_1, \ldots, d_k) \in \delta(c)$, nous créons une relation r de type r_k et d'arité k telle que, pour $1 \leq i \leq k$, $\gamma_i(r)$ est le sommet dont le type est $t_i \sqcap d_i$.

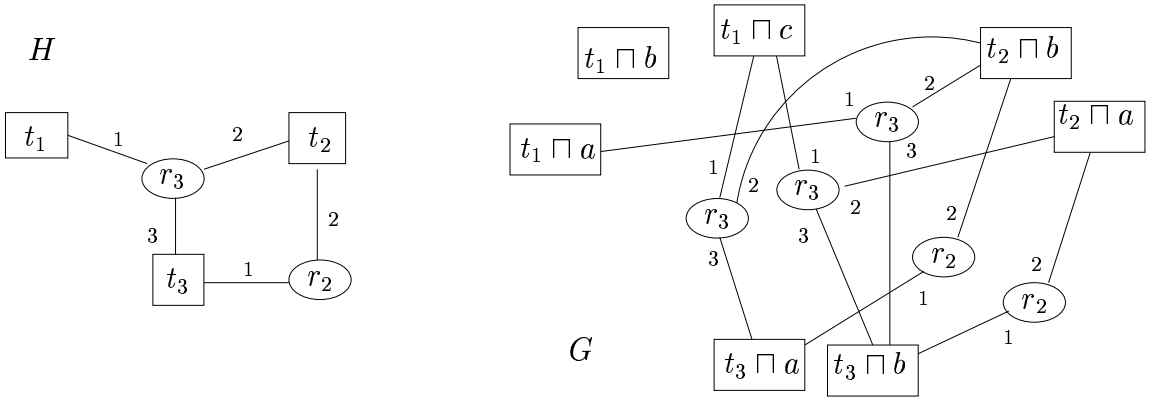


FIG. 5.4 – Les graphes H et G obtenus par transformation du réseau de la FIG. 5.3.

La FIG. 5.4 illustre le résultat de cette transformation, appliquée au réseau de contraintes de la FIG. 5.3. Notons que les graphes élémentaires obtenus sont minimaux, et que la structure du graphe H obtenu est exactement celle du réseau de contraintes initial.

Dans notre exemple, les seules projections de H dans G sont : π_1 qui associe au sommet concept de type t_1 le sommet concept de type $t_1 \sqcap c$, au sommet concept de type t_2 celui de type $t_2 \sqcap a$, et au sommet concept de type t_3 celui de type $t_3 \sqcap b$; et π_2 qui à ces sommets concepts associe, dans l'ordre, les sommets concepts de type $t_1 \sqcap c$, $t_2 \sqcap b$, et $t_3 \sqcap a$.

Propriété 5.1 *Soit $\mathcal{N}$ un réseau de contraintes, et C2P($\mathcal{N}$) = $(\mathcal{S}, H, G)$. Alors $\mathcal{N}$ est consistant ssi il existe une $\mathcal{S}^*$-projection de H dans G. De plus, il existe une bijection entre les solutions du réseau de contraintes et les projections de H dans G.*

Preuve: Nous montrons comment toute solution du réseau de contraintes peut être transformée en projection par une application injective, et réciproquement.

($\Rightarrow$) Voir que si $((x_1, d_{i_1}), \ldots, (x_n, d_{i_n}))$ est une solution du réseau de contraintes, alors $((t_1, t_1 \sqcap d_{i_1}), \ldots, (t_n, t_n \sqcap d_{i_n}))$ (où nous identifions de manière unique les sommets concepts par leur type) est une projection de H dans G. Il est aussi immédiat de vérifier que cette transformation des solutions est injective.
($\Leftarrow$) Si $((t_1, u_1), \ldots, (t_n, u_n))$ est une projection de H dans G, alors cette projection est nécessairement de la forme $((t_1, t_1 \sqcap d_{i_1}), \ldots, (t_n, t_n \sqcap d_{i_n}))$ (sinon, les types ne seraient pas compatibles). Alors $((x_1, d_{i_1}), \ldots, (x_n, d_{i_n}))$ est une solution du réseau de contraintes. Là aussi, vérifier l'injectivité de cette transformation est immédiat. □

De Projection à CSP

Réciproquement, prenons $\mathcal{S}$ un support, et deux graphes minimaux G et H sur $\mathcal{S}^*$. Nous construisons le réseau de contraintes $\mathcal{N} = P2C(\mathcal{S}, H, G)$ de la façon suivante :

- les variables du réseau $\mathcal{N}$ sont les sommets concepts du graphe H ;
- le domaine de chaque variable x est constitué de l'ensemble des sommets concepts de G ;
- les contraintes du réseau $\mathcal{R}$ sont les relations du graphe élémentaire H (*i.e.* $c = (x_1, \ldots, x_k)$ est une contrainte de $\mathcal{N}$ ssi il existe $c \in U(H)$ telle que $\gamma(c) = (x_1, \ldots, x_k)$: remarquons que si une telle relation c existe, elle est unique puisque H est minimal) ;
- pour chaque relation $r \in U(H)$, pour chaque relation $r' \in U(G)$, si $\tau(r') \leq \tau(r)$, alors nous ajoutons $\gamma(r')$ aux tuples autorisés de la contrainte associée à r.

Ici aussi, la structure du graphe H est conservée.

Voir qu'en appliquant *P2C* aux graphes H et G de la Fig. 5.4, on retrouve le réseau de contraintes de la Fig. 5.3. Ceci n'est cependant pas toujours vrai.

Propriété 5.2 *Soient $\mathcal{S}$ un support, H et G deux graphes élémentaires minimaux sur $\mathcal{S}^*$, et $\mathcal{N} = P2C(\mathcal{S}, H, G)$. Alors $\mathcal{N}$ est consistant ssi il existe une $\mathcal{S}^*$-projection de H dans G. De plus, il existe une bijection entre les solutions du réseau de contraintes et les projections de H dans G.*

Preuve: La preuve est immédiate : voir que $((x_1, y_{i_1}), \ldots, (x_n, y_{i_n}))$ est une projection de H dans G si et seulement si $((x_1, y_{i_1}), \ldots, (x_n, y_{i_n}))$ est une solution du réseau de contraintes. □

Discussion : projection vs CSP

Les transformations *P2C* et *C2P* démontrent que CSP et Projection(s) sont deux problèmes extrêmement similaires. On peut passer d'un problème à l'autre par des transformations qui préservent aussi bien la structure du problème (tout au moins celle du graphe H ou du réseau de contraintes) que l'ensemble des solutions de ce problème.

Alors, si ces deux problèmes sont si similaires, une question se pose « pourquoi étudier le problème de la projection[2] ? » Nous pouvons répondre à cette question en mettant en lumière les points suivants :

- PROJECTION peut se voir comme une requête (définie par l'utilisateur) dans une base de données (un graphe distinct existant). La distinction entre ces deux graphes n'est pas présente dans le formalisme des réseaux de contraintes, qui nécessite de créer, à chaque requête, un réseau encodant à la fois la requête et la base de faits.
- les extensions que nous allons considérer (les règles, par exemple) sont difficilement exprimables dans le modèle CSP (mais remarquons, dans l'autre sens, qu'il est difficile d'exprimer, par exemple, des domaines continus dans le modèle des graphes simples).

Ayant ainsi affirmé la nécessité d'étudier PROJECTION comme un problème distinct/autonome, il se pose une autre question : « pourquoi, pour résoudre PROJECTION ne pas traduire le problème dans le formalisme des réseaux de contraintes et utiliser les outils existants ? »

Ce thème sera présent, en toile de fond, dans l'ensemble de cette section. Mais nous pouvons immédiatement exposer un type de problème, auquel la projection pourrait répondre, mais qu'il serait difficile de traduire en réseau de contraintes.

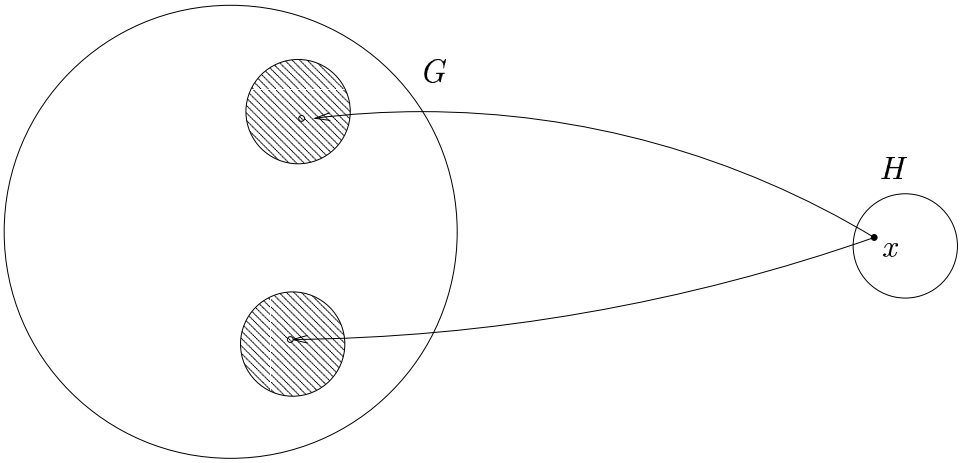


FIG. 5.5 – Un graphe G trop gros pour pouvoir construire les domaines via *P2C*.

Supposons G un graphe de très grande taille (par exemple, le graphe du web ?) et H un graphe (une requête) de taille raisonnable, tel qu'un de ses sommets concepts x a un petit nombre d'images possibles dans G (x peut être vu comme typé par un mot-clef rare). Alors il est possible (si H est connexe et si l'algorithme de recherche repose sur la relation de voisinage) que la recherche n'implique qu'un nombre réduit de sommets concepts de G. Pourtant, la construction du CSP équivalent nécessiterait de considérer l'ensemble des sommets de G... ce qui peut être impossible dans le cas de très grands graphes.

[2]Nous posons cette question plutôt que sa réciproque. « Pourquoi étudier les CSP ? » pour deux raisons : 1. Ce mémoire est dédié aux graphes conceptuels et 2. l'utilisation fréquente des CSP dans de nombreux domaines de l'IA ne nécessite pas que l'on prenne ici leur défense.

De plus, même en dehors de ce cas extrême, nous devrons considérer, dès que nous utiliserons des règles, que le graphe G sera très dynamique. Le coût de la modification du CSP, à chaque application d'une règle sur G, serait prohibitif.

Enfin, alors qu'un réseau de contraintes peut être un graphe « très connecté », le graphe H représentant une requête est souvent, lorsqu'il est construit directement ou indirectement par un utilisateur, un graphe peu complexe, *i.e.* de petite taille et ayant « peu » de cycles.

C'est pour l'ensemble de ces raisons que nous avons jugé préférable d'adapter les algorithmes étudiés dans la communauté CSP au problème de la projection de graphes élémentaires, plutôt que de traduire notre problème dans un autre formalisme.

5.1.3 Autres problèmes de $\mathcal{SG}$

Dans [Chein and Mugnier, 1992], la complexité de plusieurs problèmes liés à PROJECTION est démontrée dans le cadre des graphes conceptuels simples (sans liens de coréférence). Nous adaptons ici ces résultats aux graphes élémentaires. Le fait que ces problèmes soient NP-difficiles montre que tout repose sur l'efficacité de l'algorithme utilisé pour résoudre PROJECTION. Nous rappellerons brièvement les transformations utilisées pour prouver ces différents résultats.

Les résultats ci-dessous nécessitent la NP-complétude du problème PROJECTION ?. Nous devrons donc supposer (voir Th. 5.1) que la comparaison des types dans le support est un problème de NP, et qu'il existe dans $\mathcal{S}$ des types de relations d'arité ≥ 2.

Équivalence

ÉQUIVALENCE ?
Données : Deux graphes élémentaires G et H sur un support $\mathcal{S}$.
Question : G et H sont-ils équivalents ?

Propriété 5.3 ÉQUIVALENCE ? *est un problème NP-complet.*

Preuve: Le certificat polynomial est la donnée des deux projections. Soit $(\mathcal{S}, G, H)$ une instance de PROJECTION ?. Nous la transformons en instance de ÉQUIVALENCE ? de la façon suivante : 1) construire un graphe G_1 obtenu à partir de G en lui rajoutant un sommet concept de type New_C, relié à chaque sommet concept de G par une relation binaire de type New_R ; et 2) construire un graphe G_2 obtenu à partir de l'union disjointe S de G et de H en lui rajoutant un sommet concept de type New_C, relié à chaque sommet concept de S par une relation binaire de type New_R. Les deux nouveaux types ne sont comparables à aucun autre type du support initial $\mathcal{S}$. Alors $G \preceq H$ ssi $G_1 \equiv G_2$. □

Projection injective

Dans un certain nombre de travaux utilisant les graphes conceptuels, ce n'est pas la projection qui est utilisée comme base de raisonnement, mais la projection injective (deux sommets concepts distincts ont des images distinctes). Certains domaines d'application rendent parfois nécessaire de considérer une projection injective, par exemple pour projeter des graphes représentant des molécules chimiques [Régin, 1995, Vismara, 1995].

PROJECTION INJECTIVE ?
Données : Deux graphes élémentaires G et H sur un support $\mathcal{S}$.
Question : Existe-t-il une projection injective de H dans G ?

Propriété 5.4 PROJECTION INJECTIVE ? *est un problème NP-complet.*

Preuve: Voir que ce problème est une généralisation de ISOMOMORPHISME DE SOUS-GRAPHE, problème [GT48] de [Garey and Johnson, 1979]. □

La transformation suivante est couramment utilisée dans les modélisations en graphes conceptuels, lorsque l'on veut assurer l'injectivité de la projection (ou d'une partie de la projection). Il s'agit cette fois de transformer PROJECTION INJECTIVE ? en PROJECTION ?. Soit $(\mathcal{S}, G, H)$ une instance de PROJECTION INJECTIVE ?. Nous la transformons en instance $(\mathcal{S}', G', H')$ de PROJECTION ? en rajoutant à $\mathcal{S}$ un nouveau type de relation binaire $\neq$, incomparable avec tous les autres types, et en rajoutant, pour chaque couple c, c' de sommets concepts de G (resp. H), une relation de type $\neq$ entre ces sommets. Alors π est une projection injective de H dans G si et seulement si π est une projection de H' dans G'.

Une telle relation de type $\neq$ est souvent utilisée pour les modélisations utilisant les graphes conceptuels. Si on précise dans le graphe G que certains sommets concepts représentent des entités différentes, relier deux sommets concepts par une relation de type $\neq$ dans la requête obligera la projection à les envoyer dans deux sommets distincts.

Irredondance

IRREDONDANT ?
Données : Un graphe élémentaire G sur un support $\mathcal{S}$.
Question : G est-il irredondant ?

Propriété 5.5 IRREDONDANT ? *est un problème co-NP-complet.*

Preuve: IRREDONDANT ? est le complément du problème suivant, REDONDANT ?, que nous montrons NP-complet : « étant donné un graphe élémentaire G, existe-t-il un endomorphisme de G qui n'est pas bijectif et complet ? » (voir Def. 3.7). La transformation que nous utilisons est différente de celle donnée dans [Chein and Mugnier, 1992] (qui utilisent

une transformation à partir de PROJECTION INJECTIVE ?). Ce problème admet bien un certificat polynomial (la donnée de cet endomorphisme), et nous exhibons une transformation polynomiale qui associe à toute instance $(\mathcal{S}, G, H)$ de PROJECTION ? une instance $(\mathcal{S}', G')$ de REDONDANT ?, construite de la façon suivante : G' est l'union disjointe de G et de H. Les n sommets concepts issus de H sont incidents à des relations unaires de types $1, 2, \ldots, n$, et les n' sommets concepts issus de G sont incidents à des relations unaires de types $1', 2', \ldots, n'$. Le support $\mathcal{S}'$ est composé du support $\mathcal{S}$ et des types de relations unaires que nous avons utilisés, ordonnés de la façon suivante : les types $1, 2, \ldots, n$ sont deux à deux incomparables, les types $1', 2', \ldots, n'$ sont deux à deux incomparables, ces types sont incomparables avec tous les autres types du support, et chaque type de $\{1, 2, \ldots, n\}$ est plus général que chaque type de $\{1', 2', \ldots, n'\}$. Nous supposerons, sans perte de généralité, que les graphes G et H sont connexes.
($\Rightarrow$) Supposons une projection π de H dans G. Alors $\pi' : G' \to G'$ qui associe à tout sommet concept c issu de H le sommet issu de son image par π, et à tout sommet c issu de G c lui-même, est un endomorphisme (les relations unaires incidentes aux sommets concepts de H peuvent s'envoyer sur les relations unaires incidentes aux sommets concepts de G) qui n'est pas bijectif. Il s'ensuit que G' est redondant.
($\Leftarrow$) Si G' est un graphe redondant (il existe un endomorphisme π de G' qui n'est pas bijectif et complet). G' est composé de deux composantes connexes (une issue de G, et une issue de H). Supposons que π n'envoie aucun sommet de H dans G. Donc $\pi_{|H}$ est une projection de H dans H, et il ne peut s'agir que de l'identité (ceci est assuré par les relations unaires). L'ordre sur les relations unaires interdit toute projection d'un sommet concept issu de G dans un sommet concept issu de H. Donc $\pi_{|G}$ est une projection de G dans G, et il ne peut s'agir que de l'identité (ceci est assuré par les relations unaires). Donc π est l'identité, ce qui est absurde. Donc il existe un sommet concept issu de H dont l'image par π est un sommet concept issu de G. Comme la partie de G' issue de H et celle issue de G sont deux composantes connexes distinctes, il s'ensuit que tous les sommets de la partie issue de H se projettent dans la partie issue de G. Il s'ensuit que $\pi_{|H}$ est une projection de H dans G. □

Cas polynomiaux

Un premier cas polynomial pour PROJECTION ? est démontré, dans le cadre des graphes conceptuels simples, par [Chein and Mugnier, 1992] : il s'agit du cas où le graphe H est un « arbre » (et plus précisément du cas où les seuls cycles élémentaires admis sont formés par les multi-arêtes entre un sommet relation et un sommet concept). L'algorithme BCC que nous présentons dans ce chapitre (Sect. 5.3.3) en fournit une nouvelle preuve. Ce cas polynomial est étendu dans [Baader et al., 1999] au cas où H peut être transformé en un arbre en éclatant des sommets concepts individuels (on obtient un arbre équivalent, d'un point de vue logique à H, mais qui ne l'est pas du point de vue de la projection).

[Kerdiles, 2001] étend encore ces résultats en définissant la notion de « *guarded simple graph* », qui inclut les cas précédents. Cette notion est inspirée de la notion de « formule gardée » en logique.

Les arguments structurels sur le graphe de contraintes permettent aussi d'obtenir des cas polynomiaux, suivant la structure de la requête H, pour le problème PROJECTION ?. Le cas polynomial pour les arbres est prouvé dans [Freuder, 1982]. Dans le cas où les contraintes sont des relations n-aires, de nombreuses méthodes de décomposition permettant de se ramener à un arbre sont exposées et comparées dans [Gottlob et al., 1999].

En ce qui concerne les autres problèmes apparentés à PROJECTION ?, notons que PROJECTION INJECTIVE ? reste NP-complet lorsque H est un arbre, mais devient polynomial si G et H sont tous deux des arbres [Mugnier and Chein, 1993]. [Mugnier, 1995] présente un algorithme qui calcule la forme irredondante d'un graphe dont la complexité est polynomialement liée à celle d'un algorithme résolvant PROJECTION ?. Ainsi, dans les cas où PROJECTION ? est polynomial, on peut mettre un graphe sous forme irredondante de façon polynomiale.

Plus généralement, les cas polynomiaux connus pour ÉQUIVALENT ? et IRREDONDANT ? proviennent directement des cas polynomiaux de PROJECTION ?. Enfin, dans le cas où H est un arbre, le problème consistant à compter le nombre de projections de H dans G est lui-aussi polynomial (alors que leur énumération, du fait du nombre de projections possibles, est un problème exponentiel). Un algorithme est donné dans [Mugnier, 1992].

Cette brève présentation des autres problèmes de $\mathcal{SG}$ montre que PROJECTION ? est le problème central à résoudre si l'on veut développer des algorithmes efficaces.

5.2 Adaptation d'algorithmes de CSP

La communauté « Contraintes » a développé un important travail de recherche autour de l'optimisation des algorithmes permettant de résoudre le problème de satisfaction de contraintes.

L'algorithme de base, `Backtrack` [Golomb and Baumert, 1965], s'est vu améliorer par l'utilisation de diverses techniques. Certaines reposent sur une économie du travail réalisé à chaque étape de l'algorithme (par exemple `BackMark`), d'autres sur un ordonnancement des variables qui rend plus probable la découverte rapide d'une solution (ou alternativement l'élimination rapide d'instanciations qui ne sont pas des solutions). Des résultats intéressants ont été obtenus en sacrifiant la rapidité des opérations réalisées à chaque étape de l'algorithme, afin de réduire la taille de l'arbre de recherche. Nous parlerons ici de deux importantes familles de méthodes, le filtrage (notamment avec `Forward Checking`), et le « saut arrière » (`BackJump`).

La structure du graphe a aussi été utilisée, [Freuder, 1982] montre que le problème est polynomial dans le cas des arbres. Différentes méthodes de décomposition (voir, par exemple, [Gottlob et al., 1999]) permettent de se ramener à ce cas.

Nous n'avons pas ici l'ambition de faire un état de l'art exhaustif sur les algorithmes de résolution d'un réseau de contraintes. Nous renvoyons donc le lecteur à [Meseguer, 1989] pour un état de l'art qui, s'il n'est pas récent, présente cependant un panorama des principales techniques. Pour une comparaison des différents mécanismes de `BackJump`, le lecteur

pourra se référer à [Prosser, 1993], pour `Forward Checking` à [Haralick and Elliott, 1980] ([Bessière et al., 1999] pour les contraintes non binaires), et, pour les différentes formes de maintien de consistance, à [Debruyne and Bessière, 2001]. Enfin, [Freuder, 1982] expose un algorithme polynomial lorsque le réseau de contraintes est un arbre, et différentes généralisations aux réseaux de contraintes n-aires sont présentées dans [Gottlob et al., 1999].

Cette partie a pour but de proposer un algorithme efficace, permettant de calculer une ou toutes les projections d'un graphe dans un autre. Cet algorithme bénéficiera des améliorations apportées à `BackTrack` par différents travaux sur les réseaux de contraintes. Cette adaptation des résultats obtenus dans la communauté « Contraintes » devra prendre en compte les particularités des modèles de raisonnement qui nous intéressent dans ce mémoire :

- Le graphe G qui représente la base de faits est un grand graphe, qui évoluera dynamiquement (application de règles). Il est donc hors de question de le représenter par une matrice d'adjacence.
- Ce graphe G peut être *très* grand. Nous souhaitons ne pas avoir à le parcourir tout entier (si nous pouvons l'éviter).
- Lorsque nous appliquerons des règles, nous aurons à calculer de nombreuses projections « faciles ». Notre choix d'un algorithme de projection devra donc reposer sur son faible surcoût (*overhead cost*) dans les cas faciles, autant que sur son efficacité dans la résolution de problèmes difficiles.

Dans la Sect. 5.2.1, nous construisons, à partir d'optimisations de `BackTrack` étudiées dans la communauté « Contraintes », un algorithme de recherche de projection(s) qui répond à ces spécifications. Dans la Sect. 5.3, nous ajoutons à cet algorithme un mécanisme dont l'efficacité repose sur le nombre de composantes biconnexes du graphe, BCC [Baget and Tognetti, 2001].

5.2.1 BackTrack

Nous présentons ici quelques méthodes permettant d'améliorer l'algorithme `BackTrack`. Après avoir exposé la version générique de `BackTrack` que nous allons utiliser tout au long de ce chapitre, nous étudierons :

- une méthode permettant de limiter le surcoût lié aux comparaisons de types, qui justifie algorithmiquement l'utilisation des types conjonctifs ;
- un mécanisme permettant de calculer, à chaque étape du `BackTrack`, les candidats possibles pour le sommet courant, par une méthode d'affinage utilisée dans [Baget and Tognetti, 2001] ;
- la structure de données utilisée pour cet affinage nous permet d'écrire efficacement plusieurs algorithmes développés pour le problème de satisfaction de contraintes :
  - `BackMark` [Gaschnig, 1979]
  - `Forward Checking` [Haralick and Elliott, 1980], et principalement son extension étudiée dans [Bessière et al., 1999] aux contraintes/relations n-aires.
- cette structure de données est résistante aux « sauts arrières » effectués par les différents algorithmes `BackJump`, comme aux « sauts avants » que nous étudierons dans

la Sect. 5.3.

Générer et Tester

Il est possible de vérifier en temps polynomial ($O(m_H \times m_G \times (k + \times P(s)))$) (à condition de pouvoir exécuter les comparaisons dans $\mathcal{S}$ en un temps polynomial borné par $P(s)$) si une application quelconque π de $V(H)$ dans $V(G)$ est une projection. En effet, il suffit, pour chaque relation r parmi les m_H relations de H, de vérifier s'il existe une relation r' parmi les m_G relations de G telle que : 1) les k arguments de r' sont les images par π des k arguments de r, et 2) le type de r' est plus spécifique que celui de r (testé en temps $P(s)$). Notons que nous pouvons être un petit peu plus efficace, et ne tester, pour chaque relation r de H, que les d_G relations dont le premier argument est l'image par π du premier argument de r : ce qui nous donne une complexité en $O(m_H \times d_G \times (k + \times P(s)))$.

Ceci nous donne immédiatement un algorithme de recherche : générer toutes les applications de $V(H)$ dans $V(G)$, et tester si ce sont des projections. Il y a $n_H^{n_G}$ telles applications. Il faudra toutes les parcourir si on cherche à énumérer toutes les projections, et, dans le pire des cas, elles seront toutes parcourues même si on ne cherche qu'une seule projection. La complexité de l'algorithme PROJECTION, dans le pire des cas, est donc en $O(n_H^{n_G} \times m_H \times d_G \times (k + \times P(s)))$ L'algorithme PROJECTIONS, lui, admettra comme borne inférieure $\Omega(n_H^{n_G} \times P(s))$: il sera dans tous les cas exponentiel.

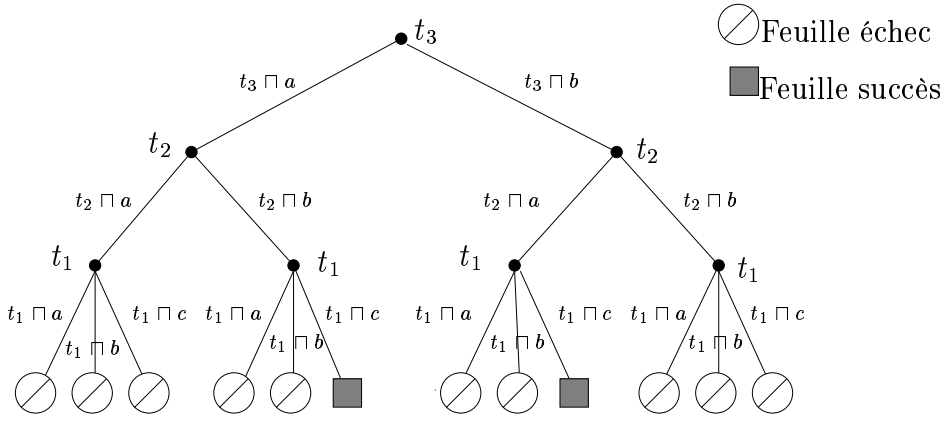


FIG. 5.6 – L'arbre de génération des applications de $V(H)$ dans $V(G)$.

Considérons les graphes G et H de la FIG. 5.4 (où tous les types distincts sont incomparables). Nous pouvons voir le processus qui génère les applications de $V(H)$ dans $V(G)$ comme un arbre : dans la FIG. 5.6, chaque branche de l'arbre représente une application de $V(H)$ dans $V(G)$ (mais nous n'avons représenté que celles qui associent à un sommet concept de type t un sommet concept de type $\leq t$). Dans H comme dans G, la donnée d'un type de concept détermine de manière unique un sommet concept, nous nous servons donc ici de ce type comme d'un identificateur. Un sommet de l'arbre indique donc, par l'intermédiaire de son étiquette, un sommet concept du graphe élémentaire H, et les arcs

(l'orientation étant induite par la hauteur dans l'arbre) représentent les sommets concepts de G dans lesquels ce sommet peut se projeter. Les feuilles de cet arbre représentent donc toutes les applications possibles de $V(H)$ dans $V(G)$, application que l'on peut lire sur la branche qui mène à cette feuille. Sur l'arbre représenté ici, deux branches seulement correspondent à une projection (feuilles succès), toutes les autres mènent à des « feuilles échecs » .

BackTrack

Algorithme 2: *BackTrack*$(\mathcal{S}, H, G)$

Données : Un support $\mathcal{S}$, deux graphes élémentaires minimaux H et G sur $\mathcal{S}^*$.

Résultat : Énumère les $\mathcal{S}^*$-projections de H dans G.

```
H ← Ordonner(H);
niveau ← 1;
remonter ← FAUX;
si (H = ∅) alors
    Solution-Trouvée(H);
sinon
    tant que (niveau ≠ 0) faire
        si (niveau = |V(H)| + 1) alors
            Solution-Trouvée(H);
            remonter ← VRAI;
        sinon
            concept-Courant ← V_H[niveau];
        si (remonter) alors
            si (Autres-Candidats(concept-Courant, G)) alors
                remonter ← FAUX;
                niveau ← Niveau-Suivant(concept-Courant);
            sinon
                niveau ← Niveau-Précédent(concept-Courant);
        sinon
            si (Chercher-Candidats(concept-Courant, G)) alors
                niveau ← Niveau-Suivant(concept-Courant);
            sinon
                remonter ← VRAI;
                niveau ← Niveau-Précédent(concept-Courant);
```

Deux idées sont à l'origine de l'algorithme `BackTrack` (que nous noterons BT).

1. un parcours en profondeur de cet arbre, si on ne cherche qu'une seule projection, nous évite d'avoir à générer toutes ses feuilles. L'algorithme s'arrêtera dès qu'il aura trouvé une feuille succès.
2. si π est une projection de H dans G, alors la restriction de π à un sous-graphe élémentaire H' de H est une projection de H' dans G. En utilisant la contraposée de cette propriété, nous allons pouvoir couper plus tôt des branches de l'arbre de recherche.

Cet algorithme est attribué à [Golomb and Baumert, 1965], mais cette méthode est beaucoup plus ancienne. Par exemple, la description de la résolution du « problème des 36 officiers », par [Euler, 1782], correspond exactement à l'exécution d'un `BackTrack`.

Nous proposons ici une version de cet algorithme (Alg. 2). Comme [Prosser, 1993], nous le présentons dans sa version itérative. En effet, bien que la version récursive soit réputée plus intuitive, cet algorithme va nous servir de noyau pour toutes les améliorations que nous allons maintenant décrire, et cette version nous permet un meilleur contrôle des sauts avant et arrière que nous utiliserons. Pour améliorer cet algorithme, nous travaillerons uniquement sur les fonctions auxiliaires qu'il utilise.

Tout d'abord, nous avons choisi d'utiliser un seul algorithme pour résoudre les problèmes PROJECTION ?, PROJECTION et PROJECTIONS. Son comportement dépend en effet de la façon dont on écrit la fonction *Solution-Trouvée*.

- pour répondre au problème PROJECTION ?, la fonction *Solution-Trouvée* stoppe l'exécution de l'algorithme et répond VRAI. Il faudra répondre FAUX à PROJECTION ? si, à la fin de l'exécution de `BackTrack`, *Solution-Trouvée* n'a pas été appelée.
- répondre au problème PROJECTION se fait de la même manière, mais *Solution-Trouvée* devra lire la projection courante dans les sommets concepts de H.
- pour énumérer les PROJECTIONS, *Solution-Trouvée* devra sauvegarder la projection courante.

Afin d'unifier ces différents problèmes, on peut, par exemple, lancer le BT dans son propre *thread* d'exécution. Le programme/utilisateur qui lance cet algorithme, à travers un schéma producteur/consommateur traditionnel, pourra lire/traiter toutes les projections trouvées, ou arrêter l'exécution de l'algorithme (après la première solution, par exemple).

Dans cet algorithme, la variable *niveau* correspond au niveau (dans l'arbre de recherche), du prochain sommet à examiner (le niveau 1 correspondant à la racine de cet arbre, et $|V(H)|$ à une feuille). La variable *remonter* aura la valeur VRAI si la branche en cours de développement ne doit pas être continuée. Nous considérerons de plus que chaque sommet concept possède un champ nommé *image*. Examinons maintenant les différentes fonctions appelées par cet algorithme.

- *Ordonner* donne un ordre (qui sera pour l'instant arbitraire) aux sommets concepts de H. Ces sommets sont rangés, dans cet ordre, dans un tableau, et chaque sommet concept connaît sa position dans ce tableau. Quitte à renommer les sommets concepts, nous noterons $x_i = (V(H))[i]$ le sommet concept placé à la position i dans ce tableau. Nous notons $pre_V(x_i)$ l'ensemble des sommets concepts de H qui sont placés avant x_i dans ce tableau (*i.e.* $pre_V(x_i) = \{x_1; \ldots; x_{i-1}\}$). Suivant cet ordre,

à chaque sommet concept x_i correspond un ensemble de relations, $pre_U(x_i) = \{r \in U(H),\ \forall x \in \gamma(x_i),\ x \in pre_V(x_i)\}$. Notons qu'une fois l'ordre désiré calculé (mais nous pouvons ne rien faire, et considérer celui dans lequel le graphe H a été construit), ces informations peuvent être calculées en $\Theta(m_H \times k)$ (et nécessitent le même espace mémoire).
- *Niveau-Suivant*, appliquée au sommet concept x_i (qui est à la position i dans le tableau), retourne $i+1$. De la même manière, appliquée au même sommet, *Niveau-Précédent* retourne $i-1$.
- *Chercher-Candidats* stocke dans le champ *image* du sommet concept x qui est son argument le premier sommet concept $y \in V(G)$ rencontré tel que, en posant $y = image(x)$, pour toute relation $r \in pre_U(x)$, avec $\gamma(r) = (x_1, \ldots, x_p)$, il existe $r' \in U(G)$ avec $\gamma(r') = (image(x_1), \ldots, image(x_p))$ et $\tau(r') \leq \tau(r)$. Cette fonction retourne VRAI si elle trouve un tel sommet (nous dirons que c'est un *candidat possible* pour x_i), et FAUX sinon. Autrement dit, lorsque la fonction retourne VRAI, si l'on considère le sous-graphe H' de H engendré par les relations de $pre_U(x)$, les champs *image* des sommets concepts de H' définissent une projection de H' dans G. Si x est le dernier sommet de H selon l'ordre défini par $Ordonner(H)$, alors $H' = H$ et on a trouvé une projection de H dans G.
- enfin, *Autres-Candidats* cherche à placer, à chaque nouvel appel, un nouveau candidat possible dans le champ *image* de x, répond VRAI s'il l'a trouvé, et répond FAUX sinon.

En vérifiant qu'il est suffisant de tester uniquement les relations de $pre_U(x)$ (où x est le sommet concept courant) à chaque étape de l'algorithme, on prouve que cet algorithme s'arrête (après avoir parcouru, au maximum, l'arbre qui correspond à toutes les applications de $V(H)$ dans $V(G)$) ; que toutes les solutions qu'il énumère sont des projections (adéquation) ; et qu'il les trouve toutes (complétude).

Avec une méthode naïve de recherche des candidats possibles, cet algorithme a une complexité, dans le pire des cas, en $\mathcal{O}(d_H \times d_G \times n_H^{n_G} \times (k + P(s)))$, équivalente à celle de l'algorithme brutal « générer et tester ». En pratique, toutefois, BT coupe souvent des branches de l'arbre de recherche avant de les avoir explorées jusqu'au bout.

Reprenons l'exemple de la FIG. 5.4, et supposons que les sommets de H soient ordonnés dans l'ordre t_3, t_2, t_1 (encore une fois, nous pouvons ici identifier les sommets par leur type). $pre_U(t_3)$ est composé de la seule relation unaire incidente à t_3, $pre_U(t_2)$ contient la relation unaire incidente à t_2 et la relation binaire incidente à t_2 et t_3, et $pre_U(t_1)$ contient la relation unaire incidente à t_1 et la relation ternaire incidente à t_1, t_2 et t_3. L'algorithme ainsi décrit génère l'arbre de BT illustré par la FIG. 5.7.

Remarquons que, pour couper au plus vite l'arbre de recherche, il faut que les $pre_U(x_i)$ soient grands le plus rapidement possible. C'est pourquoi il est utile de baser l'ordre des sommets de H sur la relation de voisinage de ce graphe, par exemple par un parcours, en largeur ou en profondeur, de ce graphe.

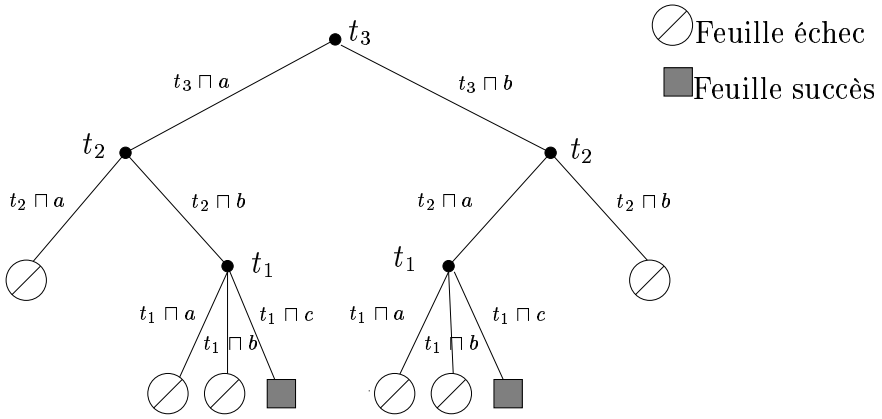


FIG. 5.7 – L'arbre de recherche associé à une exécution de `BackTrack`.

Comparaisons dans le support

Nous avons vu que le coût des comparaisons dans le support devient un coefficient multiplicatif pour la complexité de BT. Il s'agit d'un problème spécifique à PROJECTION ?, que l'on ne retrouve pas dans un CSP.

En effet, nous n'avons fait pour l'instant aucune hypothèse sur la complexité des opérations de comparaison dans $\mathcal{S}$. Nous supposerons cependant maintenant que ces comparaisons se font en temps polynomial (ou alors, la complexité de PROJECTION ? devient anecdotique devant le coût de ces opérations de comparaison). Si $t(\mathcal{S})$ borne le temps mis à comparer deux éléments $\mathcal{S}$, alors deux éléments de $\mathcal{S}^*$, de tailles respectives l_1 et l_2, pourront être comparés en $O(T(\mathcal{S}) = O(l_1 \times l_2 \times t(\mathcal{S}))$. Il s'agit d'un coefficient multiplicatif qui peut être important, et qui s'applique à *toutes* les opérations élémentaires de BT. Comment le réduire ?

Une solution classique est la compilation du support. Si T_i est un ensemble de taille n, on peut créer un tableau de taille n^2, en temps $O(n^2 \times t(\mathcal{S}))$, dans lequel on pourra lire la réponse à $t \leq_i t'$ en temps constant $O(1)$. Remarquons que la construction de ce tableau peut déjà nécessiter un espace mémoire prohibitif (par exemple, voir les 400000 types de concepts nécessaires à [Genest, 2000]). La comparaison des deux types de $\mathcal{S}^*$ se fera maintenant en $O(l_1 \times l_2)$. Il semble difficile d'étendre cette méthode à une compilation de $\mathcal{S}^*$: il peut y avoir $O(2^n)$ éléments dans $\mathcal{S}^*$.

Cependant, il n'y aura au maximum que $m_H \times m_G$ comparaisons de types distinctes au cours de la projection. Nous pourrons donc construire, pendant la projection, un tableau (si la taille de G est raisonnable) ou une table de hachage (dans le cas contraire) contenant les comparaisons déjà calculées.

La complexité de BT devient alors en $O((d_H \times k \times d_G \times n_H^{n_G}) + (m_H \times m_G \times T(\mathcal{S})))$. Le coût des comparaisons dans le support n'est alors plus un coefficient multiplicatif pour la complexité de l'algorithme. Dans le cas où $t(\mathcal{S})$ est polynomial (ce qui est toujours le cas lorsque les relations $\leq_i$ sont *données*), ce surcoût dû aux opérations de comparaison dans le support sera lui aussi polynomial. Nous pouvons le considérer comme négligeable devant

le coût (exponentiel) du BT. Sa complexité amortie peut donc être considérée comme une constante.

Nous considérerons donc, dans les algorithmes qui suivent, que la comparaison de deux types sur $\mathcal{S}^*$ se calcule en $O(1)$.

Il s'agit ici d'une justification algorithmique de la conjonction de types : en effet, au lieu de vérifier si chacune parmi p relations jumelles est supportée dans H, nous rejetons ce coût dans les comparaisons du support, et supprimons ainsi le coefficient multiplicatif p. Il est donc utile, au niveau algorithmique, de considérer les graphes élémentaires sous leur forme minimale.

Optimiser la recherche des candidats possibles

Nous essayons ici, non pas de réduire la taille de l'arbre de recherche, mais d'optimiser le temps mis à calculer les candidats possibles à chaque noeud de cet arbre. Comment peut-on calculer, à une étape de l'exécution de BT, l'ensemble des candidats possibles de x ?

La démarche naïve consiste, lorsque l'on veut calculer les candidats possibles pour un sommet concept x, à parcourir les sommets de G, puis à vérifier, pour chaque relation $r = (x_1, \ldots, x_p) \in pre_U(x)$ (nous pouvons, puisque les graphes élémentaires sont minimaux, identifier une relation par son voisinage), s'il existe $r' = (image(x_1), \ldots, image(x_p)) \in U(G)$ telle que $\tau(r') \leq \tau(r)$. On dit alors que r' est le *support*[3] de r pour la projection courante, ou que r est *supportée* par r'.

Cette démarche a un avantage, puisque les candidats possibles sont énumérés un par un : si on ne cherche qu'une seule projection, elle peut être trouvée sans avoir énuméré tous ces candidats possibles. Cependant, vérifier si un sommet de G est un candidat se fait en $O(d \times k)$. Il y a au maximum n_G sommets à tester, ce qui nous donne la complexité de BT que nous avons exposée.

La méthode que nous présentons maintenant généralise aux relations n-aires la structure de données que nous avions proposée dans [Baget and Tognetti, 2001] pour calculer les candidats.

Considérons maintenant $pre_U(x) = \{r_0, r_1, \ldots, r_q\}$, r_0 la relation unaire incidente à x, et supposons $pre_U(x) \setminus \{r_0\} \neq \emptyset$.

Alors nous notons $\Delta_1(x)$ l'ensemble construit de la façon suivante :

- prenons d'abord l'ensemble de tous les y tels que, si $image(x) = y$, alors r_1 est supportée dans G ;
- enlevons de cet ensemble tous les y tels que r_0 n'est pas supportée.

Cet ensemble $\Delta_1(x)$ peut être calculé en $O(d \times k)$. En effet, soit z un argument de r_1 différent de x (nous supposons que z existe, sinon r_1 devrait être considéré de la même façon qu'une relation unaire). Alors, les y doivent être recherchés dans les relations incidentes

[3]Le support d'une relation, terme importé de la communauté contraintes, ne devra pas être confondu avec le support $\mathcal{S}$ qui encode l'ontologie

à *image*(z), et il y en a au plus $O(d)$. Chacune de ces relations sera testée en $O(k)$ (son arité). Notons que, cette fois-ci, nous n'avons pas nécessairement besoin de parcourir l'ensemble des sommets concepts de G. Enfin, nous construisons itérativement les ensembles $\Delta_2(x), \ldots, \Delta_q(x)$ définis par :

$$\Delta_{i+1}(x) = \{y \in \Delta_i(x) \text{ tels que } y = \textit{image}(x) \Rightarrow r_{i+1} \text{ est supportée dans } G\}$$

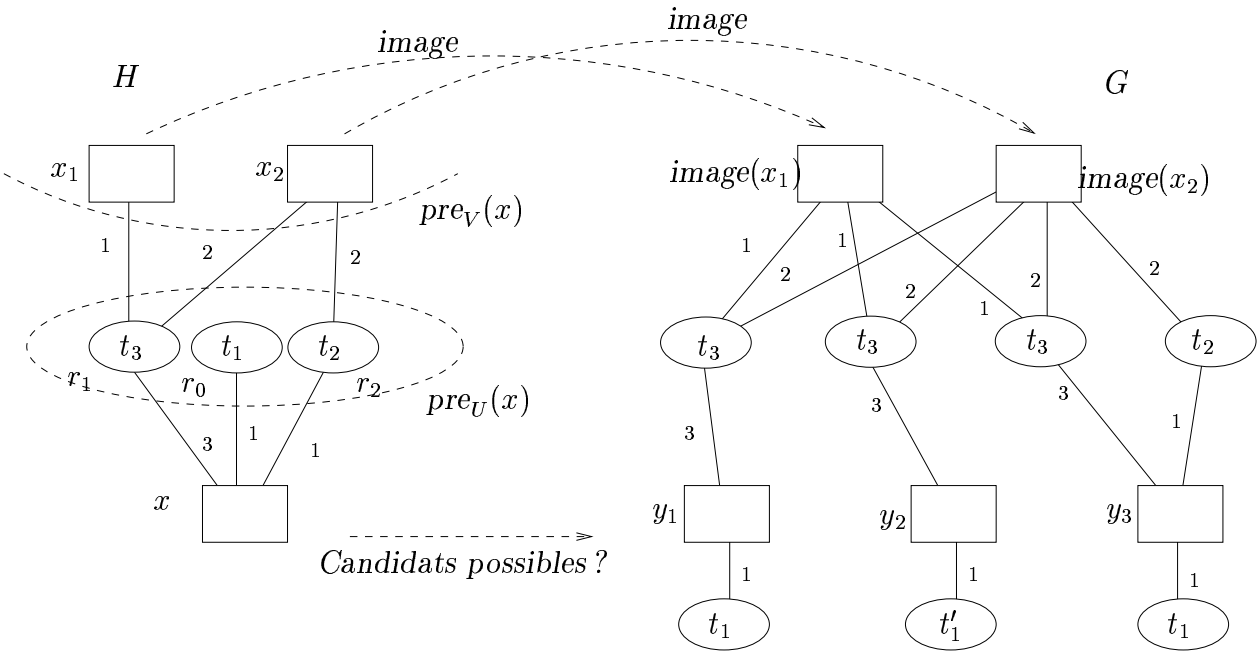


FIG. 5.8 – Recherche des candidats possibles à chaque étape de l'algorithme.

Considérons l'exemple de la FIG. 5.8. Nous cherchons à calculer, à une étape donnée de l'exécution de BT, les candidats possibles du sommet concept x. Nous avons $pre_U(x) = \{r_0, r_1, r_2\}$, où r_0 est la relation unaire, r_1 est la relation ternaire, et r_2 la relation binaire. Alors, pour calculer $\Delta_1(x)$, nous considérons tout d'abord l'ensemble des $y \in V(G)$ tels que, si $y = \textit{image}(x)$, r_1 est supportée par G, c'est à dire $\{y_1, y_2, y_3\}$, et ôtons de cet ensemble les sommets concepts pour lesquels r_0 n'est pas supportée : il nous reste $\Delta_1(x) = \{y_1, y_3\}$. Nous affinons maintenant cet ensemble par r_2, en ôtant de $\Delta_1(x)$ tous les sommets concepts pour lesquels r_2 n'est pas supportée, et obtenons $\Delta_2(x) = \{y_3\}$.

Propriété 5.6 *L'ensemble $\Delta_q(x)$ ainsi obtenu est l'ensemble de tous les candidats possibles pour x.*

Preuve: En effet, si y est un candidat possible pour x, alors voir que $y \in \Delta_1(x)$ et qu'il ne peut être supprimé par les affinages successifs. Réciproquement, si $y \in \Delta_q(x)$, alors toutes les relations de $pre_U(x)$ sont supportées dans G. □

Estimons maintenant le coût du calcul de $\Delta_{i+1}(x)$ à partir de $\Delta_i(x)$, pour une relation $r_{i+1} = (x_1, \ldots, x_p) \in pre_U(x)$. Il nous faut vérifier, pour chaque $y = image(x) \in \Delta_i(x)$, si $(image(x_1), \ldots, image(x_p))$ est un support de r_{i+1}. Si nous disposions d'une représentation matricielle du graphe G, ce test pourrait s'effectuer en $O(1)$. Cependant, nous nous interdisons cette facilité, qui augmenterait le coût des modifications du graphe G induites par l'application de règles (voir Chap. 6). Nous devrons donc parcourir les relations incidentes à l'image d'une extrémité de r_{i+1}, et vérifier si l'une d'entre elles est un support de r_{i+1}. La génération de $\Delta_{i+1}(x)$ se fera donc en temps $O(d \times k \times |\Delta_i(x)|)$. Le temps total pour générer tous ces Δ sera donc en $O(d \times k \times (1 + \Sigma_{2 \leq j \leq q} |\Delta_j(x)|))$. Le pire des cas ($d = n_G$, et aucun affinage ne réduit la taille des Δ) nous donne une complexité en $O(n_G^3 \times k)$. Si nous comparons ce résultat au pire des cas de l'algorithme naïf ($d = n_G$), dont la complexité devient $O(n_G^2 \times k)$, nous voyons que nous avons *augmenté* la complexité d'un ordre de grandeur.

Cependant, une analyse en moyenne montre que cet algorithme est plus efficace. Considérons un générateur aléatoire de graphes élémentaires, construit, par l'intermédiaire de la transformation *C2P*, à partir d'un générateur aléatoire de réseaux de contraintes (voir, par exemple [Smith, 1996]). Alors si la *dureté* [4] (tightness) des contraintes est $\rho \in]0; 1[$, la taille estimée de $\Delta_{i+1}(x)$ sera $(1 - \rho) \times \Delta_i(x)$ (l'algorithme que nous décrivons ici a le même comportement que celui que nous avons présenté dans [Baget and Tognetti, 2001]). La complexité du calcul des candidats sera alors de $O(d \times k \times |\Delta_1(x)| \times \Sigma_{1 \leq j < q} (1 - \rho)^j)$, ce qui peut se simplifier en $O(d \times k \times (1 + 1/\rho))$, si on considère que q est assez grand (une hypothèse du pire des cas). Ceci justifie l'idée intuitive que si les relations de H ont peu de certificats possibles dans G, alors la taille des Δ diminuera rapidement, au fur et à mesure des affinages successifs.

BackMark

[Gaschnig, 1979] propose un algorithme de CSP appelé `BackMark`. Cet algorithme tente de minimiser le nombre de vérifications redondantes de contraintes. Traduit en termes de projection, il faut essayer de ne pas retester inutilement si une relation incidente au sommet courant admet un certificat dans G. Nous allons utiliser la structure de données (les Δ_i) vue ci-dessus pour proposer une autre version de `BackMark`, qui présente deux avantages par rapport à la version originale : elle fonctionne avec les relations n-aires, et elle est compatible avec les sauts avant et arrière que nous utiliserons par la suite.

Soit x un sommet concept de H, et $pre_U(x) = \{r_0, \ldots, r_q\}$. Pour chacune des relations $r_1, \ldots, r_q$, nous notons $\max(x, r_i)$ l'argument de r_i (différent de x) qui sera examiné le dernier par BT (*i.e.* celui qui est le plus grand pour l'ordre induit par la position des sommets concepts dans le tableau). Alors l'ordre dans lequel nous considérons les r_i doit vérifier la propriété : si $\max(x, r_i)$ est avant $\max(x, r_j)$, alors r_i est avant r_j.

Nous avons illustré cet ordre dans la FIG. 5.9. Les sommets de $pre_V(x)$ sont considérés dans l'ordre A, B, C, D, E, F, G. Ceci induit plusieurs ordres possibles sur $pre_U(x)$, par

[4] La dureté d'une contrainte r d'arité k est définie par $1 - |\delta(r)|/|\mathcal{D}|^k$, *i.e.* le complément du nombre de tuples autorisés par la contrainte sur le nombre de tuples possibles.

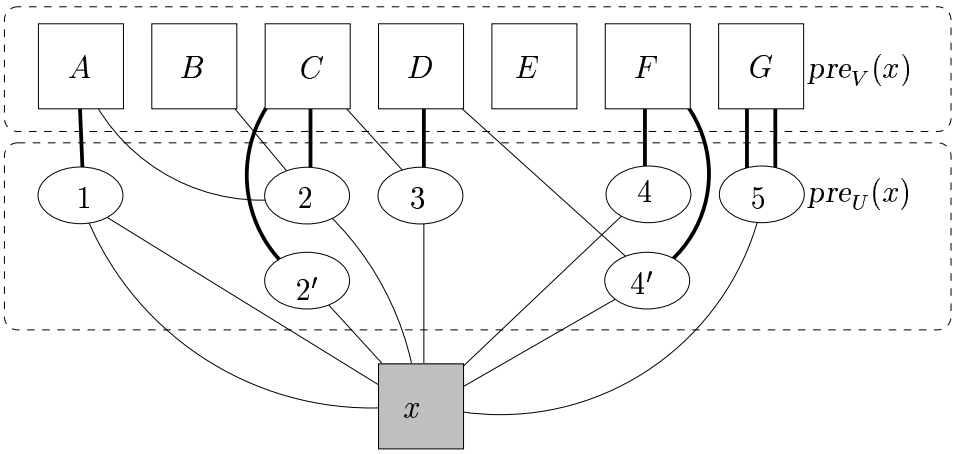


FIG. 5.9 – Calcul de l'ordre sur les relations de $pre_U(x)$.

exemple $1, 2, 2', 3, 4', 4, 5$. Cet ordre sur les $pre_U(x)$ sera celui avec lequel nous construirons les Δ_i.

Supposons maintenant qu'à une étape de BT, nous ayons construit les différents $\Delta_i(x)$, puis qu'un échec nous force à remonter dans l'arbre de recherche.

Propriété 5.7 *Soient $r_1, \ldots, r_k$ les relations non unaires de $pre_U(x)$, ordonnées comme indiqué ci-dessus. Alors si $image(max(x, r_1)), \ldots, image(max(x, r_p))$ n'ont pas changé depuis la dernière recherche de candidats possibles pour x, alors $\Delta_1(x), \ldots, \Delta_p(x)$ restent identiques. La recherche des candidats possibles ne nécessite donc que la construction de $\Delta_{p+1}(x), \ldots, \Delta_k(x)$.*

Ceci nous donne donc un algorithme pour calculer les candidats : parcourir $pre_U(x)$, de r_1 à r_k, jusqu'à ce qu'on trouve un r_p tel que $image(max(x, r_p))$ a été modifié depuis le dernier calcul de candidats pour x, puis commencer le calcul des candidats à partir de Δ_{p+1}. Nous avons donné cette technique, équivalente à `BackMark`, dans [Baget and Tognetti, 2001]. Nous la généralisons ici au cas n-aire (il s'agit bien d'une généralisation, si nous n'avons que des relations binaires, l'ordre des relations est celui des sommets concepts qui sont l'« autre extrémité »). Notons déjà que cette technique est compatible avec les méthodes de saut arrière (`BackJump`), mais que, pour l'utiliser en conjonction avec les techniques de filtrage avant, il nous faudra effectuer une légère modification.

