N° d'ordre : 3698

UNIVERSITÉ DE RENNES 1

THÈSE

Fouille de données sans information a priori sur la structure de la connaissance

Application à l'analyse de journaux d'alarmes réseau

Alexandre Vautier

Pour obtenir le grade de

Docteur de l'Université de Rennes 1 Mention Informatique

Équipe d'accueil :	Dream à L'Irisa
École Doctorale :	Matisse
Composante universitaire :	Ifsic

soutenue le 8 févier 2008 devant la commission d'examen :

Président	Olivier Ridoux	Professeur	Université Rennes 1
Rapporteurs	Dominique Duval	Professeure	Université Joseph Fourier
	Bruno Crémilleux	Professeur	Université de Caen
Examinateur	Pierre Le Maigat	Ingénieur	France-Télécom R&D Lannion
Co-encadrant	René Quiniou	Chercheur	INRIA
Directrice de thèse	Marie-Odile Cordier	Professeure	Université Rennes 1

Remerciements

Je remercie vivement Dominique Duval, professeure à l'université Joseph Fourier à Grenoble et Bruno Crémilleux, professeur à l'université de Caen, d'avoir bien voulu être rapporteurs de cette thèse. Leurs remarques et leurs suggestions ont beaucoup contribué à son amélioration. Je les remercie particulièrement de l'effort qu'ils ont fait de s'intéresser à un travail à la fois proche de leur préoccupation mais aussi intégrant de nombreuses notions qui étaient éloignées de leur domaine.

Un grand merci à Olivier Ridoux qui a bien voulu participer au jury de thèse et s'intéresser à mon travail.

Je tiens à remercier tout particulièrement Marie-Odile Cordier, ma directrice de thèse et René Quiniou, mon encadrant, qui m'ont fait confiance tout en supervisant mon travail. Ils m'ont à la fois donné la liberté de travailler sur des sujets qui me passionnaient tout en orientant mes recherches durant ces trois années.

Je souhaiterais, encore une fois, exprimer mes remerciements à Dominique Duval, pour m'avoir pris en stage lorsque j'étais en quatrième année à l'INSA de Rennes et pour m'avoir fait découvrir les catégories et plus particulièrement les esquisses. Son aide pour la formalisation de mes idées en fouille de données en utilisant les esquisses a été un point précieux de cette thèse.

Je remercie également Christophe Dousson de France-Télécom Lannion qui a été notre contact principal pour le CRE Curar avec France-Télécom ainsi que Pierre Le Maigat qui a bien voulu être examinateur de cette thèse. Tout en étant intéressés par les applications concrètes pour France-Télécom, ils se sont aussi intéressés au contenu théorique de cette thèse. Leurs remarques tout au long de ce contrat ont largement contribué à la réussite de cette thèse.

Un énorme merci à tous les membres de l'équipe Dream pour leur sympathie et l'environnement de travail incomparable dans lequel j'ai pu évoluer. Je remercie les anciens doctorants de l'équipe, Élisa, François et Alban présents lors de mon arrivée, ils ont su me guider lors de mes premiers pas de jeune doctorant. Je remercie vivement les filles du bureau d'à coté, Sophie et Véro, pour leur soutien et leur bonne humeur omniprésente. Je n'oublie pas Philippe, le joueur de l'équipe qui m'a fait découvrir un très grand nombre de jeux dont le poker, ainsi que Pascal pour les nombreuses parties qui ont suivi cette initiation. Un grand merci à Xavier et Goulven pour leur humour rassurant car aussi douteux que le mien ;-)

Je remercie Ronan, mon collègue de bureau qui a commencé et fini en même temps que moi sa thèse et qui a supporté pendant ces trois ans, mon humeur quelques

fois brusque, de longs discours sur mes nouvelles idées et quelques gommes lancées à travers le bureau ;-)

Je remercie affectueusement mes parents et ma soeur qui m'ont supporté pendant cette thèse et qui m'ont poussé à persévérer dans cette voie. Je remercie aussi mes amis qui m'ont permis de changer d'air dans les moments où la thèse devenait trop prenante. Je remercie en particulier mes colocs, François, Arnaud, Perrine et Nico qui ont vécu avec moi durant ces trois années. Enfin, le plus tendre des mercis à Lucie qui m'a accompagné pendant la rédaction et a su rendre un peu plus légère cette période très exigeante.

Table des matières

Liste des définitions

Liste des propositions et théorèmes

Liste des exemples

Liste des schémas

Liste des programmes

Liste des algorithmes

Liste des figures

Liste des tableaux

Introduction

Contexte

La fouille de données se présente comme un ensemble de techniques permettant à un utilisateur de découvrir des informations utiles et potentiellement inconnues à partir d'une masse de données. Plusieurs tentatives ont été proposées dans le but de généraliser toutes ces techniques. Le point commun de ces généralisations est que l'application de la fouille de données sur une masse de données consiste à enchaîner l'exécution d'opérateurs dans le but d'aboutir à une généralisation appelée *modèle des données* que l'utilisateur interprète. Du point de vue des bases de données qui entreposent les données, cela correspond à une succession de requêtes (Imielinski et Mannila, 1996; Johnson *et al.*, 2000). Du point du vue de logiciels visuels de fouille de données (Witten et Frank, 2005; Mierswa *et al.*, 2006; Demsar *et al.*, 2004), cela correspond à définir un processus de fouille de données d'extraction puis à l'exécuter.

Divers objectifs de processus de fouille de données sont envisageables. La plupart reposent sur la vérification des hypothèses d'un utilisateur sur des données. À partir d'une hypothèse qui structure ses connaissances, l'utilisateur est en mesure de définir un processus de fouille de données puis de vérifier que le modèle extrait valide ou infirme l'hypothèse. Cela suppose que l'utilisateur a suffisamment d'informations pour formuler cette hypothèse. Or, dans le cas où il a peu d'informations a priori sur la structure de la connaissance dans les données, il lui est impossible de définir une quelconque hypothèse. La découverte de connaissances sans hypothèse préalable empêche la définition précise d'un processus de fouille de données. Cette thèse propose une façon de répondre à ce problème.

Le contexte applicatif de cette thèse rentre dans le cadre d'un CRE (Contrat de Recherche Externalisée) avec France-Télécom R&D Lannion. Les applications portent sur l'analyse de journaux d'alarmes réseau. Ces données sont particulièrement intéressantes car elles sont présentes en grand nombre et sont difficiles à appréhender. En effet, l'utilisateur ne possède pas d'informations a priori sur les phénomènes qui ont généré ces alarmes. Or, il est probable que des informations intéressantes résident dans ces journaux. La première application porte sur des alarmes produites par un concentrateur VPN (Virtual Private Network) pour la détection d'intrusions. La seconde application concerne l'analyse des flux réseau importants sur « l'épine dorsale » d'internet pour la détection d'attaques DDoS (Distributed

Denial Of Service).

Contributions

Le principal objectif de cette thèse est de montrer l'intérêt d'employer des outils de spécification dans le but d'automatiser la découverte de connaissances dans les données.

L'approche proposée consiste à recourir aux outils de spécification issus de la théorie des catégories de manière à spécifier le processus de fouille de données. Plus précisément cela consiste à étendre la structure catégorique des *esquisses* dans le but de structurer les opérateurs de fouille de données. Une esquisse peut être vue comme un graphe dans lequel des constructions particulières sont spécifiées. Certaines de ces constructions sont interprétées comme le produit cartésien ou l'union disjointe ensembliste. L'extension des esquisses aux *esquisses relationnelles* consiste à ajouter la notion de relation et d'ensemble des parties. De la même façon que les arcs d'une esquisse s'interprètent comme des fonctions, les relations d'une esquisse relationnelle s'interprètent comme des relations *orientées*. De façon à faire cohabiter une spécification et une de ses implémentations, nous introduisons la notion de *schéma*. À partir de la définition des opérateurs de fouille de données sous la forme de schémas, il est possible de générer automatiquement des processus de fouille de données en vue de rechercher des modèles de qualité relativement aux données qu'ils résument. Notre approche catégorique de la fouille de données est à mettre en parallèle avec la modélisation des bases de données par les esquisses (Rosebrugh et Wood, 1992).

L'évaluation de modèles de qualité requiert la définition d'un *critère générique d'évaluation de modèles*. Ce critère doit être générique puisque les opérateurs de fouille de données génèrent des modèles de types différents. De plus il ne doit pas nécessiter l'intervention de l'utilisateur puisque celui-ci n'a pas la connaissance d'un critère précis d'évaluation. Le seul objectif est d'obtenir des modèles qui résument bien les données sans être trop complexes. La méthode employée, liée au domaine de la sélection de modèle en statistiques, applique le *principe MDL* (Minimum Description Length) pour évaluer les modèles. Plus un modèle décrit d'une façon précise et condensée des données, meilleur il est. L'évaluation de cette qualité est basée sur la spécification de la *relation de couverture* qui lie les modèles aux données.

L'application de cette approche sur des journaux d'alarmes a été complétée par la mise en œuvre d'un algorithme d'extraction d'attributs sur des données non structurées et d'un outil complémentaire de visualisation de graphes dynamique permettant une meilleure compréhension des flux réseau. Ces outils sont venus compléter une base d'opérateurs de fouille de données implémentés dans une plate-forme. Ce prototype, écrit en Prolog, couvre toutes les notions abordées dans cette thèse.

approches

Plan

Les dépendances entre chapitres sont résumées dans la figure page suivante. Cette thèse est divisée en deux parties. La première partie introduit la notion d'esquisse relationnelle qui est l'outil employé dans la seconde partie pour spécifier des opérateurs de fouille de données. Le premier chapitre présente divers outils de spécification et la définition des esquisses qui sont la base des esquisses relationnelles. Le second chapitre décrit formellement les esquisses relationnelles en reprenant la même présentation que le premier chapitre. De plus, ce chapitre introduit les schémas et en montre un exemple simple. Des résumés des concepts nécessaires à la compréhension de la seconde partie sont fournis dans les conclusions de ces deux chapitres.

La seconde partie décrit l'utilisation des esquisses relationnelles pour la fouille de données. Le troisième chapitre présente les bases de la fouille de données accompagnées des travaux les plus proches de notre méthode, les méthodes de sélection de modèles et l'exploration des journaux d'alarmes réseau. Le quatrième chapitre montre l'intérêt des esquisses relationnelles. Une architecture de la fouille de données sous forme d'esquisses relationnelles est introduite et une méthode d'unification d'opérateurs de fouille de données avec les données est présentée. Enfin, l'évaluation générique de modèles basée sur MDL est exposée. Ces notions sont expérimentées sur des données simulées à la fin de ce chapitre. Le cinquième chapitre décrit comment la plate-forme a été utilisée pour deux applications réelles qui concernent l'extraction de connaissances dans les journaux d'alarmes réseau. Ces journaux étant assez riches, de nouveaux algorithmes spécialisés ont été ajoutés à la plate-forme et sont présentés dans ce même chapitre. Enfin la conclusion de cette thèse fait la synthèse du travail effectué et présente plusieurs perpectives.

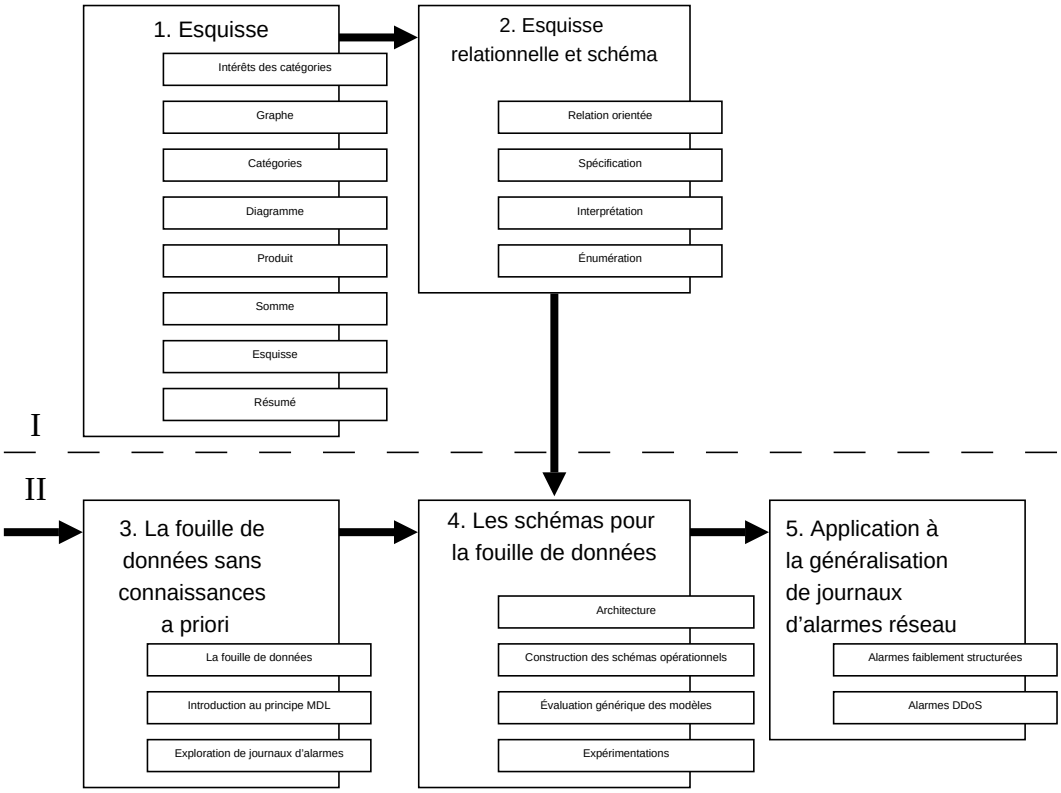


FIG. 0.1: Graphe des dépendances entre chapitres.

Première partie

Spécifications

Chapitre 1

Esquisse

La théorie des esquisses (Ehresmann, 1965) a été introduite par Ehresmann comme une sous-discipline de la théorie des catégories (Mac Lane, 1971). Ce chapitre regroupe un ensemble de définitions et d'exemples illustrant les esquisses et les catégories en général. Ils sont extraits, pour la plupart de livres dont le but est de montrer l'intérêt des catégories en informatique (Barr et Wells, 1990) ou de présenter d'une façon exhaustive les constructions catégoriques (Mac Lane, 1971; Borceux, 1994).

Dans un premier temps, les intérêts des catégories en informatique sont commentés. Ensuite, les graphes sont brièvement définis car ils sont les précurseurs des catégories. Puis les catégories sont introduites accompagnées d'exemples concrets. Ensuite plusieurs constructions sont présentées : les diagrammes, une notion proche des équations dans les catégories, les produits (ou cônes), une notion comparable au produit cartésien dans le cas des ensembles et les sommes (ou cocônes), une notion équivalente à l'union dans le cas d'ensembles. Enfin, ces définitions permettent d'aboutir à la définition d'esquisse finie et discrète.

1.1 Intérêts des catégories

Après avoir illustré deux points de vue opposés sur les catégories, nous montrons leur intérêt pour l'informatique (Goguen, 1991b). Le pouvoir de généralisation des catégories est actuellement vu comme son principal défaut par ses détracteurs mais aussi comme sa principale qualité par ses fervents utilisateurs.

D'un côté, les catégories peuvent être utilisées afin de généraliser des problèmes dans lesquels elles n'apportent rien de plus que la théorie existante. Cette généralisation implique d'utiliser des constructions catégoriques qui compliquent inutilement

le problème pour des non initiés. Ainsi, au lieu de simplifier le problème, une généralisation inutile induit en erreur et correspond à une mauvaise utilisation des catégories.

D'un autre côté, de nombreuses théories sont souvent trop proches de leur problématique. En effet, elles proposent des définitions et des théorèmes qui généralisent très peu leur approche. L'établissement de liens avec d'autres théories est d'autant plus complexe voire même complètement impossible sans tout reformuler. Dans ce cadre, le pouvoir d'expression des catégories est un atout pour formaliser ces théories afin d'éclaircir les relations entre théories.

La théorie des catégories est souvent comparée à la théorie des ensembles puisqu'elle fournit un cadre général à presque toutes les disciplines mathématiques. Elle sert de fondations aux mathématiques car elle permet d'organiser, de découvrir des analogies et de généraliser de nombreuses branches des mathématiques. La transposition de ces intérêts pour l'informatique est présentée dans les sous-sections suivantes accompagnées d'exemples concrets.

Formuler des définitions

L'un des premiers apport de la théorie des catégories est de permettre de formuler les structures à partir desquelles sont effectués des calculs. En informatique, ces structures sont habituellement définies à partir de langage de programmation ou de spécification. Par exemple, les *spécifications algébriques* (Ehrig et Mahr, 1985; Bert *et al.*, 1995) peuvent être généralisées par les catégories qui apportent une nouvelle façon de concevoir les langages de spécification (Goguen, 1991a). Les *esquisses* présentées dans la suite de cette thèse font parties des outils de spécification issues des catégories.

D'autres langages informatiques sont liées à des catégories particulières qui ont le même pouvoir d'expression. Par exemple, le λ-calcul est généralisable par une catégorie dite *cartésienne et fermée* (Barr et Wells, 1990). Les bases de données ont aussi été formalisées avec les catégories (Dampney *et al.*, 1992; Rosebrugh et Wood, 1992).

Enfin, dans d'autres domaines, comme le diagnostic, les catégories apportent une formulation élégante (Li et Pereira, 1995) du problème. Le diagnostic consiste à déterminer les causes d'un comportement anormal d'un système. Les états d'un système sont considérés comme les *objets* d'une catégorie et les changements d'état sont considérés comme des *morphismes* entre objets.

Calculer simplement

Lorsqu'un concept a été défini correctement dans un langage catégorique, établir des preuves revient à utiliser des structures catégoriques comme les *diagrammes* (une notion équivalente aux équations). Dans le contexte des liens entre les langages impératifs et les langages de spécification, il existe des méthodes (Henkel et Diwan, 2003) permettant de définir des diagrammes à partir de programmes Java convertis en

catégories. Cette méthode montre qu'il est possible de prouver automatiquement certaines propriétés simplement en utilisant la puissance des catégories.

En informatique, De nombreuses opérations sont vues comme des constructions catégoriques appelées *pushout*. Que ce soit le passage de paramètres aux fonctions, l'instanciation d'une classe et bien d'autres structures (Dumas et Duval, 2006; Williamson *et al.*, 2001), le pushout est un outil catégorique puissant permettant de transformer des opérations complexes en des calculs catégoriques simples. En particulier, le système Specware (Srinivas et Jüllig, 1995) utilise le pushout pour construire un code dérivé de spécifications formelles.

Enfin dans le cadre du diagnostic, les opérations de génération de diagnostic peuvent être définies dans les catégories. Les algorithmes existants peuvent être réutilisés (Li et Pereira, 1995) à partir de la définition de la théorie du problème de diagnostic dans une catégorie.

Lier des théories différentes

L'établissement de liens ou même l'utilisation d'un même support entre théories issues de domaines différents peuvent être importants dans différents domaines. Par exemple dans le domaine de la logique, les institutions (Goguen et Burstall, 1992) ou les spécifications diagrammatiques (Duval, 2003) fournissent un moyen de définir différentes logiques (du premier ordre, équationnelle, clauses de Horn, intuitionniste...). Cela permet de montrer comment différentes théories peuvent être rassemblées ou encore sous quelles conditions les preuves d'un théorème peuvent être mises à contribution d'une logique à une autre. Deux exemples sont donnés. Le premier concerne la définition d'exception dans un programme et la seconde est en relation avec les ontologies.

La gestion des exceptions peut être formalisée en utilisant différentes logiques (Duval et Reynaud, 2006). On définit une logique basique dans laquelle les exceptions ne sont pas autorisées puis une autre logique qui réutilise cette logique en ajoutant seulement le mécanisme d'exception. La correspondance entre ces deux logiques peut être faite en utilisant les catégories.

Une *ontologie* est à l'origine une notion philosophique qu'il faut différencier de la notion informatique (Zúniga, 2001). En philosophie, l'ontologie est une discipline qui consiste à décrire les propriétés générales de tout ce qui existe dans la nature dans le but d'une *vérité générale*. En informatique, une ontologie, ou plus précisément un système ontologique d'informations, est un ensemble structuré de concepts permettant de donner un sens aux informations dans un *domaine particulier*. Les concepts sont organisés sous la forme d'un graphe représentant des relations sémantiques ou hiérarchiques. Un des problèmes majeurs des ontologies est d'intégrer les informations provenant de multiples sources. En effet, même pour un domaine de connaissances spécifique, il existe souvent plusieurs points de vue à partir desquels émergent plusieurs ontologies. De plus, des domaines connexes définissent des ontologies complètement différentes alors que des parties de l'information qu'elles abstraient sont identiques. Les catégories sont un moyen de modéliser la distribu-

tion des connaissances (Krötzsch *et al.*, 2005). La structuration des ontologies selon les catégories (Johnson et Dampney, 2001; Jannink *et al.*, 1998) font l'objet de recherches actives dans le but de proposer un système où des ontologies structurées différemment cohabitent et d'établir des liens explicites entre les informations qu'elles contiennent.

Voir de nouvelles directions de recherche

Établir des liens entre leur propre domaine de recherche et d'autres domaines différents peut suggérer aux chercheurs de nouvelles perspectives de recherche. Par exemple, il est possible qu'une opération relative à un traitement particulier défini dans une catégorie soit applicable dans une autre catégorie. Se poser la question de l'application d'une construction catégorique sur les objets de sa propre théorie est un moyen de produire de nouvelles idées.

La notion de réécriture, omniprésente en informatique, en logique ou en mathématiques est un exemple où les catégories ont influencé la recherche. La réécriture peut être utilisée aussi bien pour déduire (on parle alors de règle d'inférence) ou pour calculer. La description et l'enchaînement des étapes élémentaires de réécriture sous forme de constructions catégoriques ont permis d'élaborer de nouvelles techniques basées sur des constructions catégoriques plus complexes mais plus efficaces (Corradini *et al.*, 1997).

Unifier

Le dernier intérêt qui conclut cette section est le pouvoir d'unification des catégories. L'informatique, un domaine récent vis-à-vis des mathématiques, est fragmentée en domaines. D'un domaine à un autre, les terminologies employées sont totalement différentes alors que ces domaines participent tous à une meilleure *gestion* de l'information. Dans toutes ces disciplines, l'utilisation des catégories permet de structurer les concepts afin de dégager une vision unifiée de l'informatique. Cette structuration permet la mise en place de liens entre domaines

Les catégories sont un outil mathématique très puissant pour organiser dans un même cadre une majorité de théories informatiques. La contrepartie de ce pouvoir d'expression est une connaissance approfondie des mécanismes catégoriques parfois lourde à supporter pour les utilisateurs. Sans utiliser directement les catégories, beaucoup de théories informatiques produisent des connaissances très intéressantes. C'est pourquoi, il n'est pas possible d'affirmer que les catégories sont absolument nécessaire à l'informatique.

Les travaux présentés dans cette thèse utilisent les catégories pour la fouille de données à des fins très concrètes. Avant de présenter ces travaux, les sections suivantes détaillent précisément quelques constructions catégoriques existantes évoquées dans cette section.

1.2 Graphe

Le concept de graphe est un précurseur du concept de catégorie : une catégorie est approximativement un graphe dans lequel les flèches peuvent être composées. Dans la suite du document, un graphe orienté est appelé un graphe. Les concepts de graphe fini et de graphe discret sont utiles à partir de la section 1.7 page 29.

Définition 1.1 (Graphe)
Un graphe est un couple (G_0, G_1) où
- G_0 est un ensemble de nœuds (ou objets).
- G_1 est un ensemble de couples ordonnés de nœuds, appelés arcs (ou flèches).

La première composante d'un arc est son nœud source (ou domaine) tandis que la seconde composante est le nœud cible (ou codomaine). ▪

Définition 1.2 (Graphe discret)
Un graphe (G_0, G_1) est discret si il n'a pas d'arc : $G_1 = \emptyset$. ▪

Définition 1.3 (Graphe fini)
Un graphe est fini si le nombre de nœuds et le nombre d'arcs sont finis. ▪

1.3 Catégories

Avant de définir les catégories, quelques définitions doivent être introduites au préalable sur les chemins dans les graphes.