Enfin, nous remarquons que `BackMark` ne crée aucun surcoût quant à la génération de candidats, et ne fait que diminuer le coût amorti de cette génération. En effet, plus la recherche de la projection sera difficile, plus il y aura de « petits » backtracks, et plus `BackMark` réussira à factoriser les calculs de recherche des candidats possibles. Il ne fait cependant rien, contrairement aux algorithmes que nous allons maintenant présenter, pour diminuer la taille de l'arbre de recherche.

5.2.2 Forward Checkings

L'idée de `Forward Checking` (ou FC) est de regarder si les candidats que l'on choisit pour un sommet sont tels que leurs voisins pourront eux-mêmes être projetés. Dans le cas des réseaux de contraintes binaires, cette idée était déjà dans [Golomb and Baumert, 1965], mais a été formalisée et développée par [Haralick and Elliott, 1980]. Nous présenterons tout d'abord cette technique dans le cas binaire, puis nous inspirerons de [Bessière et al., 1999] pour la généraliser au cas n-aire. Enfin, nous montrerons la modification que l'on doit apporter à `BackMark`, dans le cas n-aire, pour qu'il reste compatible avec `Forward Checking`.

Forward Checking : le cas binaire

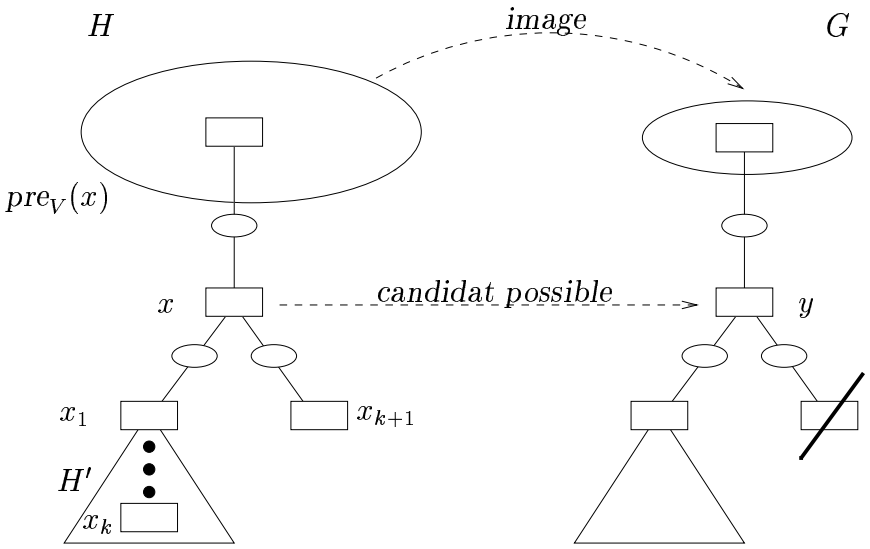


FIG. 5.10 – Nécessité de regarder plus loin que le sommet courant.

Considérons l'exemple de la FIG. 5.10. Après avoir projeté le sous-graphe de H engendré par $pre_V(x)$ dans G, nous tentons maintenant d'étendre cette projection à une projection de $pre_V(x) \cup \{x\}$. Nous avons trouvé y comme candidat possible pour x. Supposons maintenant que nous examinions les sommets $x_1, \ldots, x_k$ pour tenter d'étendre la projection à $pre_V(x) \cup \{x, x_1, \ldots, x_k\}$, ce qui peut être exponentiellement long. Il se peut que nous nous apercevions lors de l'examen de x_{k+1} que tout ce travail était inutile : on ne peut pas étendre la projection initiale à une projection de $pre_V(x) \cup \{x, x_{k+1}\}$.

Afin d'éviter ce problème, FC propose de *propager* les choix effectués dans le voisinage de $pre_V(x) \cup \{x\}$. Dans le cas binaire, la propriété que nous souhaitons voir vérifiée à chaque étape de l'algorithme (où le sommet concept courant est x, et ses candidats possibles sont $y_1, \ldots, y_k$) s'exprime par :

(FC) « Pour chaque projection de x dans y_i, pour chaque sommet concept z voisin d'un sommet de $pre_V(x) \cup \{x\}$, la projection courante du sous graphe de H engendré par les

sommets concepts $\text{pre}_V(x) \cup \{x\}$ *peut s'étendre à une projection du sous-graphe de* H *engendré par les sommets concepts* $\text{pre}_V(x) \cup \{x, z\}$ *».*

Un algorithme utilisant cette propriété s'implémente facilement dans le cas binaire, sans surcoût, en utilisant la structure de données que nous avons définie pour `BackMark`.

Supposons que le sommet concept courant x du BT a au moins un voisin dans $pre_V(x)$. Puisque les sommets du graphe H sont ordonnés par un parcours de ce graphe, ce sera toujours le cas sauf si x est le premier sommet d'une composante connexe du graphe H. Nous notons $pre_U(x) = \{r_1, \ldots, r_k\}$ les relations *binaires* entre x et $pre_V(x)$, ordonnées comme nous l'avons indiqué.

Supposons maintenant que $\Delta_1(x), \ldots, \Delta_k(x)$ sont correctement construits. Alors nous écrivons les fonctions *Chercher-Candidats* et *Autres-Candidats* de la façon suivante :

- *Chercher-Candidats* pose un marqueur au début (juste avant le premier élément) de $\Delta_k(x)$, puis appelle *Autres-Candidats* (qui fera effectivement le parcours des candidats possibles) ;
- *Autres-Candidats* avance le marqueur, répond FAUX si toute la liste a été parcourue, et sinon, fait un appel à la fonction *FC-Propage*. Si cette fonction répond VRAI (*i.e.* la propriété *(FC)* est respectée), alors *Autres-Candidats* retourne VRAI. Sinon, *Autres-Candidats* s'appelle lui-même, pour examiner les candidats suivants.
- *FC-Propage* considère l'ensemble $post_U(x)$ des relations entre x et les sommets qui ne sont pas encore projetés (ceux qui ne sont pas dans $pre_V(x)$). Pour chacune de ces relations r (dont z est l'autre extrémité), x possède un pointeur sur le $\Delta_i(z)$ correspondant à la relation r. Nous le noterons $\Delta'(x, r)$. Toutes ces informations doivent être construites et sauvegardées à l'initialisation (*Ordonner*) du graphe. Alors, pour chacune des relations $r \in post_U(x)$, *FC-Propage* construit la liste $\Delta'(x, r) = \Delta_i(z)$, comme indiqué pour `BackMark`. Si un des $\Delta'(x, r)$ devient vide, *FC-Propage* s'arrête et retourne FAUX, sinon, il retourne VRAI après avoir construit tous les $\Delta(x, r)$.

Propriété 5.8 *Dans le cas où toutes les relations sont binaires, l'algorithme* `BackTrack`*, utilisant les fonctions auxiliaires définies ci-dessus, vérifie les conditions suivantes :*

- *si* x *n'est pas le premier sommet concept d'une composante connexe, alors l'ensemble des* $\Delta_i(x)$ *aura été construit* avant *l'appel à Chercher-Candidats*(x).
- *l'ordre de construction des* $\Delta_i(x)$ *est respecté, ce qui nous permet à chaque fois de construire* $\Delta_i(x)$ *à partir de* $\Delta_{i-1}(x)$.

L'algorithme est donc adéquat et complet dans le cas des relations binaires, et respecte la propriété (FC).

Remarquons aussi que cet algorithme intègre naturellement les avantages de `BackMark`.

Cependant, lorsqu'on considère le cas des relations n-aires, ni la structure de données considérée ni la propriété *(FC)* ne sont satisfaisantes.

Forward Checkings n-aires

Une façon immédiate de généraliser `Forward Checking` au cas n-aire est de considérer les graphes conceptuels très simples associés aux graphes H et G (les relations sont alors

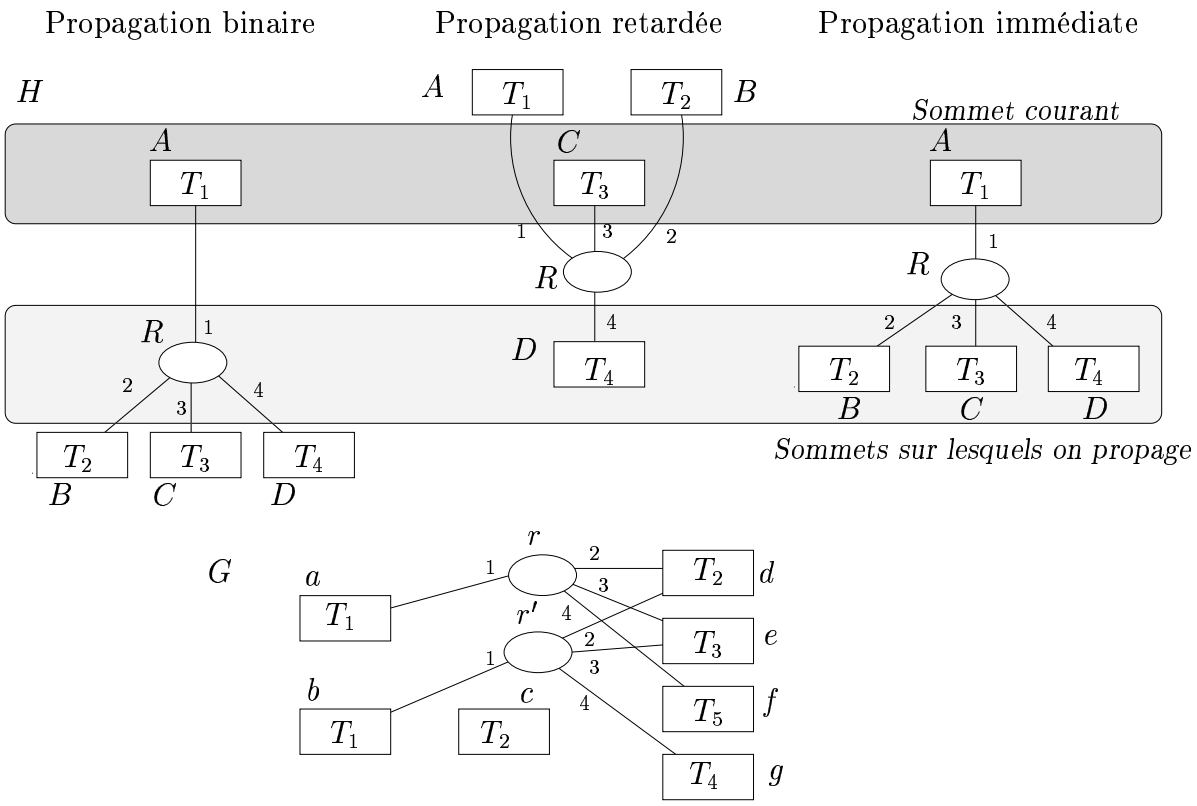


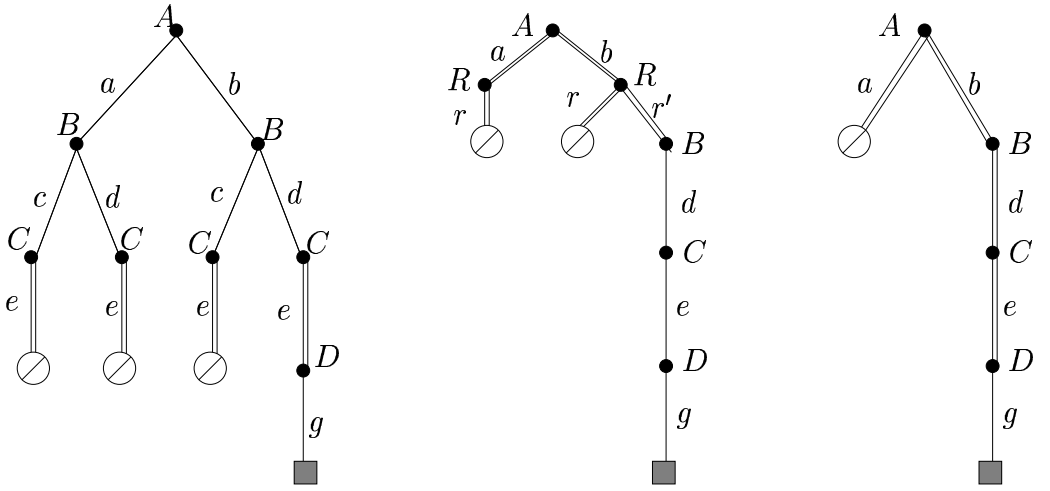
FIG. 5.11 – Quand propager les conséquences de la projection des arguments ?

des sommets, voir Sect. 4.1.2). Cette méthode est identique à celle consistant à utiliser le « *hidden graph* » d'un réseau de contraintes n-aire. Le `Forward Checking` binaire que nous venons d'exposer peut alors être utilisé. Cet algorithme, que nous appellerons **FC**, a cependant une faiblesse : ne progressant que de sommet concept en sommet relation, puis de sommet relation en sommet concept, il ne regarde en fait qu'un demi-coup en avant, et ne peut être considéré comme une vraie généralisation de `Forward Checking` aux relations n-aires. Il s'agit de l'exemple (*propagation binaire*) de la FIG. 5.11. Lorsque l'on essaie d'associer a à A, la propagation ne nous indique aucune erreur puisque le sommet relation R pourra être projeté sur r. Cet algorithme ne regarde pas plus loin, ce qui aurait indiqué que D ne pourra pas se projeter.

Une première idée de généralisation, en considérant toujours le graphe binaire, a été d'augmenter la profondeur de propagation de FC, afin qu'il propage, a une distance 2, de sommet concept en sommet concept. Cet algorithme est appelé **FC+**. Mais si nous voulons vraiment considérer les relations comme des hyperarcs, et généraliser `Forward Checking` aux hypergraphes, plusieurs choix s'offrent à nous.

Tout d'abord, quand doit-on vérifier qu'une relation sera supportée dans G ? Comme illustré dans la FIG. 5.11 (où tous les types de relations unaires distincts sont incompatibles), nous pouvons considérer les cas suivants :

- Nous pouvons vérifier qu'une relation est supportée dès que tous ses arguments sauf un ont été projetés. C'est ce que nous appelons *propagation retardée*. Dans ce cas, on ne cherche si la relation R est supportée que lorsque l'on examine C comme sommet concept courant (A et B ont déjà été projetés). Supposons que A ait été projeté dans a, et B dans c. Nous examinons maintenant la possibilité de projeter le sommet concept courant C dans e, et propageons ce choix. Quelle que soit l'image que l'on associe au sommet concept D, la relation ne sera pas supportée. Nous devons donc prendre un autre choix pour C. Le problème est que nous aurions pu, dès la projection de B, nous apercevoir que cette projection partielle ne pouvait pas s'étendre.
- Nous pouvons vérifier qu'une relation pourra être supportée dès qu'un de ses arguments est projeté, ce que nous appelons *propagation immédiate*. Ici, dès que nous essayons d'associer a au sommet concept A, nous nous apercevons que la relation R incidente à A n'aura aucun support incident à a. La propagation immédiate coupe donc plus vite des branches de l'arbre de recherche, au prix d'un coût accru lors de la génération d'un noeud de cet arbre.



Propagation retardée Propagation binaire Propagation immédiate

FIG. 5.12 – Arbres de recherche associés à ces différentes propagations.

Dans la FIG. 5.12, nous avons représenté les arbres de recherche associés à ces différents choix de propagation. Les arêtes représentées par un double trait sont celles où une propagation est effectuée. La propagation immédiate coupe toujours plus rapidement l'arbre de recherche, mais il existe des cas où la propagation retardée est plus efficace que la propagation binaire (dans le cas, par exemple, ou les relations sont binaires). Cependant, couper plus rapidement l'arbre de recherche ne voudra pas dire que l'algorithme s'exécutera plus rapidement : le coût des propagations peut éventuellement être prohibitif.

Et ce n'est pas l'unique choix que nous ayons à faire. Si, dans le cas binaire, il est suffisant de propager par les relations incidentes au sommet concept courant, ce n'est plus le cas avec les relations n-aires, comme le montre la FIG. 5.13. En effet, supposons que

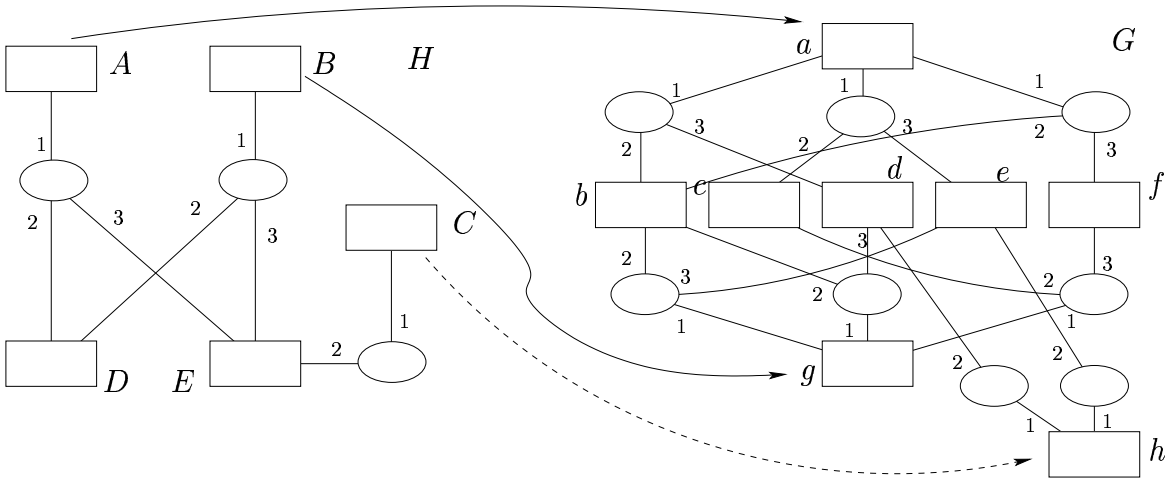


FIG. 5.13 – Propagation par des relations non incidentes au sommet concept courant.

nous ayons projeté le sommet concept A du graphe H dans le sommet concept a du graphe G, et le sommet B dans g. Après la propagation immédiate à partir de A, nous pouvons en déduire que les sommets (D, E) peuvent se projeter dans $(b, d), (c, e)$ ou (b, f). Les images possibles de D sont donc $\{b, c\}$, et celles de E sont $\{d, e, f\}$. Après la propagation (immédiate) à partir de B, les sommets (D, E) peuvent se projeter dans (b, e), (b, d) ou (c, f), ce qui ne restreint pas les images possibles de D et E. Le sommet concept courant est maintenant C. Tentons de le projeter dans h, et examinons les conséquences de ce choix : il faut supprimer f des images possibles de E. Réexaminons maintenant les supports de la relation incidente à A. Les sommets (D, E) peuvent encore se projeter dans (b, d) et (c, e), et nous n'avons rien de plus à enlever aux images possibles de D et E. Passons maintenant à la relation incidente à B. Cette fois-ci, les seules images possibles de (D, E) sont (b, e) et (b, d), ce qui restreint les images possibles de D à $\{b\}$ et celles de E à $\{d, e\}$. Réexaminons (de nouveau) les supports de la relation incidente à A, et les images possibles de D et E ne sont plus que $\{b\}$ et $\{d\}$.

En examinant une relation non incidente au sommet concept courant (le sommet C), nous avons pu filtrer des images possibles (qui n'avaient pas été filtrées auparavant). De plus, plusieurs examens de cette relation ont été nécessaires. Nous pouvons développer les deux enseignements de cet exemple :

1. contrairement au cas binaire, il est possible de filtrer davantage dans $post_V(x)$ en considérant non seulement toutes les relations qui sont incidentes à x et qui ont un argument dans $post_V(x)$, mais aussi toutes les relations qui ont un argument dans $pre_V(x)$ et un autre dans $post_V(x)$. Si nous ne considérons que les relations incidentes à x (ce qui filtre moins), nous dirons que la propagation est *locale*. Si nous considérons toutes les relations entre $pre_V(x)$ et $post_V(x)$, nous dirons que la propagation est *globale*.

2. de plus, nous avons vu que le filtrage des candidats possibles par l'examen d'une

relation peut rendre utile le réexamen d'une autre. De la même manière, ceci n'est pas nécessaire à l'adéquation et à la complétude de l'algorithme, mais peut éventuellement couper plus rapidement l'arbre de recherche. Nous dirons que la propagation se fait *en une passe* si chaque relation n'est examinée qu'une fois, *en plusieurs passes* sinon.

Nous avons donc différents mécanismes de propagation possibles, certains réussissant à couper plus rapidement l'arbre de recherche, mais toujours au prix d'un travail accru lors de l'examen du sommet concept courant. Ces différents mécanismes sont étudiés et expérimentés dans [Bessière et al., 1999].

- **FC** considère le graphe conceptuel très simple (biparti d'incidence) induit par l'hypergraphe (*hidden CSP*, suivant la terminologie contraintes) ;
- **FC+** considère aussi le biparti d'incidence, et utilise un FC binaire modifié, qui propage à une distance de 2 ;
- **nFC0** utilise la propagation retardée, locale ;
- **nFC1** utilise un type de propagation particulier ; équivalent à celui de **FC+** ;
- **nFC2** utilise la propagation immédiate, locale, en une passe ;
- **nFC3** utilise la propagation immédiate, locale, en plusieurs passes ;
- **nFC4** utilise la propagation immédiate, globale, en une passe ;
- **nFC5** utilise la propagation immédiate, globale, en plusieurs passes.

Un algorithme de propagation **FC'** est dit *plus fort* que **FC"** si, en considérant le même ordre sur les sommets concepts, à chaque étape de l'algorithme (où x est le sommet concept courant), pour chaque sommet $y \in post_V(x)$, les candidats possibles pour y obtenus par **FC'** sont inclus dans ceux obtenus par **FC"**. Les résultats de [Bessière et al., 1999], qui comparent les forces respectives de ces différents algorithmes, sont illustrés dans la FIG. 5.14.

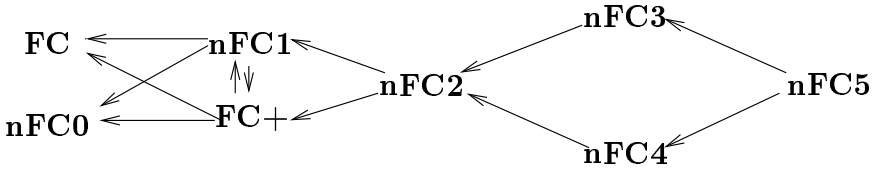


FIG. 5.14 – Forces respectives des différents `Forward Checking` n-aires.

Cependant, seule l'expérimentation peut montrer si l'espoir d'avoir un filtrage plus important justifie le coût engendré par la sophistication des méthodes de propagation. Tout d'abord, il est bien connu [?] que **FC** est bien moins efficace que **FC+**. Ensuite, les expérimentations de [Bessière et al., 1999] concernant tous ces algorithmes, excepté **FC**, montrent que :

- **FC+** est toujours le plus mauvais choix que l'on puisse faire[5] (ce qui est une autre justification, outre le nombre de projections, de l'intérêt de considérer les relations

[5]Des expérimentations récentes [Mamoulis and Stergiou, 2001] montrent cependant qu'il se comporte bien lorsque la dureté des contraintes est *très* importante.

comme des hyperarcs plutôt que comme des sommets), son efficacité est souvent inférieure de plusieurs ordres de grandeur à celle des autres algorithmes ;
- **nFC0**, le plus faible en terme de filtrage, est le meilleur de ces algorithmes quand la dureté des contraintes est faible (en terme de projection, il y a énormément de choix pour les supports possibles), en effet, dans ce cas, multiplier les tests ne réduit pas fortement les images possibles, pour un surcoût important ;
- **nFC5**, celui qui filtre le plus, est le plus efficace quand la dureté des contraintes est importante (il y a peu de choix possible pour les supports) ;
- **nFC2** est un bon choix de compromis, il n'est le plus efficace dans aucun cas, mais son efficacité est toujours satisfaisante. C'est celui dont nous proposons ici une implémentation.

Le `Forward Checking` choisi : nFC2

Nous choisissons d'adapter **nFC2** au problème PROJECTION. En effet, bien que nous ayons déjà fait quelques suppositions sur la structure d'un graphe requête H défini par un utilisateur, le paramètre de *dureté* sur lequel repose le choix d'un des `Forward Checking` n-aires est essentiellement du ressort de la structure de la base de faits G, sur laquelle nous ne souhaitons pas faire de suppositions. Aussi, le choix du « candidat de compromis » **nFC2** semble approprié. De plus, cet algorithme est simple à implémenter, ne nécessitant qu'une légère modification de la structure de données (les Δ) que nous avons déjà présentée, et ne génère pas un surcoût important.

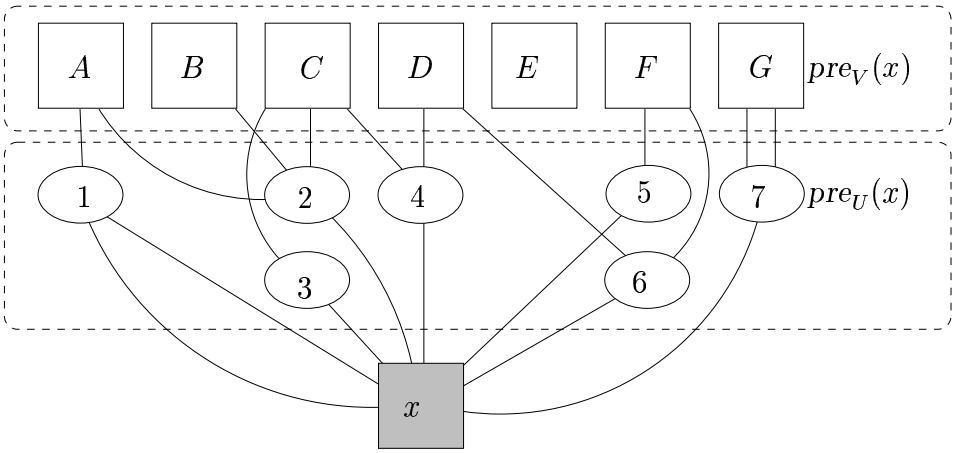


FIG. 5.15 – Exemple utilisé pour la construction des $\Delta(x)$.

Commençons tout d'abord par modifier quelque peu les champs $\Delta(x)$ de chaque sommet du graphe H. En effet, cette fois-ci, ces champs ne seront pas modifiés une fois par relation appartenant à $pre_U(x)$ (quand tous les autres arguments ont été projetés), mais à chaque fois qu'un argument d'une de ces relations sera projeté. Il faudra donc construire ces champs en considérant, pour chaque sommet $y \in pre_V(x)$, pour chaque relation $r \in pre_U(x)$

incidente à y, un champ $\Delta(y, r)$. En considérant un ordre total sur les sommets concepts, qui est celui donné par leur position dans le tableau, et un ordre total (arbitraire) sur les relations, l'ordre lexicographique résultant de ces deux ordres nous permet d'ordonner les différents $\Delta(y, r)$. Ainsi, d'après l'exemple de la FIG. 5.15, le sommet x contient, dans cet ordre, les champs $\Delta(A, 1)$, $\Delta(A, 2)$, $\Delta(B, 2)$, $\Delta(C, 2)$, $\Delta(C, 3)$, $\Delta(C, 4)$, $\Delta(D, 4)$, $\Delta(D, 6)$, $\Delta(F, 5)$, $\Delta(F, 6)$ et $\Delta(G, 7)$.

Ainsi, lorsque le sommet concept courant est un de ces y, le choix du candidat que l'on examine est propagé, pour toutes les relations r entre y et $x \in pre_V(y)$, directement dans le champ $\Delta(y, r)$ du sommet x, et sera calculé à partir du champ immédiatement précédent, déjà construit. Toujours sur le même exemple, si le candidat courant est D, le choix d'un candidat pour D sera propagé dans $\Delta(D, 6)$ du sommet concept F, dans $\Delta(D, 4)$ du sommet x, et dans $\Delta(D, 6)$ du sommet x (en utilisant le même ordre sur les relations qu'à la construction).

Algorithme 3: *Chercher-Candidats*(x, G)

Données : Le sommet concept courant x, et le graphe G dans lequel on doit le projeter.

Résultat : VRAI s'il existe un candidat possible y pour x tel que la propagation de ce choix par **nFC2**, qui met à jour tous les Δ, n'en vide aucun (les voisins de x dans $post_V(x)$ auront un ensemble non vide de candidats possibles), et FAUX sinon.

si *(*$pre_V(x) = \emptyset$*)* **alors**
  $\Delta(x) \leftarrow$ Demander-Candidats(x, G);

candidat-courant$(x) \leftarrow$ Début-Liste(Dernier-$\Delta(x)$);

retourner *Autres-Candidats*(x, G);

L'algorithme *Chercher-Candidats* (Alg. 3) s'écrit directement, et fait appel à *Autres-Candidats* pour la propagation. Notons que si x est le premier sommet concept d'une composante connexe, alors ses candidats possibles n'auront pas été construits. Nous devrons donc chercher ces candidats possibles dans le graphe G (*Demander-Candidats*). Ceci peut se faire par un parcours de tous les sommets concepts du graphe G, en vérifiant si leur type est compatible avec celui de x, ou peut utiliser un mécanisme d'indexation des sommets concepts du graphe G, afin de ne pas avoir à parcourir l'ensemble de ce graphe. Un mécanisme d'indexation, reposant sur les graphes étoiles, est proposé dans [Guinaldo, 1996].

L'algorithme *Autres-Candidats*, quant à lui, est présenté dans l'Alg. 4. Il repose sur une propagation (nous avons choisi **nFC2**), dont l'algorithme est donné dans l'Alg. 5.

Algorithme 4: *Autres-Candidats*(x, G)

Données : Le sommet concept courant x, et le graphe G dans lequel on doit le projeter.

Résultat : VRAI s'il existe un autre candidat possible y pour x tel que la propagation de ce choix par **nFC2**, qui met à jour tous les Δ, n'en vide aucun (les voisins de x dans $\text{post}_V(x)$ auront un ensemble non vide de candidats possibles), et FAUX sinon.

```
candidat-courant(x) ← Suivant-Liste(Dernier-Δ(x));
si (candidat-courant(x) = FIN-DE-LISTE) alors
    retourner FAUX;
sinon
    si (nFC2-Propage(x, candidat-courant(x))) alors
        retourner VRAI;
    sinon
        retourner Autres-Candidats(x, G);
```

Algorithme 5: *nFC2-Propage*(x, y)

Données : Le sommet concept courant x de H, et un de ses candidats possibles y.

Résultat : Propage le choix de y pour x, suivant l'algorithme nFC2, et répond VRAI si et seulement si aucun domaine de $post_V(x)$ n'a été vidé.

```
// Les deux premières boucles « pour » ne sont qu'une vue de l'esprit. En effet, la
// fonction Ordonner a construit toutes les paires (Δ', Δ) nécessaires, dans le bon
// ordre, au cours de la phase d'initialisation.
pour (r ∈ post_U(x)) faire
    // Les relations incidentes à x ayant au moins un argument dans post_V(x)
    pour (z ∈ args(r) ∩ post_V(x)) faire
        Δ ← Trouver-Delta(z, x, r);
        // Le Δ de z qui doit être mis à jour par la propagation le long de r du choix
        // d'un candidat possible pour x
        Δ' ← Précédent-Liste(Δ);
        Vider(Δ);
        pour (v ∈ Δ') faire
            si (Supportée(r, (x, y), (z, v))) alors
                // Cet appel teste si il existe une relation r' dans G, de type plus
                // spécifique que celui de r, telle que, si on considère la projection
                // partielle π qui associe aux sommets de pre_V(x) leur image
                // courante, qui associe y à x et v à z, alors si γ_i(r) est instancié
                // par π, alors γ_i(r') = π(γ_i(r)). Remarquons encore une fois qu'il
                // suffit de parcourir les relations incidentes à y pour chercher ces
                // certificats possibles r'.
                Δ ← Δ ∪ {v};
        si (Vide ?(Δ)) alors
            retourner FAUX;
retourner VRAI;
```

5.2.3 Autres améliorations de BackTrack

Nous évoquons ici d'autres améliorations de `BackTrack`, et en particulier des mécanismes de « saut arrière » (*BackJump*). L'un de ces mécanismes, CDBJ (*Conflict Directed BackJumping*) [Prosser, 1993] est considéré comme particulièrement efficace en collaboration avec `Forward Checking`.

Nous expliquerons pourquoi nous ne jugeons pas nécessaire, dans les modèles que nous étudions dans ce mémoire, d'utiliser des mécanismes de filtrage plus efficaces comme MAC [Bessière and Régin, 1996].

BackJumps

Alors que les algorithmes de type `Forward Checking` (comme les techniques de filtrage que nous évoquerons plus loin) cherchent à couper plus rapidement l'arbre de recherche en testant la possibilité de projection un peu plus loin que le sommet concept courant, les algorithmes de type `BackJump` cherchent à couper cet arbre en remontant plus haut dans cet arbre après un échec. L'idée commune à ces algorithmes est que, après un échec dans l'arbre de recherche, il faut remonter à la source de cet échec : il est inutile de tenter des projections qui renouvelleront cet échec.

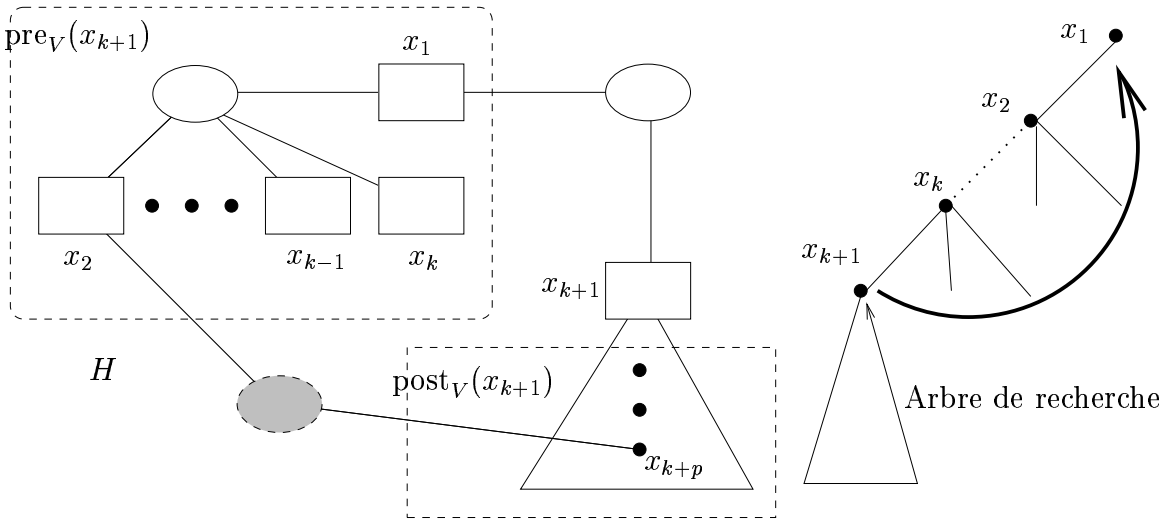


FIG. 5.16 – `BackJump` : ne pas explorer des branches qui répéteront la même erreur.

La FIG. 5.16 illustre un tel cas : H est un graphe dans lequel x_{k+1} est relié à x_1, mais n'est pas dans le voisinage de $x_2, \ldots, x_k$. Nous avons projeté les sommets concepts $x_1, \ldots, x_k$, et lorsque le sommet concept courant est x_{k+1}, l'appel à *Chercher-Candidats* ou *Autres-Candidats* retourne FAUX. Nous supposerons dans un premier temps que nous n'avons utilisé aucune technique de filtrage (en particulier, il n'y a pas eu de propagation par `Forward Checking`).

Il y a deux cas dans lesquels l'échec sur x_{k+1} ne pourra pas être corrigé en changeant les projections de $x_2, \ldots, x_k$: dans un premier cas, nous supposerons que nous avons un *échec terminal* (qui correspond à une *feuille échec* dans l'arbre de recherche), dans le second cas, nous supposerons que nous avons un *échec interne* (le noeud de l'arbre de recherche n'est pas une feuille). Dans ces deux cas, on pourra remonter directement à x_1 (suivant l'algorithme que nous avons écrit, *Niveau-Précédent*(x_{k+1}) pourra retourner l'indice de x_1).

- si l'échec sur x_{k+1} est terminal, alors aucun des choix pour les images de $x_2, \ldots, x_k$ n'est coupable de l'échec sur x_{k+1}. Un sommet concept est *coupable* d'un échec (*culprit variable*, en contraintes) si le choix d'un de ses candidats est une des causes possibles[6] d'une suppression de candidats pour d'autres sommets, menant à l'échec.
- si l'échec sur x_{k+1} est interne, alors supposons qu'il n'y a pas de relations entre $\text{pre}_V(x_{k+1})$ et $\text{post}_V(x_{k+1})$ (la relation en grisé de la FIG. 5.16 est donc considérée comme absente). De la même façon, aucun des sommets de $x_2, \ldots, x_k$ n'est coupable des échecs terminaux de $\text{post}_V(x_{k+1})$ qui sont remontés jusqu'à x_{k+1}.

Différents critères peuvent être utilisés pour exploiter cette notion de sommet concept *coupable*. Nous présenterons brièvement trois de ces critères, et les algorithmes qui en découlent : GBJ (Gaschnig's BackJumping, [Gaschnig, 1979]), GBBJ (Graph-Based BackJumping, [Dechter, 1990]), et CDBJ (Conflict-Directed BackJumping, [Prosser, 1993]), qui combine les deux précédents algorithmes.

- GBJ n'est utilisé que lorsque l'échec sur un sommet concept x est terminal. Dans ce cas, nous considérons le premier sommet concept $y \in \text{pre}_V(x)$ qui a vidé un Δ (*i.e.* il existe r tel que $\Delta(y, r)$ dans x est vide, et aucun Δ précédent de x n'est vide). Alors, si nous changeons la projection de n'importe quel sommet concept entre y et x, ce Δ sera vidé de la même façon. L'appel à *Niveau-Précédent* pourra donc, sans risque de perte de solution, retourner le niveau de y. Notons que ce n'est pas vrai si l'échec est interne : si nous considérons la présence de la relation en grisé dans la FIG. 5.16, l'échec (terminal) sur le sommet de x_{k+p} de $\text{post}_V(x_{k+1})$ peut (éventuellement) être corrigé par la modification de l'image de x_2.
- GBBJ considère aussi bien les échecs terminaux que les échecs internes. Cependant, sa notion de coupable est plus faible puisqu'il remonte au dernier (dans l'ordre sur les sommets concepts) voisin du sommet concept sur lequel on repère un échec. Ceci ne pose aucun problème lorsque cet échec est terminal (nous sommes alors dans un cas particulier de GBJ), mais il faut faire attention lorsque l'échec est interne. En effet, supposons, toujours sur l'exemple de la FIG. 5.16, qu'il y ait un échec interne sur x_{k+p}, et que son dernier voisin soit x_{k+1}. Alors il n'y a aucun problème pour remonter l'arbre de recherche jusqu'à x_{k+1}. Or il n'y a maintenant plus d'autres candidats pour x_{k+1}, et nous remontons maintenant jusqu'à x_1. Nous n'avons pas considéré le sommet concept x_2, alors que changer son image aurait pu nous donner une image possible pour x_{k+p}. Lorsque nous remonterons d'un échec terminal y à un

[6] L'identification d'un coupable ou de plusieurs coupables est un problème difficile. Nous nous servirons ici plutôt de la notion, plus précise, de non coupable : changer la projection d'un tel sommet ne suffira pas à réparer l'échec.

sommet concept x, il faudra donc indiquer à x les voisins de y qui restent à explorer, ce sera le *jumpback set* de x, qu'il considérera comme ses propres voisins s'il doit faire un retour arrière. Chaque fois qu'on réussit à projeter un sommet (l'ensemble des candidats possibles du sommet concept courant est non vide), ce *jumpback set* est réinitialisé à vide.
- CDBJ combine les avantages de GBJ et GBBJ. En effet, les coupables considérés ne sont pas tous les voisins d'un sommet concept, mais seulement ceux ayant vidé un de ses Δ, comme dans GBJ. Pour pouvoir considérer n'importe quel type d'échec (terminal ou interne), CDBJ utilise, comme GBBJ, un *jumpback set* contenant ces coupables.

nFC2 + CDBJ

Des nombreuses expérimentations réalisées dans la communauté « contraintes », il a longtemps été conclu qu'un algorithme mixte, utilisant à la fois FC et CDBJ, était le meilleur algorithme existant pour résoudre un CSP [Prosser, 1993]. Cependant, les expérimentations de [Bessière and Régin, 1996] montrent que l'algorithme MAC proposé par [Sabin and Freuder, 1994] est beaucoup plus efficace si les réseaux de contraintes (binaires) sont grands et difficiles (notion liée à la transition de phase, voir, par exemple, [Smith, 1996]). L'algorithme nFC2+CDBJ reste encore un bon choix pour des instances du problème où la recherche peut s'appuyer sur une structuration forte (du réseau de contraintes donc du graphe requête), comme ce sera le cas si on considère des requêtes écrites par l'utilisateur dans le formalisme des graphes conceptuels.

MAC est un algorithme qui conserve le schéma général de FC, mais en diffère sur trois points particuliers :
- il utilise une notion de consistance d'arc au lieu de la forme particulière de celle-ci utilisée par `Forward Checking` (en termes de projection de graphes conceptuels, le dernier Δ de chaque sommet concept x de H doit contenir toutes les images possibles y telles que, pour toute relation r avec $\gamma_i(r) = x$, il existe une relation de type plus spécifique dans G et un de ces y vérifiant $\gamma_i(r') = y$) ;
- il propage à la fois lorsque l'on choisit une image possible pour un sommet concept et lorsque l'on s'aperçoit que ce choix mène à un échec ;
- enfin, la propagation n'est pas uniquement dans le voisinage du sommet concept courant (ou de ces prédécesseurs, pour les versions globales de **nFC**), mais sur tous les sommets concepts de H, et, de plus, elle est en plusieurs passes.

Nous avons choisi de ne pas utiliser MAC, car dans le cadre qui nous intéresse, celui de la projection de graphes conceptuels, MAC n'a pas que des avantages. Tout d'abord, si G est un très grand graphe, la propagation sur des sommets concepts éloignés du sommet concept courant peut déterminer un nombre de plus en plus grand d'images possibles (proche de la taille de G), ce que nous souhaitons éviter. [Freuder and Wallace, 1991] évoque une façon de limiter ce problème, en stoppant la propagation dès que les ensembles d'images possibles dépassent une certaine taille. Cette idée n'a pour l'instant pas été expérimentée. Ensuite, dès que nous ajouterons des règles, les algorithmes que nous utiliserons reposeront sur un

très grand nombre de petites projections, souvent faciles. Il vaudra alors mieux privilégier un algorithme ayant un surcoût faible, comme nFC2 + CDBJ.

De plus, avantage non négligeable, la structure de données que nous avons proposée permet une implémentation simple et efficace [7] de nFC2 + CDBJ. Nous pensons que cet algorithme peut être un bon outil générique de recherche de projections dans les modèles de la famille $\mathcal{SG}$.

5.3 Utilisation des composantes biconnexes : BCC

L'algorithme que nous avons présenté dans la Sect. 5.2 repose sur des critères locaux, décelés au cours de la recherche, pour ses différentes optimisations. Nous allons d'abord greffer à `BackTrack` une autre amélioration, qui repose cette fois-ci sur la structure globale du graphe, et en particulier sur ses composantes biconnexes. Cet algorithme, appelé BCC, a été présenté dans une version « contraintes » [Baget and Tognetti, 2001], utilisant des contraintes binaires. Nous présentons ici son adaptation à la projection de graphes élémentaires, où les relations sont n-aires.

BCC est un ajout à `BackTrack` qui repose sur un ordre particulier (statique) des sommets du graphe requête H. Ce prétraitement du graphe requête, basé sur la construction de l'arbre des composantes biconnexes, s'effectue en temps linéaire. Les informations liées à cet ordre des sommets du graphe nous permettront, au cours du `BackTrack`, de réduire l'espace de recherche : certains échecs permettent en effet de supprimer définitivement des sommets de l'ensemble des images possibles, et certains succès nous permettront de compiler des projections partielles. BCC fonctionnera par de petits sauts arrières, et des *sauts avant* permis par la compilation de solutions partielles.

Nous expliquons cet algorithme, évaluons sa complexité, et montrons qu'il est polynomial (et optimal) dans le cas des arbres. Enfin, nous montrons que cet algorithme peut être greffé sur nFC2+CDBJ, et que ces différentes méthodes coupent l'arbre de recherche de façon différente.

5.3.1 Définitions et notations

Nous rappelons dans cette section quelques définitions sur les composantes biconnexes d'un graphe, que nous adaptons au cas n-aire, et présentons quelques notions (ordres compatibles, accesseurs, sommets concepts terminaux), qui seront indispensables à la compréhension de l'algorithme BCC.

[7] Cependant, cet algorithme serait plus efficace si nous bénéficions d'une représentation matricielle du graphe G, ce que nous nous interdisons si nous voulons utiliser cet algorithme dans les modèles de la famille $\mathcal{SG}$ utilisant des règles de graphes. Dans un cadre limité à la recherche de projection(s), et si le graphe G n'est pas trop grand, une telle représentation est à adopter.

Ordre compatible

Soit H un graphe élémentaire, et $V_1, \ldots, V_k$ des sous-ensembles de $V(H)$, ordonnés de V_1 à V_k, tels que $\cup_{1 \leq i \leq k} V_i = V(H)$ (mais leur intersection n'est pas nécessairement vide). Nous appelons $pred(V_i)$ l'union des $V_1, \ldots, V_{i-1}$. Nous disons qu'un ordre total des sommets concepts de H est *compatible avec cette décomposition* si, pour chaque V_i, pour chaque sommet concept x dans V_i, si $x \notin pred(V_i)$, alors x est plus grand que tous les sommets concepts de $pred(V_i)$. Un tel ordre sera obtenu en numérotant d'abord les éléments de V_1, puis ceux de V_2 qui ne sont pas dans V_1, ..., puis ceux de V_k qui ne sont pas dans $pred(V_k)$.

Si nous nous donnons un tel ordre, nous appelons *accesseur* d'un sous-ensemble V_i le plus petit élément de V_i.

Exemple introductif : le cas des composantes connexes

Nous rappelons qu'un graphe non vide H est dit *connexe* s'il existe un chemin entre tout couple de sommets concepts de H. Une *composante connexe* de H est un sous-graphe de H, connexe, de taille maximale.

Supposons maintenant un graphe H dont les composantes connexes sont $V_1, \ldots, V_k$. Nous dirons qu'un ordre total sur les sommets concepts de H est *CC-compatible* s'il est compatible avec un ordre quelconque de ses composantes connexes. Un ordre CC-compatible sera obtenu en numérotant les sommets concepts de H composante connexe par composante connexe.

Si nous exécutons `BackTrack` pour chercher les projections de H, ayant au moins deux composantes connexes, dans un graphe G quelconque, en utilisant un ordre CC-compatible pour les sommets concepts de H, alors `BackTrack` risque de réaliser énormément de travail inutile : s'il échoue sur la composante V_i (*i.e.* il détecte un échec sur l'accesseur de V_i), alors il générera toutes les autres projections du sous-graphe de H engendré par les sommets concepts de $pred(V_i)$, avant de répéter exactement la même erreur sur V_i pour chacune de ces projections.

Ce problème est bien connu, et la solution habituelle est d'exécuter `BackTrack` de façon indépendante sur chacune des composantes connexes de H (que l'on peut considérer comme des sous-problèmes indépendants), et de s'arrêter si une de ces composantes connexes ne se projette pas dans G. Mais ceci pourrait aussi s'implémenter de façon très simple, avec une légère modification de la fonction *Niveau-Précédent* de `BackTrack` : si le sommet concept courant est l'accesseur d'une composante connexe (information stockée lors de la phase *Ordonner*), et aucune solution n'a été trouvée (*i.e.* aucune solution n'a été stockée par l'appel à *Solution-Trouvée*), alors *Niveau-Précédent* doit retourner 0, stoppant effectivement l'exécution de l'algorithme BT.

Bien que très simple, cet exemple illustre la démarche que nous avons utilisée pour l'algorithme BCC : prendre en compte dans l'ordre des sommets concepts des informations sur une propriété globale du graphe H, et utiliser cette propriété pendant l'exécution de BT. Avec un ordre CC-compatible, nous bénéficions de l'indépendance des sous-problèmes correspondant aux composantes connexes de H ; avec un ordre BCC-compatible, que nous

allons maintenant définir, nous tirerons parti d'une « dépendance restreinte » entre ces problèmes.

Composantes biconnexes d'un graphe binaire

Nous définissons tout d'abord les composantes biconnexes dans le cas binaire, puis dans le cas n-aire, suivant la présentation qui en est faite dans une version longue de [Gottlob et al., 1999], présentée à CP'2000.

Soit H un graphe *binaire*. Un sous-ensemble de k sommets d'un graphe connexe H est appelé un *k-séparateur* de H si la suppression de ces k sommets déconnecte H. Un graphe H est dit *k-connexe* s'il n'admet pas de $(k-1)$-séparateur, *i.e.* si lui ôter $k-1$ sommets quelconques n'est pas suffisant pour le déconnecter ou le réduire à un sommet. Un arbre, par exemple, est une composante 1-connexe mais pas 2-connexe, car on le déconnecte en lui enlevant un sommet qui n'est pas une feuille. Une composante k-connexe de H est un sous-graphe de H, k-connexe, de taille maximale.

Une *composante biconnexe* est une composante 2-connexe. Si un graphe n'est pas biconnexe, alors il existe un sommet dont la suppression déconnecte le graphe. Ce 1-séparateur est appelé *séparateur* (ou point d'articulation).

Dans le graphe H de la Fig. 5.17, les séparateurs sont dessinés en noir, et les zones grisées entourent les sommets appartenant à une même composante biconnexe.

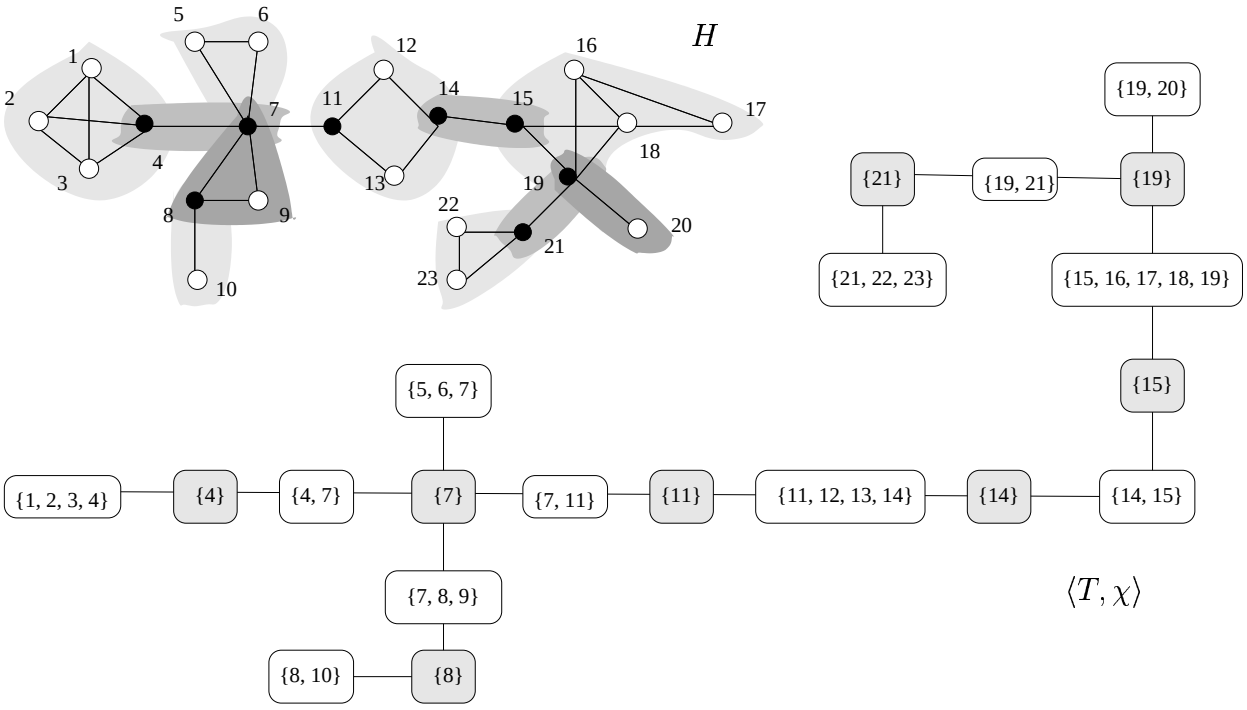


Fig. 5.17 – Séparateurs, composantes biconnexes, et construction de $\langle T, \chi \rangle$.

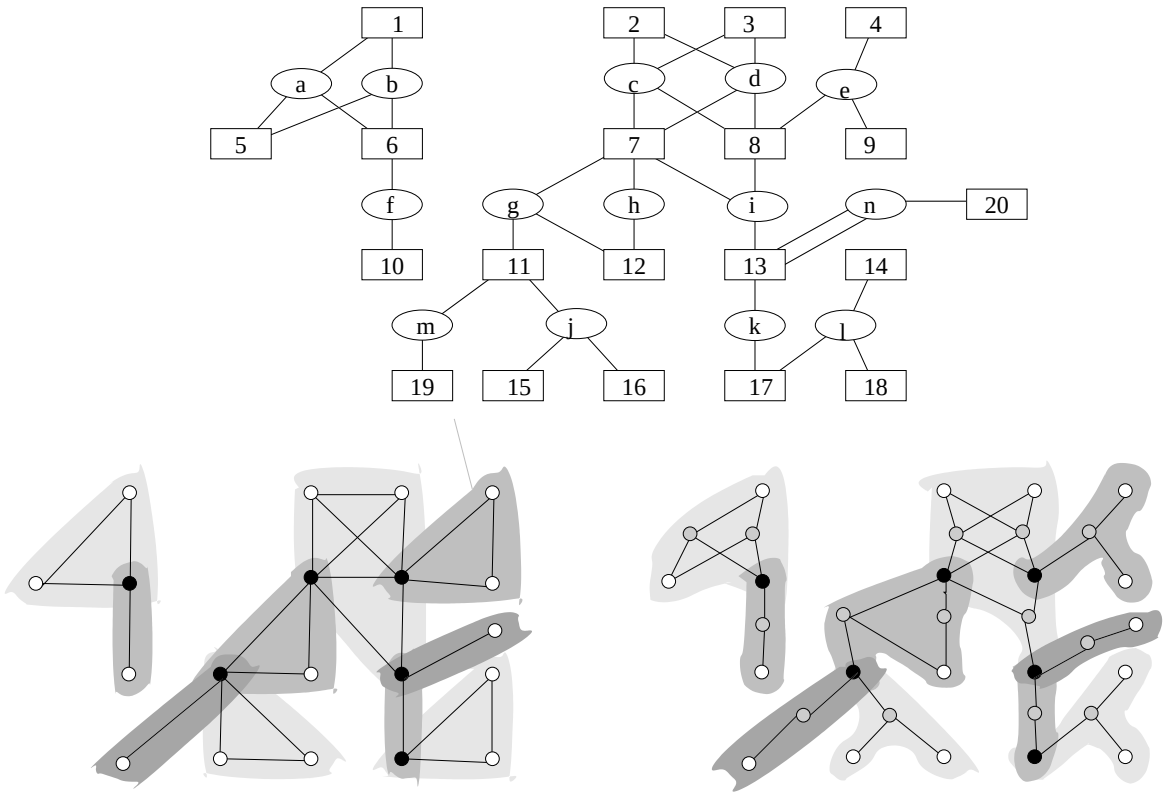
Soit H un graphe connexe binaire. Alors l'*arbre des composantes biconnexes* de H est

un arbre étiqueté $\langle T, \chi \rangle$ où χ est une bijection entre l'ensemble des sommets de T et un ensemble formé des composantes connexes et des séparateurs de H.[8] Il y a une arête pq dans l'arbre T si et seulement si $\chi(p)$ est un séparateur et $\chi(q)$ est une composante biconnexe contenant $\chi(p)$.

Si H n'est pas connexe, la définition précédente produit non pas un arbre, mais une forêt (chaque composante connexe ayant son arbre des composantes biconnexes associé). Nous l'appellerons la forêt des composantes biconnexes de H. Il est possible de calculer cette forêt en temps linéaire ([Tarjan, 1972], voir aussi l'exercice guidé dans [Cormen et al., 1990]).

Dans l'exemple de la FIG. 5.17, nous avons représenté en grisé les sommets de $\langle T, \chi \rangle$ qui correspondent à des séparateurs de H.

Composantes biconnexes d'un hypergraphe



Binarisation par le graphe primaire

Binarisation par le graphe caché

FIG. 5.18 – Constructions de l'arbre des composantes biconnexes d'un hypergraphe.

Comment adapter ces définitions aux hypergraphes orientés que sont les graphes élémentaires ? Notons tout d'abord que, la connexité étant une notion « non orientée », nous

[8] Notons que la représentation des séparateurs dans l'arbre des composantes biconnexes suffit à corriger l'erreur qui s'était glissée dans [Baget and Tognetti, 2001], où l'arbre tel qu'il était défini était en fait un graphe.

oublierons (jusqu'au calcul des projections) l'ordre sur les arguments des relations. Nous considérerons pour l'instant que ces relations sont des hyperarêtes, et les représenterons comme précédemment, par leur biparti d'incidence (plutôt que par la représentation plus traditionnelle consistant à entourer les extrémités d'une même hyperarête), mais en ignorant les numéros inscrits à côté des traits.

[Gottlob et al., 1999] binarisent un hypergraphe H en utilisant son graphe primaire (*primal graph*) $P(H)$. Ce graphe est aussi appelé 2-section de l'hypergraphe [Berge, 1970]. Il s'agit du graphe (binaire) dont les sommets sont les sommets du graphe H, et tel qu'il existe une arête entre deux sommets distincts si ces sommets appartiennent à une même hyperarête de H (chaque hyperarête est donc remplacée par une clique). Les séparateurs et les composantes biconnexes de l'hypergraphe H sont définies comme étant celles de $P(H)$. La FIG. 5.18 illustre cette transformation. Comme dans l'exemple précédent, les séparateurs de ce graphe sont dessinés en noir et ses composantes biconnexes sont mises en valeur par des zones grisées.

Cependant, nous préférons travailler, comme nous en avons pris l'habitude, sur le graphe caché $C(H)$ (*i.e.* le graphe biparti correspondant, où les hyperarêtes sont considérées comme des sommets). Nous appellerons *vrai sommet* un sommet de $C(H)$ qui représente un sommet de H. Un *vrai séparateur* est un vrai sommet séparateur. Une *vraie composante biconnexe* est la restriction aux vrais sommets d'un sous-graphe maximal de $C(H)$ n'admettant pas de vrai séparateur.

Dans la FIG. 5.18, nous avons représenté en noir les vrais séparateurs, en blanc les autres vrais sommets, et en gris les sommets qui représentent des hyperarêtes de H. Les vraies composantes biconnexes sont également mises en valeur par des zones grisées.

L'équivalence entre ces deux méthodes est précisée par la propriété suivante, qui se démontre de façon immédiate :

Propriété 5.9 *Soit H un hypergraphe. Alors les séparateurs de $P(H)$ sont exactement les vrais séparateurs de $C(H)$, et les composantes biconnexes de $P(H)$ sont exactement les vraies composantes biconnexes de $C(H)$.*

Nous pourrons donc appeler séparateur d'un hypergraphe H aussi bien un séparateur de $P(H)$ qu'un vrai séparateur de $C(H)$, et appeler composante biconnexe d'un hypergraphe aussi bien une composante biconnexe de $P(H)$ qu'une vraie composante biconnexe de $C(H)$. Dans les deux cas, la forêt des composantes biconnexes obtenue sera identique.

Bien que la caractérisation par le graphe caché semble plus compliquée, elle a un réel intérêt. En effet, il est immédiat d'adapter l'algorithme construisant l'arbre des composantes biconnexes d'un graphe binaire pour qu'il construise l'arbre des vraies composantes biconnexes d'un graphe biparti. L'algorithme obtenu reste linéaire. Or si H est un hypergraphe ayant m hyperarêtes d'arité k, le graphe primaire aura dans le pire des cas $m \times k(k-1)$ arêtes, alors que le graphe caché n'aura que $m \times k$ arêtes. L'algorithme de construction de $\langle T, \chi \rangle$ utilisant le graphe primaire est donc en $O(mk^2)$, tandis que celui utilisant le graphe caché est en $O(mk)$.

La FIG. 5.19 représente la forêt des composantes biconnexes de l'hypergraphe H de la FIG. 5.18. Cet exemple sera utilisé tout au long de cette section. Les sommets de la forêt

qui correspondent à des séparateurs ont été représentés en grisé. Les autres sommets (qui correspondent à des composantes biconnexes) sont identifiés par des lettres $A, B, \ldots$

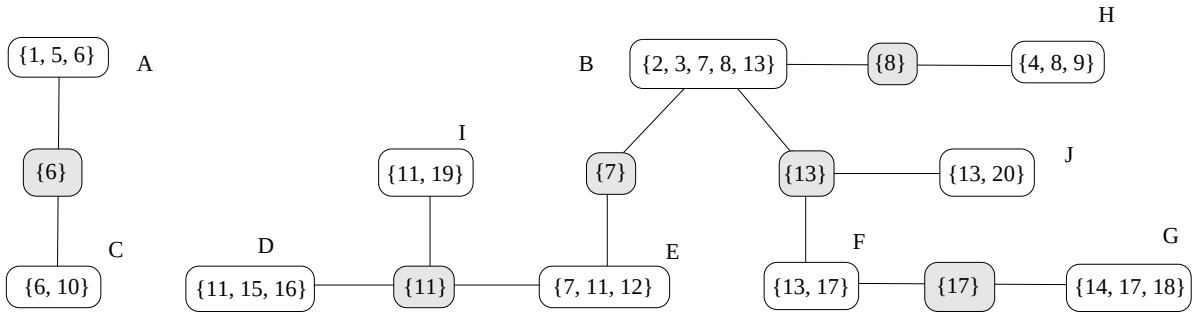


FIG. 5.19 – Forêt des composantes biconnexes de l'hypergraphe de la FIG. 5.18

Ordre BCC-compatible

Soit H un hypergraphe et $\langle T, \chi \rangle$ sa forêt des composantes biconnexes. Pour chaque arbre de T, choisissons un sommet de cet arbre qui sera la racine. Nous pouvons donc considérer T comme une forêt enracinée. Un *ordre naturel* des composantes biconnexes est un ordre total sur ces composantes, compatible avec une partition de H en composantes connexes (*i.e.* numérotant les composantes connexes arbre par arbre), tel que toute composante biconnexe est plus grande que toutes celles qui sont ses ancêtres dans T. De façon générale, aussi bien un parcours en profondeur qu'un parcours en largeur de T à partir de chacune de ses racines suffisent à générer un ordre naturel des composantes biconnexes.

Considérons l'exemple de la FIG. 5.19. En choisissant A et B comme racines, nous pouvons considérer notamment les ordres naturels suivants :

- $A < C < B < E < F < J < H < D < I < G$ est obtenu par un parcours en largeur ;
- $A < C < B < E < D < I < F < G < J < H$ est obtenu par un parcours en profondeur ;
- $A < C < B < E < F < D < H < G < I < J$ n'est obtenu ni par un parcours en largeur, ni par un parcours en profondeur.

Par contre, les ordres suivants ne sont pas naturels :

- $A < C < B < D < E < F < G < H < I < J$, car D est plus petit que son ancêtre E ;
- $A < B < C < E < D < F < G < H < I < J$, car cet ordre n'est pas compatible avec la partition en composantes connexes.

Définition 5.2 *Un ordre* BCC-compatible *des sommets d'un hypergraphe H est un ordre compatible avec un ordre naturel de sa forêt des composantes biconnexes.*

Remarquons qu'un ordre BCC-compatible est compatible avec une décomposition en composantes connexes (il est CC-compatible). Nous pourrons nous intéresser à un tel ordre,

calculé par un parcours en profondeur (DFS,pour *Depth-First Search*) de la forêt des composantes biconnexes, et par un parcours en largeur à l'intérieur des composantes (BFS pour *Breadth-First search*) : il sera appelé un ordre BCC(DFS,BFS). Il peut être calculé en temps linéaire.

Calculons un ordre BCC(DFS, BFS) de l'hypergraphe de la FIG. 5.18, dont la forêt des composantes biconnexes est représentée dans la FIG. 5.19 :

1. nous choisissons un ordre naturel pour les composantes biconnexes, calculé par un parcours en profondeur : $A < C < B < E < D < I < F < G < J < H$;
2. nous faisons un parcours en largeur des sous-graphes engendrés par chacune de ces composantes : $\{1 < 5 < 6\} < \{6, 10\} < \{2 < 3 < 7 < 8 < 13\} < \{7 < 11 < 12\} < \{11 < 15 < 16\} < \{11, 19\} < \{13 < 17\} < \{14 < 17 < 18\} < \{13, 20\} < \{4 < 8 < 9\}$
3. nous renumérotons les sommets, en partant de la plus petite de ces composantes, ce qui nous donne l'ordre suivant, illustré dans la FIG. 5.20 : $\{1 < 2 < 3\} < \{3, 4\} < \{5 < 6 < 7 < 8 < 9\} < \{7 < 10 < 11\} < \{10 < 12 < 13\} < \{10, 14\} < \{9 < 15\} < \{16 < 15 < 17\} < \{13, 18\} < \{19 < 8 < 20\}$.

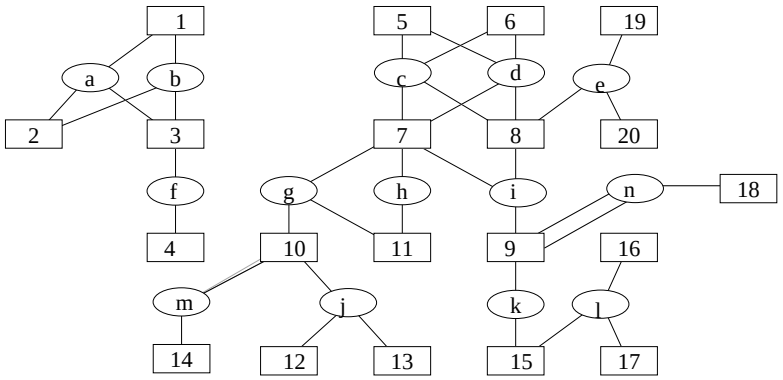


FIG. 5.20 – Un ordre BCC(DFS, BFS) pour l'hypergraphe de la FIG. 5.18

Dans l'algorithme BCC, `BackTrack` devra considérer les sommets dans un tel ordre BCC-compatible, calculé linéairement lors de la phase de prétraitement (lors de l'appel à la fonction *Ordonner*).

Arbre BCC, accesseurs et sommets terminaux

Soit H un graphe élémentaire. Nous noterons de la même façon H son hypergraphe associé, obtenu en considérant H comme non orienté. Soit $\langle T, \chi \rangle$ sa forêt des composantes biconnexes associée, et soit un ordre BCC-compatible sur les sommets de H obtenu en considérant un enracinement de T. Alors l'*arbre BCC* $\langle A, \beta \rangle$ associé à H pour cet ordre BCC-compatible est obtenu à partir de $\langle T, \chi \rangle$ de la façon suivante :

– les sommets et les arêtes de A sont ceux de T, et les arbres composant T sont enracinés par un nouveau sommet, relié aux racines utilisées pour construire l'ordre BCC-compatible ;

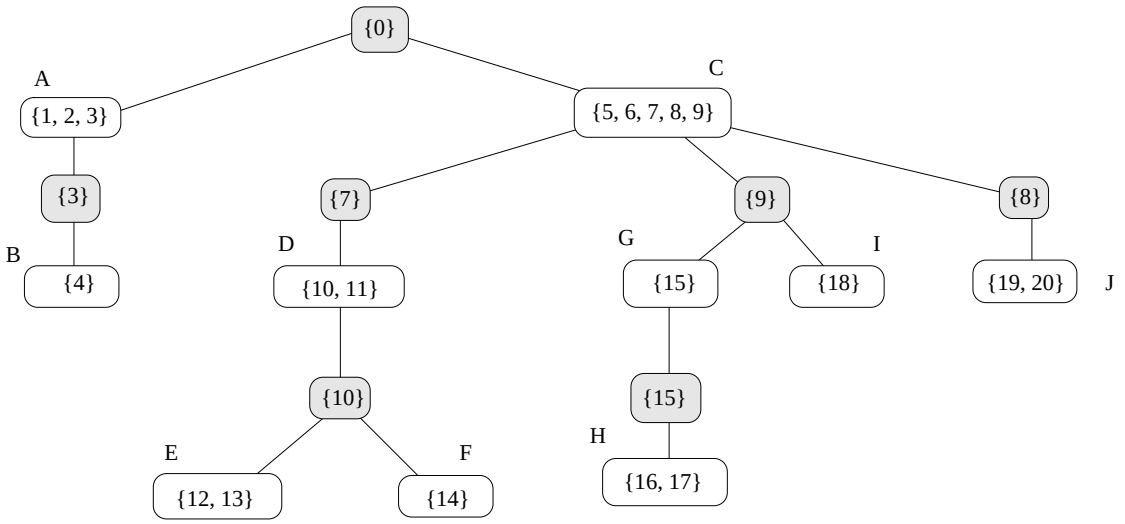


FIG. 5.21 – Arbre BCC correspondant à l'ordre BCC(DFS, BFS) de la FIG. 5.20

- β étiquette le nouveau sommet par $\{0\}$;
- si x est un sommet de A représentant un séparateur de H, alors $\beta(x) = \chi(x)$
- si x est un sommet de A représentant une composante biconnexe de H, remarquons que le sommet père de x, noté $y = père(x)$, est soit la nouvelle racine étiquetée par $\{0\}$, soit représente un séparateur s. Alors $\beta(x)$ est obtenu à partir de $\chi(x)$, en lui ôtant s si y n'est pas la racine de A.

L'arbre de l'hypergraphe H de la FIG. 5.18, pour l'ordre BCC-compatible calculé précédemment, est représenté dans la FIG. 5.21. Notons que nous avons « renommé » les sommets représentant les composantes $A, B, \ldots$ pour que l'ordre naturel que nous avons considéré sur les composantes biconnexes corresponde à l'ordre alphabétique.

Une *composante* de l'arbre BCC $\langle A, \beta \rangle$ est un sommet qui représente une composante biconnexe (elles sont représentées en blanc dans la FIG. 5.21). Un sommet x de l'hypergraphe H appartient à une composante C si $x \in \beta(C)$. Soit x un sommet de H appartenant à la composante C. L'*accesseur* d'une composante C est l'unique élément de $\beta(père(C))$. Remarquons que si $père(C)$ est la racine de A, alors l'accesseur de x n'est pas un sommet de H (mais l'index 0 de cette entité indéterminée servira de signal d'arrêt au `BackTrack`, comme indiqué dans l'exemple introductif consacré aux composantes connexes). L'accesseur d'un sommet est l'accesseur de l'unique composante à laquelle il appartient. Le *point d'entrée* d'une composante C est le plus petit sommet de $\beta(C)$.

Le point d'entrée d'une composante est dit *naturel* s'il est dans le voisinage de son accesseur. Nous ne considérerons par la suite que des ordres BCC-compatibles pour lesquels tous les points d'entrée sont naturels.

Une *composante feuille* de l'arbre BCC est une feuille de A. Un sommet de H est dit *terminal* s'il est le plus grand sommet d'une composante feuille. Nous appelons *composantes engendrées par un sommet x de A*, et notons $A_C(x)$ l'ensemble des composantes de

$\langle A, \beta \rangle$ faisant partie du sous-arbre enraciné de A ayant x pour racine. Dans la FIG. 5.21, nous avons $A_C(D) = \{D, E, F\}$. De la même façon, nous définissons l'ensemble $A_H(x)$ des sommets de H engendrés par un sommet x de A comme l'ensemble des sommets de H étiquetant $A_C(x)$: $A_H(x) = \cup_{C \in A_C(x)} \beta(C)$. Soit t un sommet terminal de H, C sa composante (feuille), et $C = C_1, \ldots, C_k$ les composantes ancêtres de C (obtenues par un parcours de l'arbre enraciné A remontant de C jusqu'à la racine). Alors, pour $1 \leq i \leq k$, si t est le plus grand sommet de $A_H(c_i)$, le point d'entrée de C_i est appelé un *compilateur* de t. Notons que si t est l'unique sommet d'une composante feuille, alors t est terminal et t est un compilateur de t.

Composante	A	B	C	D	E	F	G	H	I	J
Éléments	1, 2, 3	4	5, 6, 7, 8, 9	10, 11	12, 13	14	15	16, 17	18	19, 20
Accesseur	0	3	0	7	10	10	9	15	9	8
Sommets terminaux	/	4	/	/	13	14	/	17	18	20
Compilateurs	/	4, 1	/	/	12	14, 10	/	16, 15	18	19, 5

TAB. 5.1 – Accesseurs, sommets terminaux et compilateurs pour l'exemple courant.

La table ci-dessus indique les accesseurs, les sommets terminaux et leurs compilateurs pour le graphe H de l'exemple courant (FIG. 5.18), dont les sommets ont été ordonnés suivant un ordre BCC-compatible comme illustré dans la FIG. 5.20.

5.3.2 Présentation de l'algorithme

Le noyau de l'algorithme BCC est `BackTrack`, comme il est décrit dans l'ALG. 2. L'appel à *Ordonner* donne aux sommets concepts du graphe élémentaire requête H un ordre BCC-compatible et calcule les composantes de $\langle A, \beta \rangle$, les accesseurs, les sommets concepts terminaux, et les compilateurs. Ce prétraitement s'effectue en temps linéaire.

Cet algorithme repose sur deux propriétés essentielles, que nous allons présenter avant de proposer une implémentation.

Suppression définitive d'un candidat possible

La première idée sur laquelle repose BCC est que, lorsqu'un échec remonte d'une composante biconnexe jusqu'à son accesseur, alors seule une modification de l'image de cet accesseur peut permettre de trouver une solution. Plus précisément :

Propriété 5.10 *Si x est le point d'entrée d'une composante de $\langle A, \beta \rangle$, alors l'appel à Niveau-Précédent(x) peut, de manière sûre et optimale (si on ne considère que les informations sur le voisinage), retourner l'index de l'accesseur y de x. De plus, aucune projection de H dans G ne pourra associer au sommet concept y son image courante.*

Un appel à *Niveau-Précédent*(x) veut dire que nous avons détecté un échec (qui peut être interne ou terminal) sur le sommet concept x. Si x est un point d'entrée, la première partie de cette propriété assure que nous pouvons effectuer un saut arrière (BackJump)

directement à l'accesseur de x, sans risquer de rater une seule projection (sûr), et qu'un saut arrière plus important pourrait nous faire rater une projection (optimal). Des définitions formelles qui précisent cette notion de sauts arrières sûrs et optimaux peuvent être trouvées par exemple, dans [Dechter and Frost, 1999]. Par exemple, un échec décelé sur le sommet concept dont l'index est 19 (c'est un point d'entrée) dans la FIG.5.20 entraîne un saut arrière vers le sommet concept dont l'index est 8. Un échec décelé sur le sommet concept x dont l'index est 5 (c'est aussi un point d'entrée) veut dire que nous n'avons trouvé aucune projection pour la composante connexe contenant 5, c'est pourquoi *Niveau-Précédent*(x) pourra retourner 0, stoppant effectivement l'exécution du `BackTrack`.

De plus, la seconde partie de cette propriété signifie que nous pouvons supprimer définitivement l'image courante de y de ses images possibles. Ceci s'implémentera aisément en maintenant, pour chaque accesseur de H, un champ contenant les images interdites.

Afin de démontrer la propriété précédente, nous utiliserons le lemme suivant, qui localise les échecs terminaux ayant provoqué l'échec sur x.

Lemme 5.1 *Soit x le sommet concept sur lequel est appelé Niveau-Précédent(x), et C sa composante. Notons X l'ensemble de sommets concepts de H sur lesquels un échec a été détecté après la dernière projection réussie d'un sommet de $post_V(x)$. Alors X est un ensemble de sommets appartenant à la même branche de composantes de $A_C(C)$.*

Preuve: Voir que, pour une série d'échecs successifs, chaque appel à *Niveau-Précédent*(z) retourne l'index du sommet immédiatement précédent suivant l'ordre BCC-compatible, si x n'est pas un point d'entrée. Donc, tant qu'on n'atteint pas un point d'entrée, les appels successifs à *Niveau-Précédent* retournent les indexs de sommets appartenant à la même composante. Dès que le sommet z considéré est un point d'entrée, l'appel à *Niveau-Précédent*(z) retourne l'accesseur de ce sommet, qui appartient à une composante mère de celle de z. Donc tous les sommets de X appartiennent à une même branche de composantes, enracinée par la composante de x. □

Nous pouvons maintenant prouver la propriété 5.10 :

Preuve: Nous montrons que, si x est un point d'entrée naturel (et nous avons considéré un ordre compatible tel que tous les points d'entrée sont naturels), alors *Niveau-Précédent*(x), tel qu'il est décrit dans la Prop. 5.10, opère une forme particulière de *Graph-Based Back-Jumping* (GBBJ, [Dechter, 1990]). En effet, le plus grand sommet de $pre_V(x)$ qui est un voisin de x est son accesseur y (car x est un point d'entrée, donc le plus petit de sa composante, et est naturel, donc voisin de y). Cependant, GBBJ doit aussi transmettre à y un « jumpback set » contenant tous les autres voisins de x dans $pre_V(x)$: comme cet ensemble est vide (sinon les composantes de x et de y seraient fusionnées), ne pas le maintenir ne pose aucun problème. Comme GBBJ opère des sauts sûrs et optimaux si on ne considère que les informations sur le voisinage, alors le saut arrière opéré par *Niveau-Précédent*(x) a les mêmes propriétés.

Maintenant, pour prouver la deuxième partie de la Prop. 5.10, supposons qu'il existe une projection qui associe à y son image courante y', et montrons que ceci est absurde.

Soit C la composante du point d'entrée x, considérons le sous-graphe H' de H engendré par les sommets de $A_H(C)$, et ordonnons ses sommets par une translation de l'ordre des sommets de H. Donnons maintenant à y, premier sommet de H', le sommet concept y' de G comme image courante. Alors il existe une projection π de H' (notre hypothèse), et `BackTrack` va trouver la première de ces projections pour l'ordre considéré. Réexaminons maintenant le `BackTrack` sur H qui nous avait mené à un échec sur x, et considérons le premier échec terminal (sur un sommet concept noté x_e) qui nous avait fait dépasser cette projection. Remarquons que, grâce au lemme 5.1, x_e est bien un sommet de H'. Comme avant de considérer x_e, la projection courante de H incluait une partie de la projection (existante) π de H', alors l'image possible $\pi(x_e)$ n'a pu être vidée que par un sommet concept qui n'est pas dans H' (sinon, π ne serait pas une projection). Donc il existe une relation entre x_e est un sommet concept x'_e situé dans une composante qui n'est pas dans H' : ceci est absurde car il y aurait un cycle élémentaire $x_e, \ldots, x, y, \ldots, r, \ldots, x'_e, x_e$ où r est un sommet concept qui apparaît dans une composante qui est ancêtre de celle de y et de celle de x_e, et x_e et x'_e seraient dans la même composante biconnexe. □

Compilation d'une solution partielle

La seconde idée à la base de l'algorithme BCC est une version duale de celle exposée dans la Prop. 5.10. En effet, nous allons voir que si nous avons trouvé une projection partielle de tout un sous-arbre de $\langle A, \beta \rangle$, alors aucune modification à l'extérieur de ce sous-arbre ne pourra réfuter cette projection partielle.

Nous aurons besoin d'associer à chaque compilateur de H une *liste des valeurs compilées*, qui recevra des couples sous la forme (clé, valeur). Cette liste sera initialement vide. Notons que, si (k, v) appartient à cette liste, alors l'ajout de (k, v') effacera (k, v). Suivant le nombre d'images possibles pour les sommets de H, cette liste pourra être implémentée par une liste d'association ou par une table de hachage.

Propriété 5.11 *Soit x un sommet terminal dont l'index est i, alors tout appel à Niveau-Suivant(x), avant de retourner $i+1$, peut ajouter, pour tout compilateur c de x, le couple $(v, i+1)$ à la liste des valeurs compilées de c, où v est l'image courante de l'accesseur de c.*

Alors à chaque fois qu'un appel à Niveau-Suivant(x) devra retourner l'index i d'un compilateur c, si la liste des valeurs compilées de c contient un couple (v, j), où v est l'image courante de l'accesseur de c, alors Niveau-Suivant(x) retournera j à la place de i.

Preuve: La démonstration est duale de celle de la deuxième partie de la Prop. 5.10. Nous avions montré que si un échec se propage vers l'accesseur x de la racine d'un sous arbre A' de $\langle A, \beta \rangle$, alors seule une modification de l'image courante de x pouvait mener à une projection des sommets concepts de A'. Ici, nous devons montrer que, si nous avons trouvé une projection des sommets concepts de A', alors aucune modification à l'extérieur de A' autre que celle de l'image courante de x ne peut réfuter cette projection. La preuve en sera similaire. □

Un exemple

L'appel à *Niveau-Suivant*(x) sur un sommet terminal x veut dire que tout un sous-arbre de $\langle A, \beta \rangle$ a été projeté. Considérons de nouveau le graphe élémentaire H de la FIG. 5.20. Supposons maintenant un appel à *Niveau-Suivant*(17) : ceci veut dire que nous avons une projection de l'ensemble du sous-arbre dont la racine est la composante G. Avant de retourner 18, cet appel va ajouter aux listes des valeurs compilées de ses compilateurs (16, et 15) les couples respectifs $(v_1, 18)$ et $(v_2, 18)$, où v_1 est l'image courante de l'accesseur (15) de 16, et v_2 est l'image courante de l'accesseur (9) de 15. Maintenant, `BackTrack` examine le sommet suivant 17, c'est à dire 18, et ne lui trouve aucune image possible (remarquons que nous n'utilisons pour l'instant pas `Forward Checking`). Alors l'appel à *Niveau-Précédent*(18), comme indiqué par la Prop. 5.10, retournera 9, et interdira définitivement l'image courante de 9 pour ce sommet. Supposons maintenant que 9 possède une autre image possible, alors *Niveau-Suivant*(9) devrait retourner 10. Or 10 est un compilateur (celui de 14) et comme 14 est un sommet terminal qui a été projeté (juste avant de considérer 15, 16, 17), 10 contient dans sa liste des valeurs compilées $(v, 15)$, où v était l'image courante de l'accesseur de 10 (7), au moment où on a projeté 14. Or, cette valeur v est toujours l'image courante de 7, donc *Niveau-Suivant*(9) va retourner la valeur associée à la clef v, c'est à dire 15. Supposons maintenant que la première image possible de 15 soit la même que celle au moment où nous avions projeté 17. Lorsque nous arrivons sur le point d'entrée 16 de la composante H nous nous apercevons que l'image courante de son accesseur 15 est une clef de la liste des valeurs compilées et sautons en avant directement sur 18.

Implémentation de l'algorithme

L'algorithme BCC s'implémente directement à partir de la version de `BackTrack` que nous avons proposée (Alg. 2). La propriété 5.10 est traduite dans l'algorithme qui implémente *Niveau-Précédent* (Alg. 6), et la propriété 5.11 est traduite dans celui qui implémente *Niveau-Suivant* (Alg. 7).

5.3.3 Évaluation de BCC

Nous allons tout d'abord évaluer la complexité de BCC et montrerons qu'il est polynomial et optimal si H est un arbre. Ensuite nous comparerons BCC à des algorithmes de BackJump, ou utilisant des techniques de filtrage, et montrerons l'utilité d'implémenter des algorithmes mixtes.

Complexité

Lemme 5.2 *Lorsque l'on utilise l'algorithme BCC alors toute composante C de H n'est accédée qu'au plus une fois pour chaque image possible de son accesseur.*

Algorithme 6: *Niveau-Précédent*(x)

Données : Le sommet concept courant x, sur lequel a été détecté un échec.

Résultat : L'index du sommet sur lequel BCC doit retourner, comme indiqué dans la Prop. 5.10. Tout accesseur est doté d'un champ *Interdits* qui contient tous les sommets de G dans lesquels cet accesseur ne peut pas se projeter. Ce champ devra être pris en compte par *Autres-Candidats ?* (Alg. 4).

```
si (point-d'entrée ?(x)) alors
    si (accesseur(x) = 0) alors
        retourner 0;
    sinon
        accesseur ← accesseur(x);
        Interdits(accesseur) ← Interdits(accesseur)∪{candidat-courant(accesseur)};
        retourner index(accesseur);
sinon
    retourner index(x) − 1;
```

Algorithme 7: *Niveau-Suivant*(x)

Données : Le sommet concept courant x, dont on a calculé une image possible.

Résultat : L'index du sommet que BCC doit maintenant examiner, comme indiqué dans la Prop. 5.11.

```
index ← index(x) + 1;
si (sommet-terminal ?(x) alors
    pour c ∈ compilateurs(x) faire
        clé ← candidat-courant(accesseur(c));
        Ajout-CLE(Liste-valeurs-compilées(c), clé, index);
        // Ajout du couple (clé, index) dans la liste des valeurs compilées
si (index ≠ |V(H)| + 1) alors
    c ← V[index];
    si (point-d'entrée ?(c)) alors
        accesseur ← accesseur(c);
        image ← candidat-courant(accesseur);
        si (Teste-CLE(Liste-valeurs-compilées(c), image)) alors
            // Teste l'existence d'un couple de la forme (image, ∗) dans la liste
            index ← Cherche-VAL(Liste-valeurs-compilées(c), image);
            // Retourne l'unique index tel que (image, index) est dans la liste
retourner index;
```

Preuve: Il s'agit d'une conséquence des Props. 5.10 et 5.11. Si BCC entre dans un sous-arbre T, alors soit nous trouvons une projection de T, et l'image courante de l'accesseur de la racine de T sera compilée ; soit nous ne trouvons pas de projection, et l'image courante de cet accesseur sera interdite. Donc, à chaque fois que nous entrerons dans un sous-arbre T, l'image courante de son accesseur sera soit compilée, soit interdite, et nous n'aurons plus besoin d'entrer dans ce sous-arbre avec la même image pour son accesseur. □

Corollaire 5.1 *Lorsque H est un arbre (ou, plus précisément, quand sa binarisation par $P(H)$ est un arbre), alors BCC trouve la réponse à* PROJECTION ? *en temps $O(n_H \times n_G \times d_G \times k)$ (en considérant comme constant le coût lié aux tests de compatibilité des types), et en temps $O(n_H \times n_G \times k)$ si nous adoptons une représentation de G par sa matrice d'adjacence.*

Preuve: Les composantes biconnexes d'un arbre sont des graphes complets à deux sommets, l'accesseur et le point d'entrée. Donc toute composante ne contient qu'un sommet, le point d'entrée x. Lorsqu'on entre dans une composante, en considérant le point d'entrée x comme le sommet courant, alors les candidats possibles pour x sont générés en temps $O(d_G \times k)$ ($O(k)$ avec une matrice d'adjacence). Nous n'entrerons qu'au plus n_G fois dans les n_H composantes de H, d'où la complexité. □

Ce résultat, dans sa version contraintes (où la complexité est alors en $O(nd^2)$, voir [Baget and Tognetti, 2001]), est équivalent à celui de [Freuder, 1982] pour les réseaux de contraintes qui sont des arbres. L'algorithme de [Freuder, 1982] repose sur une phase d'arc-consistance, puis par un appel à `BackTrack` qui sera alors assuré d'être *libre de backtrack* (*BackTrack Free*). Notons aussi que cette complexité est optimale dans le cadre des réseaux de contraintes [Dechter and Pearl, 1988]. En utilisant la transformation *E2C* pour transformer les paramètres décrivant les graphes G et H en paramètres décrivant le réseau de contraintes, nous vérifions immédiatement que la complexité que nous avons obtenue est, elle aussi, optimale.

Enfin, le théorème suivant donne cette complexité dans le cas général où H est un hypergraphe quelconque :

Théorème 5.2 *Soit n_b le nombre de composantes biconnexes du graphe élémentaire H que l'on cherche à projeter dans G, et t_b la taille de la plus grande de ces composantes. Alors BCC trouve la réponse à* PROJECTION ? *en temps $O(n_b \times n_G \times (d_H \times d_G \times {t_b}^{n_G} \times k))$ (en considérant comme constant le coût lié aux tests de compatibilité des types), et en temps $O(n_b \times n_G \times (d_H \times {t_b}^{n_G} \times k))$ si nous adoptons une représentation de G par sa matrice d'adjacence.*

Si nous comparons cette complexité dans le pire des cas avec celle que nous avons calculée pour `BackTrack`, le temps mis par BCC est obtenu à partir de celui mis par `BackTrack` en le multipliant par $n_b \times n_G \times (t_b/n_H)^n_G$ (où $t_b/n_H \leq 1$). À titre d'illustration, si H est un graphe à 25 sommets, admettant 5 composantes biconnexes de taille 5, et G est un graphe de taille 50, alors le temps mis par `BackTrack` dans le pire des cas (et ce

pire des cas peut être atteint malgré la structure particulière de H, qui comprend un grand nombre de composantes biconnexes) est 10^{34} fois supérieur au pire des cas de BCC.

Enfin, nous remarquons que, si cette complexité dans le pire des cas est la même que celle de l'algorithme proposé par [Freuder, 1982], BCC sera plus efficace en pratique. En effet, BCC opère un `BackTrack` entre les composantes, alors que l'algorithme de [Freuder, 1982] peut se voir comme un « générer et tester » sur ces composantes.

Compatibilité de BCC et de différents BackJumps

Nous avons vu (Prop. 5.10) que BCC repose sur une forme particulière, très limitée, de *BackJump*, qui est en fait un *Graph-Based BackJumping* (GBBJ) ne fonctionnant que sur les points d'entrée. Une question naturelle se pose alors : que se passe-t-il lorsque l'on utilise des formes plus efficaces de *BackJump* ?

Propriété 5.12 *Soit Niveau-Précédent une fonction implémentant une forme de* Back-Jump *sûr (*i.e. *qui ne peut rater aucune projection) XBJ. Supposons de plus que Niveau-Précédent retourne toujours l'index d'un sommet appartenant à la même composante biconnexe que le sommet courant (*i.e. *appartenant à la même composante, ou étant l'accesseur de cette composante).*

Alors si y est l'accesseur de x, et aucune suite de X-BackJumps *successifs ne peut retourner dans la composante de x, alors l'image courante de y peut être interdite de façon permanente.*

L'algorithme obtenu BCC+XBJ est adéquat et complet.

Preuve: Voir que les restrictions que nous nous sommes données suffisent à prouver le Lem. 5.1. La deuxième partie de la Prop. 5.10 est alors démontrée de la même façon. □

Corollaire 5.2 *Les algorithmes BCC+GBJ, BCC+GBBJ et BCC+CDBJ sont adéquats et complets.*

Preuve: Ces trois algorithmes respectent les spécifications de la Prop. 5.12. L'appel à *Niveau-Précédent*, si l'index retourné est celui de l'accesseur y du sommet courant x, pourra interdire définitivement l'image courante de y. Remarquons que, pour les algorithmes nécessitant le maintien d'un *backjump set* (GBBJ et CDBJ), cette suppression ne pourra être effectuée que si le *backjump set* sur y est vide. □

Enfin, nous notons que BCC et les différents XBJ étudiés ici optimisent le parcours de façon différente, et que les avantages de ces deux types d'optimisation vont s'additionner. En effet, les différents XBJ remontent plus haut dans l'arbre de recherche que BCC, et sont donc en ce sens plus efficaces. D'un autre côté, si XBJ remonte dans une composante mère d'une composante C, rien ne l'empêchera de revenir dans C en utilisant la même image courante pour son accesseur (et, de façon duale, de recalculer une projection partielle d'un sous-arbre que l'on a déjà calculée).

BCC+XBJ combine donc les avantages des deux algorithmes, et le surcoût lié à la greffe de BCC sur XBJ ne réside que dans le prétraitement générant l'ordre BCC-compatible, qui est linéaire dans la taille de H.

Compatibilité de BCC et de différentes techniques de filtrage

Examinons maintenant comment BCC se combine avec différentes méthodes de filtrage.

Tout d'abord, BCC se combine de façon naturelle avec les différents types de `Forward Checking` que nous avons présentés. Nous exposons ici les modifications immédiates qu'il faut apporter à un `Forward Checking` pour lui greffer BCC, en prenant l'exemple de **nFC2** :

- L'appel à NIVEAU-PRÉCÉDENT se comporte exactement de la même façon que pour BCC (Alg. 6).
- L'appel à NFC2-PROPAGE risque de vider le Δ du point d'entrée d'une composante fille (dans le cas ou le sommet courant x est un accesseur). Il est alors possible d'interdire définitivement le candidat de x. En effet, il s'agit d'un cas où BCC aurait interdit ce candidat en remontant depuis y, mais où la méthode de propagation décèle l'échec un cran en avant, donc sans avoir à remonter. Ce comportement est implémenté par *(nFC2+BCC)-Propage* (Alg. 8).
- L'appel à NIVEAU-SUIVANT se comporte exactement de la même façon que pour BCC (Alg. 7).

Là encore, BCC+nFC2 combine les avantages des différents algorithmes, et la greffe de BCC ne génère qu'un surcoût linéaire dans la taille de H, lors de la phase de prétraitement.

Bien que nous n'ayons pas jugé nécessaire d'implémenter un algorithme tel que MAC pour résoudre le problème PROJECTION ?, notons toutefois que la combinaison de BCC et de MAC produit un résultat très intéressant, comme l'indique la propriété suivante [Baget and Tognetti, 2001], que nous traduisons en termes de projection :

Propriété 5.13 *Soit x le sommet concept courant de H examiné par l'algorithme MAC, et soit y un sommet concept de $post_V(x)$ (notons qu'il n'appartient pas forcément au voisinage de x), dont le Δ a été vidé par la phase de propagation. Supposons $C_1, \ldots, C_p$ des composantes appartenant à une même branche de l'arbre BCC, avec $x \in C_1$ et $y \in C_p$, et $z_2, \ldots, z_p$ les accesseurs de $C_2, \ldots, C_p$.*

Alors, pour $2 \leq i \leq p$, toutes les images possibles de c_i (celles qui n'ont pas été vidées temporairement par la phase de propagation) peuvent être définitivement supprimées *par BCC.*

Conclusion

Nous avons proposé un algorithme original, BCC que l'on peut greffer sur un algorithme efficace de recherche de projection entre graphes élémentaires, nFC2+CDBJ. Utiliser cet algorithme mixte n'occasionne aucun inconvénient (le seul surcoût, linéaire, étant engendré par le calcul de l'arbre BCC à la phase d'initialisation), et permet de couper efficacement l'arbre de recherche si l'hypergraphe H contient plusieurs composantes biconnexes.

Si on s'intéresse aux graphes conceptuels (qu'ils soient très simples, simples, ou emboîtés), les algorithmes que nous avons proposés justifient notre choix de calculer les projections sur les graphes élémentaires associés plutôt que directement sur les graphes conceptuels. Rappelons les intérêts de ce choix :

Algorithme 8: *(nFC2+BCC)-Propage*(x, y)

Données : Le sommet concept courant x de H, et un de ses candidats possibles y.

Résultat : Propage le choix de y pour x, suivant l'algorithme nFC2, et répond VRAI si et seulement si aucun domaine de $post_V(x)$ n'a été vidé. Si le Δ du point d'entrée d'une composante fille est vidé, alors y sera interdit pour x.

```
pour (r ∈ post_U(x)) faire
    pour (z ∈ args(r) ∩ post_V(x)) faire
        Δ ← Trouver-Delta(z, x, r);
        Δ' ← Précédent-Liste(Δ);
        Vider(Δ);
        pour (v ∈ Δ') faire
            si (Supportée(r, (x, y), (z, v))) alors
                Δ ← Δ ∪ {v};
        si (Vide ?(Δ)) alors
            // Seul ce test diffère de nFC2-Propage implémentée dans l'Alg. 5
            si (point-d'entrée ?(z)) alors
                // Dans ce cas x est nécessairement l'accesseur de z
                Interdits(x) ← Interdits(x)∪{candidat-courant(x)};
            retourner FAUX;
retourner VRAI;
```

- Tout d'abord, traduire les différents formalismes vers un formalisme commun, dans lequel les projections sont calculées, répond à un besoin de généricité des algorithmes.
- Ensuite, la vue hypergraphe permet d'écrire des algorithmes plus efficaces, que ce soit du point de vue du nombre de projections que l'on énumère (Sect. 4.1.2), ou parce que les hyperarêtes permettent une propagation plus efficace (comparaison de **nFC2** et **FC+**, Sect. 5.2.2).
- Si l'on s'intéresse aux graphes conceptuels emboîtés, alors un algorithme calculant des projections de graphes emboîtés devra prendre en compte de nouveaux objets, les boîtes. La traduction de ces boîtes en sommets permet non seulement d'utiliser le même algorithme, mais de bénéficier des différentes optimisations : par exemple, propager sur des sommets qui représentent des boîtes.
- De plus, si nous considérons un graphe emboîté (sans liens de co-identité), alors son graphe élémentaire associé est un hypergraphe qui aura souvent de nombreuses composantes biconnexes. Il s'agit là d'un cas où BCC est particulièrement efficace.

L'intérêt de BCC est en effet directement relié au nombre de composantes biconnexes de l'hypergraphe H. S'il n'y a qu'une composante biconnexe, l'algorithme BCC ne sera pas plus efficace que celui sur lequel il est greffé (son surcoût sera alors celui, linéaire, de la phase d'initialisation). Nous avons proposé dans [Baget and Tognetti, 2001] une méthode

permettant de généraliser BCC (dans sa version CSP) aux composantes k-connexes, lorsque l'on dispose d'une telle décomposition en composantes k-connexes.

Une généralisation de BCC aux composantes k-connexes, pour la projection de graphes élémentaires, nécessitera :

- de calculer une telle décomposition : il s'agit d'un problème NP-difficile dans sa version « exacte », mais l'objectif est de trouver une bonne heuristique permettant d'obtenir une décomposition en sous-ensembles k-connexes « les plus grands possibles », et « les plus équilibrés possibles » ;
- d'adapter l'algorithme BCC, qui ne gèrera plus, par exemple, une liste d'images interdites, mais des tableaux de taille $k-1$.

Enfin, à un hypergraphe peut être associé un grand nombre d'ordres BCC-compatibles des sommets, et il nous faudra étudier quels sont ceux qui nous permettront de trouver plus rapidement une solution. De plus, nous envisageons d'examiner les limites qu'il faut imposer aux méthodes d'ordonnancement dynamique des sommets (voir les travaux actuels dans le domaine des CSP) pour qu'elles restent compatibles avec BCC.

Deuxième partie

Autres modèles de la famille $\mathcal{SG}$

Note à l'attention des rapporteurs : Certaines preuves dans les deux chapitres suivants sont données en anglais, et sont un copier/coller de celles de [Baget and Mugnier, 2001a]. Une traduction en sera bien évidemment donnée dans la version définitive de cette thèse.

Chapitre 6

Le modèle $\mathcal{SR}$

Sommaire

Les règles de graphes conceptuels simples représentent des connaissances de la forme « Si ... alors ... ». Il s'agit cette fois-ci d'un réel enrichissement du modèle, puisque ces connaissances ne sont plus exprimables dans le fragment positif, conjonctif, existentiel de la logique du premier ordre. Bien que ne permettant pas d'exprimer la totalité de la logique du premier ordre représentée dans les graphes conceptuels de [Sowa, 1984], l'intérêt de ces règles en terme de représentation de connaissances a encouragé de nombreux auteurs (par exemple [Fargues et al., 1986, Rao and Foo, 1987, Gosh and Wuwongse, 1995]) à les étudier. Le modèle que nous adoptons ici est celui de [Salvat and Mugnier, 1996, Salvat, 1997], qui les premiers, ont proposé des algorithmes utilisant des opérations de graphes, adéquats et complets par rapport à la sémantique Φ de [Sowa, 1984].

La FIG. 6.1 (exemple tiré de [Salvat, 1997]) illustre trois façons équivalentes de représenter une règle de graphes conceptuels simples. La sémantique intuitive de cette règle est :

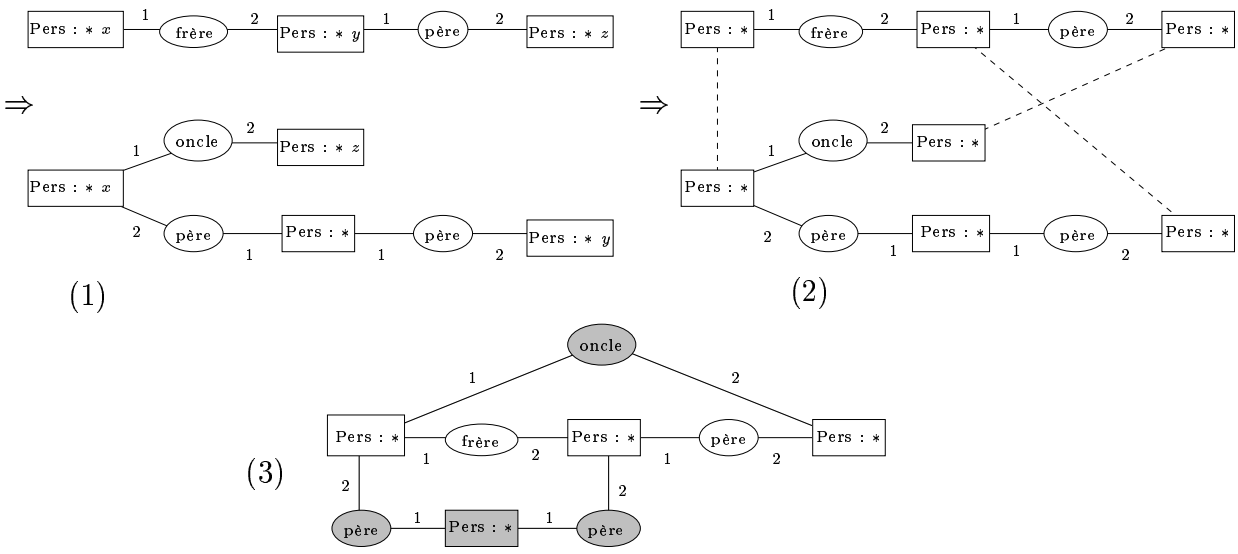


FIG. 6.1 – Trois représentations d'une même règle.

« *Si* (la personne) x est le frère de (la personne) y, et y est le père de z, *alors* x est l'oncle de z et il existe une personne qui est le père à la fois de x et de y ».

Les deux premières représentations (1 et 2) sont très similaires. L'*hypothèse* et la *conclusion* de la règle sont deux graphes distincts, identifiés par le symbole d'implication ($\Rightarrow$) qui les sépare. Afin de montrer quels sommets concepts représentent la même entité dans l'hypothèse et la conclusion, nous pouvons indiquer qu'ils sont co-référents, que ce soit en leur attribuant un même nom de variable (cas 1) ou en les reliant par des traits en pointillés (cas 2). Bien que, suivant [Sowa, 1984], ce type de représentation soit utilisé par une grande partie de la communauté « graphes conceptuels », nous avons adopté, depuis [Baget, 1998], une représentation différente, utilisant des *graphes colorés*. Avec cette représentation, nous ne manipulons qu'un seul graphe, dont les sommets concepts et les sommets/hyperarcs relations ont été colorés. Les sommets concepts et les relations de la première couleur (en blanc, dans le graphe 3) représentent l'hypothèse, et ceux de la deuxième couleur (en gris) représentent la conclusion. Grâce à cette représentation, il n'est pas nécessaire d'indiquer quels sommets représentent les mêmes entités. Nous trouvons à cette représentation équivalente les avantages suivants :

- nous pensons que cette représentation est plus facile à comprendre, puisque le lecteur n'a à faire aucun effort pour identifier quels sont les sommets concepts de la conclusion qui correspondent à des sommets concepts de l'hypothèse ;
- la co-référence (et les problèmes qu'elle pose sur les types) n'est pas nécessaire à la définition du modèle initial, et sera donc intégrée par la suite, comme une extension. Le modèle initial en sera simplifié d'autant.

Ce chapitre est consacré à l'étude de ces règles de graphes. Dans la Sect. 6.1, nous définirons formellement ces objets et le mécanisme, basé sur la projection, qui permet

d'appliquer une règle à un graphe, afin de l'enrichir par de nouvelles connaissances. Nous pouvons définir ce mécanisme indifféremment pour les graphes élémentaires ou les graphes conceptuels simples, munis ou non des extensions que nous avons exposées au Chap. 4. Ce mécanisme est à la base du problème de déduction dans le modèle $\mathcal{SR}$: étant donné un graphe initial G représentant des faits, et un ensemble $\mathcal{R}$ de règles (utilisant le même formalisme de graphes), est-il possible de construire, par une séquence d'application de règles de $\mathcal{R}$, un graphe G' qui contient une réponse à une requête H ? Nous préciserons également les liens entre déduction dans ce modèle et déduction en logique du premier ordre.

Dans la Sect. 6.2, nous prouvons que $\mathcal{SR}$-DÉDUCTION est un problème semi-décidable. Notre démonstration, différente de celle de [Coulondre and Salvat, 1998], a l'avantage d'établir que $\mathcal{SR}$ est un modèle de calcul : toute machine de Turing peut être simulée par un ensemble de règles. Nous montrerons à cette occasion la complexité inhérente à ce modèle de réécriture de graphes basé sur la projection, en prouvant que ce problème reste semi-décidable même en considérant des restrictions importantes sur les règles. Nous présentons toutefois deux cas décidables, dont l'un a été utilisé pour notre solution du problème SISYPHUS [Baget et al., 1999].

Enfin, dans la Sect. 6.3, nous proposons une amélioration du mécanisme de chaînage avant (celui utilisé dans les modèles plus généraux de la famille $\mathcal{SG}$), basée sur un prétraitement de l'ensemble de règles. Cette *compilation de la base de règles* repose sur la façon dont deux règles peuvent interagir entre elles, et le *graphe de dépendance des règles* que nous obtenons permet, d'une part, de répondre plus rapidement à une requête, et d'autre part, de définir des critères plus larges de décidabilité.

6.1 Règles de graphes

Nous commençons, dans la Sect. 6.1.1, par définir les règles et leur mécanisme d'application en nous basant sur le formalisme, à la fois plus simple et plus général, des graphes élémentaires. Nous introduisons à cette occasion la notion de *règle bien formée* pour un formalisme. Nous n'aurons plus, dans la Sect. 6.1.2, qu'à définir ce que sont les règles bien formées pour les graphes conceptuels simples et emboîtés, avec ou sans liens de co-référence. Enfin, dans la Sect. 6.1.3, nous rappelons l'extension de la sémantique Φ [Salvat, 1997] pour prendre en compte les règles de graphes.

6.1.1 Règles de graphes élémentaires

Comme les contraintes que nous étudierons au chapitre suivant, les règles sont représentées par des graphes colorés. Étant donné que le formalisme de base considéré pour la projection est celui des graphes élémentaires, ce sont aussi ces graphes que nous allons initialement colorer pour obtenir les règles.

Graphes colorés

Définition 6.1 (Graphe coloré) *Soit $\mathcal{S}$ un support. Nous appelons graphe (élémentaire) coloré sur $\mathcal{S}$ une paire (G, κ) où G est un graphe élémentaire* sans sommets concepts inutiles *sur $\mathcal{S}$, et κ est une application de $U(G)$ dans $\{0, 1\}$, associant à chaque relation sa couleur. La coloration des relations détermine celle des sommets concepts : pour tout sommet concept x nous notons $\kappa(x) = 1$ si toutes les relations incidentes à x sont colorées par 1, et $\kappa(x) = 0$ sinon.*

Si $R = (G, \kappa)$ est un graphe coloré sur $\mathcal{S}$, et $G = (V, U, \tau, \gamma)$ alors nous notons $R_{(0)}$ la structure obtenue en ne considérant que les éléments de V et de U auxquels κ associe la couleur 0, et la restriction de τ et γ à ces éléments. Dans les règles, $R_{(0)}$ correspondra à la partie *hypothèse*. Plus formellement, si nous notons $V_0(R) = V \cap \kappa^{-1}(0)$, et $U_0(R) = U \cap \kappa^{-1}(0)$, alors $R_{(0)} = (V_0(R), U_0(R), \tau|_{U_0(R)}, \gamma|_{U_0(R)})$. La structure $R_{(1)}$ obtenue en ne considérant que les éléments de V et de U auxquels κ associe la couleur 1 est définie de la même façon. Dans les règles, cette partie correspondra à la *conclusion*. Nous remarquons que $R_{(1)}$ n'est pas nécessairement un graphe élémentaire (car le sommet concept argument d'une relation de $U_1(R)$ peut être dans $V_0(R)$), mais que $R_{(0)}$ est toujours un graphe élémentaire : en effet, si $r \in U_O(R)$, alors cette couleur détermine la couleur 0 pour tous ses arguments, qui appartiennent donc à $V_0(R)$.