Définition 1.4 (Chemin)
Dans un graphe $\mathcal{G}$, un chemin d'un nœud A à un nœud B de taille k est une suite $(f_1, f_2, \ldots, f_k)$ d'arcs pour lesquelles :
i) $source(f_k) = A$,
ii) $cible(f_i) = source(f_{i-1})$ pour $i = 2, \ldots, k$, et
iii) $cible(f_1) = B$ ▪

On observe que si on dessine un tel chemin, on obtient :

$$A \xrightarrow{f_k} \cdot \xrightarrow{f_{k-1}} \cdots \xrightarrow{f_2} \cdot \xrightarrow{f_1} B$$

f_k est à gauche et l'indice décroît de gauche à droite, ainsi la composition est cohérente (définition 1.6 page suivante).

Définition 1.5 (Chemins de longueur définie)
L'ensemble des chemins de longueur k dans un graphe $\mathcal{G}$ est noté G_k tel que $k \geq 0$.▪

En particulier, G_2 sera utilisé dans la définition des catégories. C'est l'ensemble des paires d'arcs (g, f) pour lesquels la cible de f est la source de g. Ces paires sont appelées paires composables d'arcs.

Nous avons donné deux sens à G_1 et G_0 mais cela ne pose pas de problème particulier. En effet, G_1 est à la fois l'ensemble des arcs de $\mathcal{G}$ et l'ensemble des

chemins de longueur 1, ce qui est essentiellement la même chose. De la même façon, G_0 représente à la fois les nœuds de $\mathcal{G}$ et les chemins nuls. Or on peut dire qu'il existe un chemin nul correspondant à chaque nœud. Une catégorie est intuitivement un graphe muni d'une loi de composition sur ses arcs. À partir de plusieurs arcs qui se suivent, on forme un nouvel arc. Bien qu'une catégorie soit un graphe, la terminologie diffère un peu. En particulier, dans les catégories, les mots « objet », « morphisme » (ou homomorphisme), « domaine » et « codomaine » sont plus communs que « nœud », « arc », « source » et « cible ». Dans la suite la terminologie catégorique est utilisée.

Définition 1.6 (Catégorie)
Une catégorie $\mathcal{C}$ est un graphe associé à deux fonctions $c : G_2 \to G_1$ et $u : G_0 \to G_1$ avec les propriétés C-1 à C-4 ci-dessous. La fonction c est appelée composition, et si (g, f) est une paire composable, $c(g, f)$ est écrit $g \circ f$. Si A est un objet alors on peut définir le chemin $u(A)$ de longueur 1 qui est noté id_A et est appelé l'identité de l'objet A. L'ensemble des objets est noté $|\mathcal{C}|$.

C-1 Le domaine de $g \circ f$ est le domaine de f et le codomaine de $g \circ f$ est le codomaine de g.
C-2 $(h \circ g) \circ f = h \circ (g \circ f)$.
C-3 Le domaine et le codomaine de id_A sont A.
C-4 Si f est un morphisme de A à B alors $f \circ id_A = id_B \circ f = f$. ▪

Dans les deux sections suivantes, deux exemples illustrent l'utilisation des catégories. D'une part en informatique, où on peut constater qu'un langage de programmation fonctionnelle est très proche d'une catégorie et d'autre part, en mathématiques où des structures bien connues telles que les ensembles partiellement ordonnés peuvent être représenté sous la forme de catégories. Ensuite, plusieurs catégories sur les ensembles et les graphes sont définies car nécessaires pour le reste de la lecture.

1.3.1 Catégories des langages de programmation fonctionnelle

Un langage de programmation fonctionnelle peut être approximativement vu comme un ensemble de constructeurs et d'opérations primitives typées à partir desquelles on peut construire des types et des opérations plus complexes. Si on fait trois hypothèses sur la programmation fonctionnelle, on peut voir directement qu'un langage de programmation fonctionnelle L correspond à une catégorie $C(L)$.

- H-1 On suppose que pour chaque type A, il existe une opération id_A. Quand elle est appliquée, elle ne fait rien (elle retourne le paramètre d'entrée de type A).
- H-2 On ajoute au langage un type appelé 1. On impose que pour tout type A, il y a une unique opération vers 1. De plus, nous interprétons chaque constante c de A comme une opération c de 1 dans A. Cela permet d'introduire les constantes sous la forme d'opérations, ainsi elles n'apparaissent plus comme des données séparées.

- On suppose que le langage a un constructeur de composition : soient une opération f qui prend en entrée un élément de type A et retourne un élément de type B et une autre opération du type B vers le type C. Exécuter ces opérations l'une après l'autre représente une opération dérivée (un programme) qui a pour entrée A et pour sortie C.

Sous ces conditions, un langage de programmation fonctionnelle L peut être associé à une catégorie $C(L)$ telle que

CPF-1 Les types de L sont les objets de $C(L)$.

CPF-2 Les opérations (primitives ou dérivées) de L sont les morphismes (arcs) de $C(L)$.

CPF-3 Le domaine et le codomaine d'un morphisme sont respectivement le type d'entrée et le type de sortie de l'opération correspondante.

CPF-4 La composition est donnée par le constructeur de composition.

CPF-5 Les identités sont les opérations qui rendent l'unique paramètre donné en entrée.

1.3.2 Catégories des ensembles pré-ordonnés et partiellement ordonnés

Nous définissons les relations binaires en vue de décrire un ensemble pré-ordonné.

Définition 1.7 (Relation binaire)
Soient A et B deux ensembles. Une relation binaire α est un sous-ensemble du produit cartésien[1] $A \times B$, A est l'ensemble de départ et B l'ensemble d'arrivée. Soit la relation $\alpha : A \leftrightarrow B$, on écrit $x\alpha y$ pour raccourcir l'écriture $(x, y) \in \alpha$. L'ensemble des éléments de B en relation avec un élément $a \in A$ est appelé l'image de a par α et est noté $\alpha(a)$. ▪

Voici quelques propriétés sur les relations :

- Si $A = B$ alors α est une relation interne. On note S l'ensemble départ et d'arrivée de α. De plus :
  - si pour tout $x \in S$, $x\alpha x$ alors α est réflexive et
  - si pour tout $x, y, z \in S, x\alpha y$ et $y\alpha z$ impliquent que $x\alpha z$ alors α est transitive.

Un ensemble S muni d'une relation α interne, réflexive et transitive est une structure (S, α) appelée ensemble pré-ordonné. Cette structure détermine une catégorie $C(S, \alpha)$ définie ainsi :

CO-1 Les objets de $C(S, \alpha)$ sont les éléments de S.

CO-2 Si $x, y \in S$ et $x\alpha y$ alors $C(S, \alpha)$ a exactement un morphisme de x à y noté (y, x).

CO-3 Si x n'est pas en relation avec y par α alors il n'y a pas de morphisme de x à y.

Les morphismes identité de $C(S, \alpha)$ sont de la forme (x, x), ils appartiennent à la relation α car elle est réflexive. La transitivité de α est nécessaire pour assurer

[1] Le produit cartésien est défini en page 20.

l'existence de la composition (définition 1.6 page ci-contre) telle que $(z, y) \circ (y, x) = (z, x)$.

Un ensemble pré-ordonné (S, α) pour lequel α est antisymétrique (si $x \alpha y$ et $y \alpha x$ alors $x = y$) est appelé un ensemble ordonné ou partiellement ordonné. En voici deux exemples :

- les nombres réels $(\mathbb{R}, \leq)$ muni de l'ordre habituel (ensemble ordonné)
- l'ensemble des parties de tout ensemble S, $(\mathcal{P}(S), \subseteq)$ muni de l'inclusion (ensemble partiellement ordonné).

1.3.3 Catégories utiles

Nous définissons deux catégories, la première correspondant aux ensembles et la seconde correspondant aux graphes.

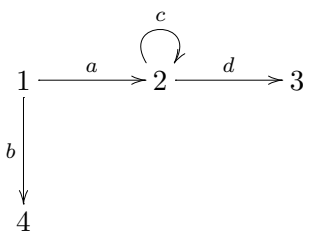
Définition 1.8 (Catégorie des ensembles)
La catégorie des ensembles a pour objet des ensembles et pour morphismes des fonctions avec la composition des fonctions pour c et la fonction identité de S à S pour id_S. Cette catégorie est noté **Ens**. .

On peut définir une catégorie dont les objets sont des graphes et les morphismes des applications entre graphes. On les appelle des morphismes de graphes.

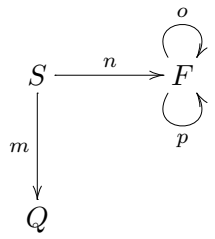
Définition 1.9 (Morphisme de graphe)
Un morphisme ϕ d'un graphe $\mathcal{G}$ à un graphe $\mathcal{H}$, noté $\phi : \mathcal{G} \to \mathcal{H}$, est un couple de fonctions $\phi_0 : \mathcal{G}_0 \to \mathcal{H}_0$ et $\phi_1 : \mathcal{G}_1 \to \mathcal{H}_1$ vérifiant la propriété suivante : si $u : m \to n$ est un morphisme de $\mathcal{G}$, alors $\phi_1(u) : \phi_0(m) \to \phi_0(n)$ dans $\mathcal{H}$. .

On vient de voir que la notation de la forme $a : B \to C$ peut être interprétée de plusieurs façons: (1) une fonction dans le cadre de la théorie des ensembles, (2) un morphisme de graphe ou (3) un morphisme dans n'importe quelle catégorie. Cela pourrait porter à confusion mais ces trois possibilités sont des instances de la troisième. En effet, il existe une catégorie dont les objets sont des ensembles et les morphismes des fonctions et une autre catégorie où les objets sont des graphes et les morphismes sont des morphismes de graphe. Il n'y a donc pas d'ambiguïté.

Exemple 1.10 (Un morphisme de graphe)
Si $\mathcal{G}$ est le graphe suivant :



et $\mathcal{H}$ ce graphe :



alors il existe un morphisme $\phi : \mathcal{G} \to \mathcal{H}$ pour lequel $\phi_0(1) = S, \phi_0(2) = \phi_0(3) = F$ et $\phi_0(4) = Q$, et ϕ_1 envoie les arcs c et d sur l'arc o. Pour les deux arcs restants, ϕ_1 suit la définition de morphisme de graphe : les arcs a et b sont respectivement envoyés sur les arcs n et m. La fonction ϕ_1 détaillée ici correspond à une des quatre possibilités de ϕ pour la fonction ϕ_0. Ces quatre possibilités sont dues aux deux boucles sur F.▪

Dans la suite du document nous simplifions l'écriture des détails d'un morphisme de graphe quand cela ne porte pas à confusion. Soit un morphisme de graphe $F : \mathcal{G} \to \mathcal{H}$. Pour tout élément A (nœud ou arc) de $\mathcal{G}$ on note FA son image dans $\mathcal{H}$.

Définition 1.11 (Catégorie des graphes)
Les objets de la catégorie des graphes sont des graphes et les morphismes de la catégorie des graphes sont des morphismes de graphes. Cette catégorie est notée **Grf**. ▪

1.3.4 Propriétés des objets et des morphismes

En général, en mathématiques, le mot « isomorphe » est employé pour dire de forme indiscernable. Nous allons le définir de façon catégorique :

Définition 1.12 (Isomorphisme)
Soient $\mathcal{C}$ une catégorie et A et B deux objets de $\mathcal{C}$. Un morphisme $f : A \to B$ est un isomorphisme s'il existe un morphisme $g : B \to A$ tel que $f \circ g$ est l'identité de B et $g \circ f$ est l'identité de A. Dans ce cas f est l'inverse de g et g est l'inverse de f. Si un tel couple de morphismes existe entre A et B alors on dit que A est isomorphe à B, ce qui est noté $A \cong B$. ▪

Afin d'illustrer cette définition, on peut faire l'analogie dans la catégorie **Ens**. On peut montrer que dans **Ens**, un morphisme est un isomorphisme si et seulement si c'est une bijection.

Preuve. D'un part, supposons que $f : S \to T$ est un isomorphisme; le morphisme inverse existe : $g : T \to S$. Alors (i) f est injective car si $f(x) = f(y)$ alors $x = g(f(x)) = g(f(y)) = y$. De plus (ii) f est surjective car pour $t \in T$ il existe $u = g(t)$ tel que $t = f(u)$. D'après (i) et (ii) f est bijective.

D'autre part, supposons que $f : S \to T$ soit bijective. On définit $g : T \to S$ en disant que $g(t)$ est l'unique élément $x \in S$ pour lequel $f(x) = t$. Cet élément x existe car f est surjective et de plus il est unique car f est injective. De par la définition de g, on a (iii) $f(g(t)) = t$ pour tout $t \in T$. De plus, la définition de g implique que

$f(g(f(x))) = f(x)$. Or comme f est injective on a (iv) $g(f(x)) = x$. D'après (iii) et (iv), f est un isomorphisme. □

Un monomorphisme est un type de morphisme dans une catégorie qui généralise le concept de fonction injective. En particulier, un monomorphisme dans la catégorie **Ens** est exactement une fonction injective. Une fonction $f : A \to B$ dans **Ens** est injective si pour tout $x, y \in A$, si $x \neq y$ alors $f(x) \neq f(y)$. Si f est un morphisme dans une catégorie, nous utilisons la même définition mis à part un changement lié au fait que les éléments d'un objet ne peuvent plus être considérés.

Définition 1.13 (Monomorphisme)
Soient une catégorie $\mathcal{C}$ et un morphisme $f : A \to B$ de $\mathcal{C}$. $f : A \to B$ est un monomorphisme si pour tout objet T de la catégorie $\mathcal{C}$ et tout morphisme $x, y : T \to A$ si $x \neq y$ alors $f \circ x \neq f \circ y$. .

Après avoir défini quelques catégories courantes, une première construction permettant d'exprimer des équations dans une catégorie est détaillée.

1.4 Diagramme

Dans un premier temps l'utilisation d'un diagramme est expliquée au moyen d'une application en fouille de données. Ensuite cette construction, équivalente à une équation, est définie.

Exemple 1.14 (Visualisation sans perte d'information)
Soient un ensemble I d'informations de n'importe quel type (nombres, phrases, tableaux...) et un ensemble V de visualisation graphique lui aussi de n'importe quel type : des graphes, des lignes de texte, des pages HTML.... On définit une fonction $voir : I \to V$ qui visualise des informations et une fonction $restituer : V \to I$ qui restitue des informations à partir d'une visualisation. Si on impose que les opérations de visualisation et de restitution n'entraînent aucune perte d'information alors on dit que l'application de la visualisation puis de la restitution $restituer \circ voir$ est équivalente aux informations de base (id_I), autrement dit $restituer \circ voir = id_I$. Ceci peut-être contraint par un diagramme. .

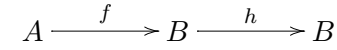
Définition 1.15 (Diagramme)
Soient $\mathcal{I}$ et $\mathcal{G}$ deux graphes. Un diagramme de forme $\mathcal{I}$ dans $\mathcal{G}$ est un morphisme de graphe $D : \mathcal{I} \to \mathcal{G}$. Le graphe $\mathcal{I}$ est appelé le graphe de forme du diagramme D. .

Exemple 1.16 (Une subtilité dans les diagrammes)
Voici un exemple illustrant quelques subtilités dans le concept de diagramme. Soit $\mathcal{G}$ un graphe avec les nœuds A, B et C (et peut-être d'autres nœuds) et les arcs

$f : A \to B$, $g : B \to C$ et $h : B \to B$. Considérons ces deux graphes :

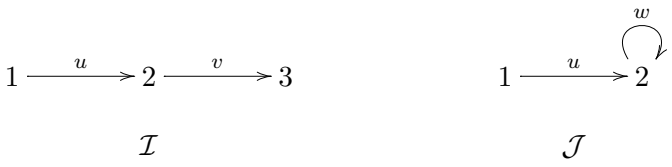
(a) (b) (1.17)

Ces deux graphes sont de formes différentes (le mot forme est utilisé de manière informelle). Mais le graphe suivant :



(1.18)

a la même forme que (1.17)(a) bien que ce soit le même graphe que (1.17)(b). De manière à saisir la différence illustrée ici entre un graphe et un diagramme, on considère ces deux graphes de forme:

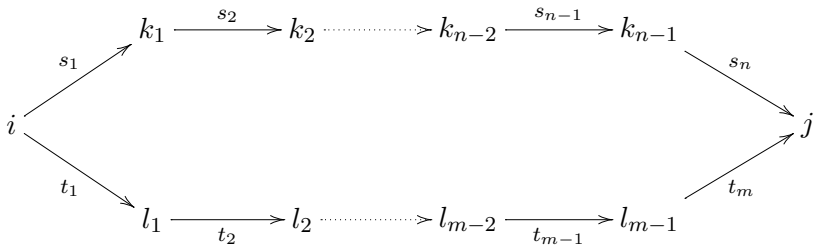


(par convention, les nombres représentent les nœuds des graphes de formes). Ainsi, le graphe (1.17)(a) est vu comme le diagramme $D : \mathcal{I} \to \mathcal{G}$ avec $D(1) = A, D(2) = B, D(3) = C, D(u) = f$ et $D(v) = g$; tandis que le graphe (1.17)(b) correspond au diagramme $E : \mathcal{J} \to \mathcal{G}$ avec $E(1) = A, E(2) = B, E(u) = f$ et $E(w) = h$. De plus, le graphe (1.18) peut être vu comme le diagramme D hormis v qui est envoyé sur h et 3 qui est envoyé sur B. ▪

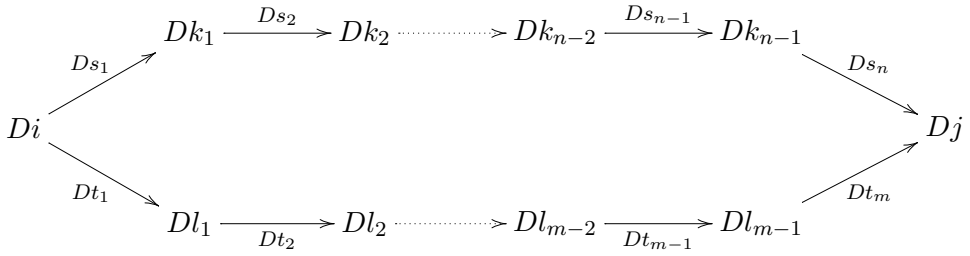
Dans la suite, un diagramme $D : \mathcal{I} \to \mathcal{G}$ est représenté comme les graphes (a) et (b) (1.17) et (1.18). C'est à dire en utilisant la forme du graphe de forme $\mathcal{I}$ mais les étiquettes des nœuds et arcs du graphe $\mathcal{G}$.

Quand le graphe de destination d'un diagramme est le graphe sous-jacent d'une catégorie, il en résulte plusieurs possibilités. En particulier, le concept de diagramme commutatif qui est le moyen d'expression des équations dans les catégories.

Définition 1.19 (Diagramme commutatif)
Soit une catégorie $\mathcal{C}$ et un diagramme $D : \mathcal{I} \to \mathcal{C}$. Le diagramme D est commutatif (ou commute) si pour tout nœud i et j de $\mathcal{I}$ et tous les chemins



de i à j dans $\mathcal{I}$, les deux chemins :



(1.20)

sont égaux dans la catégorie $\mathcal{C}$. Cela veut dire que :

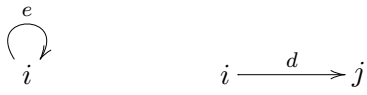
$$Ds_n \circ Ds_{n-1} \circ \ldots \circ Ds_1 = Dt_m \circ Dt_{m-1} \circ \ldots \circ Dt_1$$.

Exemple 1.21 (Commutativité sur un chemin vide)
Il y a une subtilité dans la définition précédente : que se passe-t-il si m ou n dans le diagramme (2.21) page 46 est égal à 0 ? Posons $m = 0$. On interprète l'équation suivante si les nœuds i et j sont les mêmes :

$$Ds_n \circ Ds_{n-1} \circ \ldots \circ Ds_1 = id_i$$

En particulier, un diagramme D basé sur le graphe de forme

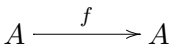
commute si et seulement si $D(e)$ est l'identité de $D(i)$. Il est important de noter que les deux graphes de forme suivants :



(a) (b) (1.22)

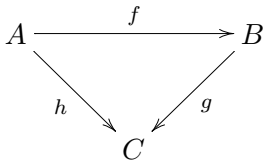
peuvent faire penser qu'on peut les représenter par le diagramme suivant (en considérant un graphe avec au moins un nœud A et une boucle f sur A) :

Cependant le diagramme basé sur (1.22)(a) commute si et seulement si $f = id_a$, tandis que le diagramme basé sur (1.22)(b) commute automatiquement. En effet, pour ce diagramme il n'existe pas deux chemins ayant même source et même destination. Ainsi on préférera représenter le diagramme de forme (1.22)(b) par le diagramme :

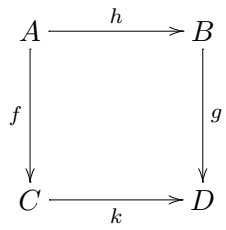


.

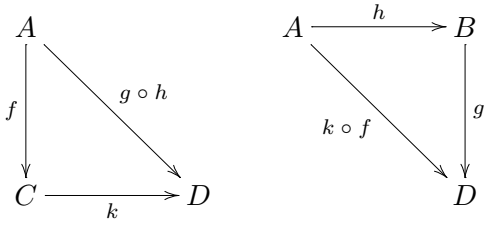
Exemple 1.23 (Le triangle : la base de la commutativité)
Le triangle suivant commute si et seulement si $h = g \circ f$.



On dit que le triangle est à la base de tout diagramme commutatif (sauf s'il implique un chemin vide) car tout diagramme commutatif peut être remplacé par un ensemble de triangles commutatifs. Un exemple illustre cette proposition. Le diagramme



commute si et seulement si l'un des deux diagrammes

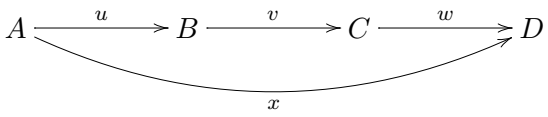


commute.

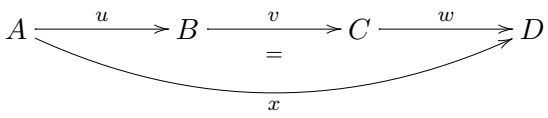
.

Il existe une autre façon de représenter un diagramme commutatif $D : \mathcal{I} \to \mathcal{C}$: soit la façon précédente où le diagramme de forme est extrait de la catégorie $\mathcal{C}$ soit en insérant le signe « = » entre les chemins qui commutent. L'exemple suivant donne un aperçu de ces deux façons de faire.

Exemple 1.24 (Représentation graphique des diagrammes)
Soit une catégorie $\mathcal{C}$ composée des nœuds A, B, C, D et des arcs $u : A \to B, v : B \to C, w : C \to D$ et $x : A \to D$. Pour indiquer que les chemins $w \circ v \circ u$ et x commutent, on dit que le diagramme suivant commute



ou de façon équivalente, on écrit :



.

1.5 Produit

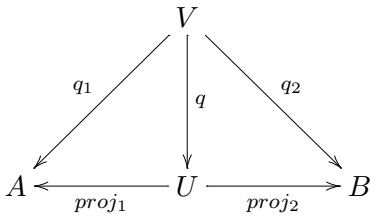
Cette section introduit les produits. En informatique, on peut voir les produits comme une construction permettant de définir des opérations avec une arité arbitraire.

1.5.1 Produit de deux objets dans une catégorie

En commençant par la définition du produit cartésien, il est aisé de comprendre le produit dans les catégories. En effet, le produit dans la catégorie des ensembles est le produit cartésien.

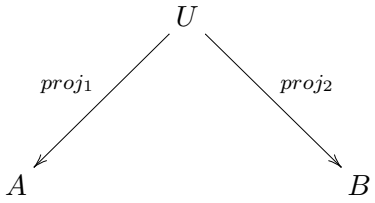
Définition 1.25 (Produit cartésien)
Si S et T sont deux ensembles, le produit cartésien $S \times T$ est l'ensemble des couples ordonnés avec une première coordonnée dans S et une seconde coordonnée dans T, autrement dit $S \times T = \{(s,t) | s \in S \; et \; t \in T\}$. Pour extraire un élément d'un produit on utilise des projections $proj_1 : S \times T \to S$ et $proj_2 : S \times T \to T$. .

A présent, voici une définition du produit de deux objets dans une catégorie. Soient A et B deux objets d'une catégorie $\mathcal{C}$. Le produit de A et B dans $\mathcal{C}$ est un objet U muni de deux morphismes $proj_1 : U \to A$ et $proj_2 : U \to B$ qui satisfait la condition suivante : pour tout objet V et les morphismes $q_1 : V \to A$ et $q_2 : V \to B$ il existe un unique morphisme $q : V \to U$ tel que ce diagramme commute

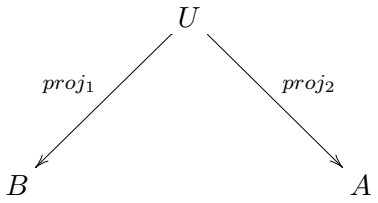


(1.26)

Cette spécification crée un objet U accompagné de deux morphismes $proj_1$ et $proj_2$ dans la catégorie $\mathcal{C}$. Le diagramme correspondant



est appelé le diagramme de produit ou cône de produit et les morphismes $proj_1$ et $proj_2$ des projections. La paire ordonné (A, B) est appelée la base du cône. Il faut noter que le diagramme de produit



est vu comme un diagramme de produit différent (identique à un isomorphisme près, voir théorème 1.31 page suivante).

Exemple 1.27 (Produit dans Ens)
Reprenons le cas du produit cartésien et considérons deux ensembles A et B dans la catégorie **Ens**. Leur produit cartésien accompagné des projections est bien le produit de A et B dans **Ens**. Supposons que l'on dispose d'un ensemble V et deux fonctions $q_1 : V \to A$ et $q_2 : V \to B$. D'après (1.26) page précédente, il existe une unique fonction $q : V \to A \times B$ définie par

$$q(v) = (q_1(v), q_2(v)) \text{ pour tout } v \in V$$

En effet, la définition de q implique que le diagramme (1.26) page ci-contre commute ($U = A \times B$) et de plus q est unique car la commutativité impose que sa valeur doit être $(q_1(v), q_2(v))$ pour tout $v \in V$. ▪

Exemple 1.28 (Produit dans un ensemble pré-ordonné)
On a déjà vu en 1.3.2 page 13 qu'à chaque ensemble pré-ordonné P correspondait une catégorie $C(P)$. Soient x et y deux objets de $C(P)$. Le produit z de ces deux éléments accompagné du couple de morphismes $z \to x$ et $z \to y$ (qui sont les projections de z) est un moyen de dire que $z \leq x$ et $z \leq y$. La définition du produit nécessite que pour tout élément $w \in P$ et deux morphismes $w \to x$ et $w \to y$, il existe un unique morphisme $w \to z$. Ce qui veut dire que

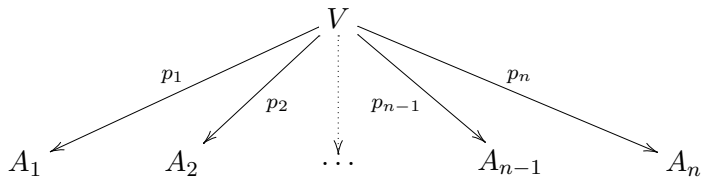
$$w \leq x \wedge w \leq y \Rightarrow w \leq z$$

associé au fait que $z \leq x$ et $z \leq y$ indique que z est l'infimum (le plus grand minorant) de x et y. Ainsi l'existence du produit dans une telle catégorie est équivalent à l'existence de l'infimum. On peut noter qu'un ensemble pré-ordonné n'admettant pas d'infimum pour tout couple d'éléments est un bon exemple de catégorie pour laquelle il n'est pas possible de définir un produit. ▪

1.5.2 Produit de n objets

Le produit de deux objets tel qu'exposé précédemment est appelé produit binaire. On définit le produit de plus de deux objets.

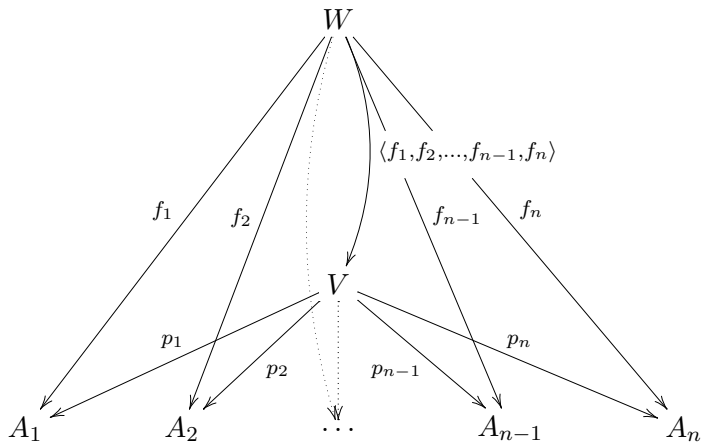
Définition 1.29 (Produit)
Le produit d'une liste d'objets $A_1, A_2, \ldots, A_n$ (qui peuvent être identiques) d'une catégorie est un objet V (noté $A_1 \times A_2 \times \ldots \times A_n$) associé à des morphismes $p_i : V \to A_i$, pour $i = 1, \ldots, n$, vérifiant la propriété suivante : étant donné un objet W et des morphismes $f_i : W \to A_i$, $i = 1, \ldots, n$, il existe un unique morphisme $\langle f_1, f_2, \ldots, f_n \rangle : W \to V$ tel que $p_i \circ \langle f_1, f_2, \ldots, f_n \rangle = f_i,\ i = 1, \ldots, n$ (équivalent au fait que le diagramme (1.30) commute). On nomme le morphisme $\langle f_1, f_2, \ldots, f_n \rangle$ une factorisation des morphismes $f_1, f_2, \ldots, f_n$. Le diagramme



est appelé diagramme de produit ou cône de produit de *sommet* V et les morphismes p_i sont appelés des projections. ▪

Dans la définition précédente, le diagramme suivant commute :

(1.30)



Théorème 1.31 (Isomorphisme de produits sur une même base)
Soient $A_1, A_2, \ldots, A_n$ des objets d'un catégorie $\mathcal{C}$, un produit de sommet C muni des projections $p_i : C \to A_i$ et un produit de sommet D muni des projections $q_i : D \to A_i$. Alors, il existe un unique morphisme $p : C \to D$ tel que $q_i \circ p = p_i$ pour tout $i = 1, \ldots, n$. De plus, p est un isomorphisme. ▪

Preuve.
D'après la définition 1.29, il existe deux uniques factorisations $p = \langle p_1, p_2, \ldots, p_n \rangle : C \to D$ et $q = \langle q_1, q_2, \ldots, q_n \rangle : D \to C$ pour lesquelles :

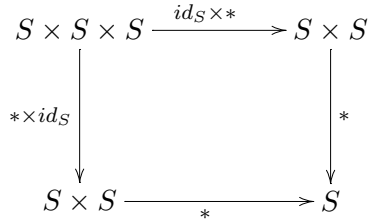
$$\begin{aligned} &p_i \circ q = q_i, \\ &q_i \circ p = p_i \end{aligned} \qquad \forall i = 1, \ldots, n$$

Il reste à prouver que p est un isomorphisme (l'inverse de q).
Or comme le morphisme $q \circ p : C \to C$ satisfait

$$p_i \circ q \circ p = q_i \circ p = p_i = p_i \circ id_C \qquad \forall i = 1, \ldots, n$$

et qu'il est unique alors $q \circ p = id_C$. De la même façon, en échangeant les « p » et les « q » on obtient que $p \circ q = id_C$ et ainsi on montre que C et D sont isomorphes. □

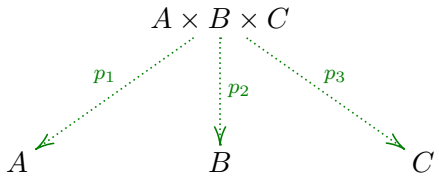
Exemple 1.32 (Associativité)
À l'aide des produits et des diagrammes commutatifs, on peut exprimer l'associativité. Soit un ensemble S et une opération binaire $*$ sur celui-ci. On veut imposer que pour tout x, y et $z \in S$ on a $(x * y) * z = x * (y * z)$. Dans la catégorie **Ens**, pour définir l'opération $*$, on définit le produit $S \times S$ et le morphisme $* : S \times S \to S$ et on dit que le diagramme suivant commute :



Ce qui suffit à exprimer l'associativité.

Dans cet exemple, le produit de morphismes $f_1 \times \ldots \times f_n$ est utilisé dans le cas de n morphismes. Dans le cas $n = 2$, il est défini à partir de deux morphismes $f : A \to A'$ et $g : B \to B'$, on définit le produit de morphismes $f \times g : A \times B \to A' \times B'$ par $\langle f \circ p_1, g \circ p_2 \rangle$ tels que p_1 et p_2 sont les projections respectives sur A et B du produit $A \times B$. Étendre cette définition à plus de deux morphismes n'est pas compliqué et n'est pas exposé ici. ▪

Deux méthodes sont utilisées pour représenter graphiquement les produits : soit la notation précédente où le sommet du produit et les projections associées sont spécifiés, soit en représentant les projections par des arcs en pointillé vert. La seconde représentation est souvent préférée dans ce document. Par exemple pour spécifier le produit $A \times B \times C$, il suffit d'écrire :



Définition 1.33 (Objet terminal)
Soit un produit de sommet V avec une base vide (il n'y a aucune projection) dans une quelconque catégorie $\mathcal{C}$. Par convention, on nomme l'objet V l'objet terminal et on le note $\mathbf{1}$. Par définition (1.29 page ci-contre), pour tout objet C de $\mathcal{C}$, il existe un unique morphisme $C \to \mathbf{1}$ que l'on nomme $\varnothing_C : C \to \mathbf{1}$. ▪

L'objet terminal a déjà été utilisé dans la sous-section 1.3.1 page 12. Le plus souvent, il est utilisé dans **Ens** pour exprimer des constantes.

1.6 Somme

Dans la théorie des catégories, la somme correspond au produit dans la catégorie duale. Voyons la définition d'une telle catégorie, cela permet de mieux comprendre les similarités entre produit et somme.

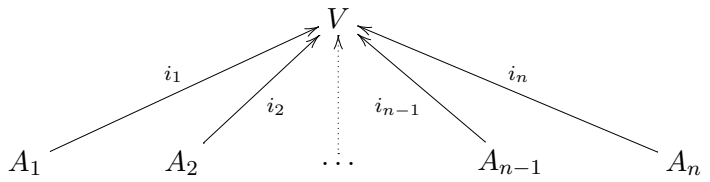
Définition 1.34 (Catégorie duale)
Soit $\mathcal{C}$ une catégorie, on peut construire une autre catégorie, notée $\mathcal{C}^{op}$, en inversant le sens des morphismes. La catégorie duale (ou opposée) $\mathcal{C}^{op}$ de $\mathcal{C}$ est définie ainsi :

D-1 Les objets et les morphismes de $\mathcal{C}^{op}$ sont les objets et les morphismes de $\mathcal{C}$.
D-2 Si $f : A \to B$ est dans $\mathcal{C}$ alors $f : B \to A$ est dans $\mathcal{C}^{op}$.
D-3 Si $h = g \circ f$ dans $\mathcal{C}$ alors $h = f \circ g$ dans $\mathcal{C}^{op}$. ▪

Il faut bien noter que cette définition est purement formelle. Par exemple, si un morphisme est une fonction dans une catégorie, ce n'est pas forcément une fonction dans la catégorie opposée. Par exemple, dans **Ens**, soit la fonction *pair* qui associe *true* à un nombre pair et $false$ à un nombre impair. Il est évident que dans $\mathbf{Ens}^{op}$, *pair* n'est plus une fonction. Néanmoins, la notion de catégorie opposée est essentiellement utilisée pour indiquer qu'une construction ou qu'une propriété a un sens *opposé* dans une catégorie opposée.

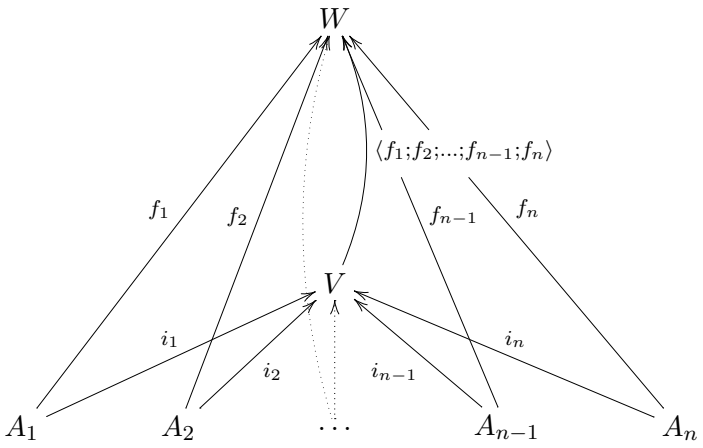
On définit la somme comme on l'a fait avec le produit, on verra ensuite que les propriétés du produit sont vraies pour une somme dans la catégorie opposée.

Définition 1.35 (Somme)
La somme d'une liste d'objets $A_1, A_2, \ldots, A_n$ (qui peuvent être identiques) d'une catégorie est un objet V (noté $A_1 + A_2 + \ldots + A_n$) associé à des morphismes (i est utilisé pour symboliser l'inclusion) $i_j : A_j \to V$, pour $j = 1, \ldots, n$, avec la propriété qu'étant donné un objet W et des morphismes $f_j : A_j \to W$, $j = 1, \ldots, n$, il existe un unique morphisme $\langle f_1; f_2; \ldots; f_n \rangle : V \to W$ tel que $\langle f_1; f_2; \ldots; f_n \rangle \circ i_j = f_j,\ j = 1, \ldots, n$ (le diagramme (1.36) page ci-contre commute). On nomme le morphisme $\langle f_1; f_2; \ldots; f_n \rangle$ une cofactorisation des morphismes $f_1, f_2, \ldots, f_n$. Le diagramme



est appelé diagramme de somme ou cocône de somme et les morphismes i_j sont appelés des inclusions. ▪

Dans la définition précédente, le diagramme suivant commute :



(1.36)

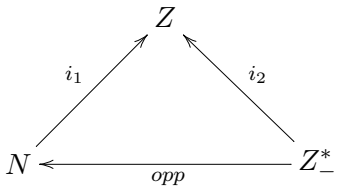
On peut noter que ce sont des points-virgules qui séparent les morphismes de la cofactorisation contrairement à la factorisation où ce sont des virgules. Le théorème 1.31 page 22 s'applique dans la catégorie opposée et ainsi prouve que deux sommes ayant le même ensemble d'objets dans leur base sont isomorphes. De plus il existe l'objet initial que l'on note 0 et qui correspond à une somme de base vide. Pour tout objet A d'une catégorie, il existe un unique morphisme de 0 à A. Dans **Ens** l'objet initial est l'ensemble vide.

Exemple 1.37 (La somme dans les langages de programmation)
Les produits permettent d'exprimer des opérations avec une arité supérieure à 1. Les sommes jouent un rôle un peu plus subtil dans les langages de programmation. Si A et B sont des types, la somme $A+B$ peut être vue comme l'union de ces types. Mais alors, que représentent les inclusions $i_1 : A \to A+B$ et $i_2 : B \to A+B$? Ce sont des conversions explicites de type. Dans les langages usuels, cette conversion est implicite car on utilise la même représentation interne d'un élément de type A quand il est mentionné de type A ou de type $A+B$. Ainsi dans ces langages, une opération ayant pour domaine $A+B$ ne peut pas connaître le type de l'élément qu'elle traite, c'est l'implémentation de l'opération qui fournit la réponse.

Une cofactorisation $\langle f_1; f_2\rangle : A+B \to W$ est vue comme un $si \ldots alors \ldots sinon$. En effet, si $x \in A$ alors $\langle f_1; f_2\rangle(x) = f_1(x)$ et sinon $x \in B$ et $\langle f_1; f_2\rangle(x) = f_2(x)$. Il faut noter que la somme $A+B$ est disjointe. ▪

Exemple 1.38 (Valeur absolue)
Soient l'ensemble N des entiers naturels et l'ensemble Z^*_- des entiers strictement négatifs. L'union de N et Z^*_- est l'ensemble Z. On définit la fonction $opp : Z^*_- \to N$ qui associe un entier négatif à son opposé. Nous pouvons décrire ces ensembles et

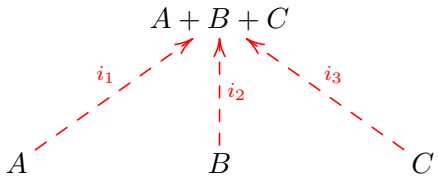
fonctions dans la catégorie **Ens**.



(1.39)

où l'objet Z est la somme des objets N et Z^*_- et les morphismes $i_1 : N \to Z$ et $i_2 : Z^*_- \to Z$ les inclusions correspondantes. Ainsi, on peut définir la fonction valeur absolue $abs : Z \to N$ par $abs = \langle id_N ; opp \rangle$. ▪

Il existe deux méthodes pour représenter graphiquement les sommes. Soit la notation précédente où le sommet de la somme et les inclusions associées sont spécifiés soit en représentant les inclusions par des arcs en tirets rouges. Par exemple, la spécification de la somme $A + B + C$ s'écrit :

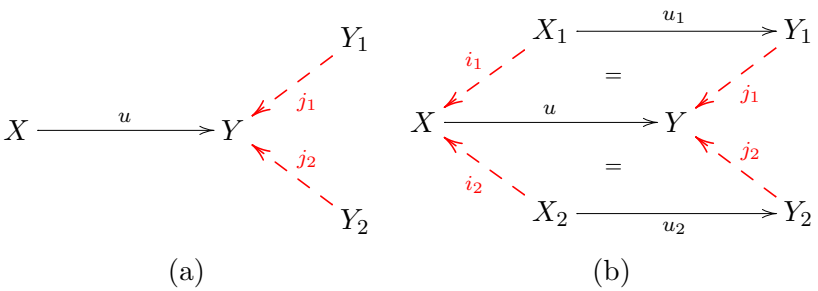


1.6.1 Propriété d'extensivité

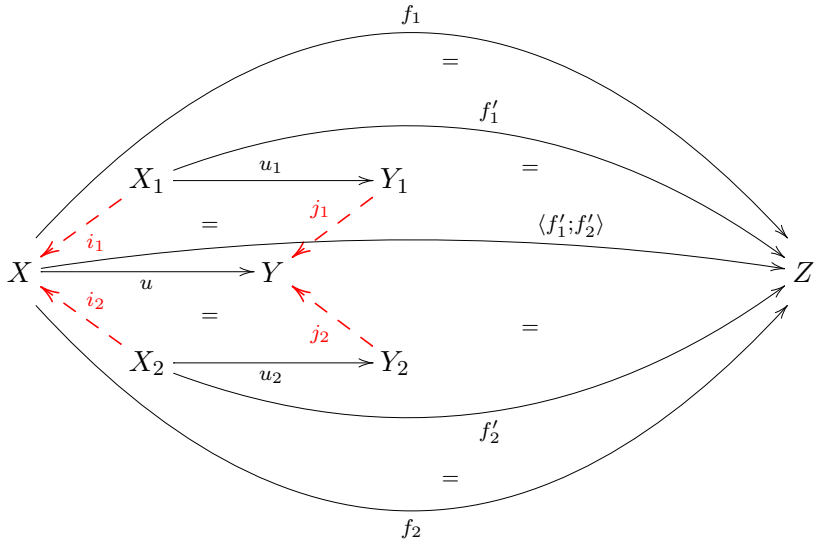
Un arc de cofactorisation représente un *si . . . alors . . . sinon* particulier. En, effet le test et les morphismes appliqués dans les deux cas ont le même type d'origine. Par exemple, il est impossible d'exprimer

$$\begin{array}{l} \text{si } u(x) \in Y_1 \text{ alors } f_1(x) \\ \text{sinon } (u(x) \in Y_2)\ f_2(x) \end{array} \tag{1.40}$$

Une transformation basée sur la propriété d'extensivité (Duval et Reynaud, 2006; Carboni *et al.*, 1993) des sommes permet de combler ce manque. Le cas à deux inclusions est illustré ici, il est facilement généralisable à n inclusions. La propriété d'extensivité établit que, à partir d'une somme (j_1, j_2) de sommet Y et d'un morphisme $u : X \to Y$, il existe une image inverse de (j_1, j_2) par u au sens des graphes suivants :



De cette façon on peut créer l'arc $\langle f'_1; f'_2 \rangle$ qui correspond à l'expression (1.40) page ci-contre. De manière à simplifier l'écriture, c.à.d sans construire les nœuds X_1 et X_2, on note la relation $\langle f'_1; f'_2 \rangle$ par $\langle f_1; f_2 \rangle_u$.

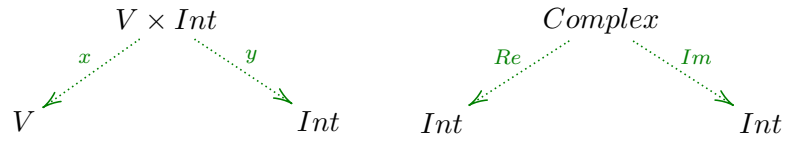


Il est possible d'utiliser l'élément sur lequel le test est effectué dans la cofactorisation. Dans l'exemple précédent $\langle f_1; f_2 \rangle_u$, cela consiste en l'utilisation dans f_1 ou f_2 de $u(x)$. Cela se fait par l'intermédiaire du mot clef *cond*. L'exemple suivant illustre son utilisation.

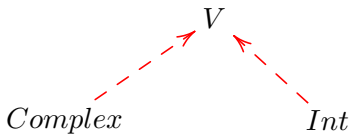
Exemple 1.41 (Nombre complexe et conditions)
Soit la fonction $f(x, y) : complex \times int \rightarrow int$ telle que si

- si x est un nombre réel, $f(x, y) = x * y$
- sinon x est un complexe, $f(x, y) = Im(x) * y$

On définit une esquisse H muni des cônes :



et du cocône :



Ainsi que du morphisme $* : Int \times Int \rightarrow Int$. On définit la fonction f par l'arc

$$f : V \times Int \rightarrow Int$$
$$f = \langle * \circ \langle Im \circ cond, y \rangle; * \circ \langle cond, y \rangle \rangle_x$$

▪

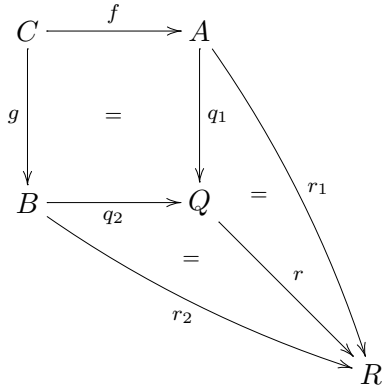
1.6.2 Pushout

Le pushout est une somme particulière. La base d'une somme est constituée seulement de nœuds alors que la base d'un pushout est constituée de deux nœuds et de morphismes. Cela implique des contraintes supplémentaires sur le sommet du pushout. Le concept de colimite généralise les concepts de somme et de pushout. Ce concept n'est pas détaillé ici car il n'est pas nécessaire.

Après avoir introduit les sommes amalgamées qui sont un exemple de pushout dans le cadre des ensembles, nous définissons formellement le pushout.