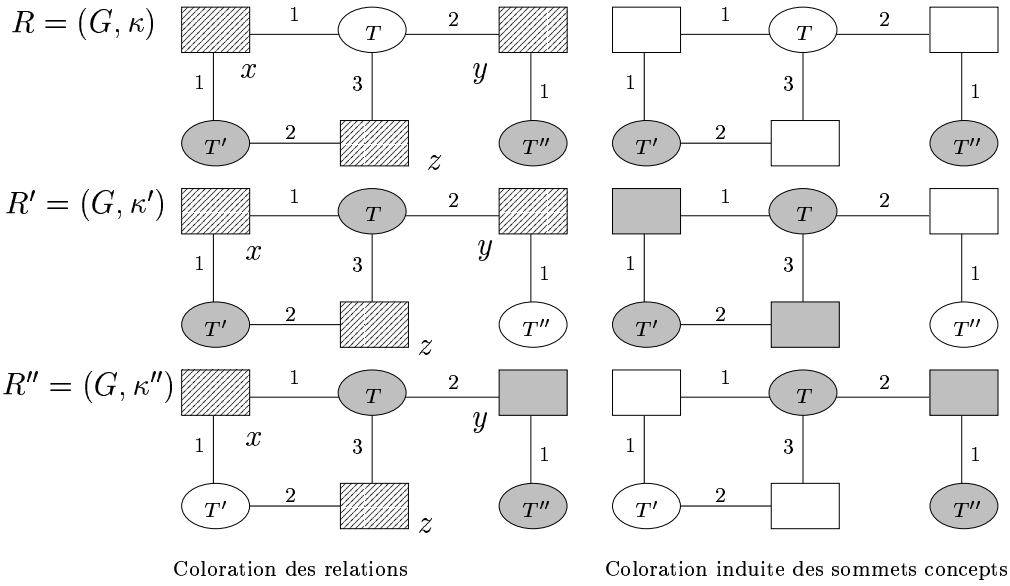


FIG. 6.2 – Colorations des sommets induites par différentes colorations des relations.

La FIG. 6.2 présente trois colorations différentes des sommets d'un même graphe élémentaire sans sommets concepts inutiles G. Les relations colorées par 0 sont représentées en blanc, et celles colorées par 1 sont représentées en gris. Cette convention sera utilisée tout au long de ce mémoire. Dans la première colonne, nous n'avons pas représenté la

couleur des sommets concepts, et cette absence de couleur est symbolisée par des sommets concepts hachurés. Dans la deuxième colonne, nous avons représenté la coloration implicite des sommets concepts. Bien qu'elle ne soit pas absolument nécessaire d'un point de vue formel (puisqu'elle est déterminée par la coloration des relations), nous ne négligerons pas son intérêt du point de vue de la lisibilité du graphe. En effet, les objets colorés en gris représentent une structure qui sera rajoutée à un graphe par l'application d'une règle, et la coloration des sommets concepts met cette structure en valeur.

Enfin, nous appelons *frontière* d'un graphe coloré l'ensemble des sommets concepts incidents à la fois à des relations de $U_0(R)$ et à des relations de $U_1(R)$. Dans l'exemple, la frontière de R est $\{x, y, z\}$, celle de R' est $\{y\}$, et celle de R'' est $\{x, z\}$. Remarquons que seuls des sommets concepts de $V_0(R)$ peuvent appartenir à la frontière d'un graphe coloré R.

La *notation prédicative* que nous avons adoptée pour les graphes élémentaires s'étend immédiatement aux graphes colorés. Nous représenterons à gauche d'un symbole de séparation, $\Rightarrow$, les relations colorées par 0, et à droite les relations colorées par 1. Nous pouvons donc représenter les graphes colorés de la FIG. 6.2 par :

$$\begin{aligned} R &= ([T(x,y,z)] \Rightarrow [T'(x,z), T''(y)]) \\ R' &= ([T''(y)] \Rightarrow [T(x,y,z), T'(x,z)]) \\ R'' &= ([T'(x,z)] \Rightarrow [T(x,y,z), T''(y)]) \end{aligned}$$

Pour tout graphe coloré R sur $\mathcal{S}$, nous avons vu que $R_{(0)}$ est un graphe élémentaire. Nous pouvons donc calculer des projections de $R_{(0)}$ dans un graphe élémentaire quelconque sur $\mathcal{S}$, ce qui est à la base de la notion d'*applicabilité* d'un graphe coloré. Cette notion d'applicabilité sera utilisée pour tous les graphes colorés (règles et contraintes) que nous utilisons dans ce mémoire.

Définition 6.2 (Applicabilité) *Soit G un graphe élémentaire sur $\mathcal{S}$, et R un graphe coloré sur $\mathcal{S}$. Alors R est dite* π-applicable *(ou simplement* applicable*, si seule l'existence de π nous intéresse) à G ssi π est une projection de $R_{(0)}$ dans G.*

Notons immédiatement qu'un graphe dont tous les sommets concepts et toutes les relations sont colorés par 1 est applicable à n'importe quel graphe. En effet, $R_{(0)}$ est dans ce cas le graphe vide $\emptyset$, qui se projette dans tous les graphes.

Règle et application d'une règle à un graphe

Une règle est une sorte de graphe coloré (au sens de l'héritage dans les langages de programmation objet).

Définition 6.3 (Règle) *Soit $\mathcal{S}$ un support. Une* règle élémentaire *R sur $\mathcal{S}$ est un graphe élémentaire coloré sur $\mathcal{S}$. Le graphe élémentaire $R_{(0)}$ est appelé l'*hypothèse *de R, et la structure $R_{(1)}$ (qui n'est pas nécessairement un graphe élémentaire) est appelée la* conclusion *de R.*

Si une règle R est π-applicable à un graphe G, alors la projection π de $R_{(0)}$ (l'*hypothèse* de la règle) dans G détermine de façon unique un graphe G' que nous pourrons construire en rajoutant à G les informations qui se trouvent dans $R_{(1)}$ (la *conclusion* de R).

Définition 6.4 (Application d'une règle à un graphe) *Soit $\mathcal{S}$ un support, G un graphe élémentaire sur $\mathcal{S}$, et R une règle élémentaire sur $\mathcal{S}$. Alors si R est π-applicable à G, nous notons $G' = \lambda(G, R, \pi)$ le graphe élémentaire obtenu de la façon suivante :*

- *$V(G') = V(G) \cup V_1(R)$ et $U(G') = U(G) \cup U_1(R)$;*
- *si $r \in U(G)$, alors son type $\tau_{G'}(r) = \tau_G(r)$, sinon $\tau_{G'}(r) = \tau_R(r)$;*
- *si $r \in U(G)$, alors $\gamma_{G'}(r) = \gamma_G(r)$, sinon, pour $1 \leq i \leq |r|$, soit $x_i = (\gamma_R)_i(r)$ est un argument de r dans R, auquel cas $(\gamma_{G'})_i(r) = x_i$ soit $x_i \in R_{(1)}$, et $(\gamma_{G'})_i(r) = \pi(x_i)$.*

En d'autres termes, le graphe élémentaire G' obtenu par l'application de R à G suivant la projection π est construit à partir d'une copie de G, d'une copie de $R_{(1)}$, et pour chaque argument d'une relation de $R_{(1)}$ qui est dans l'hypothèse de R, en remplaçant cet argument par (la copie de) sa projection par π dans G.

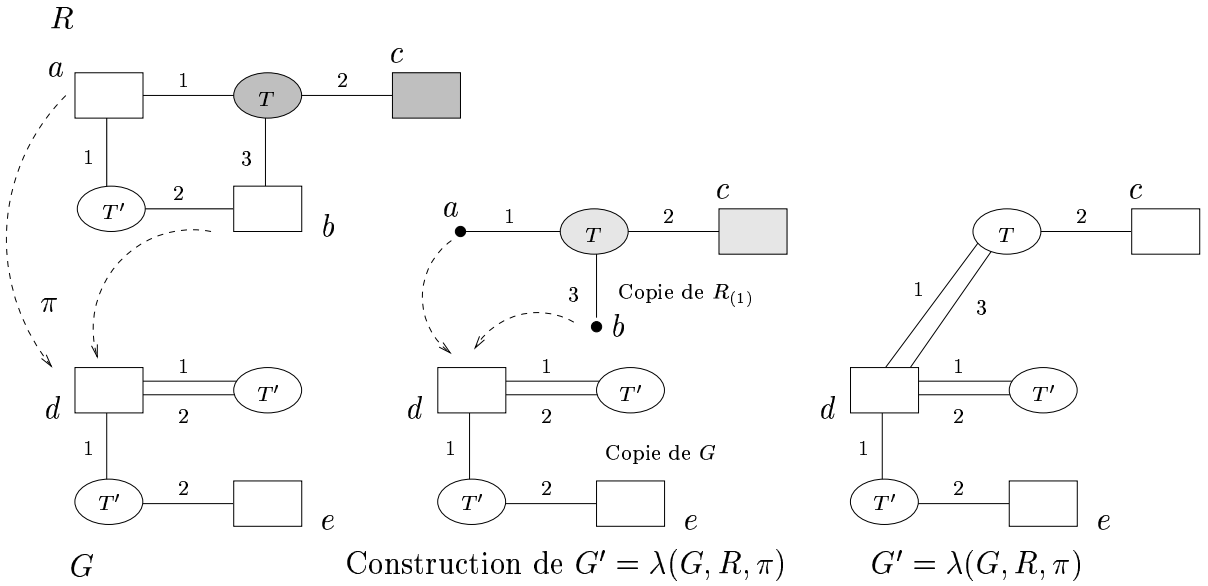


FIG. 6.3 – Application d'une règle R à un graphe G et construction de $G' = \lambda(G, R, \pi)$.

La FIG. 6.3 illustre le mécanisme d'application d'une règle. R est une règle élémentaire sur $\mathcal{S}$, G est un graphe élémentaire sur $\mathcal{S}$, et π est une projection de $R_{(0)}$ dans G (définie par $\pi(a) = \pi(b) = d$). Le graphe $G' = \lambda(G, R, \pi)$ est construit à partir d'une copie de G, d'une copie de $R_{(1)}$ (qui n'est pas un graphe élémentaire car les arguments 1 et 3 de sa relation n'existent pas dans $R_{(1)}$), puis en remplaçant les « arguments manquants » de la relation de $R_{(1)}$ par les sommets de la copie de G qui correspondent à la projection de ces arguments de $R_{(0)}$.

Le mécanisme de dérivation, et le problème de déduction

Nous considérons maintenant G un graphe de $\mathcal{G}^e(\mathcal{S})$ et un ensemble $\mathcal{R}$ de règles élémentaires sur $\mathcal{S}$. Nous disons que $G' \in \mathcal{G}^e(\mathcal{S})$ est une $\mathcal{R}$-*dérivation immédiate* de G s'il existe une règle élémentaire $R \in \mathcal{R}$ et une projection π de l'hypothèse de R dans G telles que $G' = \lambda(G, R, \pi)$. Dans ce cas, nous notons $G \rightsquigarrow G'$ (comme le support dans la projection, l'ensemble de règles est ici implicite). Comme dans tout système de réécriture, la relation de *dérivation* est la fermeture réflexo-transitive de la relation de dérivation immédiate. Nous dirons qu'un graphe G' est une $\mathcal{R}$-*dérivation* de G si $G' = G$ ou s'il existe une suite $G = G_0, G_1, \ldots, G_k = G'$ de graphes élémentaires de $\mathcal{G}^e(\mathcal{S})$ telle que, pour $1 \leq i \leq k$, G_i est une $\mathcal{R}$-dérivation immédiate de G_{i-1}. Nous notons dans ce cas $G \overset{*}{\rightsquigarrow} G'$. Nous dirons aussi que G dérive G'. Le nombre k, qui peut être nul, est la *longueur* de la dérivation.

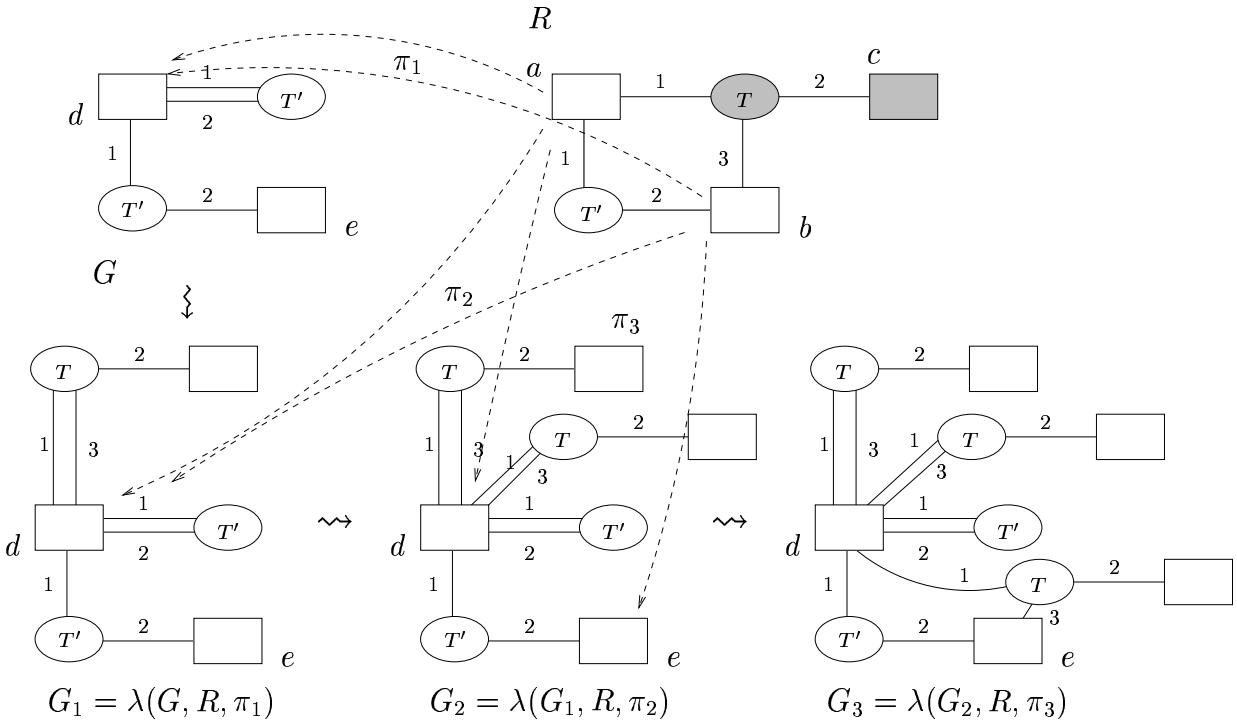


FIG. 6.4 – G dérive G_3 : une séquence de dérivations immédiates $G \rightsquigarrow G_1 \rightsquigarrow G_2 \rightsquigarrow G_3$.

La FIG. 6.4 montre une séquence de dérivations immédiates $G \rightsquigarrow G_1 \rightsquigarrow G_2 \rightsquigarrow G_3$. Les graphes G, G_1, G_2, G_3 sont donc tous des $\{R\}$-dérivations de G. Remarquons, à propos de cet exemple, que si une règle élémentaire R est π-applicable à un graphe G, alors il existe une dérivation de longueur arbitrairement grande $G = G_0 \rightsquigarrow G_1 \rightsquigarrow \cdots \rightsquigarrow G_k \ldots$, telle que, pour $1 \leq i \leq k$, $G_i = \lambda(G_{i-1}, R, \pi)$. Nous avons donné dans l'exemple le début d'une telle dérivation en construisant G_1 et G_2 à partir de la même projection. Toutefois, tous les graphes $G_1, G_2, \ldots, G_k, \ldots$ obtenus en utilisant la même projection de la même règle sont équivalents : nous pourrons donc, sans perte de généralité, ne pas considérer ces applications multiples. Notons aussi que, si des applications multiples d'une même règle

suivant une même projection créent de la redondance, les interdire ne suffit pas à l'éviter (c'est à dire à assurer qu'à partir d'un graphe irredondant on ne dérive que des graphes irredondants).

Nous pouvons maintenant définir le problème de déduction dans le modèle $\mathcal{SR}$: étant donnés une base de faits $\mathcal{G}$ constituée de l'union disjointe de plusieurs graphes élémentaires et un ensemble de règles $\mathcal{R}$, peut-on trouver une dérivation G' de $\mathcal{G}$ dans laquelle se projette une requête H ?

Définition 6.5 ($\mathcal{SR}$-déduction) *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R})$ une base de connaissances constituée d'un support $\mathcal{S}$, d'un ensemble $\mathcal{G}$ de graphes élémentaires de $\mathcal{G}^e(\mathcal{S})$, et d'un ensemble $\mathcal{R}$ de règles élémentaires sur $\mathcal{S}$. Alors un graphe élémentaire $H \in \mathcal{G}^e(\mathcal{S})$ est dit* déductible *de $\mathcal{K}$ si il existe un graphe G' $\mathcal{R}$-dérivable de $\mathcal{G}$ (où nous considérons $\mathcal{G}$ comme le graphe élémentaire qui est l'union disjointe de ses éléments) tel que $G' \preceq H$.*

6.1.2 Règles de graphes conceptuels ...

Nous avons vu (dans le Chap. 4) que la projection dans les formalismes des graphes conceptuels simples ou emboîtés, avec ou sans co-référence ou conjonction de types, pouvait être considérée comme la projection d'une classe particulière de graphes élémentaires. Nous pouvons donc utiliser les règles de graphes élémentaires pour implanter des règles dans ces différents formalismes. Cette implantation reposera sur la notion de règle élémentaire *bien formée* pour le mécanisme d'application de règle dans un formalisme donné.

Définition 6.6 (Règles bien formées) *Soit $\mathcal{G}(\mathcal{S})$ un formalisme de graphes exprimable dans $\mathcal{G}^e(\mathcal{S}')$. Alors nous disons qu'une règle R sur $\mathcal{S}'$ est bien formée pour $\mathcal{G}(\mathcal{S})$ et le mécanisme d'application λ si, pour tout graphe $G \in \mathcal{G}(\mathcal{S})$, pour toute projection π de l'hypothèse de R dans $\mathrm{elem}(G)$, le graphe élémentaire $\lambda(\mathrm{elem}(G), R, \pi)$ correspond à un graphe de $\mathcal{G}(\mathcal{S})$.*

Pour chacun des formalismes que nous avons présenté dans la Sect. 4, nous pouvons déterminer les règles bien formées pour ce formalisme et définir un mécanisme d'application λ.

Note aux rapporteurs : Cette section n'est visiblement pas terminée... Je pense que ceci ne devrait pas poser de problème pour lire la suite de ce document. Les règles de graphes élémentaires étant équivalentes à toutes les autres, nous pouvons ne considérer que celles-ci.

6.1.3 Extension de la sémantique logique Φ

Les différentes sémantiques logiques que nous avons présentées au Chap. 3 et au Chap. 4 peuvent être étendues pour traduire les règles. Notons que si ces sémantiques logiques nécessitent, pour leur complétude dans le modèle $\mathcal{SG}$, la considération de la forme normale du graphe G, il nous faudra considérer dans le modèle $\mathcal{SR}$ un mécanisme d'application

de règle λ' qui met chaque graphe obtenu par le mécanisme d'application standard λ sous forme normale.

Nous rappellons ici comment cette extension est réalisée pour la sémantique traditionnelle Φ des graphes élémentaires :

- l'interprétation $\Phi(\mathcal{S})$ du support ne change pas ;
- l'interprétation $\Phi(G)$ d'un graphe conceptuel simple ne change pas ;
- une règle $R = (G, \kappa)$ est interprétée par la formule

$$\Phi(R) = \forall x_1 \ldots \forall x_p(\phi(R_{(0)}) \rightarrow \exists y_1 \ldots \exists y_q \phi(R_(1))$$

  construite de la façon suivante :
  - $\phi(R_0)$ est la conjonction des atomes associés par Φ aux sommets de $R_{(0)}$;
  - $\phi(R_1)$ est la conjonction des atomes associés par Φ aux sommets de $R_{(1)}$;
  - $x_1, \ldots, x_p$ sont les variables apparaissant dans la formule $\phi(R_0)$;
  - $y_1, \ldots, x_q$ sont les variables apparaissant dans la formule $\phi(R_1)$ mais pas dans la formule $\phi(R_0)$.
- la traduction d'un ensemble de règles est la conjonction des interprétations de chaque règle.

Par exemple, la règle R présentée dans la FIG. 6.1 est interprétée par :

$$\exists x \exists y \exists z((\text{Pers}(x) \wedge \text{Pers}(y) \wedge \text{Pers}(z) \wedge \text{frère}(x, y) \wedge \text{père}(y, z))$$
$$\rightarrow (\exists u(\text{Pers}(u) \wedge \text{oncle}(x, z) \wedge \text{père}(u, x) \wedge \text{père}(u, y))))$$

Remarquons que, contrairement aux clauses, les variables de la conclusion sont quantifiées existentiellement. En fait, les formules associées aux règles sont celles correspondant aux TGDs (*tuple generating dependencies*) en bases de données [Coulondre and Salvat, 1998].

Théorème 6.1 (Adéquation et complétude) *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R})$ une base de connaissances, et H une requête. Alors H est déductible de $\mathcal{K}$ si et seulement si $\Phi(\mathcal{S}), \Phi(\mathcal{G}), \Phi(\mathcal{R}) \models \Phi(H)$.*

Ce théorème est démontré dans [Salvat, 1997]. Nous pouvons considérer le mécanisme d'application λ si nous utilisons des graphes élémentaires ou des hraphes conceptuels très simples. Pour les graphes conceptuels simples ou emboîtés, il nous faudra considérer (si nous tenons à la complétude), un mécanisme d'application de règles qui met le graphe produit sous sa forme normale.

6.2 Problèmes de décidabilité

Nous nous intéressons ici à la décidabilité du problème $\mathcal{SR}$-DÉDUCTION, que nous considérerons, sans perte de généralité, dans le formalisme des graphes élémentaires. La question centrale est de savoir s'il existe un algorithme $\mathcal{A}$ qui, pour toute instance $(\mathcal{K}, H)$ du problème $\mathcal{SR}$-DÉDUCTION répond :

1. OUI, en un temps fini, si et seulement si H est déductible de $\mathcal{K}$;
2. NON, en un temps fini, si et seulement si H n'est pas déductible de $\mathcal{K}$.

Si un algorithme $\mathcal{A}$ sait répondre à 1. et 2., alors le problème considéré est dit *décidable*. Si un algorithme peut répondre à 1., mais aucun ne peut répondre à 2., le problème sera dit *semi-décidable*. Si aucun algorithme ne peut répondre à 1., ni à 2., alors le problème est *indécidable*. Enfin, nous appellerons *co-semi-décidable* un problème dont le co-problème est semi-décidable.

6.2.1 Un modèle de calcul

Nous montrons ici (Sect. 6.2.1), que le problème $\mathcal{SR}$-DÉDUCTION est semi-décidable, en utilisant une démonstration différente de celle de [Coulondre and Salvat, 1998]. Notre démonstration, en explicitant les relations entre $\mathcal{SR}$-DÉDUCTION et les Machines de Turing, montre que $\mathcal{SR}$-DÉDUCTION est un modèle de calcul. Nous montrons ensuite que la complexité induite par la structure du graphe rend inopérantes des restrictions pourtant très importantes (réduire le nombre de règles, réduire le nombre de symboles, considérer des structures particulières du graphe) : le problème est encore semi-décidable. Nous examinerons ensuite un cas décidable, intéressant dans les modèles plus généraux de la famille $\mathcal{SG}$ (Chap. 7) puisque la restriction à ce type de règles sera suffisante pour plonger les problèmes de déduction associés à tous les modèles de cette famille dans la hiérarchie polynomiale.

Semi-décidabilité de $\mathcal{SR}$-DÉDUCTION

Théorème 6.2 *Le problème $\mathcal{SR}$-DÉDUCTION est semi-décidable.*

Ce résultat est prouvé dans [Coulondre and Salvat, 1998], en établissant l'équivalence entre la déduction utilisant des règles de graphes conceptuels simples et les TGDs (*Tuple Generating Dependencies*) en bases de données.

Preuve: Nous prouverons ce résultat en montrant :

1. qu'il existe un algorithme adéquat et complet pour $\mathcal{SR}$-DÉDUCTION (mais nous n'exposerons l'algorithme de résolution en marche avant de [Salvat and Mugnier, 1996] qu'à la Sect. 6.3.1) ;
2. que nous pouvons réduire le PROBLÈME DE L'ARRÊT d'une machine de Turing, qui est semi-décidable, à la $\mathcal{SR}$-DÉDUCTION.

Nous ne démontrerons donc à cette étape de notre présentation que le point 2., et commençons par un (très bref) rappel sur ce problème de l'arrêt d'une Machine de Turing.

Le problème de l'arrêt d'une Machine de Turing Une machine de Turing (MdT) est définie par un alphabet Σ, un ensemble de symboles d'état $\mathcal{T}$, comprenant deux symboles particuliers, l'état initial T_i et l'état final T_f, et un ensemble de règles de la forme $(T_j, x) \rightarrow (T_k, y, \delta)$. Dans chacune de ces règles, T_j et T_k sont des symboles d'état, x et y sont

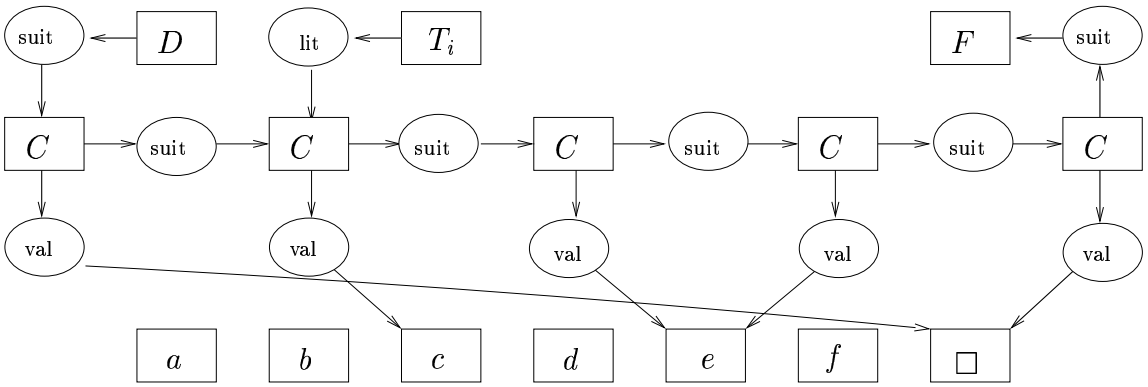
des lettres de Σ ou le symbole particulier $\Box$ (qui représente une case vide), et $\delta \in \{\text{droite}, \text{gauche}\}$. Nous considérons un ruban, infini des deux côtés, dont chaque case est remplie par le symbole vide $\Box$, inscrivons le mot m construit sur $\Sigma \cup \{\Box\}$ au milieu de ce ruban, et faisons pointer une tête de lecture dans l'état initial T_i sur la première lettre de m. Une règle $(T_j, x) \rightarrow (T_k, y, \delta)$ est applicable si la tête de lecture est dans l'état T_j et pointe sur la lettre x. Dans ce cas, l'application de la règle consiste à remplacer la lettre x par la lettre y, à faire passer la tête de lecture dans l'état T_k, et à la déplacer d'une case vers la droite ou vers la gauche, suivant la valeur de δ. Le PROBLÈME DE L'ARRÊT consiste à savoir, étant donné une machine de Turing $\mathcal{M}$ et un mot m, s'il existe une séquence d'applications de règles telle que la tête de lecture se retrouve dans l'état terminal. Ce problème est semi-décidable [Turing, 1937].

Transformation d'une Machine de Turing en instance de $\mathcal{SR}$-DÉDUCTION Nous allons maintenant exposer une transformation (calculable) Θ qui associe à toute instance $(\mathcal{M}, m)$ du PROBLÈME DE L'ARRÊT une instance $(\mathcal{K}, H)$ de $\mathcal{SR}$-DÉDUCTION telle que $\mathcal{M}$ s'arrête sur le mot m si et seulement si H est déductible de $\mathcal{K}$ (nous aurons ainsi prouvé que $\mathcal{SR}$-DÉDUCTION est semi-décidable).

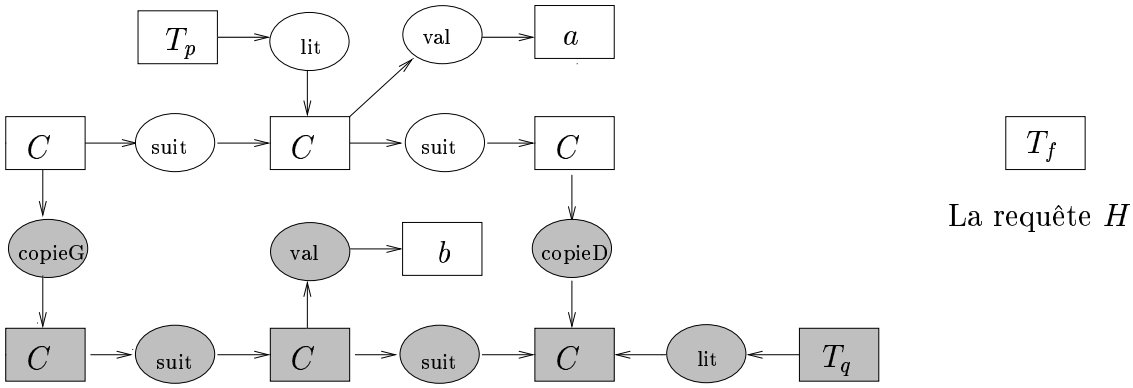
Construisons tout d'abord le support $\mathcal{S}$ pour la base de connaissances $\mathcal{K}$. Les types de concepts (ou types de relations d'arité 1) sont : C (type utilisé pour les cases du ruban infini), $T_1, \ldots, T_p$ (un type par état possible de la tête de lecture), $a, b, c, d, \ldots$ (un type par lettre de Σ), $\Box$ (représentant la case vide) et D et F (utilisés pour indiquer le début et la fin du mot initial). Les types de relation binaires sont *suit* (indique l'ordre sur les cases), *val* (une case est valuée par une lettre), *lit* (la tête de lecture lit/pointe sur une case), *copieD* (utilisé pour la reconstruction de la partie droite du mot), et *copieG* (utilisé pour la reconstruction de la partie gauche du mot). Tous ces types sont deux à deux incomparables.

Construisons le graphe G qui représentera les faits de la base de connaissances. Nous le construisons à partir du mot m et de l'alphabet Σ de la façon suivante, comme représenté dans la FIG. 6.5.

- on se donne d'une part un ensemble de $|\Sigma| + 3$ sommets concepts : trois sont typés respectivement par D, F et $\Box$, les autres par les types associés aux lettres de Σ (nous avons pris dans l'exemple $\Sigma = \{a, b, c, d, e, f\}$) ;
- d'autre part, on se donne un ensemble de $|m| + 4$ sommets concepts représentant des cases du ruban, deux sont typés par D et F (début et fin), les autres typés par C, $|m|+3$ relations binaires typées *suit* les relient par un chemin qui indiquera l'ordre des lettres du mot. Le premier (dans l'ordre indiqué par le chemin) est celui typé par D, le dernier celui typé par F. Le second et l'avant-dernier sont reliés par des relations binaires typées *val* au sommet concept typé $\Box$ (la case vide) ; et pour $1 \leq i \leq |m|$, le sommet concept à la $(i+3)$ième place est relié au sommet concept typé par la $(i+1)$ième lettre de m.
- Enfin, un sommet concept typé par T_i (la tête de lecture dans l'état initial) est relié par une relation binaire typée *lit* au sommet concept représentant la première lettre de m.

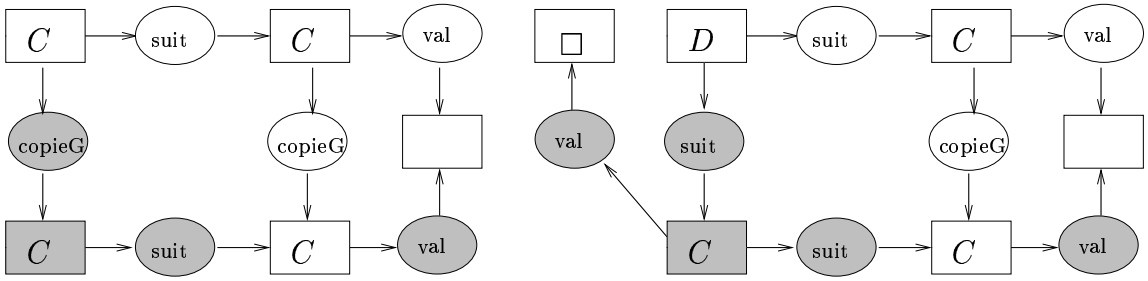


Transformation du mot initial *cee* en un graphe élémentaire G

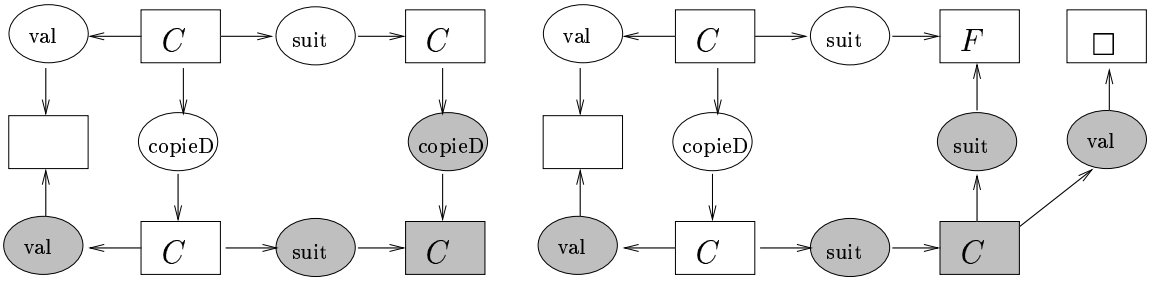


La requête H

Transformation d'une règle $(T_p, a) \to (T_q, b, \text{droite})$ en une règle R



G_1 Les deux règles de recopie à gauche G_2



D_1 Les deux règles de recopie à droite D_2

FIG. 6.5 – Transformation du PROBLÈME DE L'ARRÊT d'une MdT en $\mathcal{SR}$-DÉDUCTION.

A chaque règle $(T_p, x) \rightarrow (T_q, y, \delta)$ de la machine de Turing correspond une règle élémentaire R qui sera construite comme représenté dans la FIG. 6.5. Suivant la valeur de δ (*droite* ou *gauche*), le sommet concept typé par T_q sera relié par *lit* au sommet concept suivant celui que nous avons réécrit (typé par y) ou au sommet concept précédent. Quatre autres règles (les deux règles de recopie à gauche, et les deux règles de recopie à droite) appartiennent aussi à $\mathcal{R}$. Celles-ci auront un rôle important. En effet, contrairement au mécanisme de réécriture des MdT, celui de $\mathcal{SR}$-DÉDUCTION ne supprime/modifie aucun sommet/arc dans un graphe, et ne fait qu'ajouter de l'information. Pour simuler la modification, il faudra donc créer un nouveau sommet et recopier autour de lui tout ce qui n'a pas été modifié. Nous précisons aussi que, dans ces règles, ne pas typer un sommet concept est équivalent à le typer par $\top$, élément maximum de T_1.

Enfin, la requête H est simplement un sommet concept étiqueté par le type associé à l'état terminal T_f.

Démonstration du théorème Ayant défini la transformation $\Theta : (\mathcal{M}, m) \mapsto ((\mathcal{S}, \{G\}, \mathcal{R}), H)$, nous devons montrer que $\mathcal{M}$ s'arrête sur le mot m si et seulement si H est déductible de $(\mathcal{S}, \{G\}, \mathcal{R})$.

($\Rightarrow$) Par récurrence sur le nombre n de réécritures (applications de règles de la MdT) avant l'arrêt de la MdT. Notre hypothèse de récurrence est :

(H_n) *« s'il existe une dérivation de longueur n telle que la tête de lecture de la MdT se retrouve dans un état T_q, et lisant la lettre x d'un mot m, alors il existe une dérivation G' de G telle que le chemin associé à un mot $D\ \square^+\ m\ \square^+\ F$ se lit dans G', et un sommet concept associé à la tête de lecture dans l'état T_q est relié par lit au sommet de ce chemin qui correspond à la lettre x. »*

Cette hypothèse de récurrence étant trivialement vérifiée au rang 0, nous supposons maintenant qu'elle est vraie au rang k. Montrons qu'elle est toujours vraie au rang $k+1$. S'il existe une dérivation de longueur $k+1$ telle que la tête de lecture se retrouve, après s'être déplacée à droite, dans un état T_q, lisant la lettre x d'un mot m, alors $m = m'yxm''$, et il existe une règle $(T_p, z) \rightarrow (T_q, y, \textit{droite})$ et une dérivation de longueur k telle que la tête de lecture se retrouve dans un état T_p, lisant la lettre z d'un mot $m'zxm''$ (la démonstration est symétrique pour la tête de lecture se déplaçant à gauche). Appliquons l'hypothèse de récurrence au rang k. La FIG. 6.6 illustre les applications de règles élémentaires nécessaires pour la vérifier au rang $k+1$. Nous ne montrons ici que le résultat de l'application des règles de recopie à droite, les règles de recopie à gauche s'appliquant de façon symétrique. Le seul point délicat est de vérifier l'applicabilité de la règle élémentaire associée à une règle de MdT. Il faut pour cela vérifier qu'il y a un sommet concept typé par C « à droite » et « à gauche » du sommet concept pointé par la tête de lecture. Ceci est assuré par le fonctionnement des règles D_2 et G_2 qui « rajoutent une case vide » avant de relier le mot courant à F et D.

($\Leftarrow$) Par récurrence sur le nombre de règles correspondant aux règles de MdT utilisées au cours de la dérivation (le nombre de d'pplications de règles de graphes permettant de répondre à la requête H).

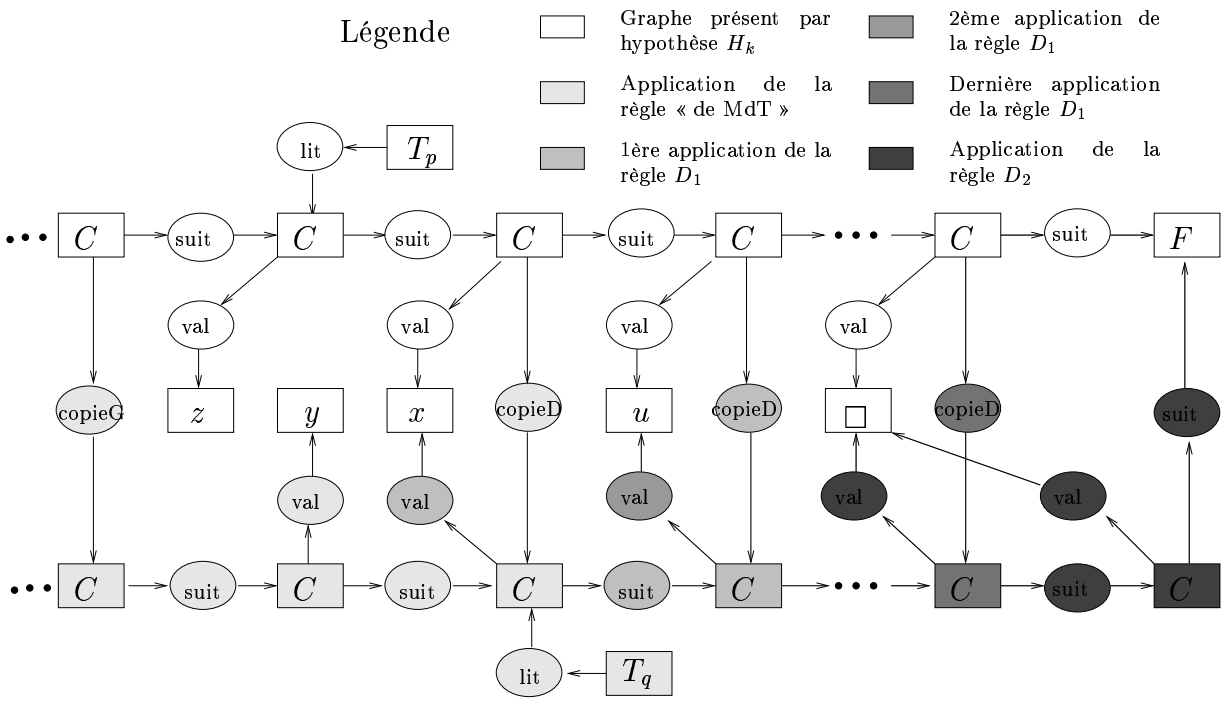


FIG. 6.6 – Recopie à droite du mot : l'hypothèse de récurrence est vérifiée au rang $k+1$.

Nous affirmons tout d'abord que, s'il existe une dérivation de longueur n menant à un G_n dans lequel se projette H, alors il existe une dérivation de longueur $n' > n$, $\mathcal{G} = G_0, G_1, \ldots, G_n, \ldots, G_{n'}$ telle que le graphe $G_{n'}$ contient une représentation de la tête de lecture dans l'état terminal, pointant sur une lettre a d'un mot *complet* (*i.e.* a value une case de type C, et cette case appartient à un chemin $[D] \rightarrow (suit) \rightarrow [C] \rightarrow (suit) \rightarrow \cdots \rightarrow (suit)[C] \rightarrow (suit) \rightarrow [F]$). Ceci se prouve immédiatement, par deux récurrences emboîtées, la première sur la longueur de la dérivation et la deuxième sur le nombre d'applications de règles de recopie à droite et de règles de recopie à gauche à chaque étape.

Notre hypothèse de récurrence peut donc s'écrire :

(H'_n) *« s'il existe une dérivation utilisant n règles correspondant à des règles de MdT, répondant à la requête H, tel que le sommet image de cette requête soit relié par une relation de type lit à un mot complet m, alors il existe une dérivation de la Machine de Turing représentée menant à l'arrêt sur un mot correspondant au mot complet m, pointant sur la même lettre.*

L'hypothèse de récurrence est trivialement vérifiée au rang 0. Supposons qu'elle soit vraie au rang k, et montrons qu'elle est vraie au rang $k+1$. Soit une dérivation $\mathcal{G} = G_0, \ldots, G_l$ utilisant $k+1$ règles correspondant à des règles de MdT, telle que H se projette dans G_l, et que la tête de lecture représentée par H soit reliée par une relation de type *lit* à un mot complet m. Considérons G_q ($q \leq l$) le premier de ces graphes dans lequel se projette H. G_q a nécessairement été obtenu par une règle R (à droite ou à gauche) représentant une règle de MdT. Soit T_p l'état de cette règle. Nous avons donc G_{q-1} un graphe obtenu

en utilisant k règles correspondant à des règles de MdT. D'après la propriété précédente, nous pouvons construire un graphe G'_q obtenu par une dérivation de G_{q-1} où T_p pointe sur une lettre a d'un mot complet, en n'utilisant que des règles de recopie. Nous pouvons donc appliquer l'hypothèse de récurrence (quitte à considérer temporairement T_p comme un état terminal) : il existe une dérivation de la MdT menant à une tête de lecture dans l'état T_p, pointant sur la lettre a du mot m. Alors il suffit d'appliquer la règle de MdT traduite par R pour obtenir un état terminal. □

Discussion

Cette démonstration présente plusieurs intérêts :

- Tout d'abord, nous montrons que la semi-décidabilité de $\mathcal{SR}$-DÉDUCTION n'est pas seulement un problème, un « inconvénient » algorithmique. En montrant que toute MdT peut être simulée dans le modèle $\mathcal{SR}$, nous mettons en valeur l'*expressivité* de ce modèle. Si l'on en croit la thèse de Church-Turing, tout problème pour lequel il existe une « procédure effective » de résolution pourra donc être implémenté dans le modèle $\mathcal{SR}$ (d'où le Cor. 6.1)
- De plus, la démonstration nous donne une idée du coût de cette simulation. Supposons qu'une MdT déterministe résolve un problème avec une complexité en temps $\Theta(t(n))$, et une complexité en espace $\Theta(e(n))$ (où n est la taille du mot initial). Nécessairement, la complexité en espace est bornée par la complexité en temps. Alors un algorithme de résolution en marche avant pour $\mathcal{SR}$-DÉDUCTION aura une complexité en temps en $\Theta((t(n))^3)$ et une complexité en espace en $\Theta((t(n))^2)$. La complexité en temps est polynomialement reliée à la complexité théorique calculée d'après le nombre d'opérations d'une MdT, mais la complexité en espace ne l'est pas. Ce problème vient en partie de l'impossibilité d'affirmer, dans $\mathcal{SR}$, qu'une branche de la dérivation ne mène nulle part, et peut être abandonnée. Certains modèles plus généraux de la famille $\mathcal{SG}$ ($\mathcal{SEC}$ et $\mathcal{SREC}$) permettent de résoudre cet aspect du problème.

Corollaire 6.1 $\mathcal{SR}$-DÉDUCTION *est un modèle de calcul.*

6.2.2 Machine de Turing universelle

La preuve de la semi-décidabilité du PROBLÈME DE L'ARRÊT est basée sur la notion de Machine de Turing universelle. Il est possible de construire :

- un codage f, qui associe à tout couple $(\mathcal{M}, m)$ constitué d'une MdT $\mathcal{M}$ et d'un mot m, un mot $m' = f(\mathcal{M}, m)$
- une machine de Turing $\mathcal{U}$ (la MdT universelle)

tels que, pour toute machine $\mathcal{M}$, pour tout mot m, $\mathcal{M}$ s'arrête sur m si et seulement si $\mathcal{U}$ s'arrête sur $f(\mathcal{M}, m)$. On montre (par un argument de diagonalisation) que la MdT universelle ne s'arrête pas sur elle-même.

Machine $\mathcal{SR}$ universelle

Par analogie, nous appellerons *machine $\mathcal{SR}$ universelle* la donnée :
- d'un support $\mathcal{S}_U$
- d'un ensemble de règles $\mathcal{R}_U$
- d'un codage g, qui associe à tout couple $(\mathcal{M}, m)$ constitué d'une MdT $\mathcal{M}$ et d'un mot m, un graphe élémentaire $G = g(\mathcal{M}, m) \in \mathcal{G}^e(\mathcal{S})$

tels que, pour toute machine $\mathcal{M}$, pour tout mot m, $\mathcal{M}$ s'arrête sur m si et seulement si le graphe constitué du sommet concept typé T_f est déductible de $(\mathcal{S}_U, g(\mathcal{M}, m), \mathcal{R}_U)$. Il est immédiat de construire une machine $\mathcal{SR}$ universelle à partir d'une MdT universelle $\mathcal{U}$ (nous la noterons $SR(\mathcal{U})$), à l'aide de la transformation Θ utilisée dans la preuve du Th. 6.2 :
- $\mathcal{S}_U$ est le support obtenu par $\Theta(\mathcal{U})$
- $\mathcal{R}_U$ est l'ensemble de règles obtenues par $\Theta(\mathcal{U})$
- le codage g associe à tout couple $(\mathcal{M}, m)$ le graphe obtenu par l'application de Θ sur le mot $f(\mathcal{M}, m)$

Lorsque l'on considérera des restrictions du modèle $\mathcal{SR}$, la possibilité de construire tout de même une machine $\mathcal{SR}$ universelle suffira à prouver que le modèle reste, heureusement, aussi expressif, et, malheureusement, semi-décidable.

Une méthode standard pour réduire l'expressivité des MdT est de restreindre le nombre de symboles d'états et la taille de l'alphabet disponibles pour construire les règles. La question qui se pose est : si on se limite à p symboles d'état, et q lettres distinctes, peut-on encore construire une machine de Turing universelle ?

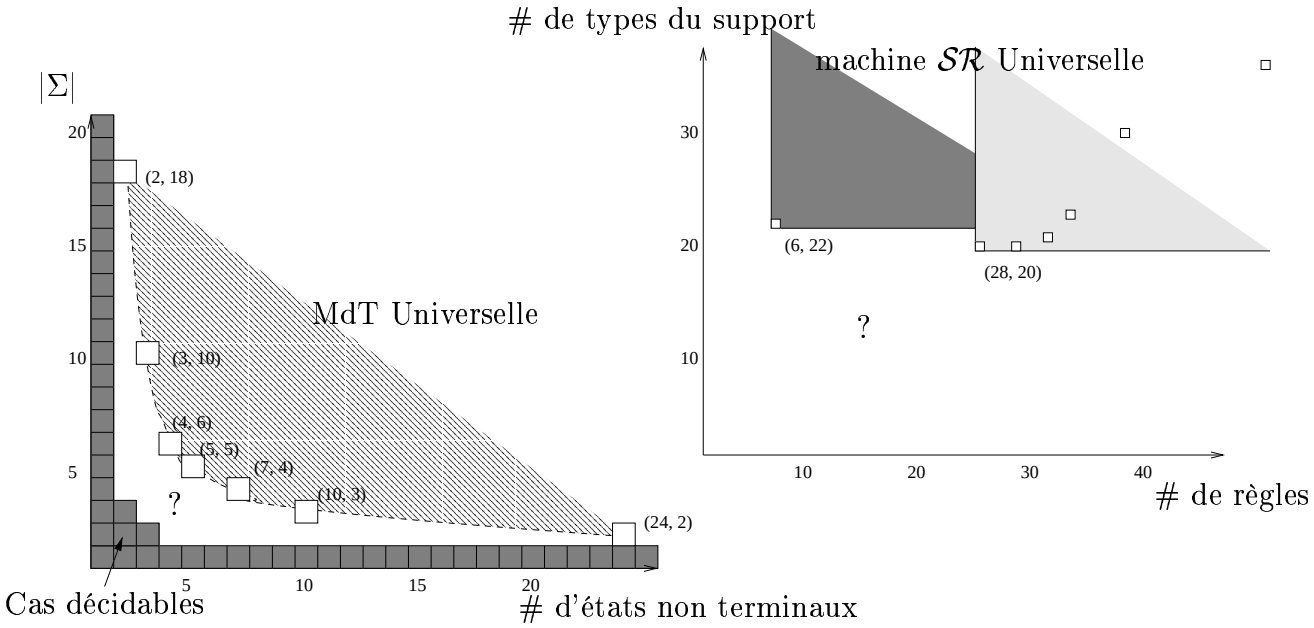


FIG. 6.7 – De la MdT universelle à la machine $\mathcal{SR}$ universelle.

Le dessin à gauche de la FIG. 6.7, très inspiré d'un schéma de [Gruska, 1997], représente les résultats exposés par [Rogozhin, 1996] pour tenter de répondre à cette question. Un carré blanc, étiqueté par (p, q) signifie qu'il existe une MdT universelle(déterministe) à p symboles d'état, et q lettres distinctes. Au dessus et à gauche de ces carrés, la région hachurée est donc une zone dans laquelle le problème est semi-décidable. En grisé, nous avons représenté la zone pour laquelle le problème est décidable : on ne peut donc y trouver de MdT universelle. Entre les frontières de ces deux zones, le problème reste ouvert.

Dans le modèle $\mathcal{SR}$, nous considérerons, par analogie avec ces travaux, des restrictions sur le nombre de règles et sur la taille du support. Nous obtiendrons une première frontière (représentée en gris clair dans le dessin à droite de la FIG. 6.7), simplement en traduisant les résultats de [Rogozhin, 1996] par l'intermédiaire de la transformation SR. En effet, si nous partons d'une MdT universelle déterministe $\mathcal{U}$ de taille (p, q), son nombre de règles est majoré par $p \times q$. Nous obtenons alors une machine $\mathcal{SR}$ universelle $SR(\mathcal{U})$ ayant $(p \times q) + 4$ règles, et dont le support est de taille $p + q + 10$ (voir la transformation Θ dans la preuve du Th. 6.2).

taille de $\mathcal{U}$	(2, 18)	(3, 10)	(4, 6)	(5, 5)	(7, 4)	(10, 3)	(24, 2)
taille de $SR(\mathcal{U})$	(40, 30)	(34, 23)	(28, 20)	(29, 20)	(32, 21)	(34, 23)	(52, 36)

Malheureusement, les tailles obtenues n'ont pas le bon goût de se disposer le long d'une frontière, seule la taille $(28, 20)$ obtenue à partir de $(4, 6)$ est relevante. Nous pouvons cependant préciser un autre point de la frontière (la partie en gris foncé du dessin à droite de la FIG. 6.7), grâce à la propriété suivante :

Propriété 6.1 *Il existe une machine $\mathcal{SR}$ universelle à 6 règles et dont le support est de taille 22 (*i.e. *une machine de taille* $(6, 22)$*).*

Preuve: Nous devons exhiber la transformation g, le support $\mathcal{S}_U$, et l'ensemble de règles $\mathcal{R}_U$. Pour construire $\mathcal{S}_U$, nous aurons besoin d'un ensemble de types d'arité 1 qui représentent les symboles d'états et les lettres de l'alphabet, en nombre suffisant pour encoder n'importe quelle machine de Turing. Prenons ceux d'une MdT universelle de taille (p, q) pour laquelle $p+q$ est minimal : $(4, 6)$. Le support est celui associé par Θ à cet alphabet et à ces symboles, auquel nous ajoutons deux relations d'arité 4, *droite* et *gauche*. Ceci nous donne bien un support de taille 22. Intuitivement, si une relation *droite* a pour arguments quatre sommets concepts typés respectivement par T_p, T_q, a et b, cela voudra dire qu'il existe une règle $(T_p, a) \rightarrow (T_q, b, droite)$.

Les règles de $\mathcal{R}_U$ sont les quatre règles de recopie de la FIG. 6.5, auxquelles nous rajoutons une règle d'application à gauche, et une règle d'application à droite. Seule cette dernière est représentée dans la FIG. 6.8. Cette règle se comporte exactement de la même manière que celles utilisées dans la démonstration du Th. 6.2, mais elle doit « lire » la façon dont elle doit s'appliquer dans le graphe initial (les paramètres d'une règle seront déterminés par sa projection).