Exemple 1.42 (Somme amalgamée)
Une somme amalgamée dans le cadre des ensembles consiste à faire l'union de deux ensembles dont certains éléments sont joints par une relation. Soient les ensembles $A = \{3, 4, 5, 6\}$, $B = \{7, 8, 9, 10, 11\}$, $C = \{0, 1, 2\}$ et les fonctions $f : C \to A$ et $g : C \to B$ telles que $f = \{0 \mapsto 5, 1 \mapsto 3, 2 \mapsto 6\}$ et $g = \{0 \mapsto 8, 1 \mapsto 9, 2 \mapsto 11\}$. La somme amalgamée correspondant aux fonctions f, g est composée des fonctions $q_1 = \{3 \mapsto 1, 4 \mapsto 4, 5 \mapsto 0, 6 \mapsto 2\}$ et $q_2 = \{7 \mapsto 7, 8 \mapsto 0, 9 \mapsto 1, 10 \mapsto 10, 11 \mapsto 2\}$ et de l'ensemble $Q = \{0, 1, 2, 4, 7, 10\}$. L'ensemble Q correspond à la somme disjointe des éléments des ensembles A et B mis en commun par les morphismes f et g. ▪

Définition 1.43 (Pushout)
Soient les morphismes $f : C \to A$ et $g : C \to B$. Le carré commutatif, représenté par $ABCQ$,



est un pushout si pour tout objet R et toute paire de morphismes $r_1 : A \to R$ et $r_2 : B \to R$ telle que $r_1 \circ f = r_2 \circ g$ il existe un unique morphisme $r : Q \to R$ tel que $r \circ q_i = r_i$ et $i = 1, 2$. ▪

Les applications d'un pushout sont multiples. À partir de deux objets et de la définition des structures identifiant les similarités entre ces objets, un pushout définit un troisième objet qui fusionne les deux objets à partir de leurs similarités. Dans le cadre de la fouille de données, ce concept symbolise l'adaptation d'un algorithme aux données qu'il doit traiter. D'un côté un objet représente la spécification d'un algorithme de fouille de données et de l'autre côté un objet représente la spécification des données à analyser. La sous-section 4.2.2 page 104 explique comment définir ces points communs.

1.7 Esquisse finie et discrète

Les types abstraits sont une méthode de spécification basée sur des types, des opérations et des équations comme structure de base. UML est une méthode de spécification basée sur différents types de diagrammes. Les esquisses sont aussi une méthode de spécification basée uniquement sur des graphes. Tout d'abord, les esquisses sont définies et illustrées par quelques exemples. Ensuite, les esquisses sont comparées aux spécifications algébriques.

1.7.1 Définitions et exemples

Dans un premier temps, le cône fini et discret est défini. Il permet de réutiliser la notion de produit des catégories.

Définition 1.44 (Cône fini et discret)
Un cône fini et discret dans un graphe $\mathcal{G}$ consiste en un graphe $\mathcal{I}$ fini et discret (un graphe fini et discret est la même chose qu'un ensemble), un diagramme $L : \mathcal{I} \to \mathcal{G}$, un nœud A de $\mathcal{G}$ et une collection de morphismes $p_i : A \to L_i$ pour tout nœud $i \in \mathcal{I}$. Le nœud A est appelé le sommet du cône et le diagramme L la base du cône. Comme $\mathcal{I}$ est discret, la base du cône est une famille indexée de nœuds. En particulier, il est possible d'avoir $L_i = L_j$ pour $i \neq j$. ▪

On parle de cône discret pour décrire le fait qu'il n'y a pas de morphisme entre les nœuds de la base. Si on n'impose pas cette propriété, on parle alors de cône ou de limite, une notion beaucoup plus générale qui n'est pas développée dans cette thèse.

De la même façon qu'un cône discret et fini, on peut définir un cocône discret et fini qui est à la somme ce que le cône fini et discret est au produit. Pour cela, il suffit dans la définition précédente d'inverser les morphismes p_i de la base vers le sommet $p_i : L_i \to n$.

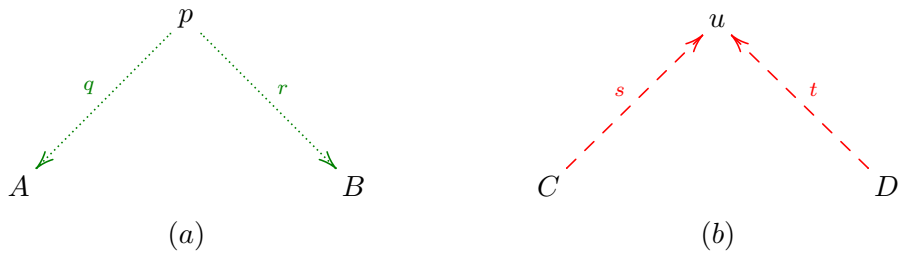
Intuitivement, une esquisse est une spécification composée de toutes les constructions vues précédemment. Pour ce faire, on utilise la notion de *modèle*. À une spécification algébrique correspond plusieurs implémentations comme à une esquisse correspond plusieurs modèles.

Définition 1.45 (Esquisse finie et discrète)
Une esquisse finie et discrète $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ consiste en un graphe $\mathcal{G}$, un ensemble fini $\mathcal{D}$ de diagrammes finis, un ensemble fini $\mathcal{L}$ de cônes finis et discrets et un ensemble fini $\mathcal{K}$ de cocônes finis et discrets. ▪

Définition 1.46 (Modèle d'esquisse finie et discrète)
Un modèle d'une esquisse $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ dans **Ens** est un morphisme $M : \mathcal{G} \to$ **Ens** qui envoie les diagrammes de $\mathcal{D}$ sur des diagrammes commutatifs, les cônes de $\mathcal{L}$ sur des cônes de produit, et les cônes de $\mathcal{K}$ sur des cocônes de somme. ▪

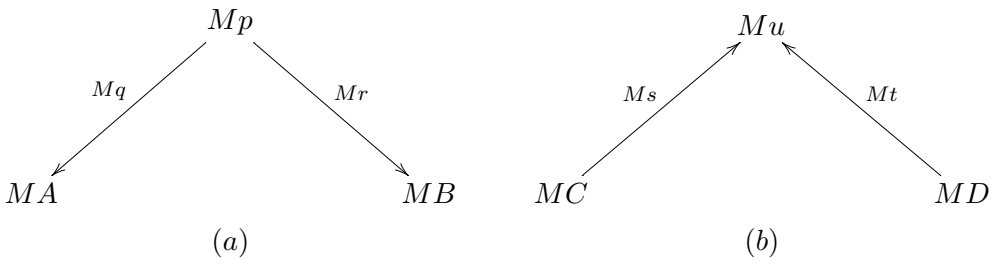
La définition exacte d'une esquisse n'impose pas que les modèles soit dans la catégorie **Ens**. Or, dans notre contexte, nous nous intéressons seulement aux modèles dans cette catégorie car elle correspond aux fonctions et ensembles que l'on peut implémenter en informatique. Dans la suite, s'il n'y pas d'ambiguïté les termes esquisse, cône et cocône sont utilisés pour parler respectivement d'une esquisse, d'un cône et d'un cocône fini et discret.

Exemple 1.47 (Esquisse simple avec un cône et un cocône)
Soit l'esquisse $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ telle que $\mathcal{D}$ soit vide, $\mathcal{L}$ soit composé du cône représenté par le diagramme (1.48)(a), $\mathcal{K}$ soit composé du cocône représenté par le diagramme (1.48)(b) et $\mathcal{G}$ soit l'ensemble des arcs et morphismes de (1.48).



(1.48)

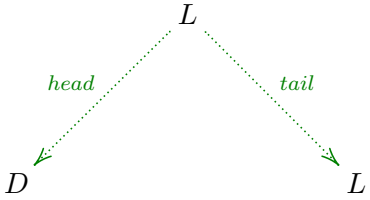
Un modèle $M : \mathcal{G} \to \mathbf{Ens}$ de l'esquisse $\mathcal{S}$ impose que (1.49)(a) soit un cône de produit dans **Ens** et (1.49)(b) soit un cône de somme dans **Ens**.



(1.49)

▪

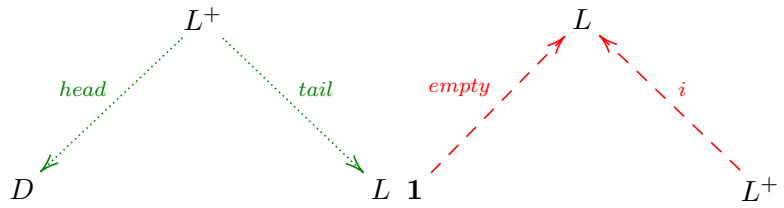
Exemple 1.50 (Esquisse des listes infinies)
Soit $\mathcal{S}$ une esquisse avec trois nœuds, $\mathbf{1}$, D et L, quatre morphismes $a, b : \mathbf{1} \to D, head : l \to D$ et $tail : L \to L$ sans diagramme et sans cocône mais avec deux cônes, le premier de sommet $\mathbf{1}$ et de base vide (correspondant à l'objet terminal) et le second représenté par ce diagramme :



Un modèle de cette esquisse est le modèle M tel que $M(L)$ est l'ensemble de toutes les séquences infinies de « a » et de « b ». $M(head)$ rend le premier caractère d'une séquence et $M(tail)$ rend la séquence obtenue en enlevant le premier caractère de la séquence d'entrée.

On ne peut pas considérer qu'un élément de $M(L)$ puisse être fini. En effet, des applications répétées de $M(tail)$ sur un tel élément devrait rendre la liste vide or celle-ci n'a pas de tête ce qui devrait l'exclure de $M(L)$. Pour contourner ce problème il faut utiliser la somme, ce qui est fait dans l'exemple suivant. ▪

Exemple 1.51 (Esquisse des listes)
Soient l'esquisse **Liste** telle que $\mathbf{1}, L, L^+, D \in |\mathbf{Liste}|$, quatre morphismes $\mathbf{1} \to L, L^+ \to L, tail : L^+ \to L$ et $head : L^+ \to D$ sans diagramme et avec un cône de sommet $\mathbf{1}$ avec une base vide, et le cône et le cocône suivant :



Pour tout ensemble S de **Ens**, il y a un modèle M de cette esquisse pour lequel $M(D) = S$, $M(\mathbf{1})$ est l'ensemble singleton contenant la liste vide, $M(L)$ est l'ensemble des listes finies composées des éléments de S et $M(L^+)$ est identique à $M(L)$ sans la liste vide. $M(head)$ et $M(tail)$ extraient respectivement la tête et la queue d'une liste non vide. ▪

Exemple 1.52 (Esquisse des entiers naturels)
Soit **Nat** l'esquisse contenant quatre nœuds $\mathbf{1}$, N , $N_{au-dessus}$ et $N + N_{au-dessus}$, deux morphismes $zero : \mathbf{1} \to N$ et $succ : N \to N + N_{au-dessus}$, aucun diagramme, un cône de sommet $\mathbf{1}$ et de base vide et un cocône construit implicitement par son nom $N + N_{au-dessus}$ et muni des inclusions $i_1 : N \to N + N_{au-dessus}$ et $i_2 : N_{au-dessus} \to N + N_{au-dessus}$. Il existe plusieurs modèles de **Nat** dans les ensembles, en voici trois d'entre eux:

- le premier modèle est celui des entiers naturels. La valeur de $zero$ est 0 et $succ(i) = i + 1$. Le modèle envoie le nœud $N_{au-dessus}$ sur l'ensemble vide.
- Le deuxième modèle envoie N sur l'ensemble des entiers naturels inférieurs ou égaux à un maximum Max. Le nœud $N_{au-dessus}$ est envoyé sur l'ensemble singleton appelé ∞. Comme précédemment la valeur de $zero$ est 0. Enfin $succ(i) = i + 1$ pour tout $i \leq Max$ et $succ(Max) = \infty$. On note que le domaine de $succ$ n'inclut pas ∞.
- Enfin le troisième modèle associe N à l'ensemble des entiers naturels inférieurs ou égaux à un maximum Max mais à la différence du second modèle il n'y a pas d'élément ∞. Le nœud $N_{au-dessus}$ est envoyé sur l'ensemble vide et $succ$ est définie par $succ(i) = i + 1$ pour tout $i \leq Max$ et $succ(Max) = 0$.

Ces trois modèles montrent qu'il faut au moins deux paramètres pour les modèles de **Nat** : la valeur de Max et la valeur du successeur de Max. .

Dans le but de réutiliser une esquisse déjà définie dans une nouvelle esquisse, on introduit le concept de morphisme d'esquisse.

Définition 1.53 (Morphisme d'esquisse)
Soient deux esquisses $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ et $\mathcal{S}' = (\mathcal{G}', \mathcal{D}', \mathcal{L}', \mathcal{K}')$. Un morphisme d'esquisse $F : \mathcal{S} \to \mathcal{S}'$ est un morphisme de graphe du graphe $\mathcal{G}$ au graphe $\mathcal{G}'$ respectant ces conditions :

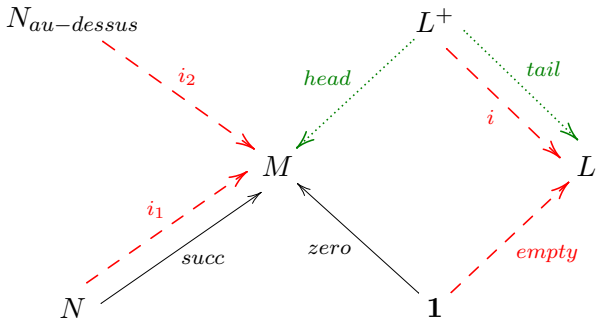
ME-1 Si D est un diagramme de $\mathcal{D}$ alors $F \circ D : \mathcal{I} \to \mathcal{G}$ est un diagramme de $\mathcal{D}'$,

ME-2 Si C est un cône de $\mathcal{L}$ de sommet n et de base L alors $F(C)$ est un cône de sommet Fn et de base FL dans $\mathcal{L}'$.

ME-3 Si O est un cocône de $\mathcal{K}$ de sommet n et de base L alors $F(O)$ est un cocône de sommet Fn et de base FL dans $\mathcal{K}'$. .

On peut noter que si dans l'esquisse $\mathcal{S}$ les ensembles de diagrammes, de cônes et de cocônes sont vides alors un morphisme d'esquisse de $\mathcal{S}$ dans $\mathcal{S}'$ est simplement un morphisme de graphe de $\mathcal{G}$ à $\mathcal{G}'$.

Exemple 1.54 (Esquisse des listes d'entiers naturels)
On souhaite définir une liste d'entiers naturels en utilisant l'esquisse **Liste** (exemple 1.51 page précédente) qui définit une liste abstraite et l'esquisse **Nat** (exemple 1.52 page précédente) qui définit les entiers naturels. Pour cela on définit une esquisse $\mathcal{S}$ et deux morphismes d'esquisse F : **Liste** $\to \mathcal{S}$ et G : **Nat** $\to \mathcal{S}$ tels que F envoie D sur M et G envoie N sur M. Pour les autres nœuds et arcs de **Liste** (resp. **Nat**), F (resp. G) les envoie sur des nœuds et arcs de même nom. Le graphe de l'esquisse $\mathcal{S}$ est le suivant :



Il est accompagné des diagrammes, des cônes et des cocônes de **Nat** et **Liste**. .

Puisque l'on dispose de morphismes d'esquisse sur les esquisses, on peut définir la catégorie des esquisses.

Définition 1.55 (Catégorie des esquisses)
La catégorie dont les objets sont des esquisses et les morphismes des morphismes d'esquisse est notée **Esq**. .

1.7.2 Spécification algébrique

Les spécifications algébriques (Ehrig et Mahr, 1985; Bert *et al.*, 1995) peuvent être représentées sous la forme d'esquisses sans cocône (avec seulement des diagrammes et des cônes). Nous n'allons pas le montrer formellement mais plutôt sous la forme d'un exemple. On peut trouver cette transformation plus en détail dans (Barr et Wells, 1990). Tout d'abord, un rappel des spécifications algébriques est donné en passant par la définition de signature.

Ensuite, un exemple de signature accompagné d'une équation illustre les spécifications algébriques. Enfin cet exemple est représenté sous la forme d'une esquisse.

Définition 1.56 (Signature)
Une signature $S = (\Sigma, \Omega)$ est une paire composée de deux ensembles finis : Σ et Ω. Les éléments de Σ sont appelés des sortes (synonyme de types dans ce contexte). Les éléments de Ω sont appelés des opérations. Une opération est composée d'une arité (un élément de l'ensemble Σ^* des mots de longueur fini dans Σ) et d'une sorte de Σ. L'arité représente le type d'entrée et la sorte représente le type de sortie.

Un modèle M d'une signature est une collection M_σ d'ensembles indexés par Σ et une collection M_f de fonctions indexées par Ω. M_σ et M_f respectent la propriété suivante : si $\sigma_1\sigma_2 \ldots \sigma_n$ est l'arité de f et τ est la sorte de f alors $M_f : M_{\sigma_1} \times M_{\sigma_2} \times \ldots \times M_{\sigma_n} \to M_\tau$. ▪

À partir d'une signature, on peut écrire des équations. Une équation est de la forme $u = v$ où u et v sont des compositions d'opérations appliquées sur des variables.

Exemple 1.57 (Équation dans les signatures)
Soit la signature $S = (\Sigma, \Omega)$ telle que $\Sigma = \{\sigma, \tau\}$ et $\Omega = \{m : \sigma\sigma \to \tau, f : \sigma\tau\tau \to \sigma\}$. On peut écrire l'équation

$$f(y, m(x, y), t) = x \tag{1.58}$$

Cette équation est satisfaite dans un modèle M de S si et seulement si

$$\forall x, y \in M_\sigma, \forall t \in M_\tau, M_f(y, M_m(x, y), t) = x$$ ▪

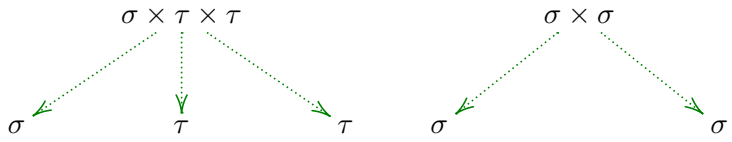
Une spécification algébrique est une signature associée à un ensemble d'équations sur ses opérations. La signature S de l'exemple 1.57 accompagnée de l'équation (1.58) constitue une spécification algébrique (S, E). Les modèles d'une spécification algébrique sont les modèles de la signature S qui satisfont les équations E.

L'esquisse $\mathcal{S}(S, E)$ dont les modèles sont identiques aux modèles d'une spécification algébrique (S, E) peut être approximativement décrite de cette manière :

- à chaque sorte de S correspond un nœud de l'esquisse $\mathcal{S}(S, E)$,
- à chaque opération $op : \sigma_1\sigma_2 \ldots \sigma_n \to \tau$ de S correspond un cône de sommet $\sigma_1 \times \sigma_2 \times \ldots \times \sigma_n$ et de base $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ et un arc $op : \sigma_1 \times \sigma_2 \times \ldots \times \sigma_n \to \tau$,
- à chaque équation de E correspond un diagramme relativement complexe pour deux raisons : dans une équation $u = v$ les variables qui apparaissent dans u et v ne sont pas forcément les mêmes et une variable peut apparaître plusieurs fois dans une équation.

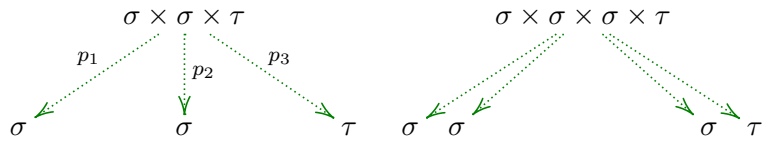
De manière à clarifier cette méthode sans passer par une formalisation du problème, l'exemple suivant illustre la transformation.

Exemple 1.59 (Esquisse d'une spécification algébrique)
Soit la spécification algébrique (S, E) telle que S est la signature de l'exemple 1.57 page précédente et E contient l'équation de ce même exemple. L'esquisse $\mathcal{S}(S, E)$ est composée des cônes



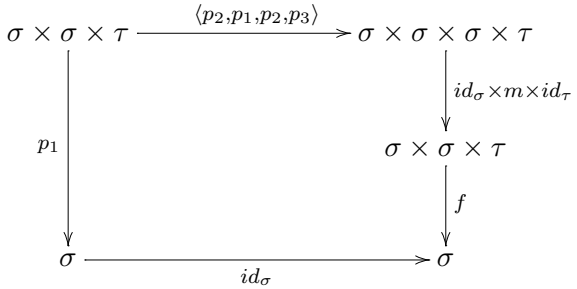
issus respectivement des opérations f et m qui sont traduites par les arcs $f : \sigma \times \tau \times \tau \to \sigma$ et $m : \sigma \times \sigma \to \tau$.

En vue de transformer l'équation $f(y, m(x, y), t) = x$, on écrit les deux cônes



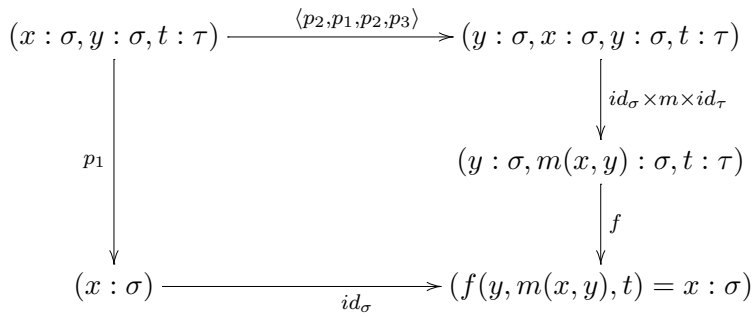
(a) (b) (1.60)

où le sommet du cône (1.60)(a) est utilisé pour représenter l'ensemble des variables x, y et t de l'équation et le sommet de (1.60)(b) est utilisé pour représenter le fait que y est utilisée deux fois dans $f(y, m(x, y), t)$. Dans le diagramme suivant, on factorise $\sigma \times \sigma \times \tau$ dans $\sigma \times \sigma \times \sigma \times \tau$ pour exprimer cette double utilisation de y. Le diagramme



traduit l'équation $f(y, m(x, y), t) = x$, le chemin $id_\sigma \circ p_1$ correspond à x et le chemin $f \circ id_\sigma \times m \times id_\tau \circ \langle p_2, p_1, p_2, p_3 \rangle$ correspond à $f(y, m(x, y), t)$. le graphe

suivant explicite un peu plus l'utilisation du diagramme de manière informelle :



Cet exemple montre que le pouvoir d'expression des esquisses est supérieur, grâce à la notion de cocône, à celui des spécifications algébriques. Des spécifications algébriques munies de l'union de type sont approximativement des esquisses. Il est possible de réutiliser toutes les applications des spécifications algébriques dans les esquisses. ▪

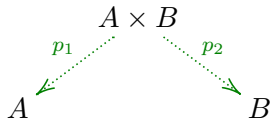
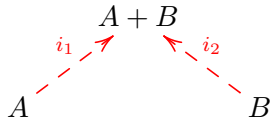
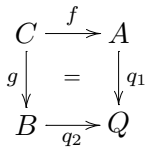
Conclusion

Les esquisses finies et discrètes ont été définies dans ce chapitre. Il faut bien noter que les concepts qui viennent d'être présentés ne représentent qu'une partie de la théorie des catégories. Néanmoins les notions abordées sont suffisantes pour la compréhension de la suite de ce document. Un résumé de ces notions est fourni afin de synthétiser toutes les constructions introduites.

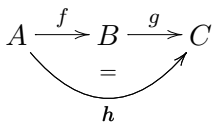
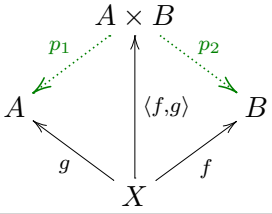
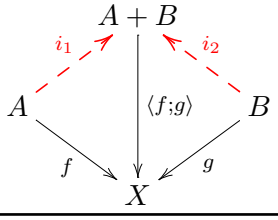
Une esquisse est une spécification sous la forme d'un graphe. On différencie cinq types de constructions dans ce graphe : les nœuds, les morphismes (ou arcs), les diagrammes (ou équations), les cônes et les cocônes. Les morphismes et les nœuds ont des étiquettes. Une esquisse étant une spécification, on l'associe à plusieurs modèles qui constituent des interprétations de l'esquisse. Un modèle est une des façons de respecter la spécification d'une esquisse en appliquant les constructions dans une structure donnée. La structure considérée ici est celle des ensembles munie des fonctions entre ensembles. Les nœuds s'interprètent comme des ensembles (des types), les morphismes s'interprètent comme des fonctions typées, les diagrammes s'interprètent comme des équations entre chemins de fonctions (un chemin est une composition de plusieurs fonctions), les cônes s'interprètent comme le produit cartésien et les cocônes s'interprètent comme l'union disjointe. Deux chemins de même source et de même cible peuvent être contraints à être identiques par un diagramme.

De la même façon qu'une esquisse, l'ensemble des esquisses peut être vu comme un graphe particulier (la catégorie des esquisses). Les nœuds sont des esquisses et les morphismes sont des morphismes d'esquisse permettant de transformer une esquisse en une autre esquisse. Concrètement, la catégorie des esquisses permet la réutilisation des constructions d'une esquisse à une autre.

Le tableau 1.61 page suivante énumère les propriétés d'un nœud et le tableau 1.62 énumère les propriétés des morphismes. Un exemple et son interprétation dans le domaine des ensembles est fourni pour chaque construction.

	Exemple	**Interprétation dans Ens**	**Référence**
Nœud terminal	$\mathbf{1}$	Le type $\mathbf{1}$ représente un singleton ou encore le type void.	Définition 1.33 page 23
Cône		Le type $A \times B$ est le produit cartésien des types A et B. Les fonctions p_1 et p_2 sont des projections : $p_1((a,b)) = a$ et $p_2((a,b)) = b$	Définitions 1.29 page 22 et 1.44 page 29
Cocône		Le type $A + B$ est l'union disjointe des types A et B. Les fonctions i_1 et i_2 sont des inclusions : $i_1(a) = a$ et $i_2(b) = b$.	Définition 1.35 page 24
Pushout		Le type Q est la somme amalgamée de A et B en tenant compte des éléments similaires définis par f et g	Définition 1.43 page 28

TAB. 1.61: Résumé des propriétés des nœuds.

	Exemple	Interprétation	Référence
Morphisme	$A \xrightarrow{f} B$	f est une fonction de A vers B	Définition 1.6 page 12
Identité	$id_A : A \to A$	Pour tout élément a de type A, $id_A(a) = a$	Définition 1.6 page 12
Composition		$h = g \circ f$ est la composition des fonction f et g (tout chemin de morphismes est composable)	Définition 1.6 page 12
Morphisme nul	$A \xrightarrow{\varnothing_A} \mathbf{1} \xrightarrow{c} B$	$c \circ \varnothing_A$ permet d'accéder à la constante c de type B	Définition 1.33 page 23
Factorisation		Pour $x \in X$, l'image $\langle f, g\rangle(x)$ est un couple de type $A \times B$ construit à partir des éléments $f(x)$ et $g(x)$	Définitions 1.29 page 22
Cofactorisation		Pour $x \in A + B$, si $x \in A$ alors $\langle f; g\rangle(x) = f(x)$ sinon $(x \in B)$ $\langle f; g\rangle(x) = g(x)$	Définition 1.35 page 24

TAB. 1.62: Résumé des propriétés des morphismes.

Chapitre 2

Esquisse relationnelle et schéma

Considérons le problème de la fouille de données de manière assez générale. Un *opérateur de fouille de données* prend en entrée un ensemble de données et retourne un modèle de données[1]. Soient *data* un type de données et *model* un type de modèles de données. On peut spécifier cet opérateur par la fonction $mine : \mathcal{P}(data) \rightarrow model$. De la même façon, la *relation de couverture*[2] qui lie un modèle de données et les données qu'il couvre s'exprime par la relation $cover : model \leftrightarrow data$

La formalisation de la fouille de données sous la forme d'esquisses nécessite l'adjonction de nouvelles constructions aux esquisses. Tout d'abord, les notions d'ensemble $\mathcal{P}(data)$ et de relations ($cover : model \leftrightarrow data$) sont nécessaires pour l'expression des problèmes de fouille de données. De plus, les opérateurs de fouille de données ($mine : \mathcal{P}(data) \rightarrow model$) qui accèdent aux données doivent pouvoir utiliser les structures de données (projections, inclusions, types...) mises en place par les esquisses. Cependant, il faut tenir compte d'un objectif des opérateurs qui est leur efficacité. En effet, pour généraliser des masses de données, des algorithmes très performants ont été développés. Il n'est pas question ici de spécifier complètement ces algorithmes mais seulement une partie représentée par les types *model*, *data* et la relation de couverture *cover*.

Nous proposons d'unifier toutes ces exigences dans le concept d' *esquisse relationnelle*. Les esquisses relationnelles sont un outil proche des esquisses permettant de spécifier des relations et l'ensemble des parties de tout ensemble. L'introduction

[1] Il faut différencier les modèles des esquisses (définition 1.46 page 29) liés à la théorie des catégories des modèles de données (détaillés dans la partie 3.1.2 page 69) qui abstraient les données en fouille de données.

[2] Ces notions de fouille de données sont détaillées dans la sous-section 3.1.2 page 69 et la section 4.1 page 94.

de ces notions impliquent des changements par rapport aux esquisses. C'est pourquoi, nous présentons les esquisses relationnelles de la même façon que les esquisses vues au chapitre précédent.

Dans la première section, nous exposons nos motivations en introduisant des relations particulières dites *orientées*. Dans la seconde section, nous définissons formellement ce qu'est une esquisse relationnelle. Dans la troisième section nous définissons les modèles d'une esquisse relationnelle à l'aide d'une *catégorie relationnelle* que nous définissons. La quatrième section introduit un outil d'énumération d'objets utile dans le cadre de la fouille de données. Enfin la cinquième section présente les schémas qui sont des esquisses relationnelles liées à des implémentations.

2.1 Relation orientée

Il existe plusieurs façons de représenter les relations dans la théorie des catégories. Celles-ci sont brièvement décrites dans l'exemple 2.1. Les relations orientées sont ensuite définies pour arriver à la catégorie où elles cohabitent avec les ensembles.

Exemple 2.1 (Les relations dans les catégories)
On peut représenter une relation $\alpha \subseteq A \times B$ de différentes façons dans une catégorie :

1. Si α est une relation interne, réflexive et transitive alors on peut représenter α par une catégorie comme exposé dans la sous-section 1.3.2 page 13.
2. Dans la catégorie **Ens**, α peut être vue comme un objet (un ensemble). En effet, A et B sont des objets de la catégorie **Ens** comme tout ensemble $\alpha \subseteq A \times B$.
3. Il existe une généralisation des relations appelée « span » qui consiste en un objet R et deux morphismes $R \to A$ et $R \to B$. Il est important de noter qu'une relation peut être exprimée sous la forme d'un span mais que tous les spans ne représentent pas des relations.
4. Enfin, l'introduction de la notion de « monade » dans **Ens** permet pour tout ensemble $S \in |\mathbf{Ens}|$ d'associer l'ensemble des parties $\mathcal{P}(S)$. Ainsi, la relation α peut être vue comme une fonction $f : A \to \mathcal{P}(B)$ ou une fonction $f' : B \to \mathcal{P}(A)$.

Les trois premières propositions considèrent une relation comme un objet d'une catégorie. Or dans notre approche, nous souhaitons que les nœuds représentent des types et les flèches des fonctions ou des relations. Ainsi ces trois propositions ne sont pas intéressantes. La quatrième proposition utilise les monades et permet d'utiliser l'équivalent d'une relation sous la forme d'une fonction en utilisant l'ensemble des parties. Cette notation est relativement lourde puisque pour toute relation spécifiée dans une esquisse, il est nécessaire d'introduire l'ensemble des parties du codomaine de la relation. Même si notre approche s'inspire largement des monades pour définir l'ensemble des parties, une autre manière de faire est utilisée pour spécifier les relations. ▪

Nous définissons une relation orientée qui n'est rien d'autre qu'une relation binaire à laquelle on ajoute une direction.

Définition 2.2 (Relation orientée)
Soient A et B deux ensembles, une relation orientée α de A vers B est une relation binaire (définition 1.7 page 13) associée à une direction de A vers B (resp. de B vers A). Une relation orientée α est notée $\alpha : A \nrightarrow B$ (resp. $\alpha : B \nrightarrow A$). Si $a \in A$ et $b \in B$ sont en relation par α, on écrit $a\,\alpha\,b$ (resp. $b\,\alpha\,a$). L'ensemble image de $a \in A$ par α est noté $\alpha(a) = \{x | a\alpha x\}$. ▪

Proposition 2.3 (Catégorie des ensembles avec relations orientées)
Le graphe dont les objets sont des ensembles et les arcs sont des relations orientées associé à la fonction de composition de relations orientées $c : G_2 \rightarrow G_1$ et à la fonction $u : G_0 \rightarrow G_1$ est une catégorie notée ***EnsRel****.*

Soient les ensembles A, B et C et les relations $\alpha : A \nrightarrow B$ et $\beta : B \nrightarrow C$:

$$c(\alpha, \beta) = \gamma \text{ tel que } \gamma : A \nrightarrow C \text{ et } x\gamma z \Leftrightarrow \exists y \in B, x\alpha y \wedge y\beta z$$
$$u(A) = id_A \text{ tel que } id_A : A \nrightarrow A \text{ et } x\, id_A\, x \Leftrightarrow x \in A$$ ▪

Preuve.

C-1 $\beta \circ \alpha : A \nrightarrow C$. Le domaine de $\beta \circ \alpha$ est celui de α et le codomaine de $\beta \circ \alpha$ est celui de β.

C-2 Soient un ensemble D et une relation $\delta : C \nrightarrow D$. On prouve que $(\delta \circ \beta) \circ \alpha = \delta \circ (\beta \circ \alpha)$:

$$\begin{aligned} &\theta = (\delta \circ \beta) \circ \alpha : A \nrightarrow D \\ &\quad w\,\theta\,z \Leftrightarrow \exists x \in B, y \in C, (y\delta z \wedge x\beta y) \wedge w\alpha x \\ &\zeta = \delta \circ (\beta \circ \alpha) : A \nrightarrow D \\ &\quad w\,\zeta\,z \Leftrightarrow \exists x \in B, y \in C, y\delta z \wedge (x\beta y \wedge w\alpha x) \end{aligned}$$

donc $\theta = \zeta$

C-3 Le domaine et le codomaine de id_A sont A.

C-4 On prouve que $\alpha \circ id_A = id_B \circ \alpha = \alpha$:
$\alpha \circ id_A = \iota$ et $x\,\iota\,z \Leftrightarrow \exists y \in A, x\,id_A\,y \wedge y\,\alpha\,z$ or $x \in A$ donc $x = y$ et ainsi $\iota = \alpha$. De la même façon $id_B \circ \alpha = \alpha$.

□

La catégorie **EnsRel** est intéressante mais elle ne supporte pas la notion de produit ou de somme. Or, ces concepts sont nécessaires pour la spécification de problématiques de fouille de données. Par exemple, l'esquisse 2.44 page 56 nécessite des cône et des cocônes. C'est pourquoi nous définissons les esquisses relationnelles dans la section suivante dans le but de faire cohabiter les relations et toutes les constructions des esquisses. De plus nous insérons un nouveau type de construction qui est l'objet des parties.

2.2 Spécification

Cette section définit exactement toutes les constructions présentes dans une esquisse relationnelle (définition 2.11 page 44). L'interprétation de ces constructions est détaillée dans la section 2.3 page 44.

2.2.1 Graphe relationnel

Les graphes sont la construction de base des esquisses. Nous définissons ici leur extension qui correspond à l'ajout d'un nouveau type d'arc correspondant aux relations.

Définition 2.4 (Graphe relationnel)
Un graphe relationnel est un triplet (G_0, G_1, G'_1) où :

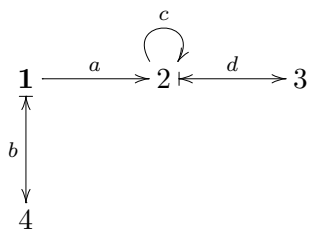
- G_0 est un ensemble de nœuds (ou objets).
- G_1 est un ensemble de couples ordonnés de nœuds, appelés *relations*.
- G'_1 est un ensemble de relations particulières incluses dans G_1, appelées *applications* : $G'_1 \subseteq G_1$.

La première composante d'une relation est son domaine tandis que la seconde composante est son codomaine. Une relation α du nœud A vers le nœud B est notée $\alpha : A \nmapsto B$. Une application f du nœud A vers le nœud B peut aussi être noté $f : A \to B$. ▪

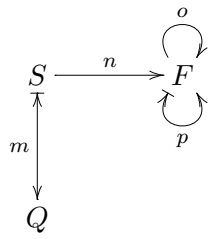
Définition 2.5 (Morphisme de graphe relationnel)
Un morphisme ϕ d'un graphe relationnel $\mathcal{G}$ à un graphe relationnel $\mathcal{H}$, noté $\phi : \mathcal{G} \to \mathcal{H}$ est un triplet de fonctions $\phi_0 : \mathcal{G}_0 \to \mathcal{H}_0$, $\phi_1 : \mathcal{G}_1 \to \mathcal{H}_1$ et $\phi'_1 : \mathcal{G}'_1 \to \mathcal{H}'_1$ vérifiant les propriétés suivantes :

- Si $\alpha : A \nmapsto B$ est une relation de $\mathcal{G}_1$, alors la relation $\phi_1(\alpha) : \phi_0(A) \nmapsto \phi_0(B)$ appartient à $\mathcal{H}_1$.
- ϕ'_1 est la restriction de ϕ_1 à $\mathcal{G}'_1$ c.à.d si $f : A \to B$ est une application de $\mathcal{G}'_1$ alors $\phi_1(f) = \phi'_1(f)$ est une application de $\mathcal{H}'_1$. ▪

Exemple 2.6 (Un morphisme de graphe relationnel)
Cet exemple est inspiré de l'exemple 1.10 page 14. Il montre qu'une relation peut être spécialisée en une application. Si $\mathcal{G}$ est le graphe suivant :



et $\mathcal{H}$ ce graphe :



alors il existe deux morphismes de graphes relationnels $\phi : \mathcal{G} \to \mathcal{H}$ pour lesquels $\phi_0(1) = S, \phi_0(2) = \phi_0(3) = F$ et $\phi_0(4) = Q$:

- $\phi_1 = \{a \mapsto n, b \mapsto m, c \mapsto o, d \mapsto p\}$ donc $\phi'_1 = \{a \mapsto n, c \mapsto o\}$
- $\phi_1 = \{a \mapsto n, b \mapsto m, c \mapsto o, d \mapsto o\}$ donc $\phi'_1 = \{a \mapsto n, c \mapsto o\}$.

2.2.2 Diagramme relationnel

Définition 2.7 (Diagramme relationnel)
Soient $\mathcal{I}$ et $\mathcal{G}$ deux graphes relationnels. Un diagramme relationnel de forme $\mathcal{I}$ dans $\mathcal{G}$ est un morphisme de graphe relationnel $D : \mathcal{I} \to \mathcal{G}$. Le graphe relationnel $\mathcal{I}$ est appelé le graphe de forme du diagramme D. .

2.2.3 Cône relationnel et cocône relationnel

Définition 2.8 (Cône relationnel fini et discret)
Un cône relationnel fini et discret dans un graphe relationnel $\mathcal{G}$ consiste en un graphe relationnel $\mathcal{I}$ fini et discret (un graphe fini et discret est essentiellement la même chose qu'un ensemble), un diagramme $L : \mathcal{I} \to \mathcal{G}$, un nœud A de $\mathcal{G}$ et une collection d'*applications* $p_i : A \to L_i$ pour tout nœud $i \in \mathcal{I}$. Le nœud A est appelé le sommet du cône, le diagramme L la base du cône et les applications p_i des projections. Comme $\mathcal{I}$ est discret, la base du cône est une famille indexée de nœuds. En particulier, il est possible d'avoir $L_i = L_j$ pour $i \neq j$. .

Définition 2.9 (Cocône relationnel fini et discret)
Un cocône relationnel fini et discret dans un graphe relationnel $\mathcal{G}$ consiste en un graphe relationnel $\mathcal{I}$ fini et discret (un graphe fini et discret est essentiellement la même chose qu'un ensemble), un diagramme $L : \mathcal{I} \to \mathcal{G}$, un nœud A de $\mathcal{G}$ et une collection d'*applications* $j_i : L_i \to A$ pour tout nœud $i \in \mathcal{I}$. Le nœud A est appelé le sommet du cocône, le diagramme L la base du cocône et les applications j_i des inclusions. Comme $\mathcal{I}$ est discret, la base du cône est une famille indexée de nœuds. En particulier, il est possible d'avoir $L_i = L_j$ pour $i \neq j$. .

Un exemple d'utilisation des cônes et des cocônes en fouille de données est donné dans le schéma 4.5 page 99.

2.2.4 Objet spécifié des parties

Définition 2.10 (Objet spécifié des parties)
L'objet spécifié des parties d'un objet A dans un graphe relationnel $\mathcal{G}$ est un nœud $\mathcal{P}(A)$ associés à l'*application unité* $\eta_A : A \to \mathcal{P}(A)$, et à la *relation d'énumération de parties* $\sigma_A : \mathcal{P}(A) \nrightarrow A$. .

Un exemple d'utilisation de l'objet spécifié des parties en fouille de données est donné dans le schéma 2.44 page 56.

2.2.5 Esquisse relationnelle

Définition 2.11 (Esquisse relationnelle finie et discrète)
Une esquisse relationnelle finie et discrète $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K}, \mathcal{P})$ consiste en un graphe relationnel $\mathcal{G}$, un ensemble fini $\mathcal{D}$ de diagrammes relationnels finis, un ensemble fini $\mathcal{L}$ de cônes relationnels finis et discrets, un ensemble fini $\mathcal{K}$ de cocônes relationnels finis et discrets et un ensemble fini $\mathcal{P}$ d'objets spécifiés des parties. .

Définition 2.12 (Morphisme d'esquisse relationnelle)
Soient deux esquisses relationnelles $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K}, \mathcal{P})$ et $\mathcal{S}' = (\mathcal{G}', \mathcal{D}', \mathcal{L}', \mathcal{K}', \mathcal{P}')$. Un morphisme d'esquisse relationnelle $F : \mathcal{S} \to \mathcal{S}'$ est un morphisme de graphe relationnel du graphe $\mathcal{G}$ au graphe $\mathcal{G}'$ respectant ces conditions :

ME-1 Si D est un diagramme relationnel de $\mathcal{D}$ alors $F \circ D : \mathcal{I} \to \mathcal{G}$ est un diagramme relationnel de $\mathcal{D}'$,

ME-2 Si C est un cône relationnel de $\mathcal{L}$ de sommet n et de base L alors $F(C)$ est un cône relationnel de sommet Fn et de base FL dans $\mathcal{L}'$.

ME-3 Si O est un cocône relationnel de $\mathcal{K}$ de sommet n et de base L alors $F(O)$ est un cocône relationnel de sommet Fn et de base FL dans $\mathcal{K}'$.

ME-4 Si $\mathcal{P}(A)$ est l'objet spécifié des parties de l'objet A dans $\mathcal{P}$ alors $F(\mathcal{P}(A))$ est l'objet spécifié des parties de l'objet FA dans $\mathcal{P}'$. .

Définition 2.13 (Catégorie des esquisses relationnelles)
La catégorie dont les objets sont des esquisses relationnelles et les morphismes des morphismes d'esquisse relationnelle est notée **EsqRel**. .

2.3 Interprétation

Toutes les définitions de cette section permettent d'aboutir à la définition de modèle d'esquisse relationnelle en 2.36 page 52. Ces définitions correspondent à l'interprétation des constructions des esquisses relationnelles vues dans la section précédente (diagramme relationnel, cône et cocône relationnels et objet spécifié des parties) dans une catégorie relationnelle.

2.3.1 Catégorie relationnelle

La notion de chemin reste valable dans un graphe relationnel. En effet, la composition de relations ou d'applications est valable. Dans le cas d'un chemin d'applications, le chemin est interprété comme une application. Dans le cas d'un chemin qui compose des relations orientées et des applications, ce chemin est interprété comme une relation orientée.

Définition 2.14 (Chemin relationnel)
Dans un graphe relationnel $\mathcal{G}$, un chemin relationnel d'un nœud A à un nœud B est une suite $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ de relations pour lesquelles :

i) $source(\alpha_k) = A$,
ii) $cible(\alpha_i) = source(\alpha_{i-1})$ pour $i = 2, \ldots, k$, et
iii) $cible(\alpha_1) = B$.

Définition 2.15 (Chemins relationnels de longueur définie)
L'ensemble des chemins relationnels de longueur k dans un graphe relationnel $\mathcal{G}$ est noté G_k tel que $k \geq 0$. .

Définition 2.16 (Chemins relationnels d'applications de longueur définie)
L'ensemble des chemins relationnels de longueur k dans un graphe relationnel $\mathcal{G}$ composés uniquement d'applications est noté G'_k tel que $k \geq 0$. .

Exemple 2.17 (Chemins de relations et d'applications)
Les relations d'un chemin peuvent être des relations ou des applications. Soit un graphe relationnel $\mathcal{G}$ définissant ces trois chemins :

$$c_1 = \quad A \overset{\alpha_4}{\mapsto\!\!\!\leftarrow} \cdot \overset{\alpha_3}{\mapsto\!\!\!\leftarrow} \cdot \overset{\alpha_2}{\mapsto\!\!\!\leftarrow} \cdot \overset{\alpha_1}{\mapsto\!\!\!\leftarrow} B$$

$$c_2 = \quad A \xrightarrow{\alpha_4} \cdot \overset{\alpha_3}{\mapsto\!\!\!\leftarrow} \cdot \xrightarrow{\alpha_2} \cdot \xrightarrow{\alpha_1} B$$

$$c_3 = \quad A \xrightarrow{\alpha_2} \cdot \xrightarrow{\alpha_1} B$$

Ces chemins appartiennent aux ensembles suivants : $c_1 \in G_3$, $c_2 \in G_4$, $c_3 \in G'_2$ ($c_3 \in G_2$). .

Définition 2.18 (Catégorie relationnelle)
Une catégorie relationnelle $\mathcal{C}$ est un graphe relationnel associé à deux fonctions $c : G_2 \to G_1$ et $u : G_0 \to G_1$ avec les propriétés CR-1 à CR-5 ci-dessous. La fonction c est appelée composition. Si (β, α) est une paire composable alors $c(\beta, \alpha)$ est écrit $\beta \circ \alpha$. Si A est un objet alors on peut définir le chemin $u(A) \in G_1$ qui est noté id_A et est appelé l'identité de l'objet A. L'ensemble des objets est noté $|\mathcal{C}|$.

CR-1 Le domaine de $\beta \circ \alpha$ est le domaine de α et le codomaine de $\beta \circ \alpha$ est le codomaine de g.