Enfin, la transformation g associe à toute machine de Turing $\mathcal{M}$ de taille $(4, 6)$ et à tout mot m sur un alphabet de taille 6 un graphe construit de la façon suivante :

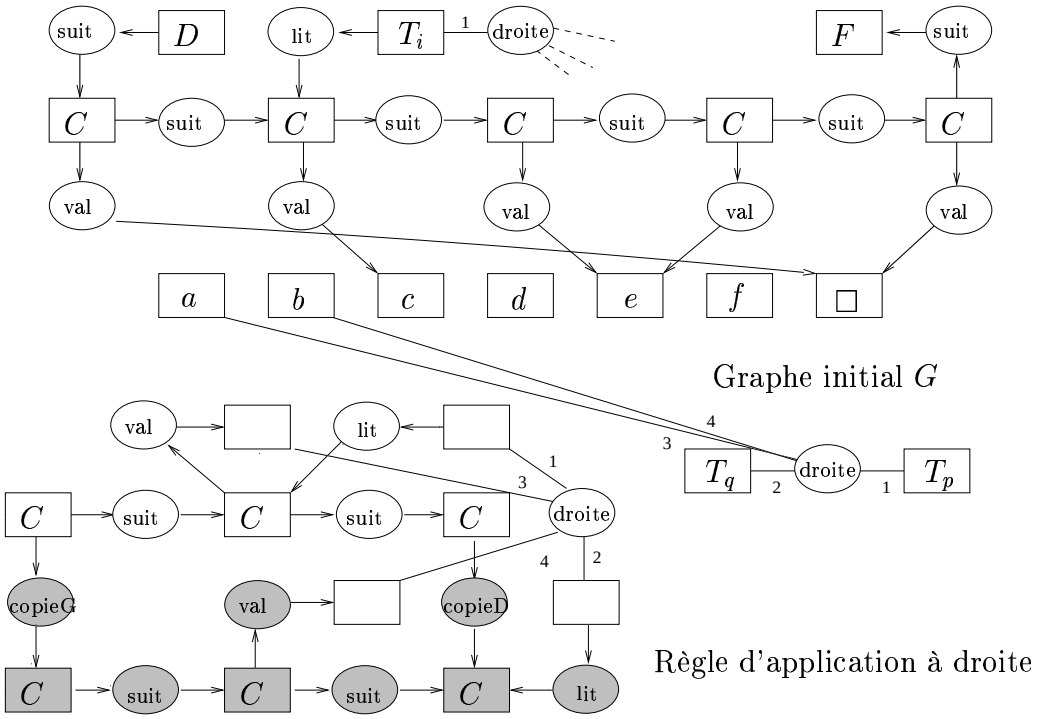


FIG. 6.8 – Construction de la machine $\mathcal{SR}$ universelle de taille $(6, 22)$.

- construire tout d'abord le graphe G de $\Theta(\mathcal{M}, m)$, comme dans la preuve du Th. 6.2 ;
- pour chacun des symboles d'état T_n différents de T_i (qui y est déjà), ajouter à G un sommet concept typé par T_n ;
- pour chaque règle $(T_p, a) \to (T_q, b, \delta)$ de $\mathcal{M}$, rajouter une relation d'arité 4, typée par δ, dont les arguments sont respectivement le sommet concept typé par T_p, celui typé par T_q, celui typé par a et celui typé par b.

Il ne reste qu'à vérifier, ce que nous laissons à l'attention du lecteur, que les 6 règles de $\mathcal{R}_U$ engendrent exactement les mêmes dérivations que celles qui auraient été obtenues avec les règles de $\Theta(\mathcal{M}, m)$. □

Machine $\mathcal{SR}$ universelle de taille $(1, 1)$.

Nous pourrions présenter ici d'autres machines $\mathcal{SR}$ universelles, pour lesquelles on arrive à réduire la taille du support ou le nombre de règles, au prix de la lisibilité de la transformation. Cependant, des arguments plus fondamentaux, que nous allons développer dans la section suivante, vont nous permettre de prouver le théorème suivant, et de clore ce problème.

Théorème 6.3 *Il existe une machine $\mathcal{SR}$ universelle de taille $(1, 1)$.*

Ce résultat montre que jouer sur le nombre de règles ou sur la taille du support n'est décidément pas la bonne méthode pour trouver des cas décidables pour $\mathcal{SR}$-DÉDUCTION.

Nous déplorons aussi de n'avoir pu trouver aucune idée permettant de retranscrire ce résultat dans le modèle des MdT, dans l'espoir de préciser la frontière décidable/semi-décidable.

Nous allons maintenant considérer différentes restrictions du problème $\mathcal{SR}$-DÉDUCTION, pour lesquelles le problème reste semi-décidable. Les deux premiers cas, basés sur le nombre de règles et la taille du support, suffiront à prouver le Th. 6.3. Le dernier cas est basé sur la structure du graphe, et montre que, même si les règles sont de la forme « si *chemin* alors *chemin* », le problème reste semi-décidable.

Restriction du nombre de règles

Lemme 6.1 *Le problème $\mathcal{SR}$-*DÉDUCTION *est réductible à sa restriction ne considérant que des bases de connaissance à une seule règle.*

Preuve: Nous souhaitons exhiber une transformation Θ qui associe à toute instance $(\mathcal{K} = (\mathcal{S}, G, \mathcal{R}), H)$ du problème $\mathcal{SR}$-DÉDUCTION, une instance $(\mathcal{K}' = (\Theta(\mathcal{S}), \Theta(G, \mathcal{R}), \Theta(\mathcal{R})), \Theta(H))$ de ce même problème telle que $|\Theta(\mathcal{R})| = 1$ et H est déductible de $\mathcal{K}$ si et seulement si $\Theta(H)$ est déductible de $\mathcal{K}'$.

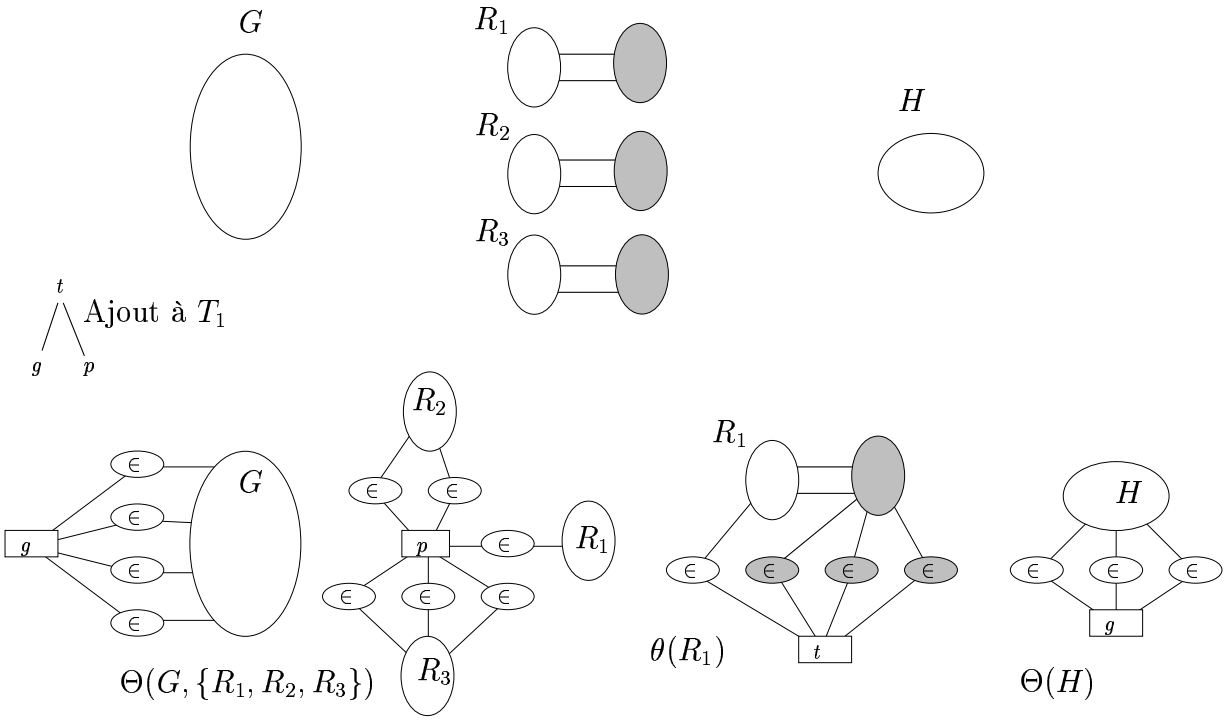


FIG. 6.9 – Instance équivalente de $\mathcal{SR}$-DÉDUCTION, n'utilisant qu'une règle.

– $\Theta(\mathcal{S})$ est obtenu à partir de $\mathcal{S}$ en rajoutant d'une part trois types d'arité 1 : g qui sera utilisé pour représenter le graphe, p qui sera utilisé comme « état poubelle », et t,

supertype de g et p ; et d'autre part $\in$, relation binaire (que nous pouvons considérer non orientée), qui sert à indiquer si un sommet concept fait partie du graphe ou d'un « état poubelle ».

- le graphe $\Theta(G, \mathcal{R})$ est obtenu à partir de l'union disjointe de $G_1 = \theta(G)$ et de $G_2 = \theta(\mathcal{R})$.
  - G_1 est obtenu en rajoutant un sommet concept typé par g à G, puis en reliant ce nouveau sommet à chacun des sommets concepts de G par une relation binaire de type $\in$.
  - G_2 est obtenu en faisant l'union disjointe des hypothèses de toutes les règles de $\mathcal{R}$, puis en rajoutant un sommet concept typé par p, relié par $\in$ à chacun des sommets concepts obtenus.
- pour chaque règle $R \in \mathcal{R}$, une règle $\theta(R)$ est construite en ajoutant à R un sommet concept typé par t, de couleur 0, et en reliant ce sommet concept à chacun des sommets concepts de R, par l'intermédiaire d'une relation $\in$. La couleur de ces relations est déterminée par la couleur de leur extrémité dans R.
- l'unique règle de $\Theta(\mathcal{R})$ est constituée de l'union disjointe de toutes les $\theta(R)$, pour chaque $R \in \mathcal{R}$.
- enfin, $\Theta(H)$ est obtenu en lui rajoutant un sommet concept typé par g, relié à chacun des sommets concepts de H par une relation typée par $\in$.

Ces transformations sont illustrées dans la FIG. 6.9. Montrons maintenant l'équivalence entre ces deux problèmes. Pour alléger cette démonstration, nous noterons $G_\theta = \Theta(G, \mathcal{R})$, Θ-dérivation une $\{\Theta(\mathcal{R})\}$-dérivation, et dérivation une $\mathcal{R}$-dérivation.

Nous prouvons tout d'abord la propriété suivante : « il existe une Θ-dérivation de G_Θ en G'_Θ si et seulement s'il existe une dérivation de G en G' telle que le sous-graphe de G'_Θ engendré par les sommets reliés par $(\in)$ au sommet concept de type g (nous le notons $P_g(G'_\Theta)$) est isomorphe à G' ».

($\Rightarrow$) Par récurrence sur la longueur de la Θ-dérivation, où l'hypothèse de récurrence est : « s'il existe une Θ-dérivation de longueur n de G_Θ en G'_Θ, alors il existe une dérivation de G en G' telle que $P_g(G'_\Theta)$ est isomorphe à G' ».

Cette hypothèse est trivialement vérifiée au rang 0 (par construction de G_Θ). Supposons qu'elle soit vérifiée au rang k, et montrons qu'elle est toujours vraie au rang $k+1$. Tout d'abord, s'il existe une Θ-dérivation de longueur $k+1$ de G_Θ en G'_Θ, alors il existe une Θ-dérivation de longueur k de G_Θ en G''_Θ et une Θ-dérivation immédiate, utilisant une projection π, de G''_Θ en G'_Θ.

Par hypothèse de récurrence, il existe une dérivation de G en G' telle que $P_g(G''_\Theta)$ est isomorphe à G''. Considérons $\theta(R_1), \ldots, \theta(R_p)$ les composantes connexes de la règle $\{\Theta(\mathcal{R})\}$ dont le sommet de type t de l'hypothèse se projette par π dans l'unique sommet de type g de G''' (ce sommet est unique car aucune application de règle ne peut rajouter un sommet de ce type). Les autres $\theta(R_i)$ ne peuvent modifier $P_g(G''_\Theta)$, et se projettent toutes dans $P_p(G''_\Theta)$ (le sous graphe de G''_Θ engendré par les sommets reliés à l'unique sommet de type p). Alors il est immédiat de voir que les applications successives de $R_1, \ldots, R_p$ à G'', suivant la restriction de π à l'hypothèse de chacune de ces règles, produisent un graphe isomorphe à $P_g(G'_\Theta)$. L'hypothèse de récurrence est donc vérifiée au rang $k+1$.

($\Rightarrow$) Par récurrence sur la longueur de la dérivation, où l'hypothèse de récurrence est « s'il existe une dérivation de longueur n de G en G' alors il existe une Θ-dérivation de G_Θ en G'_Θ telle que $P_g(G'_\Theta)$ est isomorphe à G' ».

Ici aussi, l'hypothèse est trivialement vérifiée au rang 0. Supposons-la vraie au rang k. Alors s'il existe une dérivation de longueur $k+1$ de G en G', il existe une dérivation de longueur k de G en G'' et une dérivation immédiate de G'' en G', suivant la projection π d'une règle $R \in \mathcal{R}$. Par hypothèse de récurrence, il existe une Θ-dérivation de G_Θ en G''_Θ telle que $P_g(G''_\Theta)$ est isomorphe à G''. Voir que le graphe G'_Θ obtenu en appliquant la partie $\theta(R)$ de la règle $\Theta(\mathcal{R})$ suivant π à G''_Θ, toutes les autres composantes connexes de $\Theta(\mathcal{R})$ étant projetées dans $P_p(G''_\Theta)$ est tel que $P_g(G'_\Theta)$ est isomorphe à G', ce qui prouve l'hypothèse de récurrence au rang $k+1$.

Pour achever cette démonstration, il ne reste plus qu'à remarquer que H se projette dans un graphe G si et seulement si $\Theta(H)$ se projette dans un graphe dont tous les sommets sont reliés par ($\in$) à un sommet de type g. □

Nous avons non seulement montré que $\mathcal{SR}$-DÉDUCTION est semi-décidable, même si l'ensemble de règles est réduit à une seule règle, mais nous avons construit une machine $\mathcal{SR}$ universelle de taille $(1, 24)$ à 1 règle et à 24 types de concepts. En effet, si nous partons de la machine $\mathcal{SR}$ universelle de taille $(28, 20)$ dont nous avons prouvé l'existence, alors cette transformation nous donne une machine $\mathcal{SR}$ universelle de taille $(1, 24)$.

Restriction de la taille du support

Lemme 6.2 *Le problème* $\mathcal{SR}$-DÉDUCTION *est réductible à sa restriction ne considérant que des bases de connaissances sur un support ne contenant qu'un seul type de relation (d'arité supérieure ou égale à 2).*

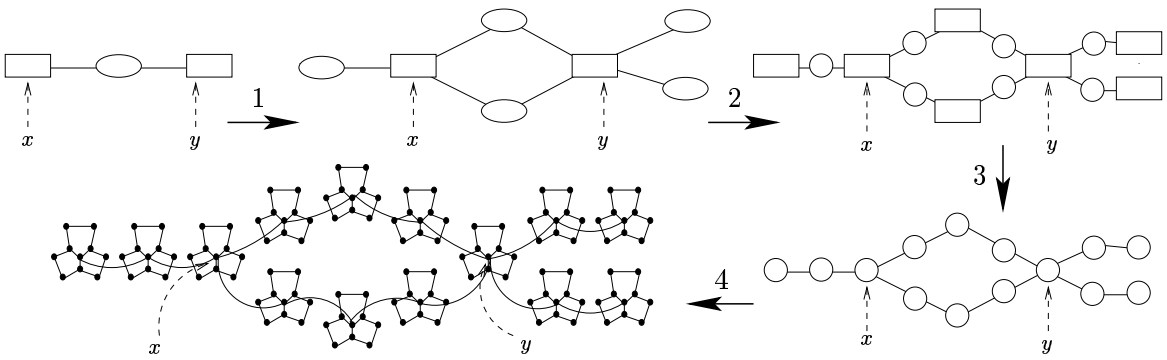


FIG. 6.10 – Transformations en graphes n'utilisant qu'un type de relation.

Preuve: Nous ne donnerons ici qu'une esquisse de démonstration, sa partie délicate (la suppression des étiquettes d'un graphe binaire étiqueté) reposant sur des résultats connus

(voir par exemple, [Hell and Nešetřil, 1979]). Cette démonstration est basée sur la séquence de transformations suivantes (illustrées dans la FIG. 6.10).

1. Soit G un graphe élémentaire, alors nous obtenons le graphe G' en rajoutant à G, pour chacune de ses relations r, l'ensemble de relations jumelles dont les types sont les types plus généraux que celui de r (cette transformation est utilisée pour démontrer la Prop. 4.4). Cette transformation supprime le besoin de comparer les types dans le support.

2. Le graphe G'' est obtenu à partir de G' en remplaçant chaque relation r d'arité k par un sommet concept de même type. Le sommet obtenu à partir de r est relié par k relations symétriques binaires, dont les types sont $1, 2, \ldots, k$ (ces types, quitte à les renommer, étant incompatibles avec ceux du support), aux sommets représentant les arguments de r. Cette transformation supprime les hyperarcs, comme les numéros sur les arêtes.

3. On peut maintenant considérer que les sommets concepts comme les relations de ce graphe sont des sommets, reliés par une relation symétrique binaire non étiquetée (de façon similaire à la transformation de graphes conceptuels élémentaires en graphes conceptuels très simple). Le graphe (il s'agit encore de G'') ainsi obtenu peut être considéré comme un graphe (au sens usuel de la théorie des graphes), étiqueté. La projection entre de tels graphes est un *homomorphisme de graphes colorés* (les étiquettes sont appelées couleurs dans la communauté « homomorphismes de graphes », elles ne sont comparables que si elles sont égales).

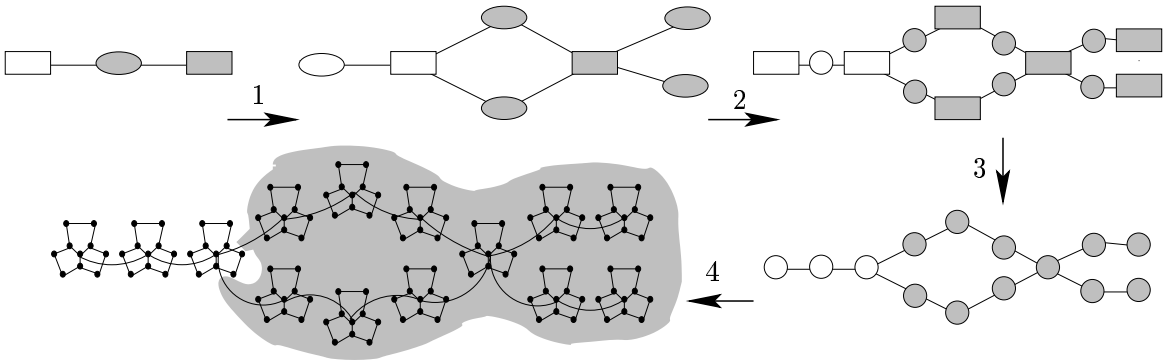
4. La dernière transformation (dont on peut trouver une version, par exemple, dans [Hell and Nešetřil, 1979], est couramment utilisée dans la communauté « homomorphismes de graphes » (elle est utilisée, par exemple, dans [Nešetřil, 2000] pour prouver que l'ensemble des *cores* est le treillis universel dénombrable, *c.f.* Th. 3.4).

   À chaque type t du nouveau support (en prenant en compte les nouveaux types $1, 2, \ldots$, nous associons un graphe (au sens de la théorie des graphes) non étiqueté $G(t)$ de façon à avoir la propriété : si t est t' sont deux types distincts, alors $G(t)$ et $G(t')$ sont incomparables. De plus, nous avons besoin d'une autre propriété : pour tout type t, le graphe $G(t)$ contient un invariant pour les endomorphismes (un sommet x de $G(t)$ tel que, pour tout endomorphisme π, $\pi(x) = x$). Nous appelons cet invariant le *centre* de $G(t)$. Nous pouvons construire de tels graphes en utilisant des *Fleurs* [**?**] : une Fleur(p, q) est une construction utilisant un nombre p de cycles élémentaires de taille q. Nous ne rentrerons pas dans les détails, mais préciserons que toute fleur admet un centre, et que, si $p < p'$ et $q' < q$ sont des nombres impairs, Fleur(p, q) et Fleur(p', q') sont incomparables. Notons que nous avons besoin pour cette construction d'un support fini, ce que nous pouvons considérer sans perte de généralité puisqu'il existe une machine $\mathcal{SR}$ universelle sur un support fini. Notons aussi que ces constructions sont polynomiales dans la taille de ces nombres. Si le support de G'' est de taille s, nous construisons (en temps polynomial) s fleurs incomparables.

Enfin, le graphe G''' est obtenu de la façon suivante : pour chaque sommet x de G'', dont l'étiquette est t, G''' contient une copie de la fleur $G(t)$. Pour chaque arête xy, nous rajoutons une arête entre le centre de la fleur représentant x et le centre de la fleur représentant y. Remarquons que, dans la FIG. 6.10, toutes les structures utilisées sont des Fleur(3, 5), il ne s'agit que d'un artifice de représentation car notre transformation génère au moins toujours deux types de Fleurs distincts.

5. Enfin, le graphe G''' n'est pas un graphe élémentaire, mais nous pouvons le voir comme une représentation simplifiée d'un graphe élémentaire symétrique dont toutes les relations sont binaires et ont un même type, t.Nous noterons $T(G)$ le graphe obtenu à partir de G par cette séquence de transformations. Nous n'avons pas, pour des raisons évidentes liées à la taille (pourtant polynomiale) du graphe obtenu, représenté le graphe élémentaire $T(G)$ dans la FIG. 6.10.

Nous affirmerons sans le démontrer la propriété P : π est une projection de H dans G si et seulement si il existe une projection π' de $T(H)$ dans $T(G)$ telle que la restriction de π' aux sommets de $T(H)$ représentant des sommets de H (les sommets x et y dans la FIG. 6.10) est égale à π.



Transformations d'une règle

FIG. 6.11 – Transformations en règles n'utilisant qu'un type de relation.

Enfin, une transformation similaire est utilisée pour construire des règles. Nous ne ferons qu'illustrer, à travers la FIG. 6.11, la façon de colorier les objets construits à chaque étape de la démonstration. Les affirmations suivantes sont une conséquence directe de la propriété P :

- une règle R est applicable suivant π à un graphe G si et seulement si $T(R)$ est applicable à $T(G)$ suivant une projection π' dont la restriction aux sommets représentant ceux de l'hypothèse de R est égale à π ;
- il s'ensuit que $G' = \lambda(G, R, \pi)$ si et seulement si $T(G') = \lambda(T(G), T(R), \pi)$;
- par une récurrence immédiate, H est déductible de $(G, \mathcal{R})$ si et seulement si $T(H)$ est déductible de $(T(G), T(\mathcal{R}))$.

Nous avons donc une transformation calculable (et même polynomiale) permettant de transformer toute instance de $\mathcal{SR}$-DÉDUCTION en une instance de sa restriction à un

support de taille 1, et qui conserve les déductions. □

Nous avons non seulement montré que $\mathcal{SR}$-DÉDUCTION est semi-décidable, même lorsque le support ne contient qu'un seul type de relation binaire, mais nous avons construit une machine $\mathcal{SR}$ universelle de taille $(1, 1)$, et avons prouvé le Th. 6.3. En effet, la traduction que nous avons proposée n'augmente pas le nombre de règles. Donc si nous traduisons la machine $\mathcal{SR}$ universelle de taille $(1, 24)$ dont nous avons proposé l'existence, nous obtenons une machine $\mathcal{SR}$ universelle de taille $(1, 1)$.

6.2.3 Autres restrictions du modèle général

Nous étudions ici d'autres cas, décidables ou encore semi-décidables, qui pourront être utilisé dans les modèles plus généraux de la famille $\mathcal{SG}$.

$\mathcal{SR}$-DÉDUCTION et le problème du mot

Nous évoquons ici une autre restriction des règles, qui ne suffit pas à rendre le problème $\mathcal{SR}$-DÉDUCTION décidable. Pourtant, cette restriction, ne considérant que des règles de la forme « *si* chemin *alors* chemin », semble à première vue très importante. Cette restriction nous sera utile pour démontrer une propriété dans le Chap. 7.

Propriété 6.2 *Même si on se restreint à des ensembles de règles ne contenant que des règles de la forme «* si *chemin* alors *chemin », alors le problème* $\mathcal{SR}$-DÉDUCTION *reste semi-décidable.*

Proof: Let us show that $\mathcal{SR}$-DEDUCTION is not decidable by building a reduction from the WORD PROBLEM IN A SEMI-THUE SYSTEM [Thue, 1914]. This problem was proven semi-decidable by [Post, 1947], using a reduction to his correspondence Problem.

The WORD PROBLEM can be expressed as : let m and m' two words, and $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ be a set of rules, each rule γ_i being a pair of words (α_i, β_i) : is there a derivation from m to m' ? There is an immediate derivation from m to m' (we note $m \to m'$) if, for some γ_j, $m = m_1\alpha_j m_2$ and $m' = m_1\beta_j m_2$. A *derivation* from m to m' (we note $m \rightsquigarrow m'$) is a sequence $m = m_0 \to m_1 \to \ldots \to m_p = m'$.

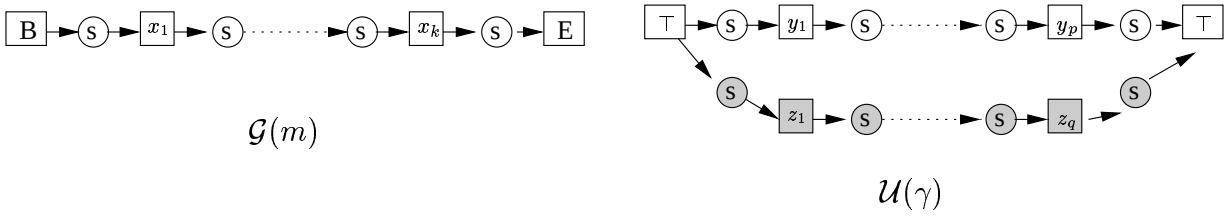


FIG. 6.12 – Transformation from the WORD PROBLEM into $\mathcal{SR}$-DEDUCTION

This problem can easily be expressed in the $\mathcal{SR}$ model. One concept type $\mathtt{x}_i$ is assigned to each letter x_i. There are three other concept types : B (for "begin"), E (for "end") and ⊤

(for "anything"). $\top$ is the greatest concept type and all other types are pairwise noncomparable. There is one relation type $\mathtt{s}$ (for "has successor"). A word $m = x_1 \dots x_k$ is associated the graph $\mathcal{G}(m)$, and to any rule $\gamma = (y_1 \dots y_p,\ z_1 \dots z_q)$ is associated the graph rule $\mathcal{U}(\gamma)$, as represented in Fig. 6.12. By a straightforward proof (a recurrence on the smallest derivation length), we obtain that to every path from the node typed B to the node typed E ("begin" to "end") in a graph $\mathcal{R}$-derived from $\mathcal{G}(m)$, corresponds a word (and not a subword) derivable from m, and reciprocally. It follows that $m \rightsquigarrow m' \Leftrightarrow \mathcal{G}(m') \geq (\mathcal{G}(m), \mathcal{U}(\Gamma))$. □

Graphes fermés et complets

L'application d'une règle peut créer de l'information redondante dans un graphe. En général, détecter la redondance est un problème difficile (nous avons vu, dans la Sect. 5.1.3, que REDONDANT ? est un problème NP-complet). Il y a cependant des cas immédiats, dont nous pouvons nous débarrasser, qui créent des dérivations dont la longueur infinie est artificielle.

Tout d'abord, si une règle a déjà été appliquée à un graphe suivant une projection, alors elle peut être appliquée de nouveau sur le graphe obtenu, suivant la même projection, et ceci indéfiniment. Ces applications répétées suivant la même projection créent de l'information redondante, elles seront dites *inutiles.*

Définition 6.7 (Graphe fermé pour une dérivation) *Soit $\mathcal{R}$ un ensemble de règles, et $G = G_0, \dots, G_k = H$ une $\mathcal{R}$-derivation d'un graphe G. Le graphe H est dit* fermé *pour cette dérivation si toute application d'une règle de $\mathcal{R}$ à H est inutile,* i.e. *si $R \in \mathcal{R}$ est applicable à H suivant une projection π, alors il existe $G_i \in \{G_1, \dots, G_k\}$ tel que $G_i = \lambda(G_{i-1}, R, \pi)$.*

Soit $\mathcal{R}$ un ensemble de règles et un graphe G, alors si un graphe fermé est $\mathcal{R}$-dérivable de G, ce graphe est unique à un isomorphisme près. De plus, si ce graphe H est obtenu par une dérivation de longueur n, alors n est la longueur maximale de toute dérivation sans application inutile, et toutes les dérivations de longueur n sans application inutile génèrent ce graphe. Lorsqu'il existe, nous l'appellerons la *fermeture* de G pour l'ensemble de règles $\mathcal{R}$, et le noterons $G^*_{\mathcal{R}}$.

Une autre notion similaire est cette fois liée à la notion de graphe irredondant.

Définition 6.8 (Graphe complet) *Un graphe irredondant G est dit* complet *pour un ensemble de règles $\mathcal{R}$, si pour toute application d'une règle $R \in \mathcal{R}$ à G suivant une prio-jection π, le graphe obtenu est équivalent à G : $\lambda(G, R, \pi) \equiv G$.*

Comme G est un graphe irredondant et que tous les graphes que nous dérivons sont équivalents à leur forme irredondante, alors si un graphe complet peut être dérivé, il est unique.

De façon informelle, la notion de graphe fermé traduit que rien ne peut être déduit qui n'a pas déjà été déduit, tandis que la notion de graphe complet traduit qu'aucune déduction ne peut apporter de nouvelles informations.

Propriété 6.3 *Lorsque la fermeture du graphe existe, alors sa forme irredondante est le graphe complet. Cependant, lorsque le graphe complet existe, sa fermeture n'existe pas nécessairement.*

Preuve: La première assertion est une conséquence directe de la remarque précédente. Pour la deuxième, considérons le cas des règles *déconnectées*. Il s'agit de règles pour lesquelles l'hypothèse n'est pas connectée à la conclusion. Remarquons que le graphe complet existe dans ce cas (chaque conclusion n'aura été rajoutée qu'au plus une fois), tandis qu'une dérivation peut ne pas être fermée. Prenons par exemple la règle « si $[t]$ alors $[t]$ » et le graphe [t] : pour toute dérivation, il existe une application qui n'est pas inutile (sur le dernier sommet rajouté). □

6.2.4 Ensembles à expansion finie et règles *range restricted*

Nous avons utilisé la notion de fermeture pour caractériser les ensembles de règles à expansion finie dans [Baget and Mugnier, 2001b]. Nous utilisons ici la notion, plus générale, de complétude.

Définition 6.9 *[Ensembles de règles à expansion finie] Un ensemble de règles $\mathcal{R}$ est dit à* expansion finie *si, pour tout graphe G, il existe une $\mathcal{R}$-dérivation de G en G' telle que $irr(G')$ (le graphe irredondant de G') est complet pour $\mathcal{R}$. Nous notons ce graphe $G^{\mathcal{R}}$.*

Si $\mathcal{R}$ est un ensemble à expansion finie, alors le problème $\mathcal{SR}$-DÉDUCTION devient décidable (mais ce n'est pas une condition nécessaire de décidabilité). Pour vérifier si H est déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$, il suffira de vérifier qu'il existe une projection de H dans $G^{\mathcal{R}}$.

Notons déjà (il s'agit de la Prop. 7.8 que nous prouverons dans le Chap. 7), que, si $\mathcal{R}$ et $\mathcal{R}'$ sont deux ensembles à expansion finie, $\mathcal{R} \cup \mathcal{R}'$ ne l'est pas nécessairement.

Nous allons maintenant nous intéresser à des règles particulières, que nous avons utilisées dans notre résolution du problème SISYPHUS [Baget et al., 1999], qui sont les règles *range restricted* (par analogie avec les règles du même nom en Datalog, où toutes les variables de la tête doivent apparaitre dans la queue, voir, par exemple [Abiteboul et al., 1995]).

Définition 6.10 *[Règles* range restricted*] Une règle R (et par extension un graphe coloré) est dite* range restricted *si sa conclusion (et par extension $R_{(1)}$ ne comprend aucun sommet concept générique (dans le cadre des graphes élémentaires, ceci veut dire qu'elle ne comprend aucun sommet concept).*

Par exemple, le premier graphe coloré de la FIG. 6.13 est *range restricted*, tandis que le second ne l'est pas.

Remarquons qu'une règle range restricted R peut être décomposée en un ensemble de règles équivalent $R_1, \ldots, R_p$ (*i.e.* pour tout graphe G' $\{R\}$-dérivable de G, il existe un graphe G'' $\{R_1, \ldots, R_p\}$-dérivable de G, et réciproquement) qui ont chacune soit exactement une relation, soit exactement un sommet concept individuel dans leur conclusion. L'interprétation logique de ces règles est celle de clauses de Horn range restricted. Notons que cette équivalence ne tiendra plus dans les modèles plus généraux de la famille $\mathcal{SG}$.

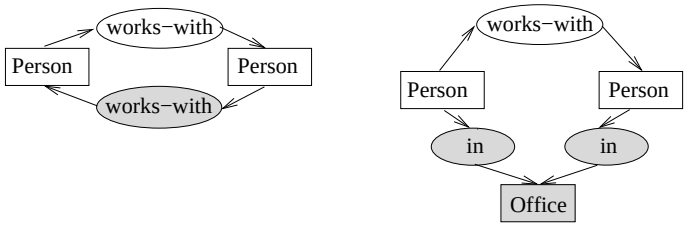


FIG. 6.13 – Deux graphes colorés : le premier est *range restricted*

Propriété 6.4 *Un ensemble de règles* range restricted *est un ensemble à expansion finie.*

Proof: Since all graphs are put into normal form, an individual marker appears at most once in a graph. The number of individual nodes created by the set of rules is bounded by $M = |\mathcal{R}| \times \max_{R \in \mathcal{R}} |R_{(1)}|$. So the number of relation nodes created (no twin relation nodes are created) is bounded by $N = \sum_{n=1}^{k} P_n(|V_C(G)| + M)^n$, where P_n is the number of relation types with a given arity n appearing in a rule conclusion, and k is the greatest arity of such a relation type. So the closure of a graph is can be obtained with a derivation of length $L \leq N + M$. We thus obtain $\mathcal{G}^{\mathcal{R}}$ in finite time. □

Avec ces règles range restricted, nous sommes assurés aussi bien de l'existence de la fermeture que du graphe complet (ces notions sont équivalentes dans le cas des règles *range restricted*). Il s'ensuit de la preuve de la Prop. 6.4 que la $\mathcal{R}$-dérivation de G en $\mathcal{G}^{\mathcal{R}}$ est de longueur bornée par $\mathcal{O}(n^{k+1})$, où n est la taille de $(G, \mathcal{R})$ et k est la plus grande arité des relations apparaissant dans une conclusion de règle. Comme cette longueur borne l'application de toute dérivation sans application inutile :

Théorème 6.4 *La restriction de* $\mathcal{SR}$-DÉDUCTION *à des règles range restricted, que nous noterons* $\mathcal{SR}^{rr}$ -DÉDUCTION*, est un problème NP-complet (si l'on considère l'arité maximale des types du support* $\mathcal{S}$ *comme une constante).*

Preuve: La dérivation menant à un graphe G' satisfaisant la requête H est de longueur polynomiale. Le graphe G', comme tous ceux intervenant dans la dérivation, est de taille polynomiale. Donc la donnée de toutes les projections utilisées pour appliquer les règles et vérifier H est un certificat de taille polynomiale, vérifiable polynomialement. Donc le problème $\mathcal{SR}^{rr}$ -DÉDUCTION appartient à NP.

Comme PROJECTION ? est un cas particulier de $\mathcal{SR}^{rr}$ -DÉDUCTION (où $|\mathcal{R}| = 0$), alors $\mathcal{SR}^{rr}$ -DÉDUCTION est un problème NP-complet. □

6.3 Un algorithme pour la marche avant : CBR

Nous allons au cours de cette section proposer une amélioration de l'algorithme naïf de résolution en marche avant dans le modèle $\mathcal{SR}$. L'idée de l'algorithme CBR que nous

présentons est basée sur un prétraitement de la base de règles (construction d'un graphe de dépendances des règles). CBR nous permettra alors de ne pas tester l'applicabilité de certaines des règles pour lesquelles il n'y a aucune aucune chance qu'elles soient applicables, et en même temps de tester cette applicabilité uniquement dans les parties du graphe pour lesquelles il y a une chance de trouver une projection. De plus, le graphe de dépendances des règles peut être utilisé pour caractériser de nouveaux cas de décidabilité (qui ne sont pas détectables par l'étude, séparément, de chacune des règles).

6.3.1 Marche avant

L'algorithme de chaînage avant est similaire à celui que l'on peut trouver dans de nombreux formalismes. Nous ne le rappelons ici que pour préciser les notations que nous allons utiliser (Alg. 9).

Algorithme 9: `MA-Déductible ?`$(\mathcal{K}, H)$

Données : Une base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R})$ et une requête H.

Résultat : VRAI si et seulement si H est déductible de $\mathcal{K}$.

graphe $\leftarrow \mathcal{G}$;
tant que *(*$\neg$ PROJECTION $?(\mathcal{S}, H, graphe)$*)* **faire**
  applications $\leftarrow \emptyset$;
  pour *(*$R \in \mathcal{R}$*)* **faire**
    pour *(*$\pi \in$ PROJECTIONS$(\mathcal{S}, hypothèse(R), graphe)$*)* **faire**
      applications $\leftarrow$ applications $\cup$ (R, π);
  si *(*$applications = \emptyset$*)* **alors**
    retourner FAUX;
  pour *(*$(R, \pi) \in applications$*)* **faire**
    graphe $\leftarrow \lambda(graphe, R, \pi)$;
retourner `vrai`;

Chaque tour de boucle **tant que** représente une *étape* de l'algorithme : appliquer au graphe courant toutes les règles suivant toutes les projections possibles. Si nous souhaitons ne pas générer d'applications inutiles (espérant générer la fermeture), nous pourrons par exemple ne considérer que les applications qui utilise au moins un sommet ajouté à l'étape précédente. Afin de ne pas générer d'application redondante (espérant générer le graphe complet), nous pourrons par exemple mettre le graphe sous sa forme irredondante à chaque application de règle (ou à chaque étape du parcours en marche avant).

[Salvat, 1997] propose un mécanisme de chaînage arrière efficace, reposant sur l'application de pièces. Cependant, nous n'avons pas étudié comment résoudre les problèmes plus généraux de la famille $\mathcal{SG}$ en utilisant un tel algorithme. Nous avons donc cherché à améliorer le mécanisme de chaînage avant.

6.3.2 Compilation d'une base de règles

Cette section est dédiée à l'amélioration du mécanisme d'application des règles en marche avant. Nous ne considérerons dans un premier temps que des règles de graphes conceptuels très simples (ou de graphes élémentaire), et discuterons ensuite de son adaptation à des formalismes plus généraux, et notamment aux graphes conceptuels simples, où la mise sous forme normale est à prendre en considération. Supposons $\mathcal{R} = \{R_1, \ldots, R_k\}$ une base de règles. Supposons de plus que, pour un graphe G donné, seule la règle R_1 s'applique, suivant une seule projection π. L'application de R_1 suivant π nous permet d'obtenir un graphe G'. Suivant le mécanisme standard d'application des règles en marche avant, il nous faut maintenant tester chacune des règles de $\mathcal{R}$ pour savoir si elle s'applique à G'.

Une première remarque est que, lorsque l'on teste si R_i peut s'appliquer à G', le fait que R_i ne s'appliquait pas à G peut être utilisé. En effet, s'il existe une projection π' de l'hypothèse de R_i dans G', toutes les images par π' ne peuvent pas être des sommets de G (sinon, R_i aurait pu se projeter dans G). Il existe donc au moins un sommet de l'hypothèse de R_i dont l'image est dans la partie de G' qui a été rajoutée par l'application de R_1. La recherche d'une projection peut donc se faire *autour* de la partie de G' qui vient d'être rajoutée.

Cette remarque en appelle une seconde : s'il n'y a aucun sommet de l'hypothèse de R_i qui se projette dans la conclusion de R_1, alors il n'y aura pas lieu de tester de nouveau l'application de R_i, si elle ne s'appliquait pas avant l'application de R_1. Nous dirons alors que R_1 est *neutre* pour R_i. Dans le cas contraire, nous dirons que R_1 est un *déclencheur possible* de R_i, et il sera nécessaire de tester de nouveau l'application de R_i après chaque application de R_1.

En développant ces deux remarques, nous obtenons un algorithme qui permet, d'une part, de réduire le nombre de règles dont on teste l'application après chaque application de règle, et d'autre part, de réduire l'espace de recherche quand on teste l'application d'une règle. Cet algorithme construit, et utilise, un *graphe de dépendance des règles*. Les sommets de ce graphe sont les règles, et l'*absence* d'un arc de R_i vers R_j signifie que R_i est neutre pour R_j. Cette information sera utilisée au cours de l'application des règles en marche avant. Notons que si le graphe de dépendance des règles est le graphe complet (boucles comprises), alors chaque application de règle nécessitera de tester de nouveau toutes les règles de la base. L'algorithme sera bien adéquat et complet (il s'agit de l'algorithme de recherche en marche avant), mais nous n'aurons absolument rien gagné en utilisant le graphe de dépendances. Il nous faudra donc un critère de neutralité permettant de supprimer le plus possible d'arcs dans ce graphe, tout en gardant l'adéquation et la complétude de l'algorithme.

Neutres et déclencheurs

Intuitivement, une règle R_1 est neutre pour une règle R_2 si une application de R_1 ne peut pas créer de nouvelles applications de R_2.

Définition 6.11 (Neutre) *Soient R_1 et R_2 deux règles définies sur un même support $\mathcal{S}$. Alors nous disons que R_1 est un* neutre *pour R_2 ssi, pour tout graphe G que l'on peut*

définir sur $\mathcal{S}$ *:*
- *si* Π *est l'ensemble des projections de l'hypothèse de* R_2 *dans* G *;*
- *si* G' *est un graphe obtenu par application immédiate de* R_1 *sur* G

Alors l'ensemble des projections de R_2 *dans* G' *est* Π.

Si on peut prouver qu'une règle R_i est neutre pour une règle R_j, alors on peut accélérer le processus de résolution en marche avant : il n'y aura pas besoin de tester de nouveau l'application de R_j après avoir appliqué R_i.

Propriété 6.5 *Soient* $\{R_{i_1}, \ldots, R_{i_p}\}$ *les règles appliquées à la kième étape du processus de résolution en marche avant. Alors si chacune de ces règles est un neutre pour* R_j*, alors l'algorithme reste adéquat et complet si on ignore le test de l'application de* R_j *à l'étape* $k+1$.

Nous utiliserons cette propriété en construisant un *graphe de dépendance des règles.* Les sommets de ce graphe sont les règles de la base $\mathcal{R}$. La présence d'un arc entre R_i et R_j veut dire que R_i *n'a pas été prouvée* neutre pour R_j. Dans ce cas, il sera nécessaire de tester l'application de R_j après toute application de R_i. Si ce graphe $\mathcal{G}_D$ est le graphe complet, alors nous obtenons l'algorithme standard de résolution en marche avant. Le but de l'opération étant de supprimer le plus de neutres possible, nous allons écrire une fonction `Supprime ?` qui, étant donné deux règles R_i et R_j, ne répond `oui` que dans le cas où R_i est un neutre pour R_j. En ne retirant du graphe de dépendance complet que les arcs pour lesquels `Supprime ?`(R_i, R_j) aura répondu `oui`, nous obtiendrons un algorithme de recherche en marche avant adéquat et complet, d'autant plus efficace que nous aurons pu supprimer plus d'arcs.

Algorithme 10: `Supprime1 ?`(R_1, R_2)

Données : Deux règles R_1 et R_2.
Résultat : N'a le droit de répondre `oui` que si R_1 est un neutre pour R_2.

retourner `non`;

L'algorithme de résolution en marche avant obtenu par l'utilisation de la fonction `Supprime1 ?` est l'algorithme standard ; en répondant `non` pour chaque couple de règles, aucun arc n'est supprimé dans le graphe de dépendance. La propriété qui suit nous donnera un algorithme plus efficace.

Un critère simple de neutralité

Propriété 6.6 *Soient* R_1 *et* R_2 *deux règles. Alors si tous les sommets de la conclusion de* R_1 *sont incompatibles avec les sommets de l'hypothèse de* R_2*,* R_1 *est un neutre pour* R_2.

Preuve: Si tous les sommets de la conclusion de R_1 sont incompatibles avec les sommets de l'hypothèse de R_2, alors soit G un graphe quelconque et G' un graphe obtenu à partir de G

par une application de R_1. Soit π une projection de l'hypothèse de R_2 dans G'. Alors toutes les images par π des sommets de cette hypothèse sont dans la partie de G' correspondant à G. Cette projection existait donc avant l'application de R_1, et R_1 est donc un neutre pour R_2. □

Cette propriété s'implémente de façon immédiate à travers l'algorithme suivant :

Algorithme 11: `Supprime2 ?`(R_1, R_2)

Données : Deux règles R_1 et R_2.

Résultat : N'a le droit de répondre `oui` que si R_1 est un neutre pour R_2.

```
pour x ∈ Hypothese(R2) faire
    pour y ∈ Conclusion(R1) faire
        si Compatible(y, x) alors
            retourner non;
retourner oui;
```

Cet algorithme détecte plus de neutres que le premier, pour un léger (puisque nous sommes dans un problème indécidable) surcoût en temps : cet algorithme a une complexité en $\mathcal{O}(n_1 \times n_2 \times T(m))$ en temps, où n_1 est la taille de la conclusion de R_1 et n_2 la taille de l'hypothèse de R_2 (où $T(m)$ est la complexité de la comparaison des étiquettes dans le support de taille m). Le problème est que cet algorithme est loin de détecter *tous* les neutres (mais cela ne modifie en rien l'adéquation et la complétude de l'algorithme de résolution). Par exemple, prenons les deux règles R_1 et R_2 données par :

$$\begin{array}{lccc} R_1 & \text{SI } [A : *] & & \text{ALORS } [B : *] \\ R_2 & \text{SI } [B : *] \rightarrow (r) \rightarrow [C : *] & & \text{ALORS } \ldots \end{array}$$

où R_1 et R_2 sont définies sur un support tel que les types A, B et C sont incomparables deux à deux. Il est clair que R_1 est un neutre pour R_2 : l'application de R_1 crée un sommet $[B : *]$, déconnecté de tous les autres sommets du graphe. En particulier, ce sommet ne peut pas être voisin par (r) d'un sommet $[C : *]$, et donc aucune projection de l'hypothèse de R_2 ne peut utiliser ce sommet $[B : *]$. Cependant, la fonction `Supprime2 ?` ne peut pas détecter ce cas de neutralité, le sommet de la conclusion de R_1 étant compatible avec le sommet $[B : *]$ de l'hypothèse de R_2. Il apparaît alors qu'une réflexion sur le voisinage s'impose pour trouver davantage de neutres.

Propriété 6.7 *La fonction* `Supprime2 ?` *ne détecte pas tous les neutres.*

De plus, bien que cette fonction `Supprime2` nous permet (éventuellement) de réduire le nombre de règles à tester au cours de la recherche, elle ne nous est d'aucune utilité pour réduire l'espace de recherche quand on doit tester la possibilité d'appliquer une règle. Pourtant, toute *nouvelle* projection de l'hypothèse de R_2 dans le graphe G' doit étendre la projection d'un sommet x de l'hypothèse de R_2 dans un sommet y, compatible avec

x, qui a été rajouté par l'application de R_1. Conserver (compiler) ces paires de sommets compatibles nous permettra donc de restreindre l'espace de recherche. L'algorithme suivant nous donne ces paires de sommets :

Algorithme 12: `Declencheurs1`(R_1, R_2)

Données : Deux règles R_1 et R_2.

Résultat : L'ensemble des paires compatibles (x, y), où x est un sommet de l'hypothèse de R_2, y est un sommet de la conclusion de R_1, et y est compatible pour x.

```
Paires ← ∅;
pour x ∈ Hypothese(R2) faire
    pour y ∈ Conclusion(R1) faire
        si Compatible(y, x) alors
            Paires ← Paires ∪ {(Ident(x), Ident(y))};
retourner Paires;
```

Si l'appel de `Declencheurs1`(R_i, R_j) retourne l'ensemble vide, alors R_i est un neutre pour R_j, et nous pouvons donc supprimer l'arc de R_i vers R_j. Sinon, nous étiquetons cet arc par l'ensemble retourné par `Declencheurs1`(R_i, R_j). Dans ce cas, si R_i est appliquée à un graphe G de taille n pour obtenir le graphe G', alors pour toute paire (n_i, n_j) d'identificateurs appartenant à l'étiquette de l'arc (R_i, R_j), il suffira de tester s'il existe une projection de l'hypothèse de R_j qui étend la projection du sommet numéroté n_j dans le sommet numéroté $n + n_i$. Nous supposons ici que les sommets de la conclusion de R_i ont été rajoutés, dans le même ordre, à la fin des sommets de G.

La complexité de `Declencheurs1`(R_i, R_j) est cette fois-ci en $\Theta(n_i \times n_j)$ (si on suppose que la complexité de la comparaison est en temps constant). La construction du graphe de dépendance, pour un ensemble de n règles, sera en $\Theta(n^2 \times n_i \times n_j)$. Ce travail initial peut être assez coûteux, mais si nous nous plaçons dans un cadre où nous disposons d'une bibliothèque de règles, utilisée pour résoudre plusieurs problèmes, ce surcoût sera rapidement amorti. En effet, pour qu'il n'y ait aucun gain d'efficacité, il faudrait :

1. que `Declencheurs1`(R_i, R_j) ne retourne $\emptyset$ pour aucun couple (ou presqu'aucun couple) de règles ;
2. que pour chaque couple (R_1, R_2) tel que R_1 est un déclencheur de R_2, presque tous les sommets de de l'hypothèse de R_2 sont compatibles avec presque tous les sommets de la conclusion de R_1.

6.3.3 Un critère de neutralité optimal

Nous exposons maintenant une caractérisation exacte du critère de neutralité :

Théorème 6.5 *Soient R et T deux règles de graphes. La règle R est un déclencheur possible pour la règle T si et seulement s'il existe :*

- *une projection π d'un sous-graphe non vide H de l'hypothèse de T dans la conclusion de R (nous notons alors $N(H)$ les sommets dans l'hypothèse de T qui sont voisins d'un sommet de H mais qui n'appartiennent pas à H),*
- *une partition $\oplus_N = \{N_1, \ldots, N_k\}$ des sommets de $N(H)$,*
- *une partition $\oplus_F = \{F_1, \ldots, F_{k+1}\}$ des sommets de la frontière F de R,*

qui respectent les deux critères suivants :

1. *Pour toute arête étiquetée par i entre un sommet h de H et un sommet n de $h(n)$, soit $N_j \in \oplus_N$ tel que $n \in N_j$. Alors il existe une arête étiquetée par i entre $\pi(h)$ et un sommet f de F_j dans R.*
2. *Pour chaque $N_j \in \oplus_N$, le support permet de créer un sommet s_j dont l'étiquette est plus spécifique que chacune des étiquettes des sommets de N_j et de F_j.*

Démonstration

Intuitivement, la projection π exprime qu'une partie de l'hypothèse de T devra se rajouter dans la partie d'un graphe correspondant à ce qui a été rajouté par l'application de R, tandis que les partitions indiquent que les sommets appartenant à N_j et les sommets appartenant à F_j pourront se projeter dans un même sommet du graphe G.

Preuve: Nous prouverons tout d'abord que, si on se donne de tels objets π, $\oplus_N$ et $\oplus_F$ entre deux règles R et T, alors R est un déclencheur possible de T. La preuve de cette partie de l'équivalence ($\Rightarrow$) se fera en construisant un graphe G tel qu'une certaine application de R crée une nouvelle application de T.Cette partie de la preuve est illustrée dans la FIG 6.14. La deuxième partie de l'équivalence ($\Leftarrow$) supposera l'existence d'un graphe quelconque G tel que l'application de R crée une nouvelle application de T. Nous construirons alors la projection π, et les partitions $\oplus_N$ et $\oplus_F$, et vérifierons que les critères 1. et 2. sont respectés.

($\Rightarrow$) Supposons qu'il existe une telle projection π et de telles partitions $\oplus_N$ et $\oplus_F$ entre R et T. Nous construisons le graphe initial G et le graphe G' obtenu par une application de R sur G de la façon suivante :

1. nous partons de l'hypothèse de R pour définir le graphe G ;
2. pour chaque $N_j \in \oplus_N$, les sommets de F_j dans la frontière de R sont fusionnés, et l'étiquette s_j du sommet résultant est plus spécifique que celle de chacun des sommets de F_j (ce qui est exigé par l'opérateur de fusion), mais aussi que celle de chacun des sommets de N_j (cette étiquette de s_j existe grâce au critère 2. du théorème) ;
3. nous faisons l'union disjointe du graphe ainsi obtenu avec le sous-graphe H' de l'hypothèse de T qui contient tous les sommets qui ne sont ni dans H, ni dans $N(H)$;
4. pour chaque arête étiquetée par i entre un sommet e de H' et un sommet n de $N(H)$ (nous supposons $n \in N_j$) dans T, nous rajoutons une arête étiquetée par i entre le sommet correspondant à e et le sommet s_j ;

Nous exhibons tout d'abord une projection π_1 de l'hypothèse de R dans ce graphe G (il peut y en avoir d'autres, mais seule celle-ci nous intéresse). Le sous-graphe de G obtenu à l'étape 2. correspond à une fusion de certains des sommets de l'hypothèse de R, qui

spécialise davantage les étiquettes des sommets fusionnés. Cette opération conserve donc une projection de l'hypothèse de R dans G, définie par :

- si $x \in F_j$ (pour $1 \leq j \leq k$), alors $\pi_1(x) = s_j$;
- sinon, $\pi_1(x) = Id(x)$.

Cette projection π_1 nous permet de construire le graphe G', obtenu par l'application de R à G suivant π_1. Nous montrerons qu'il existe une application π_2 de l'hypothèse de T dans G', qui est bien une projection, et qui ne correspond pas à une projection de l'hypothèse de T dans G. Soit π_2 l'application associant aux sommets de l'hypothèse de R_2 des sommets de G' définie par :

- si $x \in H$, alors $\pi_2(x) = \pi(x)$ (plus précisément, il s'agit du sommet de G' rajouté par l'application de R qui correspond à $\pi(x)$) ;
- si $x \in N(H)$, alors soit $N_j/x \in N_j$, alors $\pi_2(x) = s_j$;
- si $x \in H'$, alors $\pi_2(x) = Id(x)$ (plus précisément, il s'agit du sommet correspondant à x qui a été rajouté à l'étape 3.).

Si cette application est une projection, alors il s'agit bien d'une projection qui n'est pas entièrement dans G (puisque H est non vide, il existe au moins un sommet qui a pour image un sommet rajouté par l'application de R). Il reste donc à prouver que π_2 est une projection.

Tout d'abord, nous pouvons voir que la restriction de π_2 au sous-graphe engendré par les sommets de H et les sommets de H' est bien une projection. En effet, π est une projection de H dans la partie de G' rajoutée par l'application de R et Id est bien une projection de H' dans la partie de G' qui lui correspond. Comme il n'y a aucune arête entre H et H' (sinon le sommet de H' voisin d'un sommet de H devrait appartenir, par hypothèse, à $N(H)$), ces deux projections définissent bien une projection du sous graphe engendré par les sommets de H et de H' dans G'.