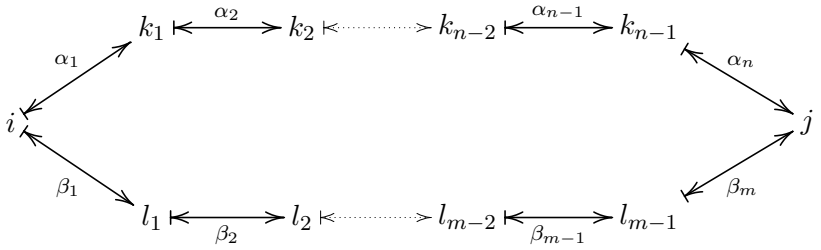
CR-2 $(\gamma \circ \beta) \circ \alpha = \gamma \circ (\beta \circ \alpha)$.
CR-3 Le domaine et le codomaine de id_A sont A.
CR-4 Si α est une relation de A à B alors $\alpha \circ id_A = id_B \circ \alpha = \alpha$.
CR-5 Si (β, α) est une paire composable, $\alpha \in G'_1$ et $\beta \in G'_1$ alors $\beta \circ \alpha \in G'_1$. ▪

Proposition 2.19 (EnsRel est une catégorie relationnelle)
La catégorie ***EnsRel*** *dont les objets sont des ensembles et les relations des relations orientées est une catégorie relationnelle dans laquelle les applications sont des relations telles que chaque élément du domaine est associé à un et un seul élément du codomaine, ce qui correspond à une fonction totale.* ▪

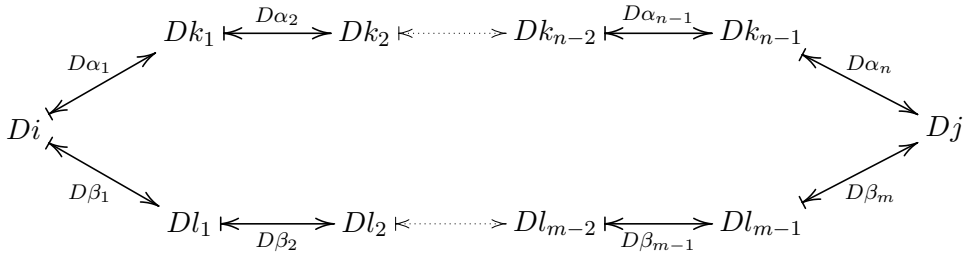
Preuve. Les propriétés CR-1 à CR-4 sont respectées puisque les propriétés C-1 à C-4 sont respectées (définition 1.6 page 12 et proposition 2.19). La composition de deux fonctions est une fonction ce qui vérifie la propriété CR-5. □

2.3.2 Diagramme commutatif relationnel

Définition 2.20 (Diagramme commutatif relationnel)
Soit une catégorie relationnelle $\mathcal{C}$ et un diagramme $D : \mathcal{I} \to \mathcal{C}$. Le diagramme D est commutatif (ou commute) si pour tout nœud i et j de $\mathcal{I}$ et pour tous les couples de chemins allant de i à j dans $\mathcal{I}$



les deux chemins correspondant dans la catégorie $\mathcal{C}$ par le diagramme D



(2.21)

sont égaux.

$$D\alpha_n \circ D\alpha_{n-1} \circ \ldots \circ D\alpha_1 = D\beta_m \circ D\beta_{m-1} \circ \ldots \circ D\beta_1$$

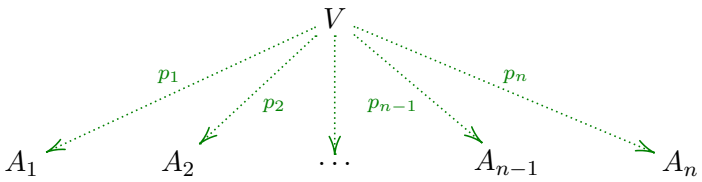
De plus si le chemin $(\alpha_n, \ldots, \alpha_1)$ appartient à I'_n alors la composition

$$D\beta_m \circ D\beta_{m-1} \circ \ldots \circ D\beta_1$$

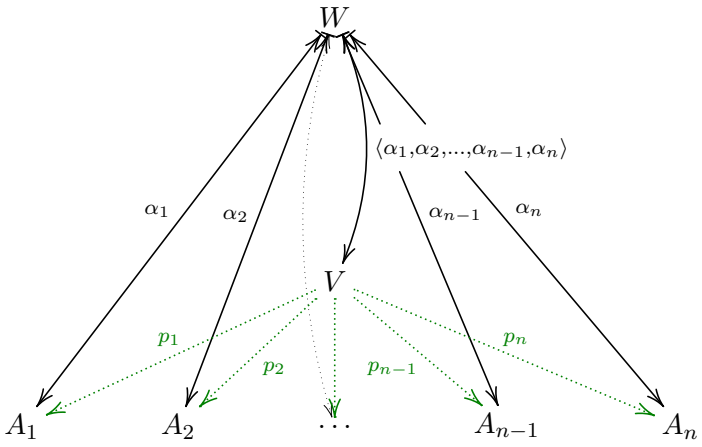
s'interprète comme une application et appartient à C'_1. ▪

2.3.3 Produit relationnel

Définition 2.22 (Produit relationnel)
Le produit d'une liste d'objets $A_1, A_2, \ldots, A_n$ (qui peuvent être identiques) d'une catégorie relationnelle est un objet V (noté $A_1 \times A_2 \times \ldots \times A_n$) associé à des applications $p_i : V \to A_i$, pour $i = 1, \ldots, n$, vérifiant la propriété suivante : étant donné un objet W et des relations $\alpha_i : W \nrightarrow A_i$, $i = 1, \ldots, n$, il existe une unique relation $\langle \alpha_1, \ldots, \alpha_n \rangle : W \nrightarrow V$ telle que $p_i \circ \langle \alpha_1, \ldots, \alpha_n \rangle = \alpha_i,\ i = 1, \ldots, n$ (équivalent au fait que le diagramme (2.23) commute). On nomme la relation $\langle \alpha_1, \ldots, \alpha_n \rangle$ une factorisation relationnelle des relations $\alpha_1, \ldots, \alpha_n$. Le diagramme



est appelé diagramme de produit relationnel ou cône relationnel de produit de *sommet* V et les applications p_i sont appelées des projections.



(2.23)

▪

Théorème 2.24 (Produit relationnel dans EnsRel)
Pour toute liste d'ensembles de la catégorie relationnelle ***EnsRel****, le produit cartésien est le produit relationnel.* ▪

PREUVE. Soient une liste d'ensembles $A_1, \ldots, A_n$, un ensemble V qui est le produit cartésien de $A_1, \ldots, A_n$, les fonctions $p_i : V \to A_i$ et pour $i = 1 \ldots n$ tels que

$$V = \{(a_1, \ldots, a_n) | a_i \in A_i \text{ pour } i = 1 \ldots n\} \text{ et}$$
$$p_i((a_1, \ldots, a_n)) = a_i$$

Soient les relations orientées $\alpha_i : W \nrightarrow A_i$ pour $i = 1 \ldots n$. On pose la relation $\gamma : W \nrightarrow V$ telle que

$$x\gamma(a_1, \ldots, a_n) \Leftrightarrow \forall i = 1 \ldots n, x\, \alpha_i\, a_i \tag{2.25}$$

On prouve l'existence et l'unicité de la relation γ telle que $p_i \circ \gamma = \alpha_i$. Pour tout $i = 1 \ldots n$, $p_i \circ \gamma$ se réduit à :

$$\begin{gathered} x\gamma(a_1, \ldots, a_n) \wedge y = p_i((a_1, \ldots, a_n)) \\ x\gamma(a_1, \ldots, a_n) \wedge y = a_i \\ x\alpha_i a_i \wedge y = a_i \\ x\alpha_i y \end{gathered}$$

Ce qui prouve que $p_i \circ \gamma = \alpha_i$ et donc l'existence de la factorisation relationnelle sous la forme de γ.

Dans le but de prouver l'unicité de γ, on suppose l'existence d'une relation $\delta : W \nrightarrow V$ qui satisfait $\alpha_i = p_i \circ \gamma$ pour tout $i = 1 \ldots n$:

$$\begin{aligned} \forall i = 1, \ldots, n, \forall x \in W, \forall z \in A_i \quad & x\alpha_i z \Leftrightarrow \exists y \in V, x\delta y \wedge y\, p_i\, z \\ & x\alpha_i a_i \Leftrightarrow y = (a_1, \ldots, a_n), x\delta y \wedge y\, p_i\, a_i \\ & x\alpha_i a_i \Leftrightarrow x\delta(a_1, \ldots, a_n) \end{aligned}$$

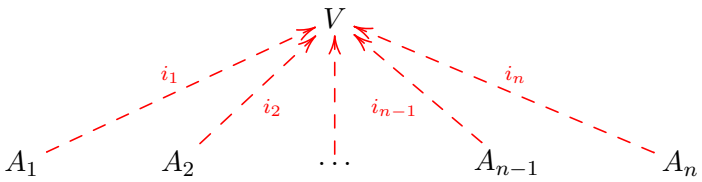
Ainsi, d'après (2.25), $\delta = \gamma$, ce qui prouve l'unicité de γ. □

Un exemple d'utilisation de la factorisation relationnelle est donné dans les schémas 4.5 page 99, 4.27 page 118 et 5.29 page 149.

2.3.4 Somme relationnelle

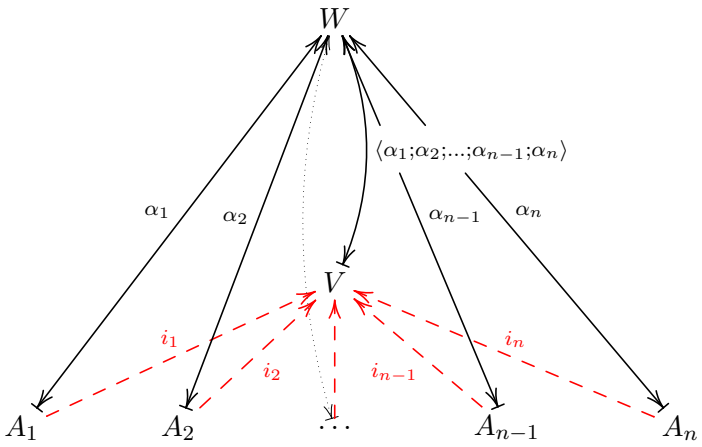
Définition 2.26 (Somme relationnelle)
La somme d'une liste d'objets $A_1, A_2, \ldots, A_n$ (qui peuvent être identiques) d'une catégorie relationnelle est un objet V (noté $A_1 + A_2 + \ldots + A_n$) associé à des applications $i_j : A_j \to V$ (i est utilisé pour symboliser l'inclusion), pour $j = 1, \ldots, n$, vérifiant la propriété suivante : étant donné un objet W et des relations $\alpha_j : A_j \nrightarrow W$, $j = 1, \ldots, n$, il existe une unique relation $\langle \alpha_1; \ldots; \alpha_n \rangle : V \nrightarrow W$ telle que $\langle \alpha_1, \ldots, \alpha_n \rangle \circ i_j = \alpha_j$, $j = 1, \ldots, n$ (équivalent au fait que le diagramme (2.27) page

suivante commute). On nomme la relation $\langle \alpha_1; \ldots; \alpha_n \rangle$ une cofactorisation relationnelle des relations $\alpha_1, \ldots, \alpha_n$. Le diagramme



est appelé diagramme de somme relationnelle ou cocône relationnel de somme et les applications i_j sont appelés des inclusions. ▪

(2.27)



Théorème 2.28 (Somme relationnelle dans EnsRel)
Pour toute liste d'ensembles de la catégorie relationnelle ***EnsRel****, l'union disjointe est la somme relationnelle.* ▪

PREUVE. Soient une liste d'ensembles $A_1, \ldots, A_n$, un ensemble V, les fonctions $i_j : A_j \to V$. Dans un premier temps, on suppose que les ensembles $A_1, \ldots, A_n$ sont disjoints entre eux et on pose

$$V = A_1 \cup \ldots \cup A_n \text{ et}$$
$$i_j(a_j) = a_j$$

Soient des relations orientées $\alpha_j : A_j \nrightarrow W$ pour $j = 1 \ldots n$. On pose la relation $\gamma : V \nrightarrow W$ telle que

$$x\gamma y \Leftrightarrow x\alpha_1 y \vee \ldots \vee x\alpha_n y$$

On prouve l'existence de la relation γ telle que $\gamma \circ i_j = \alpha_j$ pour tout $j = 1 \dots n$. Pour tout $j = 1 \dots n$, $\gamma \circ i_j$ se réduit à :

$$\begin{gathered} y\gamma z \wedge y = i_j(x) \\ x\gamma z \wedge x \in A_j \\ (x\alpha_1 z \vee \dots \vee x\alpha_n z) \wedge x \in A_j \\ x\alpha_j z \end{gathered}$$

Ce qui prouve que $\gamma \circ i_j = \alpha_j$ et donc l'existence de la cofactorisation relationnelle sous la forme de γ (l'unicité est seulement prouvée dans le cas général).

D'une manière générale, dans le cas où les ensembles $A_1, \dots, A_n$ sont non disjoints, il suffit de trouver un ensemble A'_j isomorphe de A'_j pour tout $j = 1 \dots n$ tel que les ensembles $A'_1, \dots, A'_n$ soient disjoints. Pour cela il suffit de définir A'_j tel que

$$A'_j = \{(a_j, j) | a_j \in A_j\}$$

On pose l'ensemble V et les inclusions tels que

$$\begin{aligned} V &= A'_1 \cup \dots \cup A'_n \text{ et} \\ i_j(a_j) &= (a_j, j) \end{aligned}$$

On pose la relation $\gamma : V \nrightarrow W$ telle que

$$(a, j)\gamma y \Leftrightarrow a\alpha_j y \tag{2.29}$$

Ainsi, pour tout $j = 1 \dots n$,$\gamma \circ i_j$ se réduit à :

$$\begin{gathered} y\gamma z \wedge i_j(x) = y \\ y\gamma z \wedge (x, j) = y \\ (x, j)\gamma z \\ x\alpha_j z \end{gathered}$$

Ce qui prouve que $\gamma \circ i_j = \alpha_j$ et donc l'existence de la cofactorisation relationnelle sous la forme de γ dans le cas général où les ensembles de la base du cocône sont non disjoints.

Dans le but de prouver l'unicité de γ, on suppose l'existence d'une relation δ : $V \nrightarrow W$ qui satisfait $\delta \circ i_j = \alpha_j$ pour tout $i = 1 \dots n$:

$$\begin{aligned} \forall j = 1, \dots, n, \forall x \in A_j, \forall z \in W \quad & x\alpha_j z \Leftrightarrow \exists y \in V, x\, i_j\, y \wedge y\delta z \\ & x\alpha_j z \Leftrightarrow \exists y \in V, y = (x, j) \wedge y\delta z \\ & x\alpha_j z \Leftrightarrow (x, j)\delta z \end{aligned}$$

Ainsi, d'après (2.29), on a $\delta = \gamma$, ce qui prouve l'unicité de γ. □

Un exemple d'utilisation de la cofactorisation relationnelle est donné dans le schéma 4.5 page 99.

2.3.5 Objets des parties

L'ensemble des parties habituellement utilisé en théorie des ensembles est ici défini en s'inspirant du concept catégorique des monades (Mac Lane, 1971). La multiplication des monades n'étant pas utilisée par la suite, elle n'est pas définie. Nous donnons un exemple de parties d'objet dans le cadre de la catégorie **EnsRel**.

Définition 2.30 (Ensemble des relations entre deux nœuds)
Soient A et B deux objets d'une catégorie relationnelle $\mathcal{C}$. L'ensemble des relations allant de A vers B est noté $rel(A, B)$. ▪

Définition 2.31 (Ensemble des applications entre deux nœuds)
Soient A et B deux objets d'une catégorie relationnelle $\mathcal{C}$. L'ensemble des applications allant de A vers B est noté $app(A, B)$. ▪

Définition 2.32 (Objet des parties)
Dans une catégorie relationnelle $\mathcal{C}$, pour tout objet B, il existe un objet $\mathcal{P}(B)$, une application unité $\eta_B : B \to \mathcal{P}(B)$ et une relation d'énumération de parties $\sigma_B : \mathcal{P}(B) \nrightarrow B$ avec les trois propriétés suivantes :

- Pour tout nœud $A \in |\mathcal{C}|$, il existe une bijection entre l'ensemble $rel(A, B)$ des relations allant de A vers B et l'ensemble $app(A, \mathcal{P}(B))$ des applications allant de A vers $\mathcal{P}(B)$. Cette bijection est notée $b : rel(A, B) \to app(A, \mathcal{P}(B))$.
- Pour toute application $g \in app(A, B)$ le diagramme suivant commute :

$$\begin{array}{ccc} B & \xrightarrow{\eta_B} & \mathcal{P}(B) \\ {\scriptstyle g}\uparrow & \nearrow_{b(g)} & \\ A & & \end{array} \tag{2.33}$$

- Le diagramme suivant commute (en posant $A = \mathcal{P}(B)$) :

$$\begin{array}{c} A = \mathcal{P}(B) \\ {\scriptstyle id_{\mathcal{P}(B)}}\downarrow \quad \downarrow{\scriptstyle b(\sigma_B)} \\ \mathcal{P}(B) \end{array} \tag{2.34}$$

▪

Proposition 2.35 (Objet des parties dans EnsRel)
Dans la catégorie relationnelle ***EnsRel****, l'objet des parties d'un ensemble B est l'ensemble $\mathcal{P}(B)$ des parties de B associé à la fonction $\eta_B : B \to \mathcal{P}(B), x \mapsto \{x\}$ et à la relation orientée $\sigma_B : \mathcal{P}(B) \nrightarrow B, x \mapsto \{e | e \in x\}$.* ▪

Preuve. On vérifie que les trois propriétés de la définition 2.32 sont vérifiées :

- Soit la relation $\alpha : A \nrightarrow B$. On pose la fonction $b : rel(A, B) \rightarrow app(A, \mathcal{P}(B))$

$$b(\alpha) = f \text{ tel que } \forall x \in A, f(x) = \alpha(x)$$

On prouve que b est une bijection en posant une fonction $c : app(A, \mathcal{P}(B)) \rightarrow rel(A, B)$ telle que

$$\forall f \in app(A, \mathcal{P}(B)), c(f) = \beta \text{ et } \forall x \in A, \beta(x) = f(x)$$

La fonction $c \circ b : rel(A, B) \rightarrow rel(A, B)$ est l'identité puisque

$$\forall x \in A, f(x) = \alpha(x) \text{ et } \beta(x) = f(x) \text{ et donc } \alpha = \beta$$

Ainsi, b est une bijection.

- Soit $g \in app(A, B)$ une fonction. La composition $\eta_B \circ g : A \rightarrow \mathcal{P}(B)$ est définie par

$$\forall x \in A, \eta_B \circ g(x) = \{g(x)\}$$

De plus, la fonction $b(g) : A \rightarrow \mathcal{P}(B)$ est définie par

$$\forall x \in A, b(g)(x) = \{g(x)\}$$

Ainsi $b(g) = \eta_B \circ g$, ce qui veut dire que le diagramme (2.33) page précédente commute.

- La fonction équivalente à la relation orientée $\sigma_B : \mathcal{P}(B)$ est bien l'identité puisque

$$\forall x \in \mathcal{P}(B), b(\sigma_B)(x) = \{e | e \in x\} = x$$

Ainsi le diagramme (2.34) page précédente commute.

□

2.3.6 Modèle d'esquisse relationnelle

Définition 2.36 (Modèle d'esquisse relationnelle)
Un modèle d'une esquisse relationnelle $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K}, \mathcal{P})$ dans **EnsRel** est un morphisme $M : \mathcal{G} \rightarrow$ **EnsRel** qui envoie les diagrammes relationnels de $\mathcal{D}$ sur des diagrammes commutatifs relationnels, les cônes relationnels de $\mathcal{L}$ sur des cônes relationnels de produit, les cônes relationnels de $\mathcal{K}$ sur des cocônes relationnels de somme, les objets spécifiés des parties de $\mathcal{P}$ sur des objets des parties. ▪

2.4 Énumération

L'énumération est un moyen de définir la spécification de l'énumération des éléments d'un ensemble dans la catégorie **EnsRel**. Tout d'abord, nous introduisons la disjonction de relations et la définition de nœud abstrait utiles dans la définition 2.40 page ci-contre.

Définition 2.37 (Disjonction de relations)
Soient les relations $\alpha 1 : A \nrightarrow B \ldots \alpha_n : A \nrightarrow B$. La disjonction de ces relations est une relation γ notée $\alpha_1 + \ldots + \alpha_n$ définit par :

$$x \, \gamma \, y \Leftrightarrow x \, \alpha_1 \, y \vee \ldots \vee x \, \alpha_n \, y$$

.

Définition 2.38 (Nœud opérationnel)
Soient une esquisse relationnelle S et un nœud $A \in |S|$. Le nœud A est opérationnel si :

- $A = \mathbf{1}$ ($\mathbf{1}$ correspond au nœud terminal défini en 1.33 page 23) ou
- A est le sommet d'un cône dont les objets de la base sont opérationnels ou
- A est le sommet d'un cocône dont les objets de la base sont opérationnels ou
- A est l'objet spécifié des parties ($A = \mathcal{P}(B)$) d'un objet B opérationnel.

Si et seulement si A n'est pas opérationnel alors il est abstrait. .

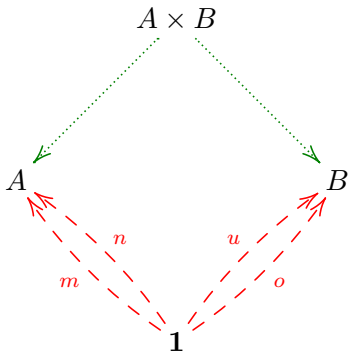
Définition 2.39 (Esquisse relationnelle opérationnelle)
Une esquisse relationnelle est opérationnelle si et seulement si tous ses nœuds sont opérationnels. Une esquisse relationnelle est abstraite si elle n'est pas opérationnelle..

Définition 2.40 (Énumération des éléments d'un nœud : Σ)
Soient une esquisse relationnelle S et un nœud $A \in |S|$. Si A est un nœud opérationnel alors il existe une relation d'énumération de A notée $\Sigma_A : \mathbf{1} \nrightarrow A$ telle que :

- Si $A = \mathbf{1}$ alors $\Sigma_{\mathbf{1}} = id_{\mathbf{1}}$.
- Si A est le sommet d'un cocône relationnel de base $(B_1, \ldots, B_n)$ par les inclusions $i_j : B_j \rightarrow A$, $j = 1 \ldots n$ alors $\Sigma_A = i_1 \circ \Sigma_{B_1} + \ldots + i_n \circ \Sigma_{B_n}$.
- Si A est le sommet d'un cône relationnel de base $(B_1, \ldots, B_n)$ alors $\Sigma_A = \langle \Sigma_{B_1}, \ldots, \Sigma_{B_n} \rangle$.
- Si A est l'objet spécifié des parties ($A = \mathcal{P}(B)$) d'un objet B opérationnel alors on peut seulement dire que Σ_A est surjective (soit un « épimorphisme » dans le langage catégorique). .

Un exemple d'utilisation de l'énumération est donné dans les schémas 2.44 page 56 et 5.3 page 131. Son utilisation principale réside dans l'adaptation de factorisations décrit dans la section 4.2 page 97.

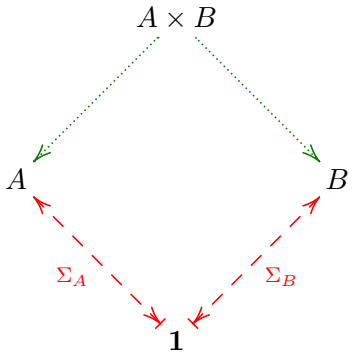
Exemple 2.41 (Énumération d'un cône et d'un cocône)
Soit l'esquisse relationnelle suivante :



telle que $A \times B$ est le sommet du cône de base (A, B) et A et B deux sommets de deux cocônes de base $\mathbf{1}$. Les quatre nœuds de cette esquisse relationnelle sont opérationnels, il existe donc quatre relations d'énumération correspondantes :

$$\begin{aligned}
\Sigma_1 &= id_{\mathbf{1}} \\
\Sigma_A &= m \circ id_{\mathbf{1}} + n \circ id_{\mathbf{1}} = \mathbf{1} \mapsto \{m, n\} \\
\Sigma_B &= o \circ id_{\mathbf{1}} + u \circ id_{\mathbf{1}} = \mathbf{1} \mapsto \{o, u\} \\
\Sigma_{A\times B} &= \langle \Sigma_A, \Sigma_B \rangle = \mathbf{1} \mapsto \{(m, o), (m, u), (n, o), (n, u)\}
\end{aligned}$$

Afin de réduire le nombre d'arcs sur le graphe, les inclusions d'un cocône relationnel sont souvent décrits par la relation d'énumération du sommet du cocône obtenu par une disjonction de relations :



$$\begin{aligned}
\Sigma_A &= \mathbf{1} \mapsto \{m, n\} \\
\Sigma_B &= \mathbf{1} \mapsto \{o, u\}
\end{aligned}$$

. ▪

2.5 Schéma

Traditionnellement, la communication entre un calculateur et un homme se fait par un langage de programmation. Actuellement, la plupart des programmes produits sont écrits en langages *opérationnels* (C, C++, Java, Perl, php, ...). À l'opposé de ces langages, on trouve les langages de *spécification*. Les langages opérationnels permettent d'exprimer *comment résoudre* tandis que les langages de spécification permettent de décrire *quel problème résoudre*.

En pratique, un langage de programmation n'est jamais seulement opérationnel ou seulement de spécification, il est bien souvent les deux à la fois. Par exemple, Prolog (Colmerauer et Roussel, 1996) est un langage basé sur la logique du premier ordre. Cependant « par les expériences des premiers utilisateurs, mais aussi par des considérations d'efficacité », en pratique, on peut voir Prolog comme un langage opérationnel (l'ordonnancement des clauses, les coupures, le back-tracking sont des outils opérationnels). Ainsi, on considère Prolog comme un langage autant opérationnel que de spécification.

Les esquisses relationnelles sont un langage de spécification. Dans une esquisse relationnelle, on ne se préoccupe pas de la manière dont sont implémentées toutes les constructions spécifiées. Afin de fixer les modèles d'une esquisse relationnelle on propose de l'associer à une implémentation dans ce que l'on appelle un *schéma*[3]. La définition donnée n'est pas donnée sous une forme catégorique puisque ce n'est pas l'un des points essentiels de notre approche.

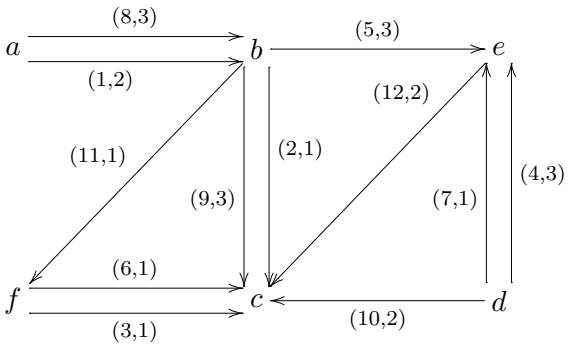
Définition 2.42 (Schéma)
Un schéma est une esquisse relationnelle pour laquelle on a « fixé » ses modèles par une implémentation. Ses modèles sont dans la catégorie relationnelle **EnsRel** et l'implémentation respecte toutes les constructions : diagramme commutatif relationnelle, produit relationnel, somme relationnelle et parties d'un objet. ▪

Le langage utilisé pour implémenter les esquisses relationnelles est Prolog. Celui-ci est particulièrement bien adapté car il permet de définir des relations. Dans la suite, puisqu'un schéma regroupe les éléments d'une esquisse relationnelle avec les éléments de son interprétation dans la catégorie **EnsRel**, les termes « application » ou « fonction » d'un schéma sont utilisés indifféremments ainsi que les termes « relation » ou « relation orientée » et les termes « objet », « nœud » ou « type ». Les morphismes entre les esquisses relationnelles qui composent des schémas sont représentés par la flèche $\Longrightarrow$. L'exemple suivant décrit le schéma des alarmes réseau qui est très utile pour expliquer les exemples du chapitre 4.

Exemple 2.43 (Alarmes réseau)
Une alarme correspond à un flot suspect d'une source à une cible (les *acteurs*) sur un réseau. Une alarme est formellement un triplet (d, e, s) où d représente la date, e représente le lien (source,cible) et s représente la sévérité (1 - faible à 3 - importante). Le graphe (2.43) page suivante représente un exemple de douze alarmes. Un nœud

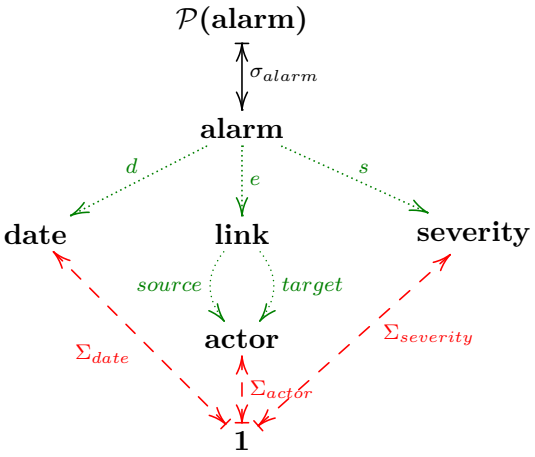
[3]Il faut différencier les schémas des schémas de Grothendieck qui sont complètement différents.

du graphe représente un acteur et un arc représente une alarme (d, s) avec une date d et une sévérité s.



Le type des alarmes est spécifié dans un schéma composé de l'esquisse relationnelle représenté en 2.44. Ce schéma est aussi représenté dans le programme 2.45. Ce programme est écrit dans un langage que nous avons mis en place basé sur Prolog..

Schéma 2.44 (Esquisse relationnelle d'alarmes réseau)
L'esquisse relationnelle des alarmes réseau est simple. Elle est composée de deux cônes et de trois cocônes.



$$\Sigma_{date} = \mathbf{1} \mapsto \{1, 2, \ldots, 2^{32}\}$$
$$\Sigma_{actor} = \mathbf{1} \mapsto \{a, b, \ldots, f\}$$
$$\Sigma_{severity} = \mathbf{1} \mapsto \{1, 2, 3\}$$

.

Programme 2.45 (Schéma d'alarmes réseau)
La correspondance entre le programme et l'esquisse relationnelle est indiquée dans les commentaires du programme. Le schéma *abstractInt* est défini en annexe par le programme B.3 page 167. Tous les mot-clés en gras génèrent automatiquement l'implémentation cachée derrière les spécifications de l'esquisse relationnelle. Les implémentations non liées aux spécifications sont directement visibles.

```
:- begin_schema(alarm_data).

    % Noeud date et cocone associe
    :- set_def_sort(date).
    :- set_def_sort_enum(date,enum_date).
       enum_date(D) :-
         Z is 2**32,
              between(1,Z,D).

    % Morphisme de schema specifiant le type date
    :- begin_functor(abstractInt_date,abstractInt).
         % Envoie le noeud int du schema abstractInt sur le noeud date
         :- functor_sort(int,date).
    :- end_functor(abstractInt_date).

    % Noeud actor et cocone associe
    :- set_def_sort(actor).
    :- set_def_sort_enum(actor,enum_actor).
       enum_actor(a).
       enum_actor(b).
       enum_actor(c).
       enum_actor(d).
       enum_actor(e).
       enum_actor(f).

    % Noeud severity et cocone associe
    :- set_def_sort(severity).
    :- set_def_sort_enum(severity,enum_severity).
       enum_severity(S) :-
            between(1,3,S).

    % Morphisme de schema specifiant le type severity
    :- begin_functor(abstractInt_severity,abstractInt).
         % Envoie le noeud int du schema abstractInt sur le noeud severity
         :- functor_sort(int,severity).
    :- end_functor(abstractInt_severity).

    % Cone link et ses projections
  :- product(link, [(source,actor),(target,actor)]).

    % Cone alarm et ses projections
    :- product(alarm, [(d,date),(e,link),(s,severity)]).

    % objet des partis P(alarm)
    :- part(alarm).

    % Morphisme de schema permettant de structurer l'operateur
    % de fouille de donnees utile au chapitre 4
    :- begin_functor(dataAlarm,dataSchema).
```

```
        :- functor_sort(data,alarm).
51    :- end_functor(dataAlarm).

:- end_schema(alarm_data).
```

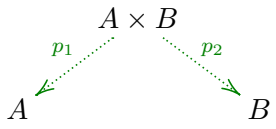
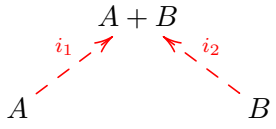
L'utilisation de Prolog pour écrire des schémas se fait naturellement. Toutes les constructions des esquisses relationnelles sont facilement exprimables dans le langage proposé. La flexibilité fournit par le fait qu'une relation ou une fonction puisse être spécifiée ou bien écrite dans le langage est particulièrement intéressante.

Conclusion

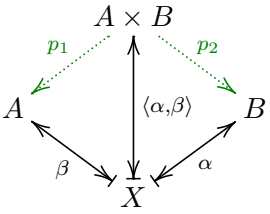
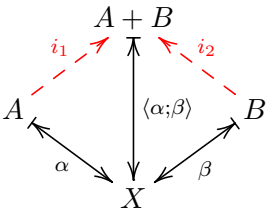
Les esquisses relationnelles finies et discrètes ont été définies dans ce chapitre. Elles étendent les esquisses présentées au chapitre précédent en introduisant un nouveau type d'arc $\nrightarrow$ et l'objet des parties $\mathcal{P}(A)$. La motivation de cette extension est la spécification de la fouille de données présentée dans le chapitre 4. Afin de parler à la fois de spécification et d'implémentation, les *schémas* ont été définis. Un schéma associe une *esquisse relationnelle* à une implémentation.

La notion de relation supporte l'introduction de la notion de parties dans une esquisse relationnelle. Une catégorie relationnelle nécessite la définition de deux types d'arcs : les relations $\nrightarrow$ et les applications $\rightarrow$ qui sont des relations particulières. Nous avons défini une catégorie relationnelle utilisant les *relations orientées* et les fonctions totales.

Par rapport aux esquisses, il n'y a pas de modifications sur le nœud terminal et le morphisme nul (appelé application nulle) qui ne sont pas détaillés dans les tableaux qui suivent. Le tableau 2.46 page suivante résume la construction des cônes et cocônes relationnels. Il est important de noter que les projections et les inclusions sont des applications. Le tableau 2.47 page 60 résume les caractéristiques des relations et des applications. Enfin, le tableau 2.48 page 61 décrit l'objet des parties et les relations associées.

	Exemple	Interprétation dans EnsRel	Référence
Cône relationnel		Le type $A \times B$ est le produit cartésien des types A et B. Les fonctions p_1 et p_2 sont des projections : $p_1((a,b)) = a$ et $p_2((a,b)) = b$	Définitions 2.8 page 43 et 2.22 page 47
Cocône relationnel		Le type $A + B$ est l'union disjointe des types A et B. Les fonctions i_1 et i_2 sont des inclusions : $i_1(a) = a$ et $i_2(b) = b$.	Définitions 2.8 page 43 et 2.26 page 48

TAB. 2.46: Cônes et cocônes.

	Exemple	**Interprétation dans EnsRel**	**Référence**
Relation	$A \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} B$	α est une relation orientée de A vers B	Définitions 2.2 page 41 et 2.18 page 45 et proposition 2.19 page 46
Application	$A \xrightarrow{f} B$	f est une fonction totale de A vers B	Proposition 2.19 page 46
Identité	$A \overset{id_A}{\longmapsto\!\!\!\!\rightarrow} A$	Pour tout élément a de type A, $id_A(a) = a$	Proposition 2.19 page 46
Composition	$A \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} B \overset{\beta}{\longmapsto\!\!\!\!\rightarrow} C$, $= \gamma$	$\gamma = \beta \circ \alpha$ est la composition des relations orientées α et β (tout chemin de relations est composable)	Proposition 2.19 page 46
Factorisation relationnelle		Pour $x \in X$, l'image $\langle\alpha,\beta\rangle(x)$ est le produit cartésien $\alpha(x) \times \beta(x)$.	Définitions 2.8 page 43 et 2.22 page 47 et théorème 2.24 page 47
Cofactorisation relationnelle		Pour $x \in A + B$, si $x \in A$ alors $\langle\alpha;\beta\rangle(x) = \alpha(x)$ sinon $(x \in B)$ $\langle\alpha;\beta\rangle(x) = \beta(x)$	Définitions 2.8 page 43 et 2.26 page 48 et théorème 2.28 page 49
Énumération	$1 \overset{\Sigma_A}{\longmapsto\!\!\!\!\rightarrow} A$	$\sigma_A(\mathbf{1}) = \{a \mid a \in A\}$	Définition 2.40 page 53

TAB. 2.47: Résumé des propriétés des relations et applications.

	Exemple	**Interprétation dans EnsRel**	**Référence**
Objet des parties	$\mathcal{P}(A)$	$\mathcal{P}(A)$ est l'ensemble des parties de l'ensemble A	Définition 2.10 page 44 et 2.32 page 51 et proposition 2.35 page 51
Relation d'énumération	$\mathcal{P}(A) \overset{\sigma_A}{\mapsto\!\!\!\!\!\!\rightarrow} A$	$\forall x \in \mathcal{P}(A), x\sigma_A y$ tel que $y \in x$	Proposition 2.35 page 51
Application unité	$A \overset{\eta_A}{\longrightarrow} \mathcal{P}(A)$	$\forall x \in A, \eta_A(x) = \{x\}$	Proposition 2.35 page 51

TAB. 2.48: Résumé des parties des objets.

Deuxième partie

Fouille de données

Chapitre 3

La fouille de données sans information a priori

Tout problème de fouille de données réunit trois composantes : des données[1] en grand nombre, un utilisateur dont l'objectif est d'extraire des connaissances à partir de ces données et des modèles[2] que l'utilisateur interprète et qui généralisent les données. Les structures des données, les structures des modèles et les objectifs de l'utilisateur sont très variés. À partir de ces différentes combinaisons, énormément de méthodes d'extraction de connaissances ont émergé. Elles emploient chacune des méthodes très particulières mais dans un seul et unique but : *aider un utilisateur à extraire des connaissances à partir de données.*

L'efficacité des méthodes de fouille de données est un point important pour juger de leur qualité. La quantité des données à gérer est bien souvent un frein à des méthodes trop complexes. Les connaissances de l'utilisateur sont un atout permettant d'accroître cette efficacité. En effet, l'introduction des connaissances de l'utilisateur permet de focaliser la recherche vers certains modèles plus que vers d'autres. Ainsi, l'espace de recherche est d'autant réduit et l'extraction en est facilitée. D'un coté cela permet d'accélérer énormément la recherche et d'un autre coté cela biaise les résultats obtenus. En effet, il peut exister de *meilleurs* modèles en dehors de l'espace de recherche considéré. Les techniques actuelles utilisent plus ou moins les connaissances de l'utilisateur. Ce chapitre propose de balayer plusieurs domaines de la fouille de données en conservant le point de vue du manque de connaissances de

[1] Les termes « masse de données », « données » et « ensemble de données » qualifient un ensemble d'éléments dont l'utilisateur souhaite extraire des connaissances.

[2] Le terme « modèle » est employé à la place du terme « modèle de données » lorsqu'il n'y a pas d'ambiguïté.

l'utilisateur sur la structure des modèles à acquérir.

Les méthodes développées en fouille de données sont à l'intersection de nombreuses disciplines comme : l'apprentissage artificiel, les statistiques, les bases de données et la visualisation d'information. Ce chapitre est composé de trois sections en relation avec ces disciplines. Dans la première section, nous présentons des généralisations de la fouille de données composées d'étapes qui utilisent des techniques issues de ces domaines. La seconde section introduit la méthode, issue des statistiques, qui a été choisie pour sélectionner des modèles dans le but de choisir ceux qui résument le mieux des données. Enfin la troisième section expose la fouille de données dans un contexte particulier : l'exploration de journaux d'alarmes. Ce domaine utilise principalement la visualisation d'information pour aider l'utilisateur à comprendre les phénomènes cachés derrière les alarmes.

3.1 La fouille de données

La fouille de données comprend de nombreuses méthodes provenant de divers domaines. C'est pourquoi, chercher à la définir de façon à couvrir toutes ces techniques ne peut amener qu'à une définition générale pas forcément exploitable. Dans notre contexte, la fouille de données est envisagée d'une manière spécifique : l'utilisateur n'a pas ou peu de connaissances sur les données. Comme on va le voir dans la suite, ce point de vue pose un certains nombre de problèmes, notamment sur le choix du type de modèle à extraire ou de la technique à appliquer.

La première sous-section présente quelques définitions de la fouille de données et les commente. Nous spécialisons ces définitions à notre contexte. La deuxième sous-section montre le rôle de la relation de couverture entre les données et les modèles. Enfin la troisième sous-section présente différents travaux qui proposent comme notre approche un cadre général à la fouille de données. À notre connaissance, ces généralisations sont les plus proches de notre méthode de spécification du processus de fouille de données.

3.1.1 Définitions

Nous commençons par commenter les deux principales définitions existantes de la fouille de données en vue d'aboutir à une définition en rapport avec notre contexte.

Définitions générales

D'une manière liée à son utilisation pratique, l'extraction de connaissances dans les bases de données est vue comme une étape d'un processus itératif d'extraction de connaissances illustré dans la figure 3.1 page ci-contre (Fayyad *et al.*, 1996). Le processus est divisé en cinq étapes qui, en partant des données, aboutissent aux connaissances. Toutes les étapes de ce processus sont effectuées avec un objectif particulier sous la conduite d'un utilisateur. Dans ce processus, le concept de fouille de

données est composé de techniques permettant de transformer les données préparées en des modèles.

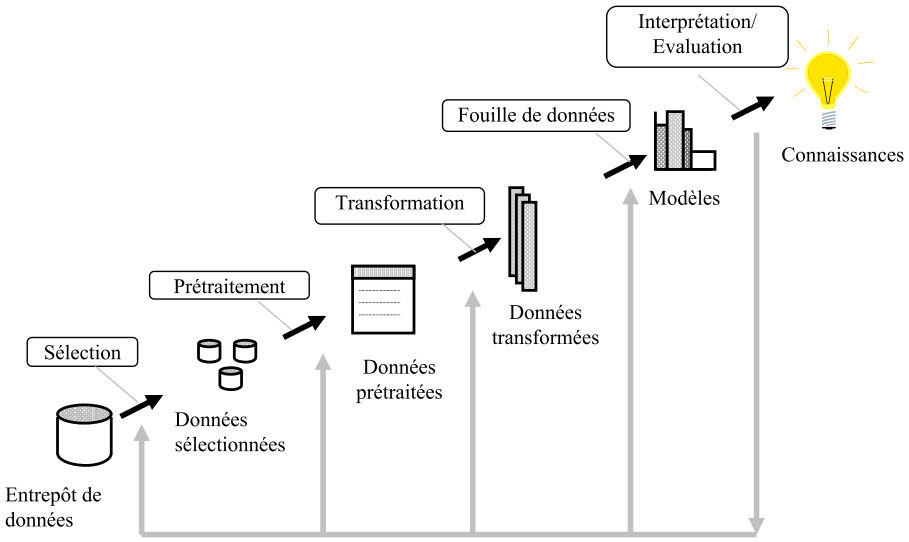


FIG. 3.1: Processus d'extraction de connaissances dans les bases de données.

Une étape de fouille de données peut être vue comme un *opérateur de fouille de données*[3]. L'utilisateur choisit les opérateurs à appliquer selon plusieurs critères : les caractéristiques des données (structure, bruit, quantité,...) et les caractéristiques des modèles à extraire (prédictivité, compréhensibilité, qualité,...). Ensuite, les modèles extraits sont interprétés par l'utilisateur selon son objectif. Enfin, au vu des connaissances acquises, l'utilisateur choisit de réitérer le processus en modifiant les étapes précédemment effectuées. Il a la possibilité d'affiner ses résultats, de changer complètement de technique, de sélectionner d'autres données,...

Une autre définition (Frawley *et al.*, 1992) plus formelle de la fouille de données introduit le langage des modèles. La voici en version originale et traduite :

« Knowledge discovery is the non trivial extraction of implicit, previously unknown, and potentially useful information from data. Given a set of facts (data) F, a language L, and some measure of certainty C, we define a pattern *as a statement S in L that describes relationships among a subset F_S of F with a certainty c, such that S is simpler (in some sense) than the enumeration of all facts in F_S ».*

« la découverte de connaissance est l'extraction non triviale d'informations implicites, précédemment inconnues et potentiellement utiles. Soient un ensemble de faits (les données) F, un langage L et une mesure de certitude C. Un modèle $S \in L$

[3]Les opérateurs de fouille de données sont définis en 4.2 page 95 et appelés opérateurs quand il n'y a pas d'ambiguïté.

décrit les relations entre les éléments d'un sous-ensemble F_S de F avec une certitude c tel que S est plus simple que l'énumération de tous les faits de F_S ».

Le procédé décrit est suffisamment général pour couvrir une grande variété d'approches qui mettent l'accent sur l'économie de la représentation du modèle par rapport aux données. Ce coût pour représenter le modèle est plus particulièrement abordé dans la section 3.2 page 75.

Néanmoins la définition préalable d'un langage L est indispensable. Le langage L définit la *structure* du modèle S. Le terme « structure » recouvre les types de modèle tels que les arbres de décision, les clusterings, les itemsets fréquents... L'utilisateur qui dirige le processus d'abstraction des données connaît la structure du modèle qu'il extrait car il utilise un langage de représentation L bien défini. Ainsi dans la définition proposée, l'information extraite n'est pas composée de la structure du modèle puisqu'elle est connue a priori.

Définition spécialisée à notre contexte

L'extraction de connaissance est dirigée par l'objectif de l'utilisateur. On différencie deux buts : la *vérification* et la *découverte*. La vérification consiste à valider les hypothèses d'un utilisateur. La découverte consiste à laisser le système découvrir de nouveaux modèles. Dans notre contexte, on suppose que l'utilisateur a peu de connaissance a priori sur les données. Il lui est très difficile d'émettre des hypothèses sur celles-ci. La découverte, quant à elle, présuppose peu de connaissances. On différencie deux tâches dans la découverte en fouille de données : la *prédiction* où le système découvre des modèles prédictifs dans le but de prédire le futur et la *description* où le système trouve des modèles descriptifs pour les représenter à un utilisateur.

Tous les systèmes de fouille de données nécessitent des connaissances de la part de l'utilisateur. Aussi minimes soient-elles, elles sont le point de départ de tout processus d'extraction de connaissances. Si on suppose que l'utilisateur a peu de connaissance sur les données alors le premier objectif est de lui faire acquérir des connaissances. Ensuite, il peut réitérer le processus à partir de son interprétation des modèles. De notre point de vue, la découverte de descriptions est le point de départ essentiel de la fouille de données. Par exemple, à la réception d'un fichier de données, le premier pas est d'examiner rapidement ce fichier en vue d'entrevoir des types de données, des moyennes de valeurs, la structure des données... Toutes ces informations constituent des descriptions de données.

L'objectif le moins contraignant de la fouille de données, relativement aux connaissances a priori de l'utilisateur, est la découverte de description. Dans la suite, nous poursuivons la spécialisation de cet aspect de la fouille de données.

3.1.2 Modèle de données et relation de couverture

L'objectif d'un modèle de fouille de données est de synthétiser l'information contenue dans l'ensemble de données à partir d'où il est extrait. Une partie de l'information contenue dans les données est absente du modèle. De ce fait, à partir d'un modèle, il est normalement impossible de retrouver exactement les données dont il provient. Cependant, il est possible de retrouver des données appelés *couverture du modèle*. C'est la *relation de couverture* qui lie un modèle aux données qu'il couvre. Dans un premier temps on suppose que la relation de couverture est définie par

$$cover : model \leftrightarrow data \tag{3.2}$$

où *model* est un type de modèle, *data* est un type de données et *cover* une relation binaire (définition 1.7 page 13). Deux exemples sont présentés dans les deux pages suivantes. Pour se guider plus facilement dans l'exploration de l'ensemble des modèles, on dote cet espace d'une relation d'ordre partiel appelée *relation de généralité*. La figure 3.3 représente l'espace des modèles muni de la relation de généralité et le lien réalisé par la relation de couverture entre espace des modèles et espace des données. La relation de généralité $\preceq$: $model \leftrightarrow model$ est compatible avec la relation de couverture telle que tout modèle $M \in model$ est plus général que le modèle $M' \in model$ si et seulement si la couverture de M' est incluse dans la couverture de M :

$$M \preceq M' \Leftrightarrow cover(M') \subseteq cover(M)$$

où $cover(x)$ représente l'ensemble des images de x par *cover*.

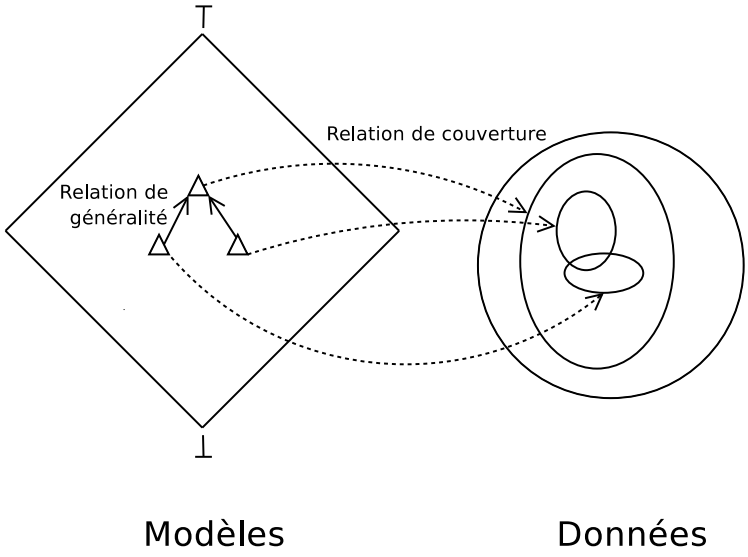


FIG. 3.3: Le lien entre la relation de couverture et la relation de généralité.

Il est important de noter que dans l'expression (3.2) page précédente, un modèle énumère *une par une* les données qu'il couvre. Cependant, pour certains modèles, les données qui sont couvertes dépendent les unes des autres (exemple 3.5). Ainsi, il faut au contraire lier un modèle à des ensembles de données dépendantes. C'est pourquoi, formellement, la relation de couverture est représentée par la relation :

$$cover : model \leftrightarrow \mathcal{P}(data)$$

Ce qui s'écrit différemment dans le cadre des esquisses relationnelles pour lesquelles les relations doivent être orientées. La définition suivante formalise une relation de couverture et un modèle de fouille de données.