Reste à étendre cette projection aux sommets de $N(H)$. Nous voyons tout d'abord que π_2 associe à tout sommet de $N(H)$ un sommet qui lui est compatible : ceci est assuré par la définition des sommets s_j à l'étape 2. de la construction de G'. Pour prouver que π_2 est un homomorphisme, il nous reste à prouver que pour toute arête xy, étiquetée par i, incidente à un sommet x de $N(H)$ (nous supposons $x \in N_j$), il existe une arête étiquetée par i entre $\pi_2(x) = s_j$ et $\pi_2(y)$. La partition de l'hypothèse de T nous impose un des trois cas suivants :

1. Soit $y \in N(H)$: ceci est impossible. En effet, un sommet qui est dans la frontière d'une règle est nécessairement un sommet concept (une conséquence immédiate de la définition des règles). La compatibilité des sommets de $N(H)$ avec des sommets de la frontière impose donc à ces sommets d'être des sommets concept. Comme nos graphes sont bipartis, il n'y a aucune arête entre ces sommets.
2. Soit $y \in H$: alors, par hypothèse (il s'agit du critère 1. dans le théorème), il existe une arête étiquetée par i entre $\pi(y)$ (dans la conclusion de R) et un sommet de F_j (dans la frontière de R). Comme les sommets de F_j ont été projetés par π_1 dans $s_j = \pi_2(x)$, cette arête est nécessairement rajoutée entre s_j et $\pi_2(y)$ dans G' (c'est le mécanisme d'application de règles).

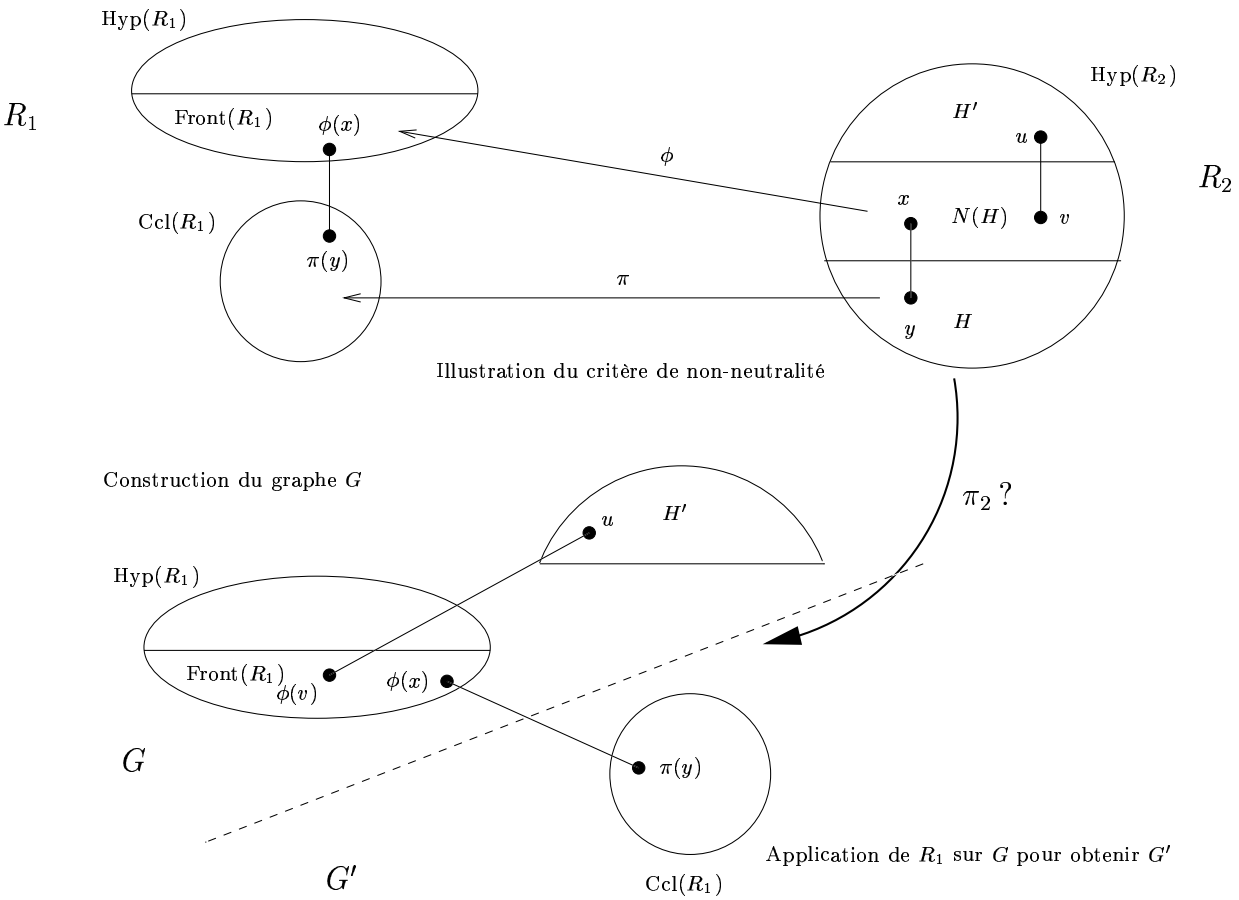


FIG. 6.14 – Preuve du théorème : trouver une projection π_2 de Hyp(R_2) dans G'.

3. Soit $y \in H'$: dans ce cas, la présence d'une arête étiquetée par i entre $Id(y)$ et s_j est assurée par l'étape 4. de la construction de G.

Donc π_2 est bien une projection de T dans G' qui n'est pas une projection dans G. L'existence d'un tel graphe G prouve que R n'est pas neutre pour T : R est un déclencheur possible de T. ◊

($\Leftarrow$) Nous supposons maintenant que R est un déclencheur possible de T. Alors il existe un graphe G, tel que, pour G' obtenu en appliquant R à G suivant une projection π_1, il existe une projection π_2 de l'hypothèse de T dans G' qui n'est pas entièrement contenue dans G. Nous allons en déduire une projection π et deux partitions $\oplus_N$ et $\oplus_F$ qui satisfont les deux critères du théorème.

Nous notons H le sous-graphe de l'hypothèse de T dont les images par π_2 sont les sommets de G' rajoutés par l'application (suivant π_1 de R à G. Nous remarquons que H est non vide (sinon la projection π_2 serait entièrement dans G), et que la restriction de π_2 à H définit bien une projection de ce sous-graphe non vide H de l'hypothèse de T dans la conclusion de R. Soit π cette projection.

Nous considérons maintenant G_f le sous-graphe de G engendré par les images par π_1

de la rontière de R_1. Nous allons tout d'abord prouver que $\pi_1^{-1}(G_f)$ induit une partition des sommets de la frontière de R_1 (immédiat, puisque, par définition d'une application, un sommet ne peut avoir deux images), et que $\pi_2^{-1}(G_f)$ induit une partition des sommets de $N(H)$.

Soit $N(H)$ le voisinage de H. Nous remarquons tout d'abord que, si x est un sommet de $N(H)$, alors $\pi_2(x)$ est un sommet de G' (et même de G) sur lequel ont été projetés (par π_1) des sommets $y_1, \ldots, y_p$ de la frontière de R_1. En effet, puisque x est voisin d'un sommet y de H, et $\pi_2(y)$ appartient à la partie de G' rajoutée par l'application de R_1, alors $\pi_2(x)$ est un voisin de $\pi_2(y)$ dans G (sinon x appartiendrait à H). Et l'application de R_1 suivant π_1 n'a pu rajouter une arête entre un sommet $\pi_2(y)$ de la partie rajoutée et un sommet $\pi_2(x)$ dans G que si π_1 a projeté au moins un sommet de la frontière de R_1 dans G (c'est le mécanisme d'application d'une règle). Nous choisissons donc arbitrairement un sommet parmi les sommets $y_1, \ldots, y_p$ de la frontière, et définissons $\phi(x) = y_1$.

Nous avons donc construit une projection π d'un sous-graphe H non vide de l'hypothèse de R_2 dans la conclusion de R_1, et une application ϕ de $N(H)$ dans la frontière de R_1. Il ne nous reste plus qu'à vérifier que ces deux applications respectent bien les critères 1. et 2. du théorème. □

Complexité

Le théorème 6.5 nous permet également de donner la complexité du problème NEUTRE ?.

NEUTRE ?
Données : Deux règles de graphes élémentaires R_1 et R_2.
Question : VRAI si et seulement si R_1 est neutre pour R_2, *i.e.* R_1 n'est pas un déclencheur de R_2.

Corollaire 6.2 NEUTRE ? *est un problème de décision co-NP-complet.*

Preuve: Voir que la caractérisation du Th. 6.5 nous indique le certificat polynomial (la projection et les deux partitions) prouvant l'appartenance du co-problème de NEUTRE ?, que nous appellerons DÉCLENCHE ?, à la classe NP. Pour prouver la complétude, nous montrons que DÉCLENCHE ? est une généralisation de PROJECTION ? : en effet, un graphe H se projette dans un graphe G si et seulement si la règle R_1 dont la conclusion, G, déconnectée de l'hypothèse, déclenche la règle R_2 dont l'hypothèse est le graphe H. □

Notons aussi qu'il existe une transformation linéaire simple d'une instance de DÉCLENCHE ? en une instance de PROJECTION ? qui conserve toutes les projections π de l'hypothèse de R_2 dans la conclusion de R_1 : ceci nous permet de calculer les déclencheurs de manière efficace, en utilisant les algorithmes du Chap. 5.

Utilisation du graphe de dépendances

Enfin, notons que l'intégration du graphe de dépendances ainsi obtenu dans le chaînage avant (Alg. 9) est immédiate :

- le graphe $GD(\mathcal{K})$ est construit préalablement à l'appel de l'algorithme, en considérant la base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R})$. Notons que G peut être introduit dans ce graphe de dépendances, en le considérant comme une règle dont l'hypothèse est vide ;
- dès qu'une requête H est posée, l'algorithme *MA-Déductible*$(\mathcal{K}, H)$ commence par intégrer H dans le graphe de dépendances $GD(\mathcal{K})$, H étant vue comme une règle à conclusion vide, le graphe $GD(\mathcal{K}, H)$ considéré est la restriction de ce graphe au sous-graphe engendré par les sommets étant sur un chemin de G à H ;
- à chaque étape E_i de l'algorithme, au lieu de considérer toutes les règles de $\mathcal{R}$, nous ne considérons que celles déclenchées par les règles appliquées à l'étape E_{i-1} (nous pouvons ne considérer que les applications utiles ou les applications non redondantes) ;
- les projections de l'hypothèse d'une règle déclenchée sont plus petites : il suffit de chercher à étendre les projections compilées.

Les trois propriétés suivantes, immédiates, permettent de caractériser des ensembles de règles décidables :

Propriété 6.8 *Si le graphe de dépendances est une forêt enracinée (le Th. 6.3 prouve que même une boucle restreinte à un arc est suffisante à générer un problème semi-décidable), alors le problème* $\mathcal{SR}$-DÉDUCTION *est décidable.*

Propriété 6.9 *Si les sommets de chaque circuit du graphe de dépendances forment un ensemble de règles à expansion finie, alors le problème* $\mathcal{SR}$-DÉDUCTION *est décidable.*

Remarquons enfin que le Th. 6.5, qui traduit exactement les possibilités de projection, ne s'applique pas tel quel lorsque nous considérons un modèle de dérivation dans lequel les graphes produits sont mis sous forme normale. Nous avons dans ce cas envisagé plusieurs solutions, mais n'en avons jugé aucune vraiment satisfaisante.

Chapitre 7

Dérivations sous contraintes

Sommaire

Ce chapitre est consacré à l'étude des autres modèles de la famille $\mathcal{SG}$, à savoir $\mathcal{SGC}$, $\mathcal{SRC}$, $\mathcal{SEC}$ et $\mathcal{SREC}$. Ces modèles sont obtenus par l'introduction de *contraintes*[1], que nous étudions d'abord comme ajout au modèle $\mathcal{SG}$. Dans ce modèle ($\mathcal{SGC}$), les contraintes sont utilisées pour valider une base de faits : les contraintes nous permettent d'exprimer la notion de *cohérence* d'une base de faits.

Nous présentons ensuite deux façons d'intégrer contraintes et règles. Pour chacun des modèles obtenus, nous déterminons la décidabilité/complexité du problème de déduction associé (et du problème de cohérence le cas échéant). Dès que les règles interviennent dans les raisonnements, les problèmes de décision associés ne sont plus décidables, mais nous exhibons des cas particuliers, liés aux règles à expansion finie présentées dans le chapitre précédent (Def. 6.9), pour lesquels le problème de déduction associé est décidable. En particulier, nous verrons qu'une restriction aux règles *range restricted* (Def. 6.10) ramène de

[1] Ces contraintes, qui sont des graphes colorés, ne doivent pas être confondues avec les contraintes d'un réseau de contraintes présentées au Chap. 5.

plus la classe de complexité de tous les problèmes de décision considérés dans la hiérarchie polynomiale.

Lorsque dans ce chapitre, nous parlerons de « graphe », ce pourra être un objet quelconque d'un formalisme de $\mathcal{SG}$. Ce n'est que lorsque la sémantique logique intervient, que nous devons préciser quels sont les objets considérés.

7.1 Contraintes : le modèle $\mathcal{SGC}$

Dans le modèle $\mathcal{SGC}$, la base de connaissances est constituée d'un support $\mathcal{S}$, d'une base de faits $\mathcal{G}$ et d'un ensemble de contraintes $\mathcal{C}$. La base de faits est dite cohérente si elle satisfait toutes les contraintes. En présence de contraintes, le problème de déduction n'a de sens que pour une base de faits cohérente : aucune requête ne peut être posée à une base de faits incohérente.

7.1.1 Contraintes positives et négatives

Nous utilisons deux types de contraintes : les contraintes positives et les contraintes négatives. Ces contraintes sont représentées par des graphes colorés.

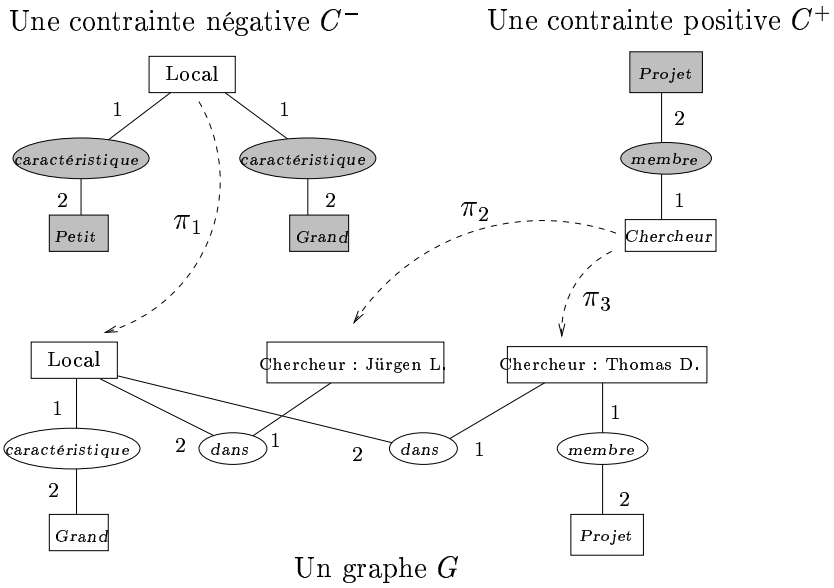


FIG. 7.1 – Une base de faits incohérente.

Définition 7.1 *Une contrainte positive (resp. négative) C est définie par un graphe élémentaire coloré. La partie $C_{(0)}$ est appelée le déclencheur de la contrainte, et la partie $C_{(1)}$ son obligation (resp. interdiction). Un graphe G π-viole une contrainte positive (resp. négative) C si π est une projection du déclencheur de C dans la forme irredondante de G (resp. dans G) qui ne peut pas être étendue (resp. qui peut être étendue) à une projection*

de C toute entière. G viole C si G π-viole C pour une projection π. Sinon, nous dirons que G satisfait C.

La FIG. 7.1 représente deux contraintes, une contrainte négative, C^-, qui peut s'exprimer par « pour tout local, il ne faut pas qu'il soit à la fois grand et petit », et une contrainte positive, C^+, qui peut s'exprimer par « tout chercheur doit être membre d'un projet » Notons que le graphe G est irredondant.

Le déclencheur de C^- peut se projeter (uniquement par la projection π_1) dans le graphe G, mais cette projection ne peut pas s'étendre à une projection de C^- toute entière : G satisfait la contrainte C^-. Le déclencheur de C^+ peut se projeter (par les projections π_2 et π_3) dans le graphe G, la projection π_3 peut s'étendre à une projection de C^+ toute entière, mais pas la projection π_2 : le graphe G viole la contrainte C^+.

Contraintes et irredondance

Pour définir la notion de violation d'une contrainte par un graphe G, nous prenons en compte G s'il s'agit d'une contrainte négative, et la forme irredondante de G dans le cas d'une contrainte positive. Pourquoi cette différence ? Imaginons que nous considérions G et non pas sa forme irredondante : on pourrait alors avoir deux graphes équivalents, tel que l'un satisfait une contrainte positive et l'autre la viole.

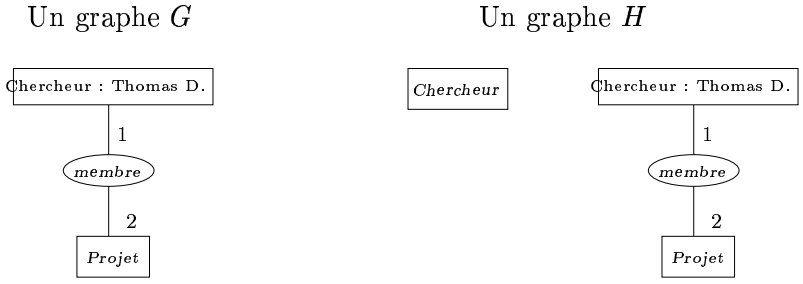


FIG. 7.2 – Deux graphes équivalents, jugés différemment par une contrainte.

La FIG. 7.2 donne un exemple de tels graphes : G satisfait la contrainte C^+ de la FIG. 7.1, mais le graphe équivalent H (qui est redondant), obtenu en faisant l'union disjointe de G et du déclencheur de C^+, viole C^+. Pour éviter d'avoir des graphes équivalents qui ne sont pas jugés de la même façon du point de vue de la cohérence (et dont on ne peut donc pas déduire les mêmes connaissances), nous avons choisi de définir la notion de satisfaction/violation d'une contrainte positive relativement à la forme irredondante des graphes.

Nous verrons que ce problème ne se pose pas avec les contraintes négatives. Nous avons tout d'abord besoin de définir la notion de contraintes équivalentes :

Définition 7.2 (Contraintes équivalentes) *Soient C_1 et C_2 deux contraintes (qui peuvent être positives ou négatives). Alors C_1 et C_2 sont dites équivalentes si, pour tout graphe G, G viole C_1 ssi G viole C_2.*

Nous voyons tout d'abord que la coloration des sommets d'une contrainte négative est une notion assez artificielle :

Lemme 7.1 *Soient $C_1 = (G, \kappa_1)$ et $C_2 = (G, \kappa_2)$ deux contraintes négatives obtenues par la coloration d'un même graphe G. Alors C_1 et C_2 sont deux contraintes équivalentes.*

Preuve: Il suffit de voir que toute contrainte négative est notamment équivalente à la contrainte négative obtenue en la coloriant complètement par la couleur 1 (la partie « déclencheur » est vide). En effet, « aucune projection d'un sous-graphe H' de H dans G ne s'étend à une projection de H » se traduit par « il n'existe pas de projection de H dans G », que l'on peut encore traduire par « dans tous les cas (*i.e.* à chaque fois qu'on a une projection du graphe vide), il n'existe pas de projection de H dans G ». □

La coloration des contraintes négatives est donc, du point de vue de leur sémantique, superflue. Ceci veut-il dire qu'il faut supprimer cette coloration des contraintes négatives ? Nous pensons que non, car cette coloration pourrait être utilisée par l'algorithme qui recherche les projections : chercher, pour chaque projection du déclencheur, si cette projection peut s'étendre à une projection du graphe tout entier. L'utilisateur peut ainsi se servir de cette coloration comme d'une possibilité d'ordonnancement des sommets pour la projection. Considérons la contrainte négative C^- de la FIG. 7.1. Si l'utilisateur sait qu'il y a très peu de sommets concepts de type *Local* dans le graphe G, alors la coloration en blanc du sommet de ce type dans C^- peut être vue comme une optimisation déclarée par cet utilisateur[2].

Propriété 7.1 *Soient G_1 et G_2 deux graphes équivalents. Alors G_1 viole une contrainte négative C si et seulement si G_2 viole C.*

Preuve: En effet, considérons deux graphes équivalents G_1 et G_2. Supposons que G_1 π-viole une certaine contrainte négative C. Ceci est équivalent à dire qu'il existe une projection π de C dans G_1 (voir preuve du Lem. 7.1). Or il existe une projection, π_1, de G_1 dans G_2, donc $\pi_1 o \pi$ est une projection de C dans G_2, d'où on déduit (encore la preuve du Lem. 7.1) que G_2 viole également C. □

Si nous ne considérons qu'un ensemble de contraintes négatives, il ne sera donc pas nécessaire de mettre la base de faits sous sa forme irredondante.

Les contraintes négatives : un cas particulier de contraintes positives

Montrons maintenant que les contraintes négatives sont en fait un cas particulier de contraintes positives.

Lemme 7.2 *Toute contrainte négative est équivalente à une contrainte positive.*

[2]On pourrait imaginer une telle aide à la recherche de projections dans le modèle $\mathcal{SG}$, par coloration de la requête (et même avec plus de deux couleurs), permettant à celui qui pose une requête de contraindre l'algorithme de recherche à adopter un certain ordonnancement de variables.

Preuve: Soit $C = (H, \kappa)$ une contrainte négative, que l'on peut voir, sans perte de généralité, comme entièrement colorée par 1 (Lem. 7.1). Alors nous pouvons construire (à une modification du support près) une contrainte positive $C' = (H', \kappa')$ équivalente à C de la façon suivante :

- H' est obtenu à partir de H en lui ajoutant un nouveau sommet concept de type $\bot$;
- ce type $\bot$ est un type particulier du (nouveau) support, qui n'apparaît dans aucun graphe de faits (ni dans la conclusion d'aucune règle) ;
- κ' colore tous les sommets de H' correspondant à ceux de H par 0, et le nouveau sommet par 1.

Voir qu'un graphe (qui ne comprend donc aucun sommet de type $\bot$) viole C si et seulement s'il viole C' : en effet, « il n'existe aucune projection de H dans G » est équivalent à « pour toute projection de H dans G, il faut pouvoir trouver le sommet $\bot$ dans G ». □

Nous pouvons donc considérer les contraintes négatives comme un cas particulier de contraintes négatives. L'étude des classes de complexité des problèmes « est-ce-que G satisfait une contrainte négative C^- » et « est-ce-que G satisfait une contrainte positive C^+ » (Th. 7.3) nous apprendra que la réciproque n'est pas vraie. En d'autres termes :

Propriété 7.2 *À moins que $\Pi_2^P = co\text{-}NP$, les contraintes positives sont une généralisation stricte des contraintes négatives.*

Dans la suite de ce chapitre, lorsque nous parlerons d' « une contrainte » ou d' « un ensemble de contraintes » sans autre précision, nous considérerons implicitement qu'il s'agit d'une, ou d'un ensemble de contraintes positives : certaines de ces contraintes pourront être équivalentes à des contraintes négatives.

Cohérence et déduction dans $\mathcal{SGC}$

Nous définissons maintenant le problème de déduction associé au modèle $\mathcal{SGC}$:

Définition 7.3 (Cohérence/Déduction dans $\mathcal{SGC}$) *Une base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$ est* cohérente *si $\mathcal{G}$ satisfait toutes les contraintes de $\mathcal{C}$ (dans le cas contraire, $\mathcal{K}$ est dite* incohérente*). Un graphe requête H peut être déduit de $\mathcal{K}$ si $\mathcal{K}$ est cohérente et si H peut être déduit de $(\mathcal{S}, \mathcal{G})$ (il existe une projection de H dans $\mathcal{G}$).*

Les problèmes de décision associés seront appelés $\mathcal{SGC}$-COHÉRENCE ($\mathcal{K}$ est-elle cohérente ?) et $\mathcal{SGC}$-DÉDUCTION ($\mathcal{K}$ est-elle cohérente, et si oui, H est-il déductible de $\mathcal{K}$?).

Notons qu'un graphe requête H qui viole une contrainte de la base $\mathcal{K}$ peut cependant être déductible de $\mathcal{K}$. Ceci ne nous semble pas poser de problème, puisque la contrainte violée est forcément une contrainte positive non équivalente à une contrainte négative (sinon H ne pourrait pas être déduite de $\mathcal{K}$). On peut alors voir H comme une représentation partielle de la connaissance déductible de $\mathcal{K}$.

7.1.2 Sémantique logique des contraintes

Nous nous intéressons maintenant à la traduction de la notion de cohérence/incohérence en logique du premier ordre.

Le cas des contraintes négatives

Si l'on ne considère que les contraintes négatives, la correspondance avec la déduction logique est immédiate et repose sur l'adéquation et la complétude de la projection relativement à la sémantique Φ (Th. 3.5). Intuitivement, un graphe G viole une contrainte négative C si et seulement si l'information représentée par C est déductible de l'information représentée par G.

Théorème 7.1 (Violation de contrainte négative et déduction) *Un graphe élémentaire G viole une contrainte négative $C = (C', \rho)$ si et seulement si $\Phi(\mathcal{S}), \Phi(G) \models \Phi(C')$, où C' est le graphe non coloré sous-jacent à C et $\Phi(C')$ est la formule qui lui est associée dans le formalisme considéré de $\mathcal{SG}$.*

Remarquons que si nous considérons d'autres formalismes de $\mathcal{SG}$ pour construire le modèle $\mathcal{SGC}$, il peut être nécessaire de considérer le graphe G sous forme normale, ou d'utiliser une sémantique qui traduit exactement la projection telle que Ψ (voir Sect. 4.2.3).

Le cas des contraintes positives

La notion de violation d'une contrainte positive par un graphe pourrait être traduite en logique du premier ordre en traduisant la notion de « projection » en une notion de « substitution logique » entre les formules associées au graphe et à la contrainte (voir la notion de S-substitution de [Chein and Mugnier, 1992] et l'utilisation qui en est faite dans [Baget and Mugnier, 2001a]). Nous trouvons plus intéressant d'établir une correspondance avec la logique en considérant les contraintes comme des règles.

L'idée est la suivante : un graphe G satisfait une contrainte positive C si et seulement si, lorsque C est vue comme une règle, toutes les applications de C sur G produisent un graphe équivalent à G. Ceci est exprimé par la Prop. 7.3. La Prop. 7.4 étend cette propriété à une dérivation quelconque, ce qui nous permettra d'utiliser la sémantique logique associée aux règles. Nous avons tout d'abord besoin du lemme suivant, dû à [Cogis and Guinaldo, 1995] :

Lemme 7.3 *Soit G un graphe et $irr(G)$ son sous-graphe irredondant équivalent (si G n'est pas redondant, on a $irr(G) = G$). Alors il existe un* pliage *f de G dans $irr(G)$, c'est-à-dire une projection f de G dans $irr(G)$, telle que la restriction de f aux sommets de $irr(G)$ est l'identité (pour chaque sommet x de $irr(G)$, $f(x) = x$).*

Propriété 7.3 *Un graphe G π-viole une contrainte positive C si et seulement si, en considérant C comme une règle, l'application de C sur G selon π produit un graphe qui n'est pas équivalent à G.*

Preuve: Nous prouverons tout d'abord ($\Leftarrow$) que, si C satisfait G, alors toute C-dérivation immédiate de G est équivalente à G, puis ($\Rightarrow$) que si C π-viole G, alors $\lambda(G, C, \pi)$ n'est pas équivalent à G.
($\Leftarrow$) Prenons C et G tels que G satisfait C. Soit π_0 une projection de $C_{(0)}$ dans G, et soit G' le graphe obtenu par l'application de C (maintenant considéré comme une règle) sur G selon π_0. Construisons la projection suivante π' de G' dans G : pour chaque sommet v, $\pi'(v) = v$ si v appartient à G ; sinon v est une copie d'un sommet w de $C_{(1)}$, et, si π est l'une des projections de C dans G qui étend π_0, on pose $\pi'(v) = \pi(w)$. π' est une projection de G' dans G, et comme G se projette trivialement dans G', ces deux graphes sont équivalents.
($\Rightarrow$) Supposons maintenant que G π-viole C. Comme la violation de contrainte est définie par rapport à la forme irredondante d'un graphe, nous pouvons considérer sans perte de généralité que G est irredondant. Notons G' le graphe obtenu par l'application de C (à nouveau considérée comme une règle) à G suivant π. Prouvons que « G et G' sont équivalents » mène à une contradiction. Si G' est équivalent à G, il existe une projection de G' dans G. Comme G est un sous-graphe irredondant de G', il existe un pliage, soit f, de G' dans G (Lem. 7.3). Construisons maintenant une projection π' de C dans G : pour tout sommet x de $C_{(0)}$, posons $\pi'(x) = f(\pi(x))$, sinon, soit x' la copie de x dans G', posons $\pi'(x) = f(x')$. Puisque, pour tout x de $C_{(0)}$, $f(\pi(x) = \pi(x)$, π' étend π. Ceci contredit l'hypothèse « G π-viole C ». G' n'est donc pas équivalent à G. □

Nous étendons immédiatement cette propriété à des dérivations de longueur arbitraire, utilisant un nombre quelconque de règles/contraintes :

Propriété 7.4 *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$ une base de connaissances. Alors $\mathcal{K}$ est cohérente si et seulement si toute $\mathcal{C}$-dérivation de $\mathcal{G}$ est équivalente à $\mathcal{G}$.*

Preuve: Prouvons successivement les deux sens de cette équivalence :
($\Rightarrow$) Soit $\mathcal{G} = G_0, \ldots, G_k$ une $\mathcal{C}$-dérivation quelconque de G. Si $\mathcal{K}$ est cohérente, alors $\mathcal{G}$ satisfait chacune des contraintes $C \in \mathcal{C}$. Alors la propriété 7.3 assure que chaque G_i, $1 \leq i \leq k$, est équivalent à G_{i-1}, donc par transitivité, est équivalent à G.
($\Leftarrow$) Réciproquement, si toute $\mathcal{C}$-dérivation de $\mathcal{G}$ est équivalente à $\mathcal{G}$, alors, en particulier, toute $\mathcal{C}$-dérivation immédiate de $\mathcal{G}$ produit un graphe équivalent à $\mathcal{G}$. Donc, pour chaque règle $C \in \mathcal{C}$, toute $\{C\}$-dérivation immédiate de $\mathcal{G}$ produit un graphe équivalent à G. Donc (Prop. 7.3), G satisfait chacune des contraintes $C \in \mathcal{C}$. Donc $\mathcal{K}$ est cohérente. □

Nous pouvons donc, grâce au Th. 6.1, en déduire immédiatement une traduction logique de la notion de cohérence d'un graphe pour un ensemble de règles :

Théorème 7.2 (Violation de contrainte positive et déduction) *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$ une base de connaissances (dans le formalisme des graphes élémentaires). Alors $\mathcal{K}$ est incohérente si et seulement si il existe un graphe élémentaire G' tel que l'on ait $\Phi(\mathcal{S}), \Phi(\mathcal{G}), \Phi(\mathcal{C}) \models \Phi(G')$ et pas $\Phi(\mathcal{S}), \Phi(\mathcal{G}) \models \Phi(G')$ (où $\Phi(\mathcal{C})$ est la formule logique associée à $\mathcal{C}$, vu comme un ensemble de règles, dans le modèle $\mathcal{SR}$).*

Comme pour les contraintes négatives, nous remarquons qu'il peut être nécessaire, suivant le formalisme de $\mathcal{SG}$ utilisé pour construire $\mathcal{SGC}$, de considérer le graphe $\mathcal{G}$ sous sa forme normale ou bien d'utiliser une sémantique traduisant exactement la projection, telle que Ψ.

7.1.3 Complexité de $\mathcal{SGC}$-COHÉRENCE et de problèmes apparentés

Nous nous intéressons maintenant à la complexité du test de cohérence d'une base de connaissances (problème $\mathcal{SGC}$-COHÉRENCE). Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$ une base de connaissances de $\mathcal{SGC}$. Il est immédiat de constater que ce problème est co-NP-complet si l'ensemble $\mathcal{C}$ ne comporte que des contraintes négatives : en effet la réponse est non si et seulement s'il existe une projection d'une contrainte de $\mathcal{C}$ dans $\mathcal{G}$ (Prop. 7.1), il s'agit donc du co-problème de PROJECTION ?.

Complexité de $\mathcal{SGC}$-COHÉRENCE

Dans le cas où $\mathcal{C}$ comporte au moins une contrainte positive, le problème devient Π_2^P-complet. Rappelons que Π_2^P est co-NP^{NP}. Nous présentons brièvement ces classes de complexité de la hiérarchie polynomiale dans l'App. D.

Théorème 7.3 (Complexité de $\mathcal{SGC}$-COHÉRENCE) $\mathcal{SGC}$-COHÉRENCE *est* Π_2^P*-complet (mais co-NP-complet si les contraintes sont toutes négatives).*

Preuve: Sans incidence sur la complexité, on peut considérer que $\mathcal{C}$ est composé d'une seule contrainte positive, que nous appellerons C. Rappelons tout d'abord que le test de satisfaction de C par un graphe G se calcule sur la forme irredondante de G. On peut considérer deux façons de prendre en compte la mise sous forme irredondante de G dans la complexité de $\mathcal{SGC}$-COHÉRENCE.

Une première façon de faire consisterait à supposer que la forme irredondante de G est calculée avant le test de cohérence. Ceci peut être fait par un nombre d'appels à un oracle de projection, qui est linéaire en la taille de G [Mugnier, 1995]. Mais nous aurions alors à résoudre un problème « fonctionnel » (calculer la forme irredondante de G) et pas un problème de décision (G est-il irredondant ?).

Nous préférons donc une deuxième façon de procéder, qui consiste à intégrer la notion d'irredondance *dans* le test de cohérence : ayant une projection π_0 du déclencheur de C dans G, la projection de C dans G que nous recherchons n'étend pas forcément π_0, mais étend la composition d'une projection de G dans l'un de ses sous-graphes (qui peut être G lui-même si G est irredondant) et de π_0.

Prouvons d'abord que $\mathcal{SGC}$-COHÉRENCE est dans Π_2^P. Ce problème correspond au langage

$$L = \{x \mid \forall y_1 \; \exists y_2 \; \exists y_3 \; R(x,\ y_1,\ y_2,\ y_3)\}$$

où :

– x représente une instance (G, C) du problème ;

– $(x,\ y_1,\ y_2,\ y_3) \in R$ si et seulement si :
  – y_1 est une projection π_0 du déclencheur $C_{(0)}$ de C dans G ;
  – y_2 est une projection π_G de G dans un de ses sous-graphes ;
  – y_3 est une projection π de C dans G
  telles que $\pi[C_{(0)}] = \pi_G \circ \pi_0$. Remarquons que si G est sous forme irredondante, alors π_G est un automorphisme.

Donc $\mathcal{SGC}$-COHÉRENCE appartient bien à la classe de complexité Π_2^P. Montrons qu'il est complet pour cette classe, par une réduction à partir du problème B_2^c :

B_2^c
Données : Une formule booléenne E, et une partition $\{X_1,\ X_2\}$ de ses variables.
Question : Est-ce-que, pour toute interprétation $\mathcal{I}_1$ des variables de X_1, il existe une interprétation $\mathcal{I}_2$ des variables de X_2 telle que E s'évalue à VRAI pour l'interprétation $\mathcal{I}_1 \cup \mathcal{I}_2$?

Ce problème est Π_2^P-complet, puisque son co-problème B_2 est prouvé Σ_2^P-complet par [Stockmeyer, 1977]. Afin de construire une transformation *polynomiale* d'une instance de B_2 en instance de $\mathcal{SGC}$-COHÉRENCE, nous utiliserons une restriction du problème B_2^c à des instances ne comprenant que des clauses de taille égale à 3 au maximum (3-CNF). Appelons 3-SAT$_2^c$ ce cas particulier où la formule E est une instance de 3-SAT. Ce problème 3-SAT$_2^c$ est, lui aussi, Π_2^P-complet : en effet, dans le même article, [Stockmeyer, 1977] montre que la restriction de B_2 à des instances où E est une 3-DNF (forme normale disjonctive, où les conjonctions sont de taille au plus 3) reste Σ_2^P-complet. Comme la négation d'une 3-DNF est une 3-CNF, il s'ensuit que le co-problème B_2^c où les formules E sont des 3-CNF est Π_2^P-complet.

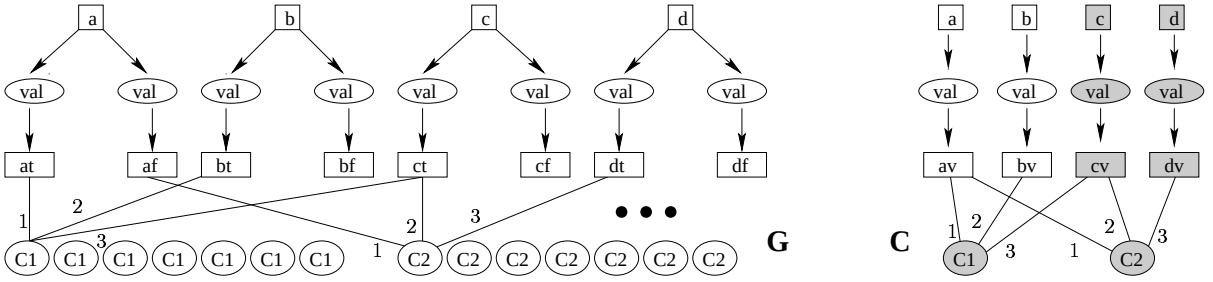


FIG. 7.3 – Transformation d'une instance de 3-SAT$_2^C$ en instance de $\mathcal{SGC}$-COHÉRENCE.

Nous proposons maintenant une transformation polynomiale *S2C* qui, à toute instance $(E, (X_1, X_2))$ de 3-SAT$_2^C$ associe une instance $(\mathcal{S}, \{G\}, \{C\})$ de $\mathcal{SGC}$-COHÉRENCE telle que la réponse à $(E, (X_1, X_2))$ est OUI si et seulement si la réponse à $(\mathcal{S}, \{G\}, \{C\})$ est OUI. Cette transformation est très similaire à celle que nous avons présentée pour réduire 3-SAT à PROJECTION ? (Sect. 5.1.2, voir la FIG. 5.2 illustrant la transformation utilisée).

Soit E une instance de 3-SAT, alors nous considérons les graphes $G = G(E)$ et $H(E)$ obtenus à partir de la formule E par la transformation présentée dans la Sect. 5.1.2. La

contrainte (positive) $C = (H(E), \rho(X_2))$ est obtenue en colorant de la façon suivante les sommets concepts et les relations de $H(E)$:

- chaque relation obtenue à partir d'une clause (celles typées par C_i) et colorée par 1 ;
- chaque sommet concept et chaque relation obtenus à partir d'une variable de X_2 (pour une variable x, il s'agit des sommets de type x et xv, et de la relation de type *val* qui les relie) est coloré par 1 ;
- tous les autres sommets concepts et relations sont colorés par 0 (*i.e.* appartiennent à l'hypothèse).

Ici aussi, nous insistons sur le fait que considérer des clauses de taille bornée nous permet d'obtenir une transformation polynomiale.

Le graphe G et la contrainte C représentées dans la FIG. 7.3 sont obtenus, grâce à cette transformation, à partir de la formule de 3-SAT $(a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d)$ et de la partition $X_1 = \{a, b\}, X_2 = \{c, d\}$.

Comme indiqué dans la Sect. 5.1.2, à chaque instanciation des variables de E telle que E est évaluée à VRAI est associée une projection de $H(E)$ dans G et réciproquement. De plus, à chaque instanciation des variables de X_1 correspond naturellement une projection de $C_{(0)}$ dans G, et réciproquement. Alors la question « est-il vrai que pour toute instanciation des variables de X_1, il existe une instanciation des variables de X_2 (une extension de l'instanciation de X_1 à celle de $X_1 \cup X_2$) telle que E est valuée à VRAI ? » est équivalente à la question « est-il vrai que pour toute projection π_0 de $C_{(0)}$ dans G, il existe une projection de C dans G qui étend π_0 ? ». □

Notons que la réduction que nous avions proposée dans [Baget and Mugnier, 2001b] est beaucoup plus simple, mais celle-ci a l'avantage de pouvoir être réutilisée dans la preuve du Th. 7.6 (Σ_3^P-complétude de la déduction dans $\mathcal{SEC}$ quand les règles de $\mathcal{R}$ sont *range-restricted*).

Complexité de quelques problèmes apparentés dans la famille $\mathcal{SG}$

Tout d'abord, la complexité de $\mathcal{SGC}$-COHÉRENCE nous donne directement la complexité de $\mathcal{SGC}$-DÉDUCTION :

Corollaire 7.1 (Complexité de $\mathcal{SGC}$-DÉDUCTION) *$\mathcal{SGC}$-DÉDUCTION est Π_2^P-complet.*

Preuve: Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{C})$ une base de connaissances, et H une requête. Alors voir que H est déductible de la base (cohérente) $\mathcal{K}$ si et seulement si la base $\mathcal{K}' = (\mathcal{S}, \mathcal{G}, \mathcal{C} \cup \{C(H)\})$ est cohérente, où la contrainte $C(H)$ est obtenue à partir de H en colorant chacun de ses sommets par 1.

Nous avons bien $\mathcal{K}'$ est cohérente $\Leftrightarrow$ $\mathcal{G}$ satisfait toutes les contraintes de $\mathcal{C}$ et satisfait la contrainte $C(H)$ $\Leftrightarrow$ $\mathcal{K}$ est cohérente et, pour toute projection de $\emptyset$ dans G, H se projette dans G $\Leftrightarrow$ $\mathcal{K}$ est cohérente et H se projette dans G $\Leftrightarrow$ H est déductible de $\mathcal{K}$. □

Cette complexité, associée à la Prop. 7.3, nous donne aussi la classe de complexité d'un problème que nous avons déjà évoqué :

COMPLET ?
Données : Un graphe G sous forme irredondante, et un ensemble de règles $\mathcal{R}$.
Question : Le graphe G est-il *complet* pour $\mathcal{R}$, *i.e.* est-ce-que toute application d'une règle de $\mathcal{R}$ à G produit un graphe équivalent ?

Corollaire 7.2 (Complexité de COMPLET ?) COMPLET ? *est un problème* Π_2^P*-complet.*

Équivalence avec le problème MIXED-CSP

Les contraintes utilisées dans les *réseaux de contraintes* (voir Sect. 5.1.2) sont plus simples que les contraintes de $\mathcal{SGC}$. En effet, nous avons vu que le problème CSP ?, équivalent à PROJECTION ?, est un problème NP-complet. Nous avons vu que la projection d'un graphe H dans un graphe G est équivalente à la satisfaction de la contrainte positive dont le déclencheur est vide et dont l'obligation est H : nous pouvons donc dire que le problème CSP ? est équivalent à la restriction de $\mathcal{SGC}$-COHÉRENCE aux contraintes dont le déclencheur est vide.

Dans le but de raisonner sur des connaissances incomplètes, [Fargier et al., 1996] proposent une extension aux réseaux de contraintes : les CSP mixtes (MIXED-CSP). Dans un MIXED-CSP, les variables du réseau de contraintes sont partitionnées en deux ensembles, les variables contrôlables, X, et les variables incontrôlables, Δ. Le problème MIXED-CSP s'exprime de la façon suivante :

MIXED-CSP
Données : Un réseau de contraintes $N = \{V, U, \delta\}$, et une partition (X, Δ) des variables de V.
Question : Est-ce-que toute instanciation consistante des variables de Δ, peut être étendue à une solution du réseau ?

[Fargier et al., 1996] montrent que MIXED-CSP est un problème NP-complet. Leur résultat nous offre non seulement une autre preuve de la Π_2^P-complétude de notre problème $\mathcal{SGC}$-COHÉRENCE, mais permet d'établir un autre pont entre PROJECTION ? et CSP ?, par l'intermédiaire de ces deux extensions similaires que sont $\mathcal{SGC}$-COHÉRENCE et MIXED-CSP.

Nous montrons que toute instance de MIXED-CSP peut être transformée en une instance de $\mathcal{SGC}$-COHÉRENCE. Nous utilisons la transformation *C2P* que nous avons présentée dans la Sect. 5.1.2, afin de transformer le réseau de contraintes composant l'instance de MIXED-CSP en deux graphes G et H. La contrainte positive $C = (H, \kappa)$ est obtenue en colorant par 0 (déclencheur) le sous-graphe de H correspondant aux sommets issus des variables de Δ. Le MIXED-CSP est consistant si et seulement si G satisfait la contrainte C.

Cette transformation est illustrée dans la FIG. 7.4. Les variables du réseau de contraintes sont partitionnées en $X = \{x_1,\ x_2\}$ et en $\Delta = \{l_1,\ l_2\}$, les trois contraintes sont C_1, C_2 et C_3. Les variables x_1 et x_2 ont même domaine, $\{1,\ 2,\ 3\}$, et les variables l_1 et l_2 ont pour

domaine $\{a,\ b\}$. Les tuples de variables autorisées par les contraintes sont définis dans la figure.

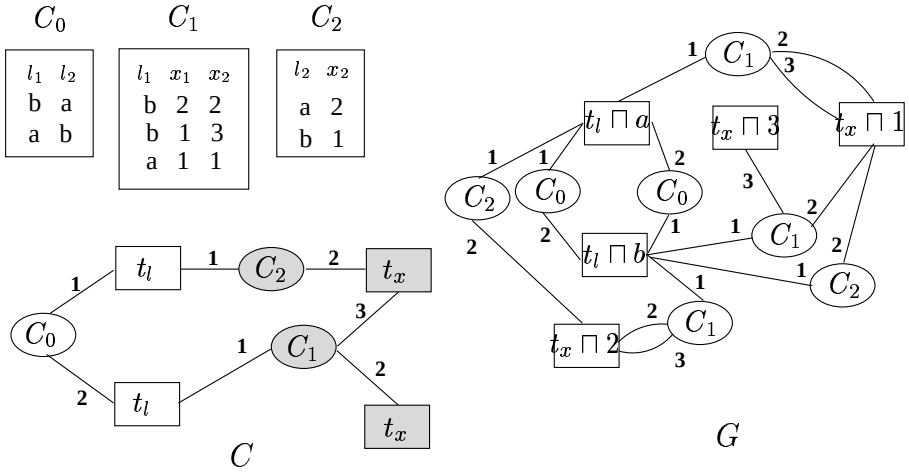


FIG. 7.4 – Transformation de MIXED-CSP en $\mathcal{SGC}$-COHÉRENCE

$\mathcal{SGC}$-COHÉRENCE et autres formalismes

D'autres définitions pour les contraintes ont été utilisées dans la communauté « Graphes Conceptuels ». Par exemple, nos contraintes (que l'on va appeler ici contraintes $\mathcal{SGC}$) sont un cas particulier des *contraintes minimales descriptives* de [Dibie et al., 1998]. Une telle contrainte peut être vue comme un ensemble de contraintes $\mathcal{SGC}$ partageant le même déclencheur. Leur signification intuitive est « si l'information A est présente dans les faits, alors je dois aussi y trouver au moins une des informations B_1 ou B_2 ou ... B_k ». Une contrainte minimale descriptive est satisfaite par un graphe G si, pour toute projection π du déclencheur dans G (il n'est pas question ici de condition d'irredondance), cette projection peut s'étendre à la projection d'au moins un des éléments de l'ensemble.

Remarquons que cette « disjonction » n'augmente pas la classe de complexité du test de cohérence (le certificat d'appartenance à la classe Π_2^P reste inchangé, et est même allégé puisqu'on ne prend pas en compte l'irredondance).

CMD-COHÉRENCE
Données : Un graphe G et un ensemble de contraintes minimales descriptives $\mathcal{C}$.
Question : Est-ce-que G satisfait toutes les contraintes de $\mathcal{C}$?

Corollaire 7.3 CMD-COHÉRENCE *est un problème* Π_2^P*-complet.*

[Dibie et al., 1998] ont montré que ces contraintes minimales descriptives généralisent un grand nombre de contraintes utilisées dans la communauté « Graphes Conceptuels ».

Citons aussi le cas des *contraintes topologiques* [Mineau and Missaoui, 1997], qui sont des contraintes $\mathcal{SGC}$ déconnectées (le déclencheur et l'obligation appartiennent à deux composantes connexes distinctes). Notons $\mathcal{SGC}^d$-COHÉRENCE ce problème :

$\mathcal{SGC}^d$-COHÉRENCE
Données : Un graphe G et un ensemble de contraintes $\mathcal{SGC}$ déconnectées $\mathcal{C}$.
Question : Est-ce-que G satisfait toutes les contraintes de $\mathcal{C}$?

Propriété 7.5 $\mathcal{SGC}^d$-COHÉRENCE *est un problème DP-complet.*

Nous utiliserons pour prouver cette propriété une réduction à partir du problème DP-complet SAT/UNSAT (voir [Papadimitriou, 1994]).

SAT/UNSAT
Données : Deux formules booléennes E_1 et E_2.
Question : Est-ce-que E_1 est satisfiable et E_2 insatisfiable ?

Remarquons que si Π est un problème DP-complet, alors le problème $\Pi^{\times N}$, dont les instances sont un nombre N d'instances de $(\pi_1, \ldots, \pi_N)$ de Π, et qui répond oui si et seulement si chacune des π_i est une instance positive de Π, est un problème DP-complet. En effet, nous pouvons sans perte de généralité considérer que Π est le problème SAT/UNSAT, et donc $(\pi_1 = (\pi_1^1, \pi_1^2), \ldots, \pi_N = (\pi_N^1, \pi_N^2))$ est une instance positive si et seulement si chacune des π_i^1 est satisfiable, et chacune des π_i^2 est insatisfiable. Ces formules ne partageant aucune variable, ceci est équivalent à $(\pi_1^1 \wedge \cdots \wedge \pi_N^1)$ est satisfiable, et $(\pi_1^2 \vee \cdots \vee \pi_N^2)$ est insatisfiable. Donc $(\pi_1, \ldots, \pi_N)$ est une instance positive de $\Pi^{\times N}$ si et seulement si $f((\pi_1, \ldots, \pi_N))$ est une instance positive de Π, donc $\Pi^{\times N}$ est plus simple que Π, et comme il s'agit de sa généralisation, les problèmes sont équivalents : $\Pi^{\times N}$ est un problème DP-complet. En d'autres termes, $\mathrm{P}^{\mathrm{DP}} = \mathrm{DP}$. Remarquons aussi que co-DP = DP.

Nous pouvons maintenant prouver la Prop. 7.5 :

Preuve: Grâce à la remarque précédente, il nous suffit de prouver que la restriction aux instances ne contenant qu'une contrainte déconnectée du problème $\mathcal{SGC}^d$-COHÉRENCE est DP-complet. Pour cela, prouvons que son co-problème, $\mathcal{SGC}^d$-INCOHÉRENCE, est DP-complet (ce qui est suffisant, puisque DP = co-DP).

Ceci est immédiat. En effet, G viole une contrainte positive déconnectée C si et seulement si $C_{(0)}$ se projette dans G, et $C_{(1)}$ ne se projette pas dans G. La transformation de SAT à PROJECTION ? présentée dans la Sect. 5.1.2 peut donc être utilisée pour prouver que $\mathcal{SGC}^d$-INCOHÉRENCE est DP-complet. □

Notons enfin que ces autres types de contraintes que nous avons présentés sont uniquement utilisées pour tester la cohérence d'une base de faits, et ne sont pas, comme dans les modèles plus généraux de la famille $\mathcal{SG}$, intégrées à un mécanisme de dérivation utilisant des règles.

Il serait peut-être possible d'établir des liens avec la vérification de bases de connaissances utilisant des règles logiques, notamment avec les travaux de [Levy and Rousset, 1996], utilisant des contraintes qui sont des TGDs (*tuple generating dependencies*), qui ont la même forme que les nôtres (voir [Coulondre and Salvat, 1998]). Toutefois, nous n'avons pas trouvé de liens « directs » entre ces problèmes.

7.2 Intégrations des règles et contraintes

Lorsque la base de connaissances comporte à la fois des contraintes et des règles, il y a plusieurs façons d'envisager la vérification de cohérence des graphes tout au long d'une dérivation :

1. une contrainte n'est-elle déclenchée que sur *(a)* le graphe courant de la dérivation, ou *(b)* doit-on envisager de tester son déclenchement sur tous les graphes qui peuvent être dérivés du graphe courant ?
2. la violation d'une contrainte par le graphe courant doit-elle *(a)* nous faire rejeter le graphe comme incohérent, ou *(b)*, doit-on vérifier si cette violation peut être réparée/restorée dans des dérivations du graphe courant ?

Si nous répondons *(a)* à ces deux questions, nous obtenons le modèle $\mathcal{SEC}$, et nous appellons les règles de ce modèle des *règles d'évolution*. Si nous répondons *(b)* à ces deux question, nous obtenons le modèle $\mathcal{SRC}$, et nous appellons les règles de ce modèle des *règles d'inférence*. Le modèle le plus général de la famille $\mathcal{SG}$, $\mathcal{SREC}$ combine règles d'évolution et règles d'inférence.

Nous présentons successivement ces trois modèles, et nous intéressons à leur classe de décidabilité. Des cas particuliers, décidables, sont proposés dans la Sect. 7.3.

7.2.1 Règles d'évolution : le modèle $\mathcal{SEC}$

Considérons le modèle $\mathcal{SEC}$ dans lequel une base de connaissances $\mathcal{K}$ est composée d'un support $\mathcal{S}$, d'un ensemble de faits $\mathcal{G}$, d'un ensemble de contraintes $\mathcal{C}$, et d'un ensemble de règles d'évolution $\mathcal{E}$. Dans ce modèle, $\mathcal{G}$ peut être vu comme un « monde initial », racine d'une arborescence potentiellement infinie de mondes possibles, $\mathcal{E}$ décrivant les évolutions (dérivations immédiates) possibles d'un monde à d'autres. Nous ne nous intéressons ici qu'à la sous-arborescence de ces mondes possibles, constituée des mondes cohérents. Étant donnés $\mathcal{K}$ et une requête H, le problème de déduction consiste à déterminer s'il existe un *chemin de mondes cohérents allant de $\mathcal{G}$ à un monde répondant à H*. Ce problème sera noté $\mathcal{SEC}$-DÉDUCTION.