Définition 3.4 (Relation de couverture et modèle de données)
Soient un ensemble $model$, un élément $M \in model$ et un type de données $data$. Une relation de couverture est définie par:

$$cover : model \nrightarrow \mathcal{P}(data)$$

L'image $cover(M)$ énumère des sous-ensembles de données. L'union de ces sous-ensembles constituent la *couverture* de M. L'élément M est appelé un *modèle de données* de type $model$. Chaque sous-ensemble de $cover(M)$ est un ensemble de données dépendantes selon le modèle M et la relation de couverture $cover$. ▪

Exemple 3.5 (Dépendance dans une séquence)
Dans le cadre des séquences, on considère une séquence d'évènements comme un ensemble d'évènements $s = \{(A,1),(B,2),(A,4),(A,5),(A,6),(B,8)\}$ où, par exemple, A représente le type de l'évènement et 1 sa date. De plus, on considère un modèle composé de types d'évènements $m = \{A, B\}$ tel qu'un modèle couvre les évènements successifs du type correspondant. Le modèle m couvre les deux couples d'évènements $\{(A,1),(B,2)\}$ et $\{(A,6),(B,8)\}$ dans la séquence s. Les deux évènements de chacun de ces couples sont dépendants selon le modèle m. En effet, on ne peut pas dire que le modèle m couvre l'évènement $(A,1)$ seul ou l'évènement $(B,2)$ seul car ils sont dépendants vis-à-vis du modèle (il en va de même pour le couple $\{(A,6),(B,8)\}$). ▪

La couverture d'un modèle est l'énumération de *toutes* les données qu'il couvre. Si on s'intéresse à un ensemble particulier D de données alors on peut s'intéresser aux données couvertes dans D, autrement-dit aux données qui appartiennent à la fois à la couverture du modèle et à D. On parle alors de *données couvertes* dans D. Ainsi un modèle M et sa relation de couverture séparent D en deux ensembles : les données couvertes dans D et les données non couvertes dans D.

Définition 3.6 (Données couvertes dans un ensemble de données)
Soient des données $D \in \mathcal{P}(data)$, une relation de couverture $cover : model \nrightarrow \mathcal{P}(data)$ et un modèle $M \in model$. Le sous-ensemble des données D couvertes par M est noté $D_{cover(M)}$ et le sous-ensemble des données D non couvertes par M est

noté $D_{\overline{cover}(M)}$.

$$D_{cover(M)} = \bigcup d \text{ tel que } d \in cover(M) \wedge d \subseteq D$$

L'ensemble des données non couvertes dans D est le complémentaire de $D_{cover(M)}$ relativement à D :

$$D_{\overline{cover}(M)} = D \setminus D_{cover(M)}$$

Quand il n'y a pas d'ambiguïté sur la relation de couverture, on omet de préciser la relation de couverture utilisée. ▪

Il est important de différencier le terme « couverture » d'un modèle qui regroupe toutes les données qui sont couvertes par un modèle du terme « données couvertes », faisant référence à un sous-ensemble D des données, qui est l'ensemble des données de D qui sont couvertes par le modèle en question.

Exemple 3.7 (Couverture d'évènements)
Considérons une séquence d'évènements $s = \{(A,1),(B,2),(A,3),(C,4),(A,5),(B,6),(A,7),(C,8),(A,9),(B,10)\}$. Un algorithme de fouille de données basé sur la fréquence extrait le modèle $M = (A,B)$. La relation de couverture r associée à ce modèle indique qu'il couvre toutes les paires d'évènements $(A,i),(B,i+1)$ avec $i \geq 0$. Si on considère la séquence s, les valeurs de i intéressantes sont $1, 5$ et 9. Les éléments de la séquence s qui sont couverts sont $s_{r(M)} = \{(A,1),(B,2),(A,5),(B,6),(A,9),(B,10)\}$ et les éléments non couverts sont $s_{\overline{r}(M)} = \{(A,3),(C,4),(A,7),(C,8)\}$. ▪

3.1.3 Généralisations de la fouille de données

Plusieurs formalisations de la fouille de données ont été proposées mais, à notre connaissance, aucune d'entre elles n'a véritablement traité le problème particulier de la découverte de descriptions. Plusieurs problèmes surgissent lors de sa mise en œuvre. Tout d'abord, l'utilisateur ne peut effectuer la tâche de préparation des données. Sans objectif, il est impensable de sélectionner, prétraiter ou transformer les données. De plus le choix d'un opérateur est particulièrement complexe car l'utilisateur ne sait pas quel type de description il veut et peut obtenir. Enfin, la dernière étape d'évaluation est simple si l'utilisateur réussit à interpréter le modèle extrait. Toutefois, si plusieurs modèles sont extraits, seule l'analyse de chacun d'entre eux, coûteuse en temps pour l'utilisateur, permet d'évaluer leur qualité. Dans la suite, nous présentons plusieurs généralisations de la fouille de données. Les deux premières, liées respectivement aux bases de données inductives et à une algèbre pour la fouille de données présentent un processus de fouille de données comme une succession de requêtes. Ensuite, une généralisation nommée aide à la fouille de données est présentée. Cette méthode est la plus proche de notre problématique car elle n'exige pas beaucoup de connaissances de la part de l'utilisateur pour exécuter un processus de fouille de données. Enfin, une librairie pour la recherche de motifs fréquents et deux logiciels permettant d'extraire visuellement des connaissances sont introduits. Ils reflètent les applications actuelles de fouille de données.

Base de données inductive

Les bases de données inductives (Imielinski et Mannila, 1996) sont nées de la nécessité de formaliser la fouille de données et d'établir des liens clairs avec les concepts et les algorithmes utilisés dans le domaine des bases de données. Le principe sous-jacent aux bases de données inductives s'appuie sur les bases de données, dont l'efficacité et la gestion sont très poussées, au profit de la fouille de données. Une base de données inductive est formée de deux composants : les données et les modèles. Le terme « inductif » vient du fait que les modèles sont induits à partir des données. Le langage de requêtes mélange des opérations sur ces deux composants. Ainsi le processus de fouille de données est vu comme une succession de requêtes à une base de données inductive. Différents langages de requêtes accompagnés des méthodes de résolutions ont été proposées (Lee et De Raedt, 2004; Kramer *et al.*, 2001; Fromont *et al.*, 2006; Vautier *et al.*, 2005b; Vautier *et al.*, 2005a). Le principe des bases de données inductives suppose que l'utilisateur possède assez de connaissances pour définir les requêtes car elles matérialisent les connaissances a priori. Des travaux (Boulicaut *et al.*, 1999; Han *et al.*, 1996) proposent d'utiliser un langage de requête commun aux diverses techniques de fouille de données. Cependant, les requêtes ne permettent pas de mélanger différentes structures de modèles.

Algèbre pour la fouille de données

Une algèbre pour la fouille de données (Johnson *et al.*, 2000) englobe à la fois les bases de données et certaines tâches de fouille de données. Elle formalise les objets manipulés en trois mondes : le monde des données, le monde des intentions et le monde des extensions. Le monde des données est une base de données relationnelles où il est possible d'exécuter les opérations classiques de base de données (union, produit, différence, sélection...). Le monde des intentions supporte la gestion de régions (modèles) qui sont des généralisations des éléments du monde des données. Une région est un ensemble de contraintes simples (numériques ou symboliques) sur des relations du monde des données. Le monde des extensions représente les relations entre données et régions (dans notre contexte, ce monde est représenté par la relation de couverture définie dans la section précédente). Une opération de fouille de données constitue un pont qui part du monde des données et qui arrive dans le monde des intentions. Son implémentation est une boîte noire attachée aux relations sur lesquelles elle peut s'appliquer. Calders a complété cette approche en remplaçant la boite noire par des opérations spécialement conçues pour la fouille de données (Calders *et al.*, 2006) qui permettent d'optimiser l'application de certains algorithmes.

Aide à la fouille de données

L'aide à la fouille de données (Bernstein *et al.*, 2005) est la méthode qui se rapproche le plus de notre problématique. L'utilisateur est aidé dans le choix du processus de fouille de données à appliquer. Les processus sont constitués d'opérations

simples en rapport avec la fouille de données et organisées dans une ontologie (voir section 1.1 pour une définition d'ontologie). Un extrait de l'ontologie utilisée dans le cadre de ces travaux est illustré dans la figure 3.8. À partir de cette ontologie et des données à explorer, il est possible d'énumérer tous les processus de fouille de données exécutables. La figure 3.9 fournit un exemple de processus. Le classement des processus selon différents critères (qualité, vitesse, compréhensibilité...) donne à l'utilisateur une information supplémentaire sur le choix du processus à exécuter effectivement.

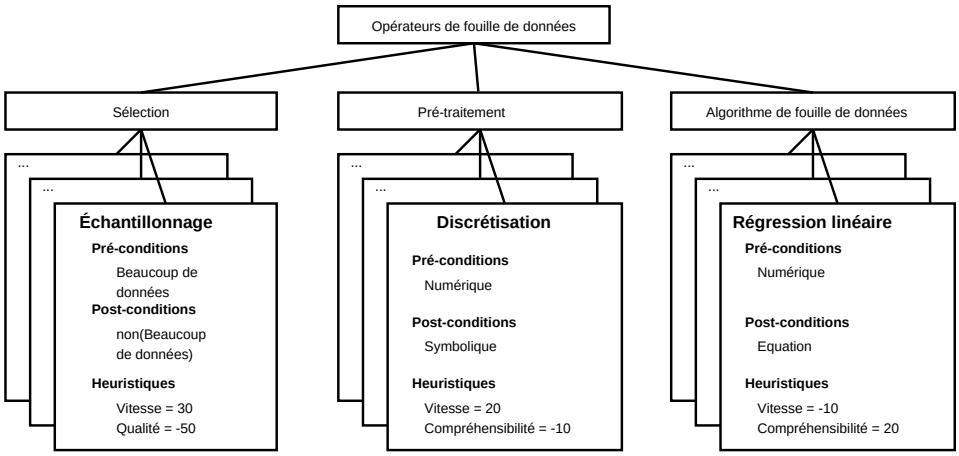


FIG. 3.8: Extrait de l'ontologie des opérateurs de fouille de données.

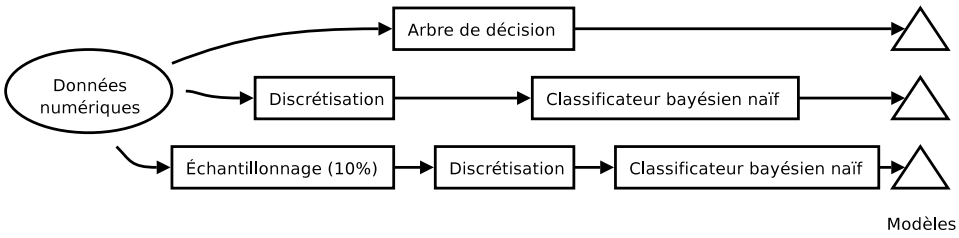


FIG. 3.9: Trois processus générés automatiquement.

Les post-conditions et les pré-conditions ne permettent pas l'exécution automatique du processus sur des données dont la structure est complexe. D'après la description de l'ontologie, il n'est pas possible de définir des opérateurs qui nécessitent des structures de données autres que numériques ou symboliques ou même des structures qui mélangent ces types. De plus, l'exécution de plusieurs processus nécessite ensuite l'analyse de leurs résultats. Dans le cas de nombreux modèles à interpréter, l'absence d'évaluation automatique des modèles implique un coût important en temps pour l'utilisateur afin d'analyser chacun des modèles extraits.

Une autre approche (Morik, 2000) consiste à disposer d'une base de processus de fouille de données appliqués avec succès. Un élément de cette base est composé des caractéristiques des données analysées, de la tâche de fouille de données effectuée et de tout le processus exécuté. Une comparaison des cas de la base avec les données et la tâche de l'utilisateur permet d'extraire les processus susceptibles d'être intéressants.

Librairie générique pour la recherche de motifs fréquents

La recherche de motifs fréquents dans les bases de données est un domaine actif depuis les années 90 (Mannila *et al.*, 1997). Les motifs, assimilables à des ensembles, des séquences ou des graphes sont recherchés dans une base de données. Ceux dont le nombre d'instances dans les données est supérieur à un seuil fixé par l'utilisateur sont extraits. La proposition d'un cadre généralisant cette recherche (Zaki *et al.*, 2005) montre que tous ces motifs sont généralisables à des graphes plus ou moins complexes. De plus, elle montre qu'il est possible de généraliser les algorithmes de manière à rechercher soit d'abord en largeur soit d'abord en profondeur. Cette généralisation est fournie dans une librairie en C++ dans le but de faciliter l'écriture de nouveaux algorithmes de recherche de motifs fréquents.

Logiciels regroupant des techniques de fouille de données

Il existe des logiciels (libres ou commerciaux) qui regroupent un certain nombre de techniques de fouille de données. Deux d'entre-eux, libres et axés sur la visualisation, sont particulièrement complets, il s'agit de Yale récemment renommé RapidMiner (Mierswa *et al.*, 2006) et Orange (Demsar *et al.*, 2004). Chaque opération est considérée comme une composant visuel d'une chaîne formant un processus d'extraction de connaissances. Les types de données autorisés par chacune des briques n'est qu'un ensemble de types simples : numérique ou symbolique. C'est l'utilisateur qui agence ces composants de façon à définir le processus à exécuter.

La formalisation de la fouille de données sous la forme d'une succession de requêtes généralise l'interactivité entre le système de fouille de données et l'utilisateur. Néanmoins, il suppose que l'utilisateur possède suffisamment de connaissances pour écrire ces requêtes. La représentation des opérations du processus d'extraction de connaissances sous la forme d'une ontologie ou de composants visuels constitue des approches pertinentes en fouille de données. Mieux ces opérations sont décrites, plus l'automatisation du processus de fouille de données peut être effectuée.

Avec peu de connaissance, fouiller des données s'avère extrêmement difficile. Pour l'utilisateur, il est impossible de décrire un objectif de fouille de données précis pourtant nécessaire au traitement des données. Or, la plupart des systèmes formalisant la fouille de données supposent que l'utilisateur possède de nombreuses connaissances sur les données qu'ils souhaitent traiter. Si l'utilisateur a peu de connaissances, une voie possible est de lui présenter des descriptions des données. De cette façon, il acquiert rapidement les capacités lui permettant d'envisager d'exécuter d'autres processus d'extraction de connaissances.

3.2 Introduction au principe MDL

Comment déterminer le modèle qui résume le mieux des données ? Si l'utilisateur avait des connaissances sur ses données, une des possibilités serait de le laisser définir un critère d'évaluation. Mais si l'utilisateur n'a pas de connaissance alors l'évaluation automatique des modèles est recommandée. Les statistiques proposent des méthodes de *sélection de modèle* basées sur différents critères d'évaluation. Un critère de sélection de modèle mesure l'écart du modèle avec les données mais aussi la complexité du modèle. Il s'agit effectivement de ne pas représenter les données par des modèles trop complexes et donc difficiles à interpréter pour l'utilisateur. Il est important de noter que dans cette section nous ne nous limitons pas aux types de modèles liés par une relation de couverture avec les données. En effet, nous considérons tous les types de modèles, quel que soit leur lien avec les données (dans la section 4.3, nous nous bornons à l'évaluation de modèles avec relation de couverture).

Dans le cadre de la sélection de modèle, une méthode particulière basée sur la complexité de Kolmogorov et plus précisément sur le *principe MDL* (Minimum Description Length) est présentée. Ce principe repose sur la recherche de la plus courte description des données dans un langage particulier. Plus la description est courte plus on considère qu'elle résume convenablement les données. Ce principe est extrêmement générique car tout modèle peut être associé à une description.

Tout d'abord, le domaine statistique de la sélection modèle est introduit. Différents critères de sélection sont comparés dont le critère basé sur le principe MDL. Ensuite, les origines du principe MDL sont exposées à partir de la complexité de Kolmogorov. Dans la troisième sous-section, le principe MDL est précisément défini et discuté. Enfin quelques applications représentatives de l'utilisation de MDL sont présentées.

3.2.1 Sélection de modèle

La sélection de modèle consiste à choisir, parmi un ensemble de structures, la structure du modèle qui décrit au mieux un ensemble de données. Ce problème a d'abord été rencontré en statistiques (Kverh et Leonardis, 2004; Akaike, 1973) et plus précisément en régression. Étant donnés n couples (x_i, y_i) tels que $i = 1, \ldots, n$ qui constituent les observations d'un certain phénomène, quelle est la fonction $y = f(x)$ qui décrit au mieux ce phénomène ? Il existe un nombre infini de fonctions mais la question principale est quelle structure de fonction choisir ? Les structures les plus connues sont les fonctions linéaires, polynomiales, logarithmiques ou encore exponentielles. Une fois que la structure a été choisie, une possibilité est de trouver la fonction qui minimise l'écart entre les données et le modèle. Cet écart est souvent calculé comme l'opposé de la *vraisemblance*. La vraisemblance d'un modèle M pour un ensemble de données D est la probabilité $p(D|M)$ que les données aient été générées à partir de ce modèle. Dans le cas d'une fonction linéaire $f(x) = ax + b$, la maximisation de la vraisemblance consiste à estimer les valeurs optimales des paramètres a et b tels que la quantité $\sum_i (y_i - f(x_i))^2$ soit minimale (méthode des moindres carrés).

Sans se borner aux fonctions, dans le cas général, supposons que pour chaque structure de modèle, il est possible d'extraire le modèle qui est le plus proche des données (qui maximise la vraisemblance). Dans ce cas, comment déterminer quelle structure est la plus à même de modéliser le phénomène observé ?

Une façon simple de décider si une structure est meilleure qu'une autre est de comparer les deux modèles optimaux pour chacune des structures et considérer comme meilleur celui qui maximise la vraisemblance avec les données. Cependant, ce n'est pas une bonne idée car une structure de modèle avec beaucoup de paramètres est plus proche des données qu'une structure avec peu de paramètres. Or, il est souvent préférable d'obtenir des modèles simples (voir le rasoir d'Occam dans la sous-section 3.2.3 page 83), autrement dit utilisant le moins de paramètres possibles. Ainsi, il y a un compromis à faire entre la vraisemblance du modèle et la complexité du modèle. On distingue les méthodes, dites paramétriques, qui traitent les modèles dont le nombre de paramètres est connu et fini des méthodes, dites non paramétriques, qui traitent les modèles dont le nombre de paramètres n'est pas connu à l'avance.

Dans le cas des méthodes paramétriques, beaucoup de solutions ont été proposées (Schwarz, 1978; Akaike, 1973; Rissanen, 1978) et introduisent différents critères de sélection de modèle notés *CSM*. Ils sont largement utilisés en vision par ordinateur (Kverh et Leonardis, 2004) et combinent l'écart du modèle avec les données et la complexité du modèle :

$$CSM(D, M) = E(D, M) + C(D, M)$$

où $E(D, M)$ représente l'écart du modèle M aux données D et $C(D, M)$ représente la complexité du modèle M relativement aux données D. Il faut noter que pour certaines méthodes la complexité du modèle ne tient pas compte des données :

$C(D, M) = C(M)$. De manière à d'éclaircir la notion de critère de sélection de modèle, les trois principaux critères BIC, AIC et MDL sont brièvement introduits.

Critère BIC

Le critère BIC (Schwarz, 1978), pour Bayesian Information Criterion, est basé sur le théorème de Bayes. Il maximise la probabilité que des données D soient générées par une structure de modèle et des informations a priori. La formule générale n'est pas donnée ici mais son approximation est :

$$CSM_{BIC}(D, M) = -log(p(D|M)) + \frac{1}{2}dim(M)log(|D|)$$

où $p(D|M)$ représente la vraisemblance du modèle M, $dim(M)$ représente le nombre de paramètres libres de la structure du modèle M et $|D|$ représente la quantité de données. La pénalité de la complexité du modèle est proportionnelle au *log* de la taille des données.

Critère AIC

Un autre façon d'évaluer un modèle M est d'estimer la distance $d(M, M^*)$ entre le modèle M et le *vrai* modèle M^*. L'un des problèmes de cette technique est qu'elle nécessite la connaissance de la distribution et de la variance du bruit. Or, dans la plupart des cas, cette information est inconnue. Cependant, si on fait l'hypothèse d'une distribution gaussienne, il est possible d'approximer $d(M, M^*)$ par le critère AIC (Akaike, 1973), pour Akaike Information Criterion :

$$CSM_{AIC}(D, M) = -log(p(D|M)) + dim(M)$$

Le critère AIC inflige une pénalité moins importante à la complexité du modèle que le critère BIC quand $|D| > 7$. En pratique, les résultats obtenus pour ces deux critères sur des simulations sont très différents suivant la taille de l'échantillon et la complexité du vrai modèle (Lebarbier et Mary-Huard, 2006). Dans le cas de modèles très simples, BIC est meilleur que AIC qui sélectionne des modèles plus complexes. Cependant, dans le cas de modèles complexes, AIC est meilleur même dans le cas d'un grand volume de données car BIC sélectionne des modèles trop simples.

Critère MDL

Le critère MDL (Rissanen, 1978; Hansen et Yu, 2001), pour Minimum Description Length, est plus longuement discuté dans les sous-sections suivantes. En statistique, il est utilisé d'une façon particulièrement rigide. Ce critère minimise le nombre de bits nécessaires pour encoder les données D en utilisant le modèle M :

$$CSM_{MDL}(D, M) = -log(p(D|M)) + L(M)$$

où, $L(M)$ représente la taille du modèle M. Contrairement à BIC ou AIC, l'influence de la pénalité due à la complexité du modèle M n'est pas seulement dépendante de sa dimension. Ainsi, le calcul de $L(M)$ doit être spécifié pour tout type de modèle. D'un point de vue technique de l'encodage des données, si on suppose que le décodeur et l'encodeur connaissent l'ensemble des données que M peut générer alors il faut en moyenne $-log(p(D|M))$ bits pour coder D sachant le modèle M.

À l'opposé des méthodes de sélection de modèles paramétriques, il existe des méthodes non paramétriques basées sur des échantillonnages de données. Ce sont les méthodes de validation croisée, bagging et boosting (Kohavi, 1995). Intuitivement, ces méthodes échantillonnent les données de façon particulière et extraient un modèle à partir de chaque échantillon prélevé. Ensuite, chaque modèle est évalué selon un certain critère sur les données non échantillonnées. Finalement, le modèle ayant la meilleure évaluation selon le critère choisi est conservé. Il faut souligner l'importance de ce critère qui est en relation avec un objectif particulier. Les méthodes paramétriques précédentes ont pour objectif d'extraire un modèle proche des données alors que dans le cas des méthodes d'échantillonnages l'objectif peut être différent. Par exemple, en classification, l'objectif est d'obtenir un modèle ayant la plus petite erreur de généralisation possible. D'autres méthodes de sélection de modèle non paramétrique (Gutierrez-Pena et Walker, 2001; Karabatsos, 2006) existent. Elles sont particulières à des modèles qui représentent des distributions modulées par des informations a priori. Ces méthodes extrêmement techniques d'un point de vue statistique ne sont pas développées ici.

Les objectifs généraux qui guident la sélection de modèles sont la recherche de modèles à la fois simples et proches des données. Cette approche est tout à fait adaptée dans le cadre de la découverte de modèles sans connaissance a priori. La sélection de modèle est un domaine très étudié en statistique. Les méthodes et surtout les critères mis en place reposent sur des bases théoriques et des expérimentations concluantes dans divers domaines. Cependant la structure des modèles envisagés est souvent restreinte à des familles de distributions particulières, paramétriques ou non.

Or, la fouille de données produit des modèles qui ne sont pas toujours des distributions. Souvent, le seul lien qui relie un modèle à des données est la relation de couverture (sous-section 3.1.2 page 69). Dans ce cas, les méthodes classiques de sélection de modèle échouent car la notion de probabilité entre le modèle et les données est simplement binaire : les données sont dans ou en dehors la couverture. En effet, considérons le modèle le plus général d'un espace de données. Ce modèle est très simple et la probabilité qu'un élément lui appartienne est certaine. Ainsi