Définition 7.4 (Déduction dans $\mathcal{SEC}$) *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{E}, \mathcal{C})$ une base de connaissances, et soit H un graphe représentant une requête. Alors H est déductible de $\mathcal{K}$ s'il existe une séquence de $\mathcal{E}$-dérivations immédiates $\mathcal{G} = G_0, \ldots, G_k$ telle que, pour $0 \leq i \leq k$, $(\mathcal{S}, \{G_i\}, \mathcal{C})$ est une base cohérente du modèle $\mathcal{SGC}$ et H est déductible de $(\mathcal{S}, \{G_k\})$ (au sens du modèle $\mathcal{SG}$,* i.e. *il existe une projection de H dans G_k).*

Un problème semi-décidable

Théorème 7.4 *[Complexité de $\mathcal{SEC}$-DÉDUCTION] Le problème $\mathcal{SEC}$-DÉDUCTION est semi-décidable.*

Preuve: $\mathcal{SR}$-DÉDUCTION est un cas particulier de $\mathcal{SEC}$-DÉDUCTION (il s'agit de sa restriction aux instances composées d'un ensemble vide de contraintes). $\mathcal{SR}$-DÉDUCTION n'étant pas décidable (Th. 6.2), $\mathcal{SEC}$-DÉDUCTION ne l'est pas non plus.

Cependant, si H est déductible de $\mathcal{K}$, un parcours en largeur de toutes les applications possibles des règles de $\mathcal{E}$ à chaque étape, associé à un test de cohérence (Π_2^P-complet donc décidable) sur chacun des graphes obtenus, nous assure que nous trouverons le graphe G_k dans lequel H peut se projeter. $\mathcal{SEC}$-DÉDUCTION est donc un problème semi-décidable. □

Nous devons malheureusement souligner l'extraordinaire coût en espace de cet algorithme nécessitant un parcours en largeur : tous les graphes correspondant à des feuilles de l'arbre de dérivation doivent être archivés.

7.2.2 Règles d'inférence : le modèle $\mathcal{SRC}$

Dans $\mathcal{SRC}$, $\mathcal{G}$ muni de $\mathcal{R}$ peut être vu comme la description finie d'un monde potentiellement infini, qui doit être cohérent. En appliquant une règle à $\mathcal{G}$, nous pouvons rendre le monde incohérent, mais une autre application de règle peut ensuite restaurer la cohérence du monde. Précisons cette notion de « restauration de conhérence ».

Définition 7.5 (Restauration de cohérence) *Soit G un graphe irredondant, C une contrainte, $\mathcal{R}$ un ensemble de règles d'inférence, et supposons que le graphe G π-viole la contrainte C. Alors cette violation (C, G, π) est dite* $\mathcal{R}$-restaurable *s'il existe une $\mathcal{R}$-dérivation de $\mathcal{G}$ dans G' et une projection π' de G' dans $irr(G')$, le graphe irredondant associé à G', telles que la projection $\pi' o \pi$ du déclencheur de C dans $irr(G')$ peut être étendue à une projection de toute la contrainte C.*

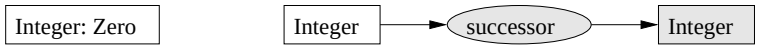
Remarquons que la violation d'une contrainte négative ne pourra jamais être restaurée (si la contrainte négative C se projette dans un graphe G, alors elle se projette dans chacune de ses $\mathcal{R}$-dérivations).

Nous pouvons maintenant définir à la fois la notion de cohérence d'une base de connaissances dans le modèle $\mathcal{SRC}$ (notion qui est définie dans ce modèle, contrairement à $\mathcal{SEC}$), et le problème de déduction associé.

Définition 7.6 (Cohérence et déduction dans $\mathcal{SRC}$) *Une base de connaissances $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{C})$ est dite* cohérente *si toute violation $\mathcal{R}$-dérivable de $\mathcal{G}$ est $\mathcal{R}$-restaurable,* i.e. *si pour toute $\mathcal{R}$-dérivation G' de $\mathcal{G}$, pour chaque contrainte $C \in \mathcal{C}$, pour chaque π-violation de C dans G', (C, G', π) est $\mathcal{R}$-restaurable.*

Un graphe requête H est dit déductible *de $\mathcal{K}$ si $\mathcal{K}$ est cohérente et H peut être déduit (au sens du modèle $\mathcal{SR}$) de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$.*

Illustrons la distinction entre les notions de cohérence dans $\mathcal{SEC}$ et $\mathcal{SRC}$ à l'aide de la FIG. 7.5. Le graphe élémentaire G exprime l'existence d'un entier : 0. Nous considérons la contrainte positive C et la règle R, toutes deux représentées par le graphe coloré K de la même figure. La contrainte C exprime que « tout entier doit avoir un entier pour successeur ». Le graphe G viole la contrainte C.



Un graphe conceptuel simple G Un graphe élémentaire coloré K

FIG. 7.5 – Deux visions différentes de la cohérence : $\mathcal{SEC}$ et $\mathcal{SRC}$

Si la règle R est considérée comme une règle d'évolution (nous sommes alors dans le modèle $\mathcal{SEC}$), comme le monde initial est incohérent, rien ne pourra être déduit de cette base de connaissances (mais nous n'en déduisons pas une incohérence de la base, notion qui, répétons-le, n'a aucun sens).

Si la règle R est considérée comme une règle d'inférence (nous sommes alors dans le modèle $\mathcal{SRC}$), l'application de la règle R permet de restaurer la violation, tout en rajoutant un nouvel entier qui n'a pas de successeur, ce qui crée une nouvelle violation. Nous pouvons prouver que toute violation de la contrainte C sera restaurable, mais ceci ne peut pas être décelé (en temps fini) par le mécanisme d'application en marche avant.

Soulignons que le modèle $\mathcal{SR}$ est obtenu par la restriction des modèles $\mathcal{SRC}$ et $\mathcal{SEC}$ à des instances contenant un ensemble vide de contraintes, et, de la même manière, $\mathcal{SGC}$ est obtenu par la restriction des modèles $\mathcal{SRC}$ et $\mathcal{SEC}$ à des instances contenant un ensemble vide de règles (respectivement d'inférence et d'évolution).

Le problème de cohérence dans $\mathcal{SRC}$, et donc de déduction, sont cette fois vraiment indécidables :

Théorème 7.5 (Complexité de $\mathcal{SRC}$-COHÉRENCE/-DÉDUCTION) $\mathcal{SRC}$-COHÉRENCE *et* $\mathcal{SRC}$-DÉDUCTION *sont des problèmes indécidables (et pas semi-décidables).*

Preuve: Prouvons l'indécidabilité du problème $\mathcal{SRC}$-COHÉRENCE (l'indécidabilité de $\mathcal{SRC}$-DÉDUCTION en découle, puisque $\mathcal{SRC}$-COHÉRENCE peut être vu comme un cas particulier de $\mathcal{SRC}$-DÉDUCTION où le graphe requête H est vide).

Considérons une base de connaissance $\mathcal{K}$ contenant une contrainte positive C^+ et une contrainte négative C^-, chacune ayant un déclencheur vide. Alors $\mathcal{K}$ est cohérente si et seulement si $C^+_{(1)}$ est déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$ et $C^-_{(1)}$ n'est pas déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$. $\mathcal{SR}$-DÉDUCTION n'étant pas décidable (Th. 6.2), il s'ensuit que $\mathcal{SRC}$-COHÉRENCE est indécidable. □

7.2.3 Utiliser à la fois évolution et inférence : le modèle $\mathcal{SREC}$

Le modèle $\mathcal{SREC}$ combine les deux schémas de dérivation des modèles $\mathcal{SEC}$ et $\mathcal{SRC}$. Maintenant $\mathcal{G}$ décrit un monde initial, les règles d'inférence de $\mathcal{R}$ complètent la description des mondes, les contraintes de $\mathcal{C}$ évaluent la cohérence des mondes, les règles d'évolution de $\mathcal{E}$ tentent de passer d'un monde cohérent à un autre monde cohérent. Étant donnés une base de connaissances $\mathcal{K}$ et une requête H, le problème de déduction consiste à déterminer s'il existe un chemin de mondes cohérents allant de $\mathcal{G}$ à un monde répondant à H. Ce

problème sera noté $\mathcal{SREC}$-DÉDUCTION. Comme pour le modèle $\mathcal{SEC}$, le problème $\mathcal{SREC}$-COHÉRENCE n'a aucun sens : la notion de cohérence est définie localement sur les mondes, et ne peut pas être définie globalement sur la base de connaissances.

Définition 7.7 (Déduction dans $\mathcal{SREC}$) *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{E}, \mathcal{C})$ une base de connaissances. Un graphe G' est une* $\mathcal{RE}$-évolution immédiate *d'un graphe G s'il existe une $\mathcal{R}$-dérivation G'' de G et une $\mathcal{E}$-dérivation* immédiate *G' de G''.*

Nous appelons $\mathcal{RE}$-évolution cohérente *d'un graphe G un graphe G' obtenu par une séquence de $\mathcal{RE}$-évolutions immédiates $G = G_0, \ldots, G_k = G'$ telle que, pour $0 \leq i \leq k$, $(\mathcal{S}, \{G_i\}, \mathcal{R}, \mathcal{C})$ est cohérent (au sens de $\mathcal{SRC}$) et, pour $1 \leq i \leq k$, G_i est une $\mathcal{RE}$-évolution immédiate de G_{i-1}.*

Soit maintenant H un graphe requête. Alors H est déductible *de $\mathcal{K}$ s'il existe une $\mathcal{RE}$-évolution G' de $\mathcal{G}$ telle que G' est déductible (au sens de $\mathcal{SR}$) de $(\mathcal{S}, \{G'\}, \mathcal{R})$.*

Lorsque $\mathcal{E} = \emptyset$ (resp. $\mathcal{R} = \emptyset$), nous obtenons le modèle $\mathcal{SRC}$ (resp. le modèle $\mathcal{SEC}$).

Corollaire 7.4 *$\mathcal{SREC}$ étant une généralisation de $\mathcal{SEC}$, le problème $\mathcal{SREC}$-*DÉDUCTION *est indécidable.*

7.3 Décidabilité et complexité

Nous étudions dans cette section des restrictions portant sur les règles ou les contraintes pour lesquelles les problèmes de déduction associés aux différents modèles de la famille $\mathcal{SG}$ deviennent décidables. Les restrictions sur les règles concernent les règles à expansion finie et les règles *range restricted* que nous avons définies au chapitre précédent (Sect. 6.2.3, Def. 6.9 et Def. 6.10). Les restrictions sur les contraintes portent sur les contraintes négatives, et les contraintes déconnectées équivalentes aux contraintes topologiques de [Mineau and Missaoui, 1997].

7.3.1 Règles à expansion finie et règles *range restricted*

Nous utiliserons dans ce chapitre les notions d'ensemble de règles à expansion finie, de graphe complet pour un ensemble de règles, et de règles *range restricted* vues au chapitre précédent (Sect. 6.2.3).

Ensembles de règles à expansion finie

Si nous nous restreignons aux ensembles de règles à expansion finie de la Def. 6.9 (en notant $\mathcal{R}^{\text{fes}}$ ou $\mathcal{E}^{\text{fes}}$ des ensembles de règles d'inférence ou d'évolution à expansion finie), nous obtenons, suivant les modèles considérés, des cas décidables. Nous aurons besoin pour notre démonstration de la propriété suivante, dont la preuve est immédiate :

Propriété 7.6 *Soit $\mathcal{K} = (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{C})$ une base de connaissances telle que $\mathcal{R}$ est un ensemble de règles à expansion finie. Alors $\mathcal{K}$ est cohérente si et seulement si $(\mathcal{S}, \{\mathcal{G}^{\mathcal{R}}\}, \mathcal{C})$ est cohérente (au sens de $\mathcal{SGC}$, où $\mathcal{G}^{\mathcal{R}}$ est le graphe complet de $\mathcal{G}$ pour $\mathcal{R}$ – voir Def. 6.9).*

De plus, une requête H est déductible de $(\mathcal{S}, \mathcal{G}, \mathcal{R})$ si et seulement si H est déductible (au sens de $\mathcal{SG}$) de $(\{\mathcal{G}^{\mathcal{R}}\})$.

Cette propriété montre aussi que la définition du problème de déduction que nous avions donnée, dans le cadre des ensembles de règles à expansion finie, dans [Baget et al., 1999], est équivalente à la restriction du cas général présenté dans ce mémoire.

Nous en déduisons les propriétés de décidabilité suivantes :

Propriété 7.7 (Décidabilité pour des ensembles de règles à expansion finie)

- *Lorsque $\mathcal{R}$ est un ensemble à expansion finie, alors :*
  - $\mathcal{SR}^{fes}$-DÉDUCTION *est décidable ;*
  - $\mathcal{SR}^{fes}\mathcal{C}$-COHÉRENCE *est décidable ;*
  - $\mathcal{SR}^{fes}\mathcal{C}$-DÉDUCTION *est décidable ;*
  - $\mathcal{SR}^{fes}\mathcal{EC}$-DÉDUCTION *est semi-décidable.*
- *Lorsque $\mathcal{E}$ est un ensemble à expansion finie, alors :*
  - $\mathcal{SE}^{fes}\mathcal{C}$-DÉDUCTION *est décidable ;*
  - $\mathcal{SRE}^{fes}\mathcal{C}$-DÉDUCTION *est indécidable.*
- *Enfin, lorsque $\mathcal{R} \cup \mathcal{E}$ est un ensemble à expansion finie :*
  - $\mathcal{S}(\mathcal{RE})^{fes}\mathcal{C}$-DÉDUCTION *est décidable.*

Proof: Suppose $\mathcal{R}$ is a f.e.s. Decidability of problems in $\mathcal{SR}$ and $\mathcal{SRC}$ follows from property 7.6. In $\mathcal{SREC}$, when the answer is "yes", it can be obtained in finite time ; we proceed as for $\mathcal{SEC}$ (see proof of theorem 7.4) but consistency checks are done on the full graph instead of the graph itself.

Now, suppose $\mathcal{E}$ is a f.e.s. $\mathcal{G}^{\mathcal{E}}$ exists, thus the derivation tree in $\mathcal{SEC}$ is finite, and consistency checks may only cut some parts of this tree. Deduction in $\mathcal{SREC}$ remains undecidable because when $\mathcal{E} = \emptyset$, one obtains the $\mathcal{SRC}$ model, in which deduction is truly undecidable.

Finally, if $\mathcal{R} \cup \mathcal{E}$ is a f.e.s., $\mathcal{G}^{\mathcal{R} \cup \mathcal{E}}$ exists, thus the derivation tree is finite, and consistency checks may only cut parts of this tree. □

Union de deux ensembles à expansion finie

Remarquons que la condition condition « $\mathcal{R} \cup \mathcal{E}$ » est un ensemble à expansion finie est une propriété plus forte que « $\mathcal{R}$ et $\mathcal{E}$ sont deux ensembles à expansion finie », ce qui est exprimé par la propriété suivante :

Propriété 7.8 *Si $\mathcal{R}$ et $\mathcal{E}$ sont tous deux des ensembles à expansion finie, alors $\mathcal{R} \cup \mathcal{E}$ n'est pas nécessairement un ensemble à expansion finie, et $\mathcal{SR}^{fes}\mathcal{E}^{fes}\mathcal{C}$-*DEDUCTION *est un problème semi-décidable.*

Proof: We build a reduction from WORD PROBLEM IN A SEMI-THUE SYSTEM [Thue, 1914] to $\mathcal{SREC}$-DEDUCTION, where the obtained rule sets $\mathcal{R}$ and $\mathcal{E}$ are both finite expansion sets. This reduction relies on the one built for proving the semi-decidability of $\mathcal{SR}$-DEDUCTION (Prop. 6.2).

Let us first present the two kind of finite expansion sets used in this reduction. $\mathcal{E}$ is a finite expansion set since only relation nodes are present in the conclusion of rules : $\mathcal{E}$ is indeed a particular case of range-restricted rules (see Prop. 6.4). $\mathcal{R}$ is also a finite expansion set since, for every rule in $\mathcal{R}$, the hypothesis is disconnected from the conclusion (we call these rules *disconnected*). Note this time that, though $\mathcal{R}$ is trivially a f.e.s., the closure of a graph w.r.t. $\mathcal{R}$ does not necessarily exist.

Recall the WORD PROBLEM takes as input m and m' two words and $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ a set of rules, each rule γ_i being a pair of words (α_i, β_i), and asks whether there is a derivation from m to m'. There is an immediate derivation from m to m' (we note $m \to m'$) if, for some γ_j, $m = m_1\alpha_j m_2$ and $m' = m_1\beta_j m_2$. A *derivation* from m to m' (we note $m \rightsquigarrow m'$) is a sequence $m = m_0 \to m_1 \to \ldots \to m_p = m'$.

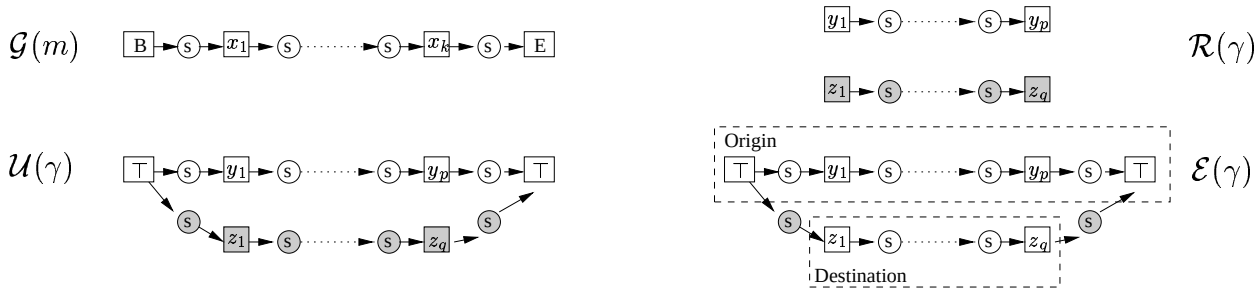


FIG. 7.6 – Transformations from the WORD PROBLEM into models of the $\mathcal{SG}$ family

We have shown how this problem can be expressed in the $\mathcal{SR}$ model : to a word $m = x_1 \ldots x_k$ is associated the graph $\mathcal{G}(m)$, and to any rule $\gamma = (y_1 \ldots y_p,\ z_1 \ldots z_q)$ is associated the graph rule $\mathcal{U}(\gamma)$, as represented in Fig. 7.6 (where $\top$ is greater than all other concept types). $m \rightsquigarrow m' \ \Leftrightarrow\ \mathcal{G}(m') \geq (\mathcal{G}(m), \mathcal{U}(\Gamma))$.

Let us now split each obtained rule $\mathcal{U}(\gamma)$ into one disconnected inference rule $\mathcal{R}(\gamma)$ and one range-restricted evolution rule $\mathcal{E}(\gamma)$. We distinguish in the hypothesis of $\mathcal{E}(\gamma)$ two subgraphs : the *origin*, which corresponds to the hypothesis of $\mathcal{U}(\gamma)$, and the *destination*, which corresponds to the conclusion of $\mathcal{R}(\gamma)$. It is easy to check that one part of the above equivalence still holds : $m \rightsquigarrow m' \ \Rightarrow\ \mathcal{G}(m') \geq (\mathcal{G}(m), \mathcal{R}(\Gamma) \cup \mathcal{E}(\Gamma))$. However, the converse is no longer valid : check by exemple that, if $\Gamma = \{\gamma = (a, c)\}$, we have $\mathcal{G}(c) \geq (\mathcal{G}(aba), \mathcal{R}(\Gamma) \cup \mathcal{E}(\Gamma))$ though $aba \not\rightsquigarrow c$ (apply $\mathcal{R}(\gamma)$ once, then two times $\mathcal{E}(\gamma)$ using different nodes `[a]` but the same node `[c]`).

We thus need the notion of a *good* application of a rule $\mathcal{E}(\gamma)$: it is such that the projection of its destination part is a mapping to a subgraph that was obtained by applying the rule $\mathcal{R}(\gamma)$, and that was never used to project the destination part of any rule of $\mathcal{E}(\Gamma)$, including $\mathcal{E}(\gamma)$ itself. Moreover, the origin and destination must be projected into disjoint paths (i.e. a node of the origin cannot have the same image has a node of the destination). If we restrict ourselves in some way to good applications of rules of $\mathcal{E}(\Gamma)$, then we can verify that $\mathcal{G}(m') \geq (\mathcal{G}(m), \mathcal{U}(\Gamma))$ iff $\mathcal{G}(m') \geq (\mathcal{G}(m), \mathcal{R}(\Gamma) \cup \mathcal{E}(\Gamma))$.

This restriction is obtained by using constraints, that will allow every good application of a rule of $\mathcal{E}(\Gamma)$, and be violated by the obtained graph otherwise. Let us note $\mathcal{R}'(\Gamma)$ and $\mathcal{E}'(\Gamma)$ the new sets of inference rules and evolution rules. The new transformation is described in Fig. 7.7. It allows to obtain the following result : $m \rightsquigarrow m' \ \Leftrightarrow\ \mathcal{G}'(m') \geq (\mathcal{G}'(m), \mathcal{R}'(\Gamma), \mathcal{E}'(\Gamma), \{C_+, C_-\})$.

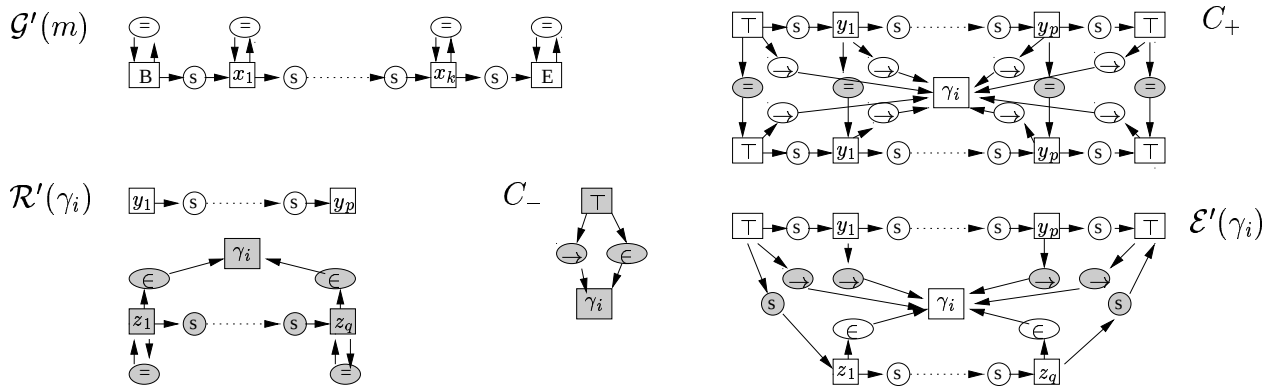


FIG. 7.7 – Reduction from the WORD PROBLEM to $\mathcal{SREC}$-DEDUCTION, with $\mathcal{E}$ and $\mathcal{R}$ f.e.s.

The names of relation types $=$, $\in$ and $\rightarrow$ have been chosen to give an intuitive idea of their role but they are just types as others. A relation node ($\in$) from a node $[z_j]$ to a node $[\gamma_i]$ means that the letter z_j has been obtained by applying the rule γ_i. A relation node ($\rightarrow$) from $[y_k]$ to $[\gamma_i]$ means that the letter y_k belongs to the subword on which the rule γ_i has been applied. $=$ is used to indicate that two concept nodes have to be projected on the same node (in CG terms, we would see it as a co-reference link). The evolution rule $\mathcal{E}'(\gamma_i)$, starting from a path representing the subword α_i used to apply γ_i and from the representation of β_i generated by $\mathcal{R}'(\gamma_i)$, produces the two relation nodes typed s simulating the application of $\mathcal{U}(\gamma_i)$, thus γ_i, and the relation nodes typed $\rightarrow$ which mark the representation of α_i as used by an application of γ_i. The negative constraint C_- prevents an application of $\mathcal{E}'(\gamma_i)$ in which two nodes of the origin and destination parts have the same image (a node necessarily obtained by some application of the rule $\mathcal{R}'(\gamma_i)$) ; while the positive constraint C_+ prevents such a subgraph to be used twice for applying $\mathcal{E}'(\gamma_i)$ with different projections of its origin : it says that in this case, the two projections of the origins must be the same. □

Règles *range restricted* et hiérarchie polynomiale

Dans tous les résultats qui suivent, nous supposons que l'arité des types de relations est bornée par une constante. Nous considérons ici des règles de type *range restricted* (Def. 6.10), et nous noterons les ensembles composés de telles règles $\mathcal{R}^{\text{rr}}$ ou $\mathcal{E}^{\text{rr}}$.

Théorème 7.6 (Complexité pour des règles *range restricted*)

- $\mathcal{SR}^{rr}$-DÉDUCTION *est un problème NP-complet ;*
- $\mathcal{SR}^{rr}\mathcal{C}$-COHÉRENCE *est un problème* Π_2^P*-complet ;*
- $\mathcal{SR}^{rr}\mathcal{C}$-DÉDUCTION *est un problème* Π_2^P*-complet ;*
- $\mathcal{SE}^{rr}\mathcal{C}$-DÉDUCTION *est un problème* Σ_3^P*-complet ;*
- $\mathcal{SR}^{rr}\mathcal{E}^{rr}\mathcal{C}$-DÉDUCTION *est un problème* Σ_3^P*-complet.*

Proof: The following results heavily rely on Prop. 6.4 : all derivations involved being of polynomial length, they admit a polynomial certificate (the sequence of projections used to build the derivation).

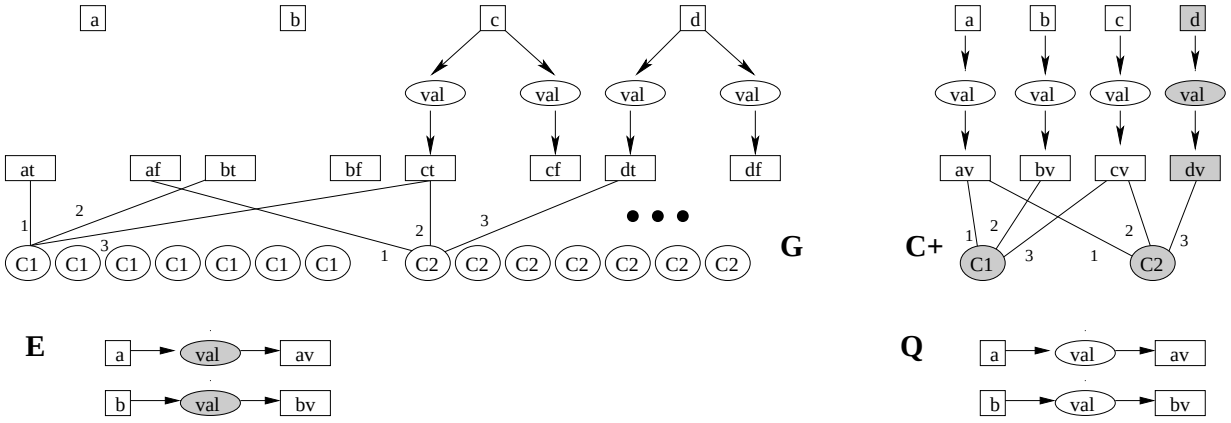


FIG. 7.8 – Example of transformation from 3-SAT_3 to $\mathcal{SEC}$-DEDUCTION

NP-completeness of $\mathcal{SR}$-DEDUCTION. The problem belongs to NP. Indeed, a polynomial certificate is given by a derivation from $\mathcal{G}$ to a graph G', followed by a projection from the query to G'. When $\mathcal{R} = \emptyset$, one obtains $\mathcal{SG}$-DEDUCTION (projection checking), thus the NP-completeness.

Π_2^P*-completeness of consistency and deduction in* $\mathcal{SRC}$. Recall the consistency check involves the irredundant form of $\mathcal{G}$. In order to lighten the problem formulation, we assume here that $\mathcal{G}$ is irredundant, but redundancy can be integrated without increasing the consistency check complexity : see the proof of theorem 7.3. $\mathcal{SRC}$-consistency belongs to Π_2^P since it corresponds to the language $L = \{x \mid \forall y_1\ \exists y_2\ R(x,\ y_1,\ y_2)\}$, where x encodes an instance $(\mathcal{G}; \mathcal{R}; \mathcal{C})$ of the problem and $(x,\ y_1,\ y_2) \in R$ iff $y_1 = (d_1; \pi_0)$, with d_1 is a derivation from $\mathcal{G}$ to G', π_0 is a projection from the trigger of a constraint $C_{i(0)}$ into G', $y_2 = (d_2; \pi_1)$, with d_2 is a derivation from G' to G'' and π_1 is a projection from C_i into G'' s.t. $\pi_1[C'_{i(0)}] = \pi_0$. R is polynomially decidable and polynomially balanced (since the lengths of d_1 and d_2 are polynomial in the size of the input). When $\mathcal{R} = \emptyset$, one obtains the problem $\mathcal{SGC}$-CONSISTENCY, thus the Π_2^P-completeness. Since $\mathcal{SRC}$-DEDUCTION consists in solving two independent problems, $\mathcal{SRC}$-CONSISTENCY (Π_2^P-complete) and $\mathcal{SR}$-DEDUCTION (NP-complete), and since NP is included in Π_2^P, $\mathcal{SRC}$-DEDUCTION is also Π_2^P-complete.

Σ_3^P*-completeness of* $\mathcal{SEC}$-DEDUCTION. As for $\mathcal{SRC}$ (see above), we assume $\mathcal{G}$ is irredundant. The question is "are there a derivation from $\mathcal{G}$ to G' and a projection from Q into G', such that for all G_i of this derivation, for all constraint C_j, for all projection π from $C_{j(0)}$ into G_i, there exists a projection π' from C_j into G_i s.t. $\pi'[C_{j(0)}] = \pi$? R is polynomially decidable and polynomially balanced (since the size of the derivation from $\mathcal{G}$ to G' is polynomially related to the size of the input). Thus, $\mathcal{SEC}$-deduction is in Σ_3^P. In order to prove the completeness, we do a reduction from a special case of the problem B_3, where the formula is a 3-CNF (i.e. an instance of 3-SAT) : given a formula E, which is a conjunction of clauses with a most 3 literals, and a partition $\{X_1, X_2, X_3\}$ of its variables, does there exist a truth assignment for the variables in X_1, such that for all truth assignment for the variables of X_2, there exists a truth assignment for the variables of X_3 such that E is true ? This problem is Σ_3^P-complete [Stockmeyer, 1977, theorem 4.1]. Let us call it 3-SAT_3.

The transformation we use is illustrated in Fig. 7.8. The 3-SAT formula used is again $(a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d)$, and the partition is $X_1 = \{a, b\}, X_2 = \{c\}, X_3 = \{d\}$. The graph G obtained is the same as in the proof of Th. 7.3, Fig. **??**, excepted that concept nodes `[x]` corresponding to

variables of X_1 are not linked to the nodes `[xt]` and `[xf]` representing their possible values.

First check that in this initial world, no constraint is violated, but the query Q is not satisfied. By applying once the evolution rule E, we try some valuation of the variables in X_1 and obtain a world G_1, that contains an answer to Q. But this world has to satisfy the positive constraint C_+, expressing that "for every valuation of the variables in $X_1 \cup X_2$, there exists a valuation of the variables in X_3 such that the formula evaluates to *true*". If G_1 satisfies this constraint, it means that we have found (by applying E) a valuation of the variables in X_1 such that for all valuations of variables in $X_1 \cup X_2$ (which can be simplified in "for all valuations of variables in X_2", since there is only one such valuation for X_1), there is a valuation of the variables in X_3 such that the formula evaluates to *true*. Then there is an answer *yes* to the 3-SAT_3 problem. Conversely, suppose an answer *no* to the $\mathcal{SEC}$ problem. It means that for every world G_1 that can be obtained by applying the rule E, the constraint C_+ is violated (otherwise Q could be projected into G_1 and the answer would be *yes*). Thus there is no assignment of the variables in X_1 satisfying the constraint, *i.e.* the answer to the 3-SAT_3 problem is *no*.

Σ_3^P*-completeness of* $\mathcal{SREC}$-DEDUCTION. $\mathcal{SREC}$-deduction stays in the same class of complexity as $\mathcal{SEC}$-deduction. Indeed, the question is "are there an $\mathcal{RE}$-derivation from $\mathcal{G}$ to G' and a projection from Q to G', such that for all G_i either equal to $\mathcal{G}$ or obtained by an $\mathcal{E}$-evolution, for all G'_i derived from G_i by an $\mathcal{R}$-derivation, for all constraint C_j, for all projection π from $C_{j(0)}$ to G'_i, there exist an $\mathcal{R}$-derivation from G'_i to G''_i and a projection π' from C_j to G''_i s.t. $\pi'[C_{j(0)}] = \pi$?" and the lengths of all derivations are polynomial in the size of the input. When $\mathcal{R} = \emptyset$, one obtains $\mathcal{SEC}$'-DEDUCTION, thus the Σ_3^P completeness. □

Il est intéressant de remarquer que, si le problème de déduction est en général plus complexe dans $\mathcal{SRC}$ (indécidable) que dans $\mathcal{SEC}$ (semi-decidable), la situation est inversée lorsque l'on considère le cas particulier des règles *range restricted*.

7.3.2 Contraintes négatives et contraintes déconnectées

Nous examinerons maintenant des types restreints de contraintes, et considérerons soit un ensemble $\mathcal{C}^-$ de contraintes négatives, soit un ensemble $\mathcal{C}^d$ de contraintes déconnectées (celles de [Mineau and Missaoui, 1997]). Ces cas particuliers de contraintes seront étudiés en conjonction avec un ensemble quelconque de règles, ou avec des règles *range restricted*.

Contraintes négatives et règles quelconques

Examinons maintenant la restriction des modèles de la famille $\mathcal{SG}$ aux bases de connaissances ne comportant que des règles négatives :

Théorème 7.7 (Complexité pour des contraintes négatives)
- $\mathcal{SGC}^-$-COHÉRENCE *devient un problème co-NP-complet ;*
- $\mathcal{SGC}^-$-DÉDUCTION *devient un problème DP-complet ;*
- $\mathcal{SRC}^-$-INCOHÉRENCE *devient un problème semi-décidable ;*
- $\mathcal{SEC}^-$-DÉDUCTION *reste un problème semi-décidable ;*
- $\mathcal{SREC}^-$-DÉDUCTION *reste un problème indécidable ;*

Proof:

Co-NP-completeness of $\mathcal{SGC}$-CONSISTENCY : immediate.

DP-completeness of $\mathcal{SGC}$-DEDUCTION *:* this problem can be expressed as "is it true that Q can be projected into G and that no constraint of C can be projected into G ?" thus belongs to DP. Now let us consider that $\mathcal{C}$ contains only one constraint. A reduction from 3-SAT to PROJECTION (see f.i. Sect. 5.1.2) provides a straighforward reduction from SAT/UNSAT to $\mathcal{SGC}$-DEDUCTION (see f.i. [Papadimitriou, 1994]), thus the DP-completeness.

Semi-decidability of $\mathcal{SRC}$-UNCONSISTENCY *:* To prove unconsistency of a KB, we must find some violation of a constraint that will never be restored. But no violation of a negative constraint can ever be restored (further rule applications can only add information, thus more possible projections, and cannot remove the culprit one). So we only have to prove that the constraints can be deduced from $(\mathcal{G}, \mathcal{R})$: it is a semi-decidable problem. *Undecidability of* $\mathcal{SRC}$-DEDUCTION follows : we must prove that Q can be deduced from $(\mathcal{G}, \mathcal{R})$, but no constraint of $\mathcal{C}$ can.

The arguments proving semi-decidability of deduction in $\mathcal{SEC}$ and undecidability of deduction in $\mathcal{SREC}$ are the same as the ones used in the general case. □

Contraintes négatives et règles *range restricted*

La restriction aux contraintes négatives fait décroitre la classe de complexité des différents problèmes du modèle $\mathcal{SGC}$, mais n'est pas d'une grande efficacité dès que les règles sont présentes. En combinant contraintes négatives et règles *range restricted*, nous obtenons des résultats beaucoup plus intéressants.

Théorème 7.8 (Complexité pour les contraintes négatives et les règles r.r.)

- $\mathcal{SR}^{rr}\mathcal{C}^{-}$-COHÉRENCE *devient un problème co-NP-complet ;*
- $\mathcal{SR}^{rr}\mathcal{C}^{-}$-DÉDUCTION *devient un problème DP-complet ;*
- $\mathcal{SE}^{rr}\mathcal{C}^{-}$-DÉDUCTION *devient un problème* Σ_2^P*-complet ;*
- $\mathcal{SR}^{rr}\mathcal{E}^{rr}\mathcal{C}^{-}$-DÉDUCTION *devient un problème* Σ_2^P*-complet ;*

Proof: Unconsistency in $\mathcal{SRC}$ admits a polynomial certificate, a derivation (of polynomial length) leading to a graph into which a constraint can be projected and this projection. Unconsistency is thus in NP, and completeness results from the particular case when $\mathcal{R}$ is empty. For deduction, we must prove that no C can be deduced from $(\mathcal{G}, \mathcal{R})$, but that Q can. So the problem is in DP. For completeness, remark that the problem is still complete when $\mathcal{R}$ is empty.

To prove that deduction in $\mathcal{SEC}$ with r.r. rules and negative constraints is Σ_2P-complete, we will first show that it belongs to Σ_2^P, then exhibit a reduction from a Π_2^P-complete problem to its co-problem (since co-$\Sigma_i^P = \Pi_i^P$).

$\mathcal{SEC}$-DEDUCTION corresponds to the language $L = \{x \mid \exists y_1 \forall y_2\ R(x, y_1, y_2)\}$, where x encodes an instance $(Q; (\mathcal{G}, \mathcal{E}, \mathcal{C}))$ of the problem, and $(x, y_1, y_2) \in R$ if y_1 encodes an $\mathcal{E}$-derivation from $\mathcal{G}$ to G' and a projection from Q to G', and the mappings y_2 from constraints to G' are not projections (so G' does not violate any constraint, and it follows that any graph in the derivation from $\mathcal{G}$ to G' neither does).

We exhibit now a reduction from the general $\mathcal{SGC}$-CONSISTENCY problem to $\mathcal{SEC}$-NON-DEDUCTION with r.r. rules and negative constraints. Let $(\mathcal{G}, \mathcal{C} = \{C\})$ be an instance of $\mathcal{SGC}$-CONSISTENCY (w.l.o.g., we restrict the problem to consider only one positive constraint). The

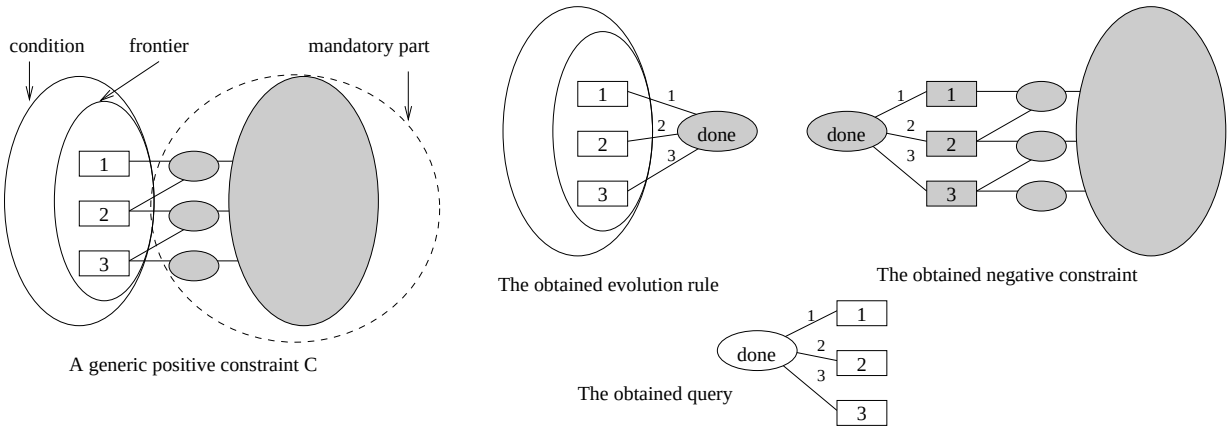


FIG. 7.9 – Transformation from $\mathcal{SGC}$-CONSISTENCY to a restricted $\mathcal{SEC}$-NON-DEDUCTION

transformation we consider builds an instance of $\mathcal{SEC}$-NON-DEDUCTION $(Q(C); (\mathcal{G}, \mathcal{E}(C), \mathcal{C}^-(C)))$ as follows : we call the *frontier* of the positive constraint C the set of nodes in the trigger (*i.e.* colored by 0) having at least a neighbor in the obligation. The definition of colored graphs implies that frontier nodes are concept nodes (their neighbors are thus relation nodes). Let us denote these frontier nodes by $1, \ldots, k$. The evolution rule $\mathcal{E}(C)$ has for hypothesis the trigger of C, and for conclusion a relation node typed `found`, where `found` is a new k-ary relation type incomparable with all other types. The i^{th} neighbor of the conclusion is the concept node i. Check that $\mathcal{E}(C)$ is a range restricted rule. The negative constraint $\mathcal{C}^-(C)$ is the subgraph of C composed of its obligation $(C_{(1)})$ added with nodes of the frontier and the relation node typed `found`, linked to the frontier nodes in the same way as above. Finally, the query $Q(C)$ is made of one relation node typed `found` and its neighbors frontier nodes. This transformation is illustrated in Fig. 7.9.

W.l.o.g. we can assume that $\mathcal{G}$ is irredundant : in that case, $\mathcal{SGC}$-CONSISTENCY is still Π_2^P-complete (see that the transformation used in the proof of Th. 7.3 produces an irredundant graph G). Now suppose that $(\mathcal{G}, C)$ is consistent : it means that either the trigger of C does not project into $\mathcal{G}$, and in that case, the rule $\mathcal{E}(C)$ will never produce the needed `found` node, or every (existing) projection of the condition of C in $\mathcal{G} = irr(\mathcal{G})$ can be extended to a projection of C as a whole. So every application of $\mathcal{E}(C)$ produces a violation of $\mathcal{C}^-(C)$. Then the query cannot be deduced from the knowledge base. Conversely, suppose that $\mathcal{G}$ π-violates C, then the application of $\mathcal{E}(C)$ following π produces a graph that does not violate $\mathcal{C}^-(C)$, and we can deduce $Q(\mathcal{G})$. □

Avec ce théorème, nous montrons une nette décroissance des classes de complexité dès que nous nous limitons à des contraintes négatives. $\mathcal{SGC}$-COHÉRENCE décroît de Π_2^P à co-NP et, lorsque nous considérons des règles *range restricted*, $\mathcal{SR}^{\mathrm{rr}}\mathcal{C}$-COHÉRENCE décroît de Π_2^P à DP, et $\mathcal{SE}^{\mathrm{rr}}\mathcal{C}$-DÉDUCTION décroît de Σ_3^P à Σ_2^P.

Relations entre les différents types de contraintes

Il serait intéressant d'exhiber des types particuliers de contraintes, plus généraux que les contraintes négatives, et qui permettent d'atteindre des classes de complexité intermédiaires entre celles des contraintes positives et celles des contraintes négatives (par exemple, Π_2^P et DP pour $\mathcal{SGC}$-COHÉRENCE).

Nous avons déjà évoqué le cas des contraintes déconnectées assimilables aux contraintes topologiques de [Mineau and Missaoui, 1997], mais nous pourrions aussi considérer la restriction, cette fois très syntaxique, à des contraintes *range restricted* (ce qui a un sens, cette notion étant définie sur les graphes colorés).

La propriété suivante met en relation ces différents types de contraintes :

Propriété 7.9 *Les contraintes négatives sont un cas particulier de contraintes déconnectées et de contraintes* range restricted.

Preuve: Comme nous l'avons déja indiqué (Prop. 7.2), les contraintes négatives sont équivalentes à des contraintes positives dont l'obligation est composée d'un seul sommet concept de type $\bot$. Cette contrainte est donc une contrainte déconnectée et, quitte dans le modèle des graphes conceptuels simples à remplacer ce type par un marqueur individuel, il s'agit aussi d'un graphe coloré *range restricted.* □

Contraintes déconnectées

Nous étudions maintenant la complexité des modèles obtenus lorsque l'on considère des contraintes déconnectées, en conjonction ou pas avec des règles *range restricted.* Nous noterons $\mathcal{C}^d$ un ensemble de contraintes déconnectées.

Théorème 7.9 (Complexité pour des contraintes déconnectées) *Lorsque l'on utilise des contraintes déconnectées et des règles quelconques :*

- $\mathcal{SGC}^{\lceil}$-COHÉRENCE *devient DP-complet ;*
- $\mathcal{SGC}^{\lceil}$-DÉDUCTION *devient DP-complet ;*
- $\mathcal{SRC}^d$-COHÉRENCE *reste indécidable ;*
- $\mathcal{SRC}^d$-DÉDUCTION *reste indécidable ;*
- $\mathcal{SEC}^d$-DÉDUCTION *reste semi-décidable ;*
- $\mathcal{SREC}^d$-DÉDUCTION *reste indécidable.*

Si nous combinons des contraintes déconnectées et des règles range restricted *:*

- $\mathcal{SR}^{rr}\mathcal{C}^d$-COHÉRENCE *devient DP-complet ;*
- $\mathcal{SR}^{rr}\mathcal{C}^d$-DÉDUCTION *devient DP-complet ;*
- $\mathcal{SE}^{rr}\mathcal{C}^d$-DÉDUCTION *devient* Σ_2^P*-complet ;*
- $\mathcal{SR}^{rr}\mathcal{E}^{rr}\mathcal{C}^d$-DÉDUCTION *devient* Σ_2^P*-complet.*

Proof: $\mathcal{SGC}$-UNCONSISTENCY belongs to DP, since we must prove that for one constraint there is a projection of its trigger and no projection of its obligation . Since DP = co-DP, $\mathcal{SGC}$-CONSISTENCY belongs to DP, and DP-completeness is proved with a reduction from SAT/UNSAT (as in proof of th. 7.7). For $\mathcal{SGC}$-DEDUCTION, we have to solve *independently* the $\mathcal{SGC}$-CONSISTENCY (DP-complete) and the $\mathcal{SG}$-DEDUCTION (NP-complete) problems, hence the result.

Arguments for undecidability of $\mathcal{SRC}$-CONSISTENCY, $\mathcal{SRC}$-DEDUCTION and $\mathcal{SREC}$-DEDUCTION, as well as semi-decidability of $\mathcal{SEC}$-DEDUCTION are the same as in the proof of Th. **??** : the constraint we used were already disconnected.

When rules are range-restricted, $\mathcal{SRC}$-UNCONSISTENCY belongs to DP : we must prove that the trigger of the constraint can be deduced from $(\mathcal{G}, \mathcal{R})$, but not its obligation. These problems belong respectively to NP and co-NP, so $\mathcal{SRC}$-CONSISTENCY belongs to DP. Completeness comes from the particular case where $\mathcal{R}$ is empty. $\mathcal{SRC}$-DEDUCTION adds independently an NP-complete problem, thus it is also DP-complete.

$\mathcal{SEC}$-DEDUCTION belongs to Σ_2^P when rules involved are range-restricted : indeed, the problem can be stated as "does there exist a sequence of graphs $\mathcal{G} = G_0, \ldots, G_p, G_{p+1}$, where $\mathcal{G} = G_0, \ldots, G_p$ is an $\mathcal{E}$-derivation and G_{p+1} is the disjoint union of G_p and $C_{(1)}$, a projection from Q to G' and a projection from $C_{(1)}$ to G_k, $0 \leq k \leq p+1$ such that for every graph $G_i, 0 \leq i < k$,*(*)* for every mapping π of $C_{(0)}$ into G_i, π is not a projection ?". Completeness follows from the particular case of negative constraints.

Proof for $\mathcal{SREC}$-DEDUCTION is similar : in the expression of the problem above, the derivation is now an $(\mathcal{E} \cup \mathcal{R})$-derivation, the G_i considered are only the ones obtained after the application of a rule from $\mathcal{E}$, and *(*)* "for every mapping π of $C_{(0)}$ into G_i" is replaced by "for every graph that can be $\mathcal{R}$-derived from G_i". □

7.3.3 Conclusion

Nous avons présenté dans ce chapitre une famille d'extensions aux modèles $\mathcal{SG}$ et $\mathcal{SR}$, basées sur l'introduction de contraintes utilisées pour tester la cohérence des graphes. Lorsque la base de connaissances ne contient que des graphes et des contraintes ($\mathcal{SGC}$), nous avons étudié les rapports entre nos contraintes et d'autres contraintes utilisées de la même façon dans la communauté « Graphes Conceptuels ». Les résultats de complexité que nous avons obtenu sont étendus à ces autres types de contraintes. Nous avons aussi montré que le problème de cohérence est équivalent au problème MIXED CSP, étendant les réseaux de contraintes.

Lorsque la base de connaissances contient aussi des règles, nous avons envisagé deux façons différentes de considérer les règles, menant à un troisième modèle lorsque l'on combine ces deux approches. Nous avons étudié de façon systématique la décidabilité et la complexité du problème de déduction dans ces différents modèles, en considérant des restrictions portant sur les contraintes, sur les règles, ou une combinaison des deux.

Les contraintes *range restricted* se sont révélées plus délicates à étudier : intuitivement, ces contraintes devraient être plus simples que les contraintes positives, mais l'impact de la mise sous forme redondante sur la complexité reste floue. Il est facile de vérifier que $\mathcal{SGC}^{\mathrm{rr}}$-DÉDUCTION (utilisant des contraintes *range restricted*) est au moins dans DP, et nous avons prouvé (bien que cette preuve ne soit pas dans ce mémoire) que ce problème appartient à Δ_2^P (i.e. P^{NP}). Cependant, nous n'avons pas su donner un résultat de complexité exact pour ce problème.

Les résultats de complexité obtenus sont résumés dans la table 7.1. Nous présentons aussi, dans la FIG. 7.10, une « carte de complexité » qui met en valeur les relations entre les différents problèmes étudiés. Dans cette figure, si X est un ensemble d'objets (contraintes ou règles), alors X^{rr}, X^d, X^- et X^{fes} représentent respectivement les restrictions à ces objets *range restricted*, déconnectés, négatifs et à expansion finie. Tous les problèmes re-

	Cas général	$\mathcal{R}$ fes	$\mathcal{E}$ fes	$\mathcal{R} \cup \mathcal{E}$ fes	$\mathcal{R}\&\mathcal{E}$ r.r.	$\mathcal{C}^-$	$\mathcal{R}\&\mathcal{E}$ r.r., $\mathcal{C}^-$	$\mathcal{R}\&\mathcal{E}$ r.r., $\mathcal{C}^d$
$\mathcal{SG}$-DED.	NP-C	/	/	/	/	/	/	/
$\mathcal{SGC}$-CONS.	Π_2^P-C	/	/	/	/	co-NP-C	co-NP-C	DP-C
$\mathcal{SGC}$-DED.	Π_2^P-C	/	/	/	/	DP-C	DP-C	DP-C
$\mathcal{SR}$-DED.	semi-dec.	dec.	/	dec.	NP-C	/	NP-C	NP-C
$\mathcal{SRC}$-CONS.	indec.	dec.	/	dec.	Π_2^P-C	co-semi-dec.	co-NP-C	DP-C
$\mathcal{SRC}$-DED.	indec.	dec.	/	dec.	Π_2^P-C	indec.	DP-C	DP-C
$\mathcal{SEC}$-DED.	semi-dec.	/	dec.	dec.	Σ_3^P-C	semi-dec.	Σ_2^P-C	Σ_2^P-C
$\mathcal{SREC}$-DED.	indec.	semi-dec.	indec.	dec.	Σ_3^P-C	indec.	Σ_2^P-C	Σ_2^P-C

TAB. 7.1 – Résumé des résultats de complexité

présentés sont complets pour la classe de complexité dans laquelle ils sont dessinés. Les arêtes doivent être considérées comme orientées du bas vers le haut. Une arête du problème P_1 vers le problème P_2 signifie que P_1 est un cas particulier de P_2. Afin d'obtenir une illustration lisible, nous n'avons pas représenté les problèmes intermédiaires entre deux problèmes de la même classe de complexité. La complexité de ces problèmes non représentés peut être obtenue en les « classifiant » dans cette hiérarchie. Par exemple, le problème $\mathcal{SEC}$-DÉDUCTION, non représenté, est plus général que $\mathcal{SR}$-DÉDUCTION (obtenu lorsque $C = \emptyset$), et plus spécifique que $\mathcal{SR}^{fes}\mathcal{EC}$-DÉDUCTION (obtenu en ajoutant un ensemble de règles d'évolution à expansion finie). Comme ces problèmes sont tous deux semi-décidables, alors $\mathcal{SEC}$-DÉDUCTION l'est aussi.

Enfin, une contrainte étant un graphe coloré tout comme une règle, et le déclenchement d'une contrainte (positive) étant identique à celui d'une règle, on peut intégrer les contraintes au graphe de dépendances : on ajoutera un sommet pour chaque contrainte, et l'on calculera les arcs (R_i, C_j) entre une règle R_i et une contrainte C_j. L'absence d'un arc (R_i, C_j) signifiera que l'application de la règle R_i ne nécessite pas de tester à nouveau si C_j est satisfaite. Cette intégration peut avoir de nombreux avantages : par exemple, si R_i déclenche sûrement une contrainte négative C_j, alors, si R_i est une règle d'évolution, nous saurons toute application de R_i génère un monde incohérent. Une telle connaissance peut servir à couper des cycles dans le graphe de dépendances, et donc à caractériser des résultats de décidabilité plus larges.

Cependant, cette intégration n'est pas aussi immédiate qu'on peut le penser. Par exemple, comment intégrer la notion de restauration dans ce schéma ? Nous reparlerons de ces problèmes dans la section consacrée aux perspectives.

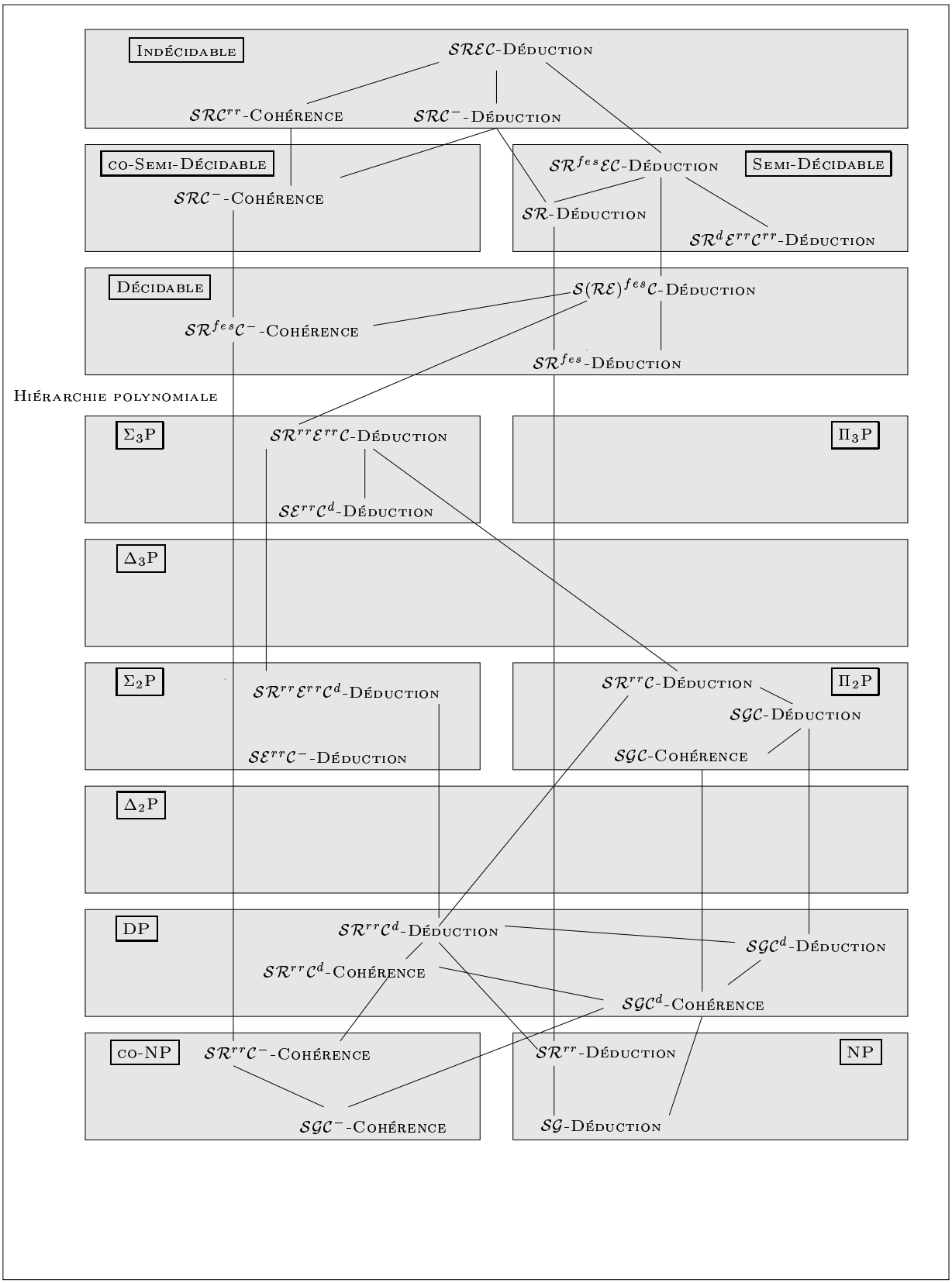


FIG. 7.10 – Une carte de complexité pour différents problèmes de la famille $\mathcal{SG}$

Chapitre 8

Conclusion

Bibliographie

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley. Traduction française : *Fondements des Bases de Données*, Vuibert Informatique, 2000.

[Alpay et al., 1995] Alpay, L., Amergé, C., Corby, O., Dieng, R., Giboin, A., Labidi, S., Lapalut, S., Després, S., Ferrandez, F., Fleury, D., Girard, Y., Jourdan, J.-L., Lechner, D., Michel, E., and Elslande, P. V. (1995). Acquisition et modélisation des connaissances dans le cadre d'une coopération entre plusieurs experts : Application à un système d'aide à l'analyse de l'accident de la route. Technical report, Rapport final du contrat DRAST n°93.0003.

[Baader et al., 1999] Baader, F., Molitor, R., and Tobies, S. (1999). Tractable and Decidable Fragments of Conceptual Graphs. Number ???? in Lecture Notes in Artificial Intelligence, pages 480–493. Springer.

[Baget, 1998] Baget, J.-F. (1998). Propositions pour l'extension du modèle des graphes conceptuels – Emboîtement, co-référence et règles de réecriture. Mémoire de dea, Université de Montpellier II.

[Baget, 1999] Baget, J.-F. (1999). A Simulation of Co-Identity with Rules in Simple and Nested Graphs. In *Proc. of ICCS'99*, pages 442–455.

[Baget, 2000] Baget, J.-F. (2000). Extending the CG Model by Simulations. In *Proc. of ICCS'00*, pages 277–291.

[Baget et al., 1999] Baget, J.-F., Genest, D., and Mugnier, M.-L. (1999). A Pure Graph-Based Solution to the SCG-1 Initiative. In *Proc. of ICCS'99*, pages 355–376.

[Baget and Mugnier, 2001a] Baget, J.-F. and Mugnier, M.-L. (2001a). Extensions of Simple Conceptual Graphs : the Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research*. Accepted for publication, sept. 2001.

[Baget and Mugnier, 2001b] Baget, J.-F. and Mugnier, M.-L. (2001b). The $\mathcal{SG}$ Family : Extensions of Simple Conceptual Graphs. In [Nebel, 2001], pages 205–210, Vol. 1.

[Baget and Tognetti, 2001] Baget, J.-F. and Tognetti, Y. (2001). Backtracking through Biconnected Components of a Constraint Graph. In [Nebel, 2001], pages 291–296, Vol. 1.

[Becker and Hereth, 2001] Becker, P. and Hereth, J. (2001). A Simplified Data Model for CGs. `http ://tockit.sourceforge.net/papers/cgDataModel.pdf`.

[Beierle et al., 1992] Beierle, C., Hedtstuck, U., Pletat, U., Schmitt, P., and Siekmann, J. (1992). An order-sorted logic for knowledge representation systems. *Journal of Arificial Intelligence*, 55 :149–191.

[Berge, 1970] Berge, C. (1970). *Graphes et hypergraphes.* Dunod.

[Bessière et al., 1999] Bessière, C., Meseguer, P., Freuder, E. C., and Larrosa, X. (1999). On Forward Checking for Non-Binary Constraint Satisfaction. In *Principles and Practice of Constraint Programming, Proc. of CP'99*, number 1713 in Lecture Notes in Artificial Intelligence, pages 88–102. Springer.

[Bessière and Régin, 1996] Bessière, C. and Régin, J. C. (1996). MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *Proc. of CP'96*, pages 61–75.

[Birkhoff, 1967] Birkhoff, G. (1967). *Lattice Theory*, volume 25 of *Coll. Publ. XXV.* American Mathematical Society.

[Bos et al., 1997] Bos, C., Botella, B., and Vanheegue, P. (1997). Modeling and Simulating Human Behaviors with Conceptual Graphs. In *Proc. of ICCS'97*, pages 275–289.

[Cao, 1999] Cao, T. H. (1999). *Foundations of Order-Sorted Fuzzy Set Logic Programming in Predicate Logic and Conceptual Graphs.* PhD thesis, University of Queensland, Australia.

[Chandra and Merlin, 1977] Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation on conjunctive queries in relational data bases. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, pages 77–90.

[Chein, 1997] Chein, M. (1997). The CORALI Project : From Conceptual Graphs to Conceptual Graphs via Labelled Graphs. Number 1257 in Lecture Notes in Artificial Intelligence, pages 65–79. Springer.

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

[Chein and Mugnier, 1995] Chein, M. and Mugnier, M.-L. (1995). Conceptual Graphs are also Graphs. In *Quatrièmes journées du Laboratoire d'Informatique de Paris-Nord*, pages 81–97.

[Chein et al., 1998] Chein, M., Mugnier, M.-L., and Simonet, G. (1998). Nested Graphs : A Graph-based Knowledge Representation Model with FOL Semantics. In *Proceedings of KR'98*, pages 524–535.

[Cogis and Guinaldo, 1995] Cogis, O. and Guinaldo, O. (1995). A linear descriptor for conceptual graphs and a class for polynomial isomorphism test. In [XXXXX, 1995], pages 263–277.

[Cook, 1971] Cook, S. A. (1971). The Complexity of Theorem-Proving Procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Comùputing*, pages 151–158. Association for Computing Machinery.

[Cormen et al., 1990] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* MIT Press/McGraw-Hill.

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

[Dau, 2000] Dau, F. (2000). Negations in Simple Concept Graphs. Number 1867 in Lecture Notes in Artificial Intelligence, pages 263–276. Springer.

[Debruyne and Bessière, 2001] Debruyne, R. and Bessière, C. (2001). Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230.

[Dechter, 1990] Dechter, R. (1990). Enhancement Schemes for Constraint Processing : Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41(3) :273–312.

[Dechter and Frost, 1999] Dechter, R. and Frost, D. (1999). Backtracking Algorithms for Constraint Satisfaction Problems. ICS technical report, University of California.

[Dechter and Pearl, 1988] Dechter, R. and Pearl, J. (1988). Network-Based Heuristics for Constraint Satisfaction Problems. *Artficial Intelligence*, 34(1) :1–38.

[Dibie et al., 1998] Dibie, J., Haemmerlé, O., and Loiseau, S. (1998). A Semantic Validation of Conceptual Graphs. In [Mugnier and Chein, 1998], pages 80–93.

[Donini et al., 1997] Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. (1997). The complexity of concept languages. *Information and Computation*, 134 :1–58.

[Euler, 1782] Euler, L. (1782). Recherches sur une nouvelle espèce de quarrés magiques. *Verh. Zeeuwsch. Genootsch. Wetensch. Vlissengen*, 9 :85–239.

[Fargier et al., 1996] Fargier, H., Lang, J., and Schiex, T. (1996). Mixed constraint satisfaction : a framework for decision problems under incomplete knowledge. In *Proc. of AAAI'96*, pages 175–180.

[Fargues et al., 1986] Fargues, J., Landau, M., Dugourd, A., and Catach, L. (1986). Conceptual Graphs for Semantics and Information Processing. *IBM Journal of Research and Development*, 30(1) :70–79.

[Feder and Vardi, 1993] Feder, T. and Vardi, M. (1993). Monotone Monadic SNP and Constraint Satisfaction. In *Proceedings of the 25th ACM STOC*, pages 612–622.

[Freuder and Wallace, 1991] Freuder, E. and Wallace, R. (1991). Selective relaxation for constraint satisfaction problems. In *Proc. of IEEE-ICTAI'91*, pages 332–339.

[Freuder, 1982] Freuder, E. C. (1982). A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29(1) :24–32.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability : a Guide to the Theory of NP-Completeness*. W. H. Freeman.

[Gaschnig, 1979] Gaschnig, J. (1979). Performance measurement and analysis of certain search algorithms. Research report CMU–CS–79–124, Carnegie-Mellon University.

[Genest, 2000] Genest, D. (2000). *Extension du modèle des graphes conceptuels pour la recherche d'informations*. PhD thesis, Université Montpellier II.

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

[Golomb and Baumert, 1965] Golomb, S. W. and Baumert, L. D. (1965). Backtrack Programming. *Journal of the ACM*, 12 :516–524.

[Gosh and Wuwongse, 1995] Gosh, B. C. and Wuwongse, V. (1995). A Direct Proof Procedure for Definite Conceptual Graphs Programs. In *Proceedings of ICCS'95*, pages 158–172.

[Gottlob et al., 1999] Gottlob, G., Leone, N., and Scarcello, F. (1999). A Comparison of Structural CSP Decomposition Methods. In *Proc. of IJCAI'99*, pages 394–399.

[Gruska, 1997] Gruska, J. (1997). *Foundations of Computing.* International Thomson Publisher Press.

[Guinaldo, 1996] Guinaldo, O. (1996). *Étude d'un gestionnaire d'ensembles de graphes conceptuels.* PhD thesis, Université de Montpellier II.

[Guinaldo and Haemmerlé, 1998] Guinaldo, O. and Haemmerlé, O. (1998). Knowledge Querying in the Conceptual Graph Model : The RAP Module. In [Mugnier and Chein, 1998], pages 287–294.

[Haemmerlé, 1995] Haemmerlé, O. (1995). *CoGITo : une plate-forme de développement de logiciels sur les graphes conceptuels.* PhD thesis, Université Montpellier II.

[Hahn and MacGillivray, 2001] Hahn, G. and MacGillivray, G. (2001). Graphs homomorphisms : Computational aspects and infinite graphs. Draft.

[Hahn and Tardif, 1997] Hahn, G. and Tardif, C. (1997). Graph homomorphisms : structure and symmetry. In *Graph symmetry*, number 497 in NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., pages 107–166. Kluwer.

[Haralick and Elliott, 1980] Haralick, R. M. and Elliott, G. L. (1980). Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14 :263–314.

[Hedrlín, 1969] Hedrlín, Z. (1969). On universal partly ordered sets and classes. *Journal of Algebra*, 11 :503–509.

[Hell and Nešetřil, 1979] Hell, P. and Nešetřil, J. (1979). Cohomomorphisms of graphs and hypergraphs. *Math. Nachr.*, 87 :53–61.

[Hell and Nešetřil, 1992] Hell, P. and Nešetřil, J. (1992). The core of a graph. *Discrete Math*, 109 :117–126.

[Hendrix, 1979] Hendrix, G. G. (1979). Encoding Knowledge in Partitioned Networks. In Findler, N. V., editor, *Associative Networks – Representation and Use of Knowledge by Computers*, pages 51–92. Academic Press.

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

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

[Kayser, 1997] Kayser, D. (1997). *La représentation des connaissances*. Hermes.

[Kerdiles, 1997] Kerdiles, G. (1997). Projection : A Unification Procedure for Tableaux in Conceptual Graphs. In *Proc. of TABLEAUX'97*, number 1227 in Lecture Notes in Artificial Intelligence, pages 138–152. Springer.

[Kerdiles, 2001] Kerdiles, G. (2001). *A landscape of conceptual graph logics*. PhD thesis, Université d'Amsterdam.

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

[Lehman, 1992] Lehman, F., editor (1992). *Semantic Networks in Artificial Intelligence*. Pergamon Press.

[Levin, 1973] Levin, L. A. (1973). Universal sorting problems. *Problemy Peredaci Informacii*, 9 :115–116. English translation in *Problems of Information Transmission*, 9, pp 265–266.

[Levinson and Ellis, 1991] Levinson, R. and Ellis, G. (1991). Multi-Level Hierarchical Retrieval. In *Proc. 6th Annual Workshop on Conceptual Graphs*, pages 67–81.

[Levy and Rousset, 1996] Levy, A. Y. and Rousset, M.-C. (1996). Verification of Knowledge Bases Based on Containment Checking. In *Proc. of AAAI'96*, pages 585–591.

[Mackworth, 1977] Mackworth, A. (1977). Consistency in Networks of Relations. *Artificial Intelligence*, 8(1) :99–118.

[Mamoulis and Stergiou, 2001] Mamoulis, N. and Stergiou, K. (2001). Solving Non-Binary CSPs Using the Hidden Variable Encoding. To appear at 7th International Conference on Principles and Practice of Constraint Programming (CP2001).

[Meseguer, 1989] Meseguer, P. (1989). Constraint satisfaction problems : An overview. *AI communications*, 2(1) :3–17.

[Mineau and Missaoui, 1997] Mineau, G. W. and Missaoui, R. (1997). The Representation of Semantic Constraints in Conceptual Graphs Systems. In *Proc. of ICCS'97*, pages 138–152.

[Montanari, 1974] Montanari, U. (1974). Networks of Constraints : Fundamental Properties and Application to Picture Processing. *Information Sciences*, 7(2) :95–132.

[Mugnier, 1992] Mugnier, M.-L. (1992). *Contributions algorithmiques pour les graphes d'héritage et les graphes conceptuels*. PhD thesis, Université Montpellier II.

[Mugnier, 1995] Mugnier, M.-L. (1995). On generalization/specialization for conceptual graphs. *Journal of Experimental and Theoretical Artificial Intelligence*, 7 :325–344.

[Mugnier, 2000] Mugnier, M.-L. (2000). Knowledge Representation and Reasonings Based on Graph Homomorphism. In *Proc. of ICCS'2000*, pages 172–192.

[Mugnier and Chein, 1993] Mugnier, M.-L. and Chein, M. (1993). Polynomial Algorithms for Projection and Matching. In *Conceptual Structures : Theory and Implementation, Selected papers from AWCG'92*, number 754 in Lecture Notes in Artificial Intelligence.

[Mugnier and Chein, 1996] Mugnier, M.-L. and Chein, M. (1996). Représenter des connaissances et raisonner avec des graphes. 10(1) :7–56.

[Mugnier and Chein, 1998] Mugnier, M.-L. and Chein, M., editors (1998). *Conceptual Structures : Theory, Tools and Applications (proceedings of the 6th International Conference on Conceptual Structures, ICCS'98)*, number 1453 in Lecture Notes in Artificial Intelligence. Springer.

[Nebel, 2001] Nebel, B., editor (2001). *Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI'01).*

[Nešetřil, 2000] Nešetřil, J. (2000). The Coloring Poset and its On-Line Universality. KAM-DIMIATA series 2000-458
`http ://nikam.ms.mff.cuni.cz/~kamserie/kamser.html#kamser-2000`.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity.* Addison-Wesley.

[Post, 1947] Post, E. L. (1947). Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12(1) :1–11. Reprinted in : M. Davis (Ed.), The Collected Works of Emil L. Post, Birkhauser, Boston 1994, pp. 503-513.

[Preller et al., 1998] Preller, A., Mugnier, M.-L., and Chein, M. (1998). Logic for Nested Graphs. *Computational Intelligence*, 14(3) :335–357.

[Prosser, 1993] Prosser, P. (1993). Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3) :268–299.

[Pultr and Trnková, 1980] Pultr, A. and Trnková, V. (1980). *Combinatorial, Algebraic and Topological Representations of Groups, Semigroups and Categories.* North Holland.

[Quillian, 1968] Quillian, M. R. (1968). Semantic Memory. In Minsky, M., editor, *Semantic Information Processing*, pages 227–270. MIT Press.

[Rao and Foo, 1987] Rao, A. and Foo, N. (1987). CONGRES : Conceptual Graph Reasoning System. In *Proc. of the 3rd Conference on Artificial Intelligence Apllications*, pages 87–92. IEEE Computer Society Press.

[Raynaud, 2000] Raynaud, O. (2000). *Codage et représentation des ensembles ordonnés.* PhD thesis, Université Montpellier II.

[Régin, 1995] Régin, J. C. (1995). *Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique.* PhD thesis, Université de Montpellier II.

[Roberts, 1992] Roberts, D. D. (1992). The Existential Graphs. In [Lehman, 1992], pages 639,663.

[Rogozhin, 1996] Rogozhin, Y. (1996). On the notion of universality and small universal Turing machines. *Theoretical Computer Science*, 168 :215–240.

[Rudolf, 1998] Rudolf, M. (1998). Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. In *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98*, volume 1764 of *Lecture Notes in Computer Science*, pages 238–251. Springer.

[Sabin and Freuder, 1994] Sabin, D. and Freuder, E. C. (1994). Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI'94*, pages 125–129.

[Salvat, 1997] Salvat, E. (1997). *Raisonner avec des opérations de graphes : graphes conceptuels et règles d'inférence*. PhD thesis, Université Montpellier II.

[Salvat and Mugnier, 1996] Salvat, E. and Mugnier, M.-L. (1996). Sound and Complete Forward and Backward Chainings of Graphs Rules. In *Proc. of ICCS'96*, pages 248–262.

[Simonet, 1996] Simonet, G. (1996). Une autre sémantique logique pour les graphes conceptuels simples ou emboîtés. Research Report RR-LIRMM 96-048, LIRMM.

[Simonet, 1998] Simonet, G. (1998). Two FOL Semantics for Simple and Nested Conceptual Graphs. In [Mugnier and Chein, 1998], pages 240–254.

[Smith, 1996] Smith, B. (1996). Locating the Phase Transition in Binary Constraint Satisfaction Problems. *Artificial Intelligence*, 81 :155–181.

[Smullyan, 1968] Smullyan, R. (1968). *XXX*. XXX.

[Sowa, 1976] Sowa, J. F. (1976). Conceptual Graphs for a Database Interface. *IBM Journal of Research and Development*, 20(4) :6–57.

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

[Stockmeyer, 1977] Stockmeyer, L. J. (1977). The polynomial-time hierarchy. *Theoretical Computer Science*, 3 :1–22.

[Tarjan, 1972] Tarjan, R. E. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM J. of Computing*, 1 :146–160.

[Thue, 1914] Thue, A. (1914). Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skr. Viedensk. Selsk. I*, 10.

[Turing, 1937] Turing, A. (1937). On computable numbers, with an application to the 'entscheidungsproblem'. *Proc. of the London Math. Soc. 2nd series*, 42 :230–265. Correction ibid Vol. 43 pp 544–546.

[Vismara, 1995] Vismara, P. (1995). *Reconnaissance et représentation d'éléments structuraux pour la description d'objets complexes. Application à l'élaboration d'un plan de synthèse en chimie organique*. PhD thesis, Université de Montpellier II.

[Welzl, 1984] Welzl, E. (1984). Symmetric Graphs and Interpretations. *Journal of Combinatorial Theory*, B 37 :235–244.

[Wermelinger, 1995] Wermelinger, M. (1995). Conceptual Graphs anf First-Order Logic. In [XXXXX, 1995].

[Wermelinger and Lopes, 1994] Wermelinger, M. and Lopes, J. G. (1994). Basic conceptual structures theory. Number 835 in Lecture Notes in Artificial Intelligence, pages 144–159. Springer.

[XXXXX, 1995] XXXXX, editor (1995). *Conceptual Structures : XXXX (proceedings of the 3rd International Conference on Conceptual Structures, ICCS'95)*, number 954 in Lecture Notes in Artificial Intelligence. Springer.

Annexe A

Documentation for the Sisyphus-I Problem

Nous rappelons ici les données du problème SISYPHUS-I tel qu'il a été posé à la communauté « Acquisition de Connaissances ». Le texte présenté ici est l'énoncé exact de ce problème, tel qu'on peut le trouver sur le site du Knowledge Acquisition Workshop (`http ://ksi.cpsc.ucalgary.ca/KAW/Sisyphus/`). La solution à ce problème que nous avons proposée [Baget et al., 1999] est donnée dans l'App. B, afin de présenter un exemple complet de modélisation dans $\mathcal{SREC}$.

We want to compare different approaches to the modeling of problem-solving processes in knowledge-based systems and the influences of the models on the knowledge acquisition activities. To this purpose we give a short description of a sample problem concerned with office assignment in a research environment. Research groups who participate in SISYPHUS-I are requested to describe how this problem can be solved and represented in their approach. To ease comparison and to allow for a well-founded discussion, we propose a structure for the reports. Please adhere to it.

In this second round of SISYPHUS-I, we decided to use almost the same problem statement, that is the office-assignment task. It has been changed a little bit, by adding a second set of slightly changed data (see Section 2.5). We assume that this will give the reader an even better chance to understand the functioning of your approach. Additionally, this introduces the issue of brittleness.We would like to know how a model reacts to unusual cases.

Furthermore, we require traces of the problem-solving processes of your system for both problem statements. This means that this year we want descriptions of operational problem solvers. [...]

Statement of the Sample Problem

The members of the research group YQT of the HNE laboratory are moved to a new floor of their château. Due to severe cuts in funding they only get a very limited number of offices. It will be quite a problem to cram them all in. To complicate matters even further some will have to share an office. After several vain attempts each ending as nightmares that would have impressed Freddy, the management of HNE is desperate. SISYPHUS-I is their last hope. HNE implores the SISYPHUS-I teams to provide knowledgeable systems that are up to the task.

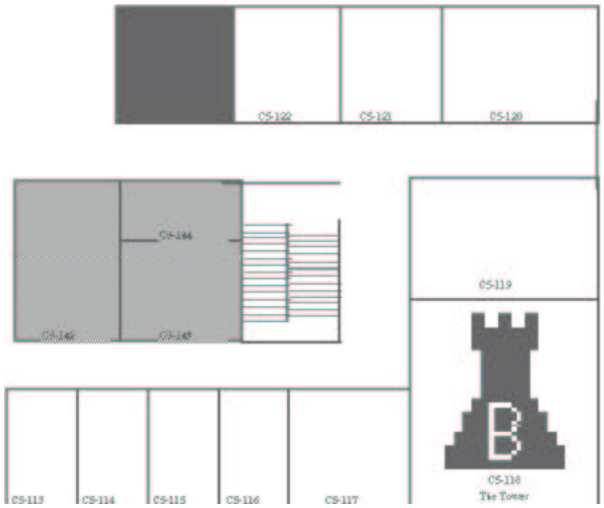


FIG. A.1 – The part of the floor-plan of the château that we will consider

It is important that the systems' ways to solve the problem follow the shining example of the wizard Siggi D., the only one who ever managed to solve the problem. The system developers should be aware of the fact that YQT's members are used to be pampered. They all have their personal preferences and professional peculiarities that should better be observed, as the dungeons of the BABYLON tower are deep and lonely.

Data on People and Offices

Description of the Members of YQT Not all members of YQT can profit from this new office space in the château, about half of the group stay in their old offices. Those that are concerned by the new assignment are :

Werner L. Role = Researcher Project = RESPECT Smoker = No Hacker = True Works-with= Angi W., Marc M.	**Jürgen L.** Role = Researcher Project = EULISP Smoker = No Hacker = True Works-with= Harry C., Thomas D.
Marc M. Role = Researcher Project = KRITON Smoker = No Hacker = True Works-with= Angi W., Werner L.	**Angi W.** Role = Researcher Project = RESPECT Smoker = No Hacker = No Works-with = Marc M., Werner L.
Andy L. Role = Researcher Project = TUTOR2000 Smoker = Yes Hacker = No Works-with =	**Michael T.** Role = Researcher Project = BABYLON Product Smoker = No Hacker = Yes Works-with = Hans W.
Harry C. Role = Researcher Project = EULISP Smoker = No Hacker = Yes Works-with = Jürgen L., Thomas D.	**Uwe T.** Role = Researcher Project = Autonomous Systems Smoker = Yes Hacker = Yes Works-with =

Thomas D. Role = Researcher Project = EULISP Smoker = No Hacker = No Works-with = Jürgen L., Harry B.	**Monika X.** Role = Secretary Smoker = No Hacker = No Works-with = Thomas D., Ulrike U., Eva I.
Ulrike U. Role = secretary Smoker = No Hacker = No Works-with = Thomas D., Monika X., Eva I.	**Hans W.** Role = Researcher Project = BABYLON Product Smoker = Yes Hacker = No Works-with = Michael T.
Eva I. Role = Manager Smoker = No Hacker = No Works-with = Thomas D., Ulrike U., Monika X.	**Joachim I.** Role = Researcher Project = ASERTI Smoker = No Hacker = No Works-with =
Katharina N. Role = Researcher Project = MLT Smoker = Yes Hacker = Yes Works-with =	

Hierarchical Structures Within YQT : Within the subset of members of YQT we have the following organizational structure : Thomas D. is the head of the group YQT ; Eva I. manages YQT ; Monika X. and Ulrike U. are the secretaries ; Werner L. and Angi W. work together in the RESPECT project ; Harry B., Jürgen L. and Thomas D. work in the EULISP project ; Michael M. and Hans W. work in the Babylon Product project ; Hans W. is the head of this large project, Marc M., Uwe T. and Andy L. pursue individual projects ; Katharina N. and Joachim I. are heads of larger projects that are not considered in this problem.

The Offices on YQT's Floor of the Château : C5-123, C5-122, C5-121, C5-120, C5-119 and C5-117 are large rooms that can host two researchers. Large rooms can be assigned to heads of groups too. C5-113, C5-114, C5-115 and C5-116 are single rooms.

A Sample Transcript of a Protocol

The words of the wizard Siggi D.		**Comments, questions and annotations**	
1	Put Thomas D. into office C5117	1a	The head of group needs a central office, so that he/she is as close to all the members of the group as possible. This should be a large office.
		1b	This assignment is defined first, as the location of the office of the head of group restricts the possibilities of the subsequent assignments.
2	Monika X. and Ulrike U. into office C5119	2a	The secretaries' office should be located close to the office of the head of group. Both secretaries should work together in one large office. This assignment is executed as soon as possible, as its possible choices are extremely constrained.
3	Eva I. into C5-116	3a	The manager must have maximum access to the head of group and to the secretariat. At the same time he/she should have a centrally located office. A small office will do.
		3b	This is the earliest point where this decision can be taken.
4	Joachim I. into C5-115	4a	The heads of large projects should be close to the head of group and the secretariat. There really is no reason for the sequence of the assignments of Joachim, Hans and Katharina.
5	Hans W. into C5-114	5a	The heads of large projects should be close to the head of group and the secretariat.
6	Katharina N. into C5-113	6a	The heads of large projects should be close to the head of group and the secretariat.

7	Andy K. and Uwe T. into C5120	7a	Both smoke. To avoid conflicts with non-smokers they share an office. Neither of them is eligible for a single office. This is the first twin-room assignment, as the smoker/non-smoker conflict is a severe one.
8	Werner L. and Jürgen L into C5123	8a	They are both implementing systems, both non-smokers. They do not work in the same project, but they work on related subjects. Members of the same project should not share offices. Sharing with members of other projects enhances synergy effects within the research group.
		8b	There really are no criteria of the sequence of these twin room assignments.
9	Marc M. and Angi W. into C5122	9a	Marc is implementing systems, Angi isn't. This should not be a problem. Putting them together would ensure good cooperation between the RESPECT and the KRITON projects.
10	Harry C. and Michael T. into C5-121	10a	They are both implementing systems. Harry develops object systems, Michael uses them. This creates synergy.

Note 1 Our wizard Siggi D. seems to pursue a general strategy of assigning the head of group and the staff personnel first, followed by the heads of large projects, who through their seniority are eligible for single offices (some are more equal then others). The offices of the head of group and the staff should be close to each other. Heads of projects should, if possible be allocated offices close to the head of group's.

Note 2 Twin offices are assigned to the members of research projects, under the consideration that synergy among projects is boosted. This means that researchers that work in same projects are, if possible not sharing an office. Coworkers that work on related subjects can share an office. It is important not to put smokers and non-smokers together into twin offices.

A Second Problem Statement

The second problem statement reflects a a more recent situation at HNE. Katharina N. left, and Christian-I joined the group. Christian I. is just a researcher, he smokes, he works in MLT, and he is a hacker. He is not a head of project.

The office situation stays the same. Again, all the members of the group must be assigned. Obviously, the situation has changed, and the system must show how it copes with this slightly modified data-set.

Remember that this second problem statement has been introduced to adress the issue of brittleness. We would like to see which general precautions your model takes, so that it can cope with borderline cases.

Annexe B

Our Modelization of Sisyphus-I

Nous donnons ici un extrait de la solution que nous avons proposée pour la modélisation (et la résolution) du problème SISYPHUS-I dans [Baget et al., 1999]. Nous présentons ici tous les graphes (faits, règles, contraintes, requête) utilisés pour représenter ce problème dans le modèle $\mathcal{SREC}$. Toutes les règles étant *range restricted*, l'ensemble des solutions peut être trouvé en un temps fini. Contrairemment aux conventions adoptées dans cette thèse, la couleur 0 est ici représentée en noir (au lieu du gris).

Support, Facts and Query

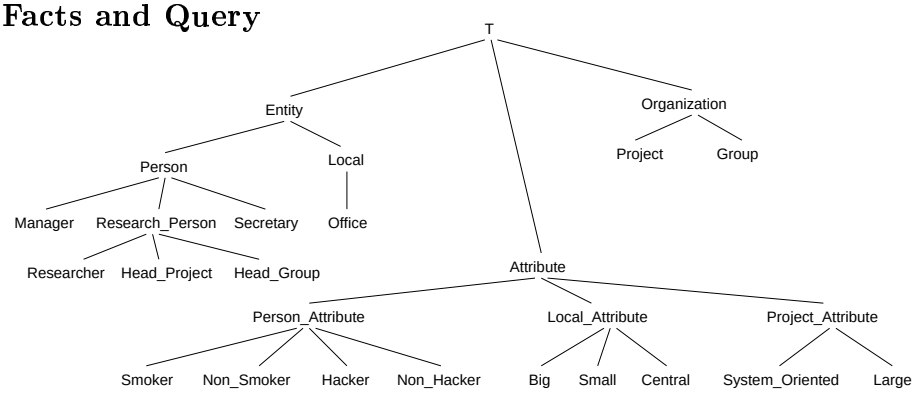


FIG. B.1 – Hierarchy of concept types

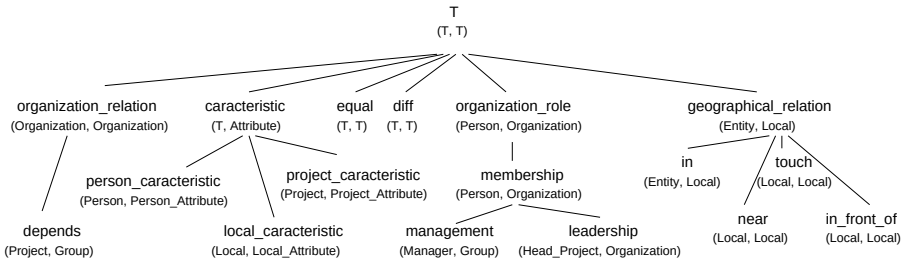


FIG. B.2 – Hierarchy of relation types

The Support The support defined in fig. B.1 and B.2 represents all types used in our modelization. Each relation type in fig. B.2 is provided with its signature.

Facts The initial graph is obtained by making the disjoint union of the graphs represented in figs. B.5, B.6, B.3 and B.4. These graphs have been separated for better readability. To ensure completeness of the reasoning process, we compute the *normal form* of the resulting graph, by merging concept nodes having the same individual marker. For a readability purpose, we did not represent *diff* relations in these graphs, though in the graph of fig. B.5, all pairs of concept nodes typed *Office* and having different individual markers are assumed to be linked by a relation node typed *diff*. The same assumption is done for the concept nodes typed *Project* in graph of fig. B.6, and for concept nodes whose type is more specific than *Person* in graph of fig. B.4.

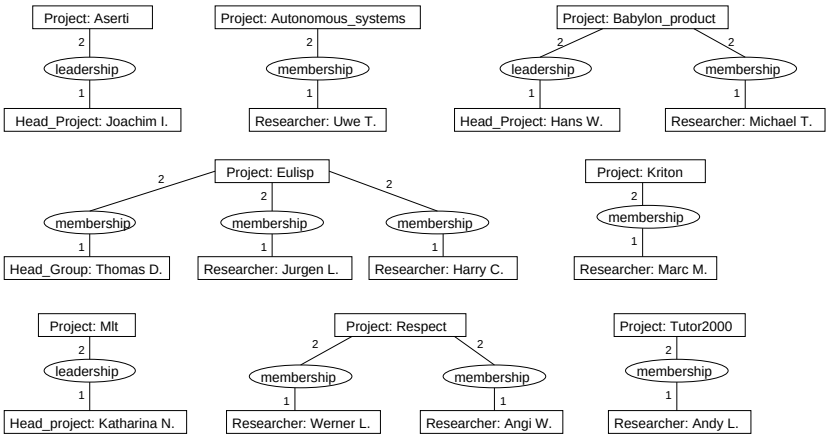


FIG. B.3 – Team members of the YQT group

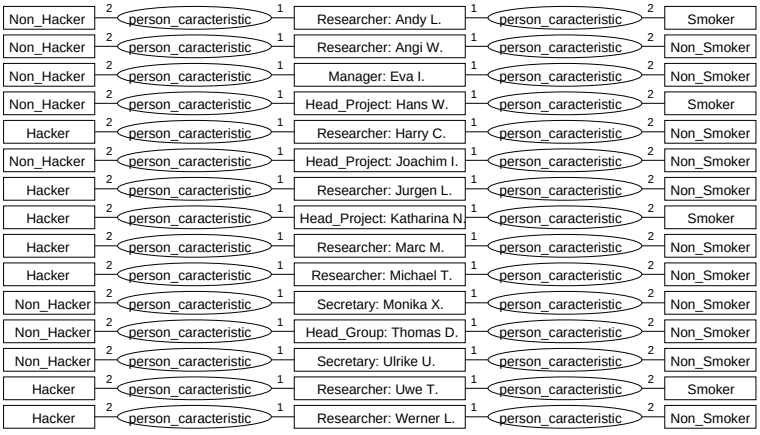


FIG. B.4 – Personal data about members of the YQT group

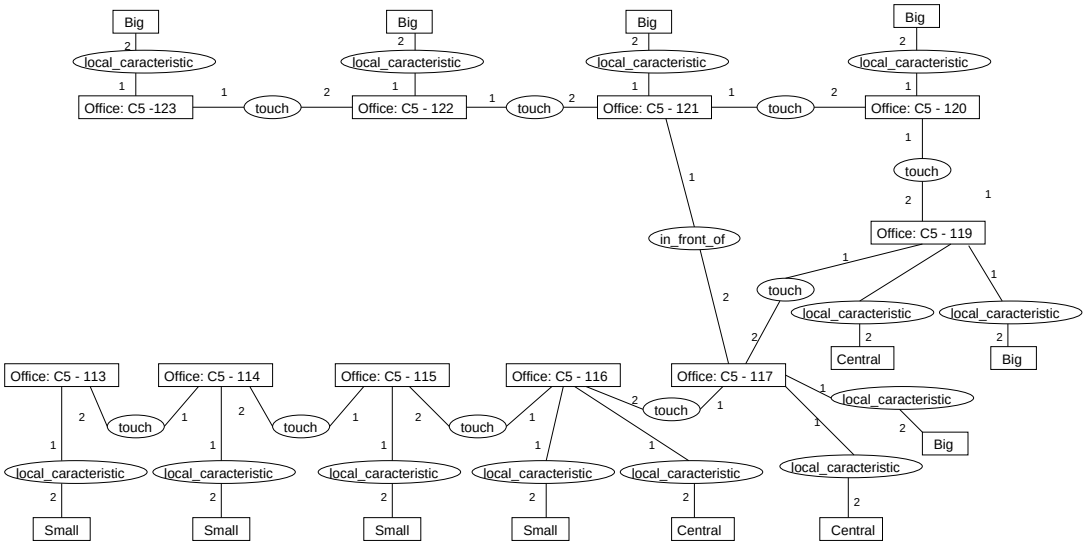


FIG. B.5 – Geographical information for the first floor of the château of HNE

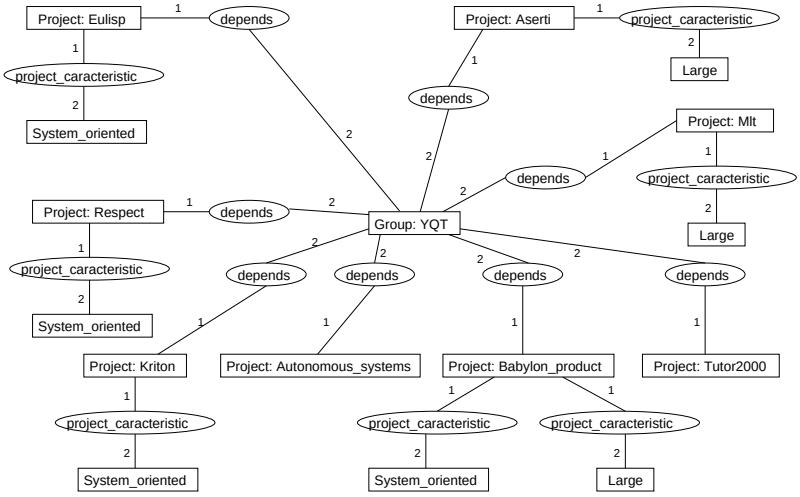


FIG. B.6 – Organizational structure of the YQT group

Query The graph we want to deduce is the solution to the Sisyphus-I problem : we want every person being placed into an office. This graph is represented in fig. B.7. Note that, due to individual markers, no *diff* relation is needed between the concept nodes representing persons.

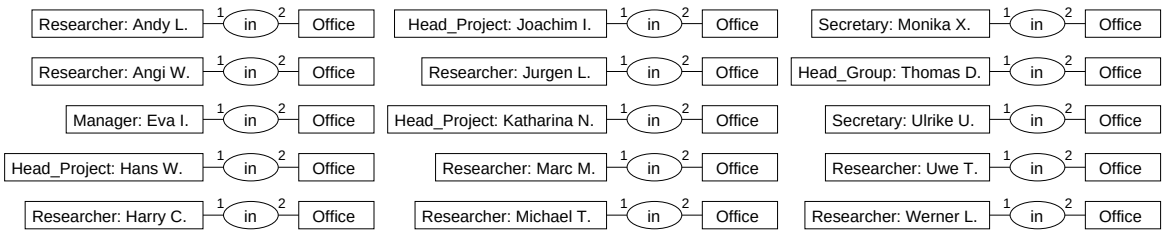


FIG. B.7 – The query

Rules and Constraints

Implicit Knowledge – the Set of Rules $\mathcal{R}$ The five rules represented in fig. B.8 express that :

A. The relation *near* is symmetrical.
B. Two locals that *touch* the same one are considered *near*.
C. The relation *touch* is symmetrical.
D. Two locals that *touch* each other are considered *near*.
E. Two locals that are *in_front_of* each other are considered *near*.

Note that the two last rules could be replaced by expressing that *in_front_of* and *touch* are two relation types more specific than *near*. We also used rules expressing that *in_front_of* and *diff* are symmetrical.

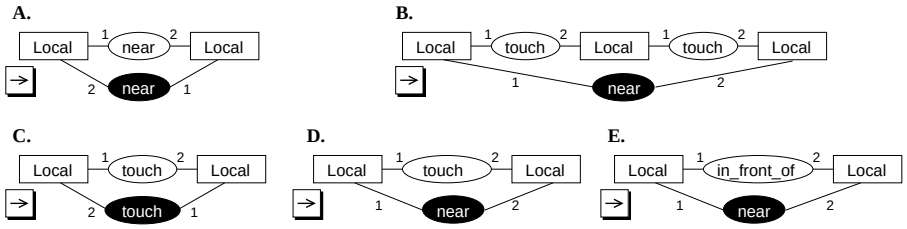


FIG. B.8 – Rules representing implicit knowledge

Constraints – the Set $\mathcal{C}$ Constraints related to possible assignments can be either positive or negative. We sorted these constraints in three categories of decreasing priority : absolute, strong and weak. It is not possible to violate an *absolute* constraint. *Strong* constraints are to be satisfied by any solution, but, if the problem is over-constrained, the user may accept to reformulate it by modifying this set of constraints. *Weak* constraints represent preferences.

Every constraint used is described below, but whenever we have very similar constraints, only one of them is drawn. These constraints are represented in order of declining priorities.

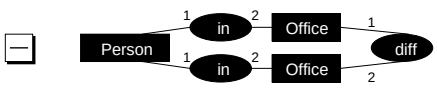


FIG. B.9 – Ubiquity ?

The graph represented in fig. B.9 is a negative constraint. It expresses that a person cannot be into two different places at the same time. This is an *absolute* constraint.

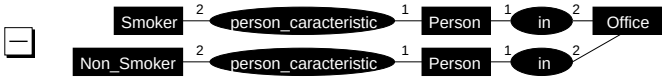


FIG. B.10 – Smoker – Non-Smoker antagonism

The graph represented in fig. B.10 is a negative constraint. It expresses that no office can host a *Smoker* and a *Non_ Smoker*. This is a *strong* constraint.

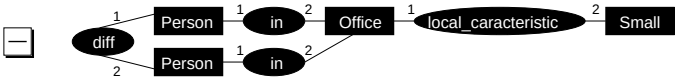


FIG. B.11 – Number of persons in small offices

The graph represented in fig. B.11 is a negative constraint. It expresses that a small office cannot host more than one person. This is a *strong* constraint.

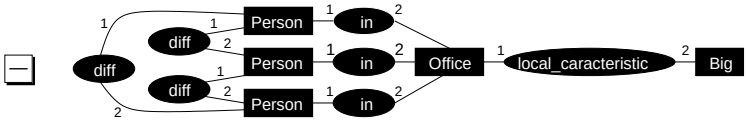


FIG. B.12 – Number of persons in big offices

The graph represented in fig. B.12 is a negative constraint. It expresses that a big office cannot host more than two persons. This is a *strong* constraint.

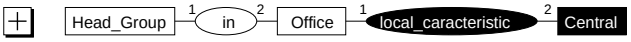


FIG. B.13 – Head of group accessibility

The graph represented in fig. B.13 is a positive constraint. It expresses that the head of group needs a central office (if the head of group is in an office, then this office has to be a central one). This is a *strong* constraint. In the same way, the manager would be pleased to have a central office, but we consider this as only a *weak* constraint.

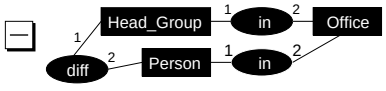


FIG. B.14 – Privileges of the head of group (1)

The graph represented in fig. B.14 is a negative constraint. It expresses that the head of group needs to be alone in his office. This is a *strong* constraint. In the same way, the manager as well as heads of large projects would be pleased to be alone in their office, but we consider this as only *weak* constraints.

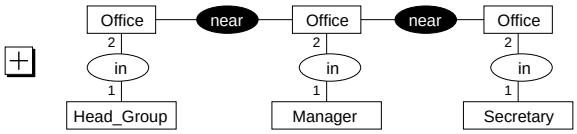


FIG. B.15 – Manager's neighborhood

The graph represented in fig. B.15 is a positive constraint. It expresses that the manager needs to be near the head of group as well as the secretariat. This is a *strong* constraint. In the same way, heads of large projects should be close to the head of group as well as the secretariat, but we only consider this as only a *weak* constraint.

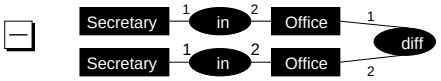


FIG. B.16 – Secretariat holds secretaries

The graph represented in fig. B.16 is a negative constraint. It expresses that no two secretaries can be in different offices. Note that this constraint can be satisfied since there are only two secretaries. Otherwise, should we want "secretaries-only" offices, we could express it by a positive constraint : "if a person is in the same office as a secretary, this person must also be a secretary". This is a *weak* constraint.

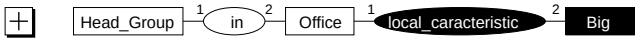


FIG. B.17 – Privileges of the head of group (2)

The graph represented in fig. B.17 is a positive constraint. It expresses that the head of group would like to have a big office. This is a *weak* constraint.

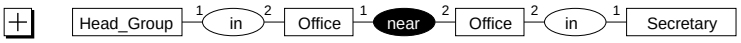


FIG. B.18 – Secretariat's accessibility

The graph represented in fig. B.18 is a positive constraint. It expresses that the secretaries' office should be located close to the office of the head of group. This is a *weak* constraint.

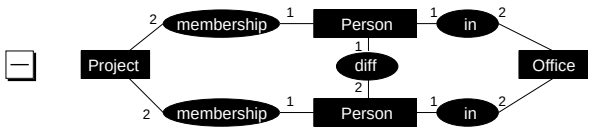


FIG. B.19 – Synergy

The graph represented in fig. B.19 is a negative constraint. It expresses that members of a same project should not share offices. This is a *weak* constraint.

Transformations – the Set of Rules $\mathcal{T}$ The simplest set of transformation rules $\mathcal{T}$ that would do the job is composed of a single rule : "if you can find a *person* and an *office*, try to place that *person* in that *office*" (fig. B.20).

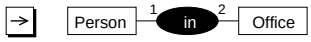


FIG. B.20 – Assigning a person to an office

Experimental Results

The research engine found *2880* solutions to the Sisyphus-I problem. These solutions satisfy all constraints excepted a weak one which expresses that heads of large projects should be next to the secretariat and the head of group. Obviously, this constraint cannot be satisfied. Given our modeling, the number of solutions can be explained as follows :

- The head of group can only be in the office 117 or 119. *(2 solutions)*
- The manager can only be placed in a small central office, i.e. the office 116. *(1 solution)*
- The positions of the head group and manager determine the position of the secretariat (the office which has not been assigned to the head of group). *(1 solution)*
- There are only three small offices left, and they must be assigned to the three heads of large projects, which should be alone in their office. *(6 solutions)*
- The two smokers must be together and have the choice between the 4 big offices left. *(4 solutions)*
- There are 15 possible sets of couples for the 6 researchers left, of them 5 are impossible (they are in the same project). There are 6 possible assignments for these couples in the 3 last big offices. *(60 solutions)*

Finally, we have the $2 \times 1 \times 1 \times 6 \times 4 \times 60 = 2880$ different solutions exhibited by our research engine. Two of the computed solutions are presented in fig. B.21. The first solution we find is the solution A in fig. B.21. The beginning of the research tree used to find this solution is presented in fig. B.22. We can see there one backtrack, as the first secretary is assigned a small office (no explicit constraint forbids it), and problems arise only when we try to assign the same small office to the second secretary. We explored 85 different worlds before generating the first valid solution (so we had 70 backtracks), and we must remind here that each of these worlds required to solve the problem of its validity, which is a co-NP-complete problem. However, the length of the computation to find the first solution for this particular problem (i.e. prove that the problem is overconstrained, remove a constraint and find an acceptable solution) was around 30 seconds.

Office	Solution A
113	Hans W. (Head of Large Project)
114	Katharina N. (Head of Large Project)
115	Joachim I. (Head of Large Project)
116	Eva I. (Manager)
117	Thomas D. (Head of group)
119	Monika X. Ulrike U. (Secretaries)
120	Andy L. Uwe T. (Researchers, smokers)
121	Michael T. Mark M. (Researchers, non smokers)
122	Jurgen L. Werner L. (Researchers, non smokers)
123	Angi W. Harry C. (Researchers, non smokers)

FIG. B.21 – One of the 2880 solutions to the Sisyphus-I problem

Thomas D. → 113 Violation : Not Central
Thomas D. → 114 Violation : Not Central
Thomas D. → 115 Violation : Not Central
Thomas D. → 116 Violation : Small Office
Thomas D.→ 117
  Eva I. → 113 Violation : Not Central
  Eva I. → 114 Violation : Not Central
  Eva I. → 115 Violation : Not Central
  Eva I. → 116
    Monika X. → 113 Violation : Far from Manager
    Monika X. → 114 Violation : Far from Manager
    Monika X. → 115
      Ulrike U. → 113 Violation : Far from Manager
      Ulrike U. → 114 Violation : Not with other Secretary
      Ulrike U. → 115 Violation : Two persons in a small office
      Ulrike U. → 116 Violation : Two persons in a small office
      Ulrike U. → 117 Violation : Head of group must be alone
      Ulrike U. → 119 Violation : Not with other Secretary
      [...] (120, 121, 122, 123) Violation : Not with other Secretary
    Monika X. → 116 Violation : Two persons in a small office
    Monika X. → 117 Violation : Head of group must be alone
    Monika X. → 119
      Ulrike U. → 113 Violation : Far from Manager
      Ulrike U. → 114 Violation : Not with other Secretary
      Ulrike U. → 115 Violation : Not with other Secretary
      Ulrike U. → 116 Violation : Two persons in a small office
      Ulrike U. → 117 Violation : Head of group must be alone
      Ulrike U. → 119
        Hans W. → ...

FIG. B.22 – A trace of the backtrack leading to the solution A

Annexe C

Quelques rappels sur les ordres

Différents types de relations d'ordre sont utilisés dans la communauté « Graphes Conceptuels » pour définir le support. Nous rappelons ici les définitions et propriétés élémentaires sur les ordres. Cette courte présentation doit beaucoup au chapitre introductif de [Raynaud, 2000]. Le lecteur pourra se référer, pour plus de précision, à [Birkhoff, 1967].

Préordre

Soit T un ensemble. On appelle *préordre* sur T une relation $\leq$ sur T, réflexive ($\forall t \in T, t \leq t$) et transitive ($\forall t, t', t'' \in T,\ t \leq t'$ et $t' \leq t'' \ \Rightarrow\ t \leq t''$). Un préordre $\leq$ peut être obtenu à partir de n'importe quelle relation R par sa *fermeture réflexo-transitive* : $\forall t \in T,\ t \leq t$ et $\forall t, t', t'' \in T,\ t \leq t'$ et $t'\ R\ t'' \ \Rightarrow\ t \leq t''$. Nous dirons que t et t' sont équivalents, et noterons $t \equiv t'$, si $t \leq t'$ et $t' \leq t$.

Ordre ou ordre partiel

Un *ordre* sur T est un préordre antisymétrique ($\forall t, t' \in T, t \equiv t' \Rightarrow t = t'$). Tout préordre $\leq_p$ sur un ensemble T induit de façon naturelle un ordre : il suffit de considérer l'ensemble C des classes d'équivalence de T pour $\equiv_p$ (en posant $c(t) = c(t')$ ssi $t \equiv_p t'$), et de définir $\leq$ sur C par $t \leq_p t' \Rightarrow c(t) \leq c(t')$. On peut voir cette opération comme l'association d'un même nom à des éléments équivalents. Notons que la fermeture réflexo-transitive d'une relation R est un ordre si et seulement si le graphe de la relation R (obtenu en associant un sommet à chaque élément de T, et en ajoutant un arc de x vers y si yRx) est sans circuit autre que ses boucles.

Soit $\leq$ un ordre sur T. On dit qu'un couple $\{t, t'\}$ d'éléments de T est *comparable* si $t \leq t'$ ou $t' \leq t$. Un ordre est dit *total* si tout couple d'éléments de T est comparable. Pour insister sur le fait qu'un ordre n'est pas *nécessairement* total, on l'appelle parfois *ordre partiel*.

Relation de couverture

Nous disons que t est couvert par t' (ou t' *couvre* t) si $t \leq t'$ et $\forall t'' \in T,\ t \leq t'' \leq t' \Rightarrow t = t''$ ou $t'' = t'$. La *relation de couverture* d'un ordre est donnée par l'ensemble des paires (t', t) telles que t couvre t'.

Élément maximal, maximum, borne supérieure

On dit que $t \in T$ est *maximal* s'il n'existe aucun autre élément $t' \in T$ tel que $t \leq t'$. On dit que t est *maximum* si, pour tout élément $t' \in T$, $t' \leq t$. Les notions d'élément *minimal* et *minimum* se définissent de façon duale. On dit que t est un *majorant* du couple $\{t', t''\}$ si $t' \leq t$ et $t'' \leq t$. Un majorant t de $\{t', t''\}$ est la *borne supérieure* de $\{t', t''\}$ si t est inférieur à tous les autres majorants de $\{t', t''\}$. Encore une fois, les notions de *minorant* et de *borne inférieure* se définissent de façon duale.

Treillis

Une relation d'ordre sur T est un *sup-demi-treillis* si tout couple d'éléments de T admet une borne supérieure. Il s'agit d'un *inf-demi-treillis* si tout couple d'éléments de T admet une borne inférieure. Un *treillis* est à la fois un sup-demi-treillis et un inf-demi-treillis.

Propriété C.1 ([Birkhoff, 1967]) *Tout treillis fini admet un minimum et un maximum. Un sup-demi-treillis qui admet un élément minimum est un treillis. De façon duale, un inf-demi-treillis qui admet un élément maximum est un treillis.*

Sous-ordre

Soient $\leq$ un ordre sur X et $\leq'$ un ordre sur X'. On dit que $\leq$ est un *sous-ordre* de $\leq'$ s'il existe une application *injective* $f : X \to X'$ telle que $x \leq' y \Leftrightarrow f(x) \leq f(y)$ (remarquons que f est une projection injective fidèle).

Annexe D

Brève présentation de la hiérarchie polynomiale

[Papadimitriou, 1994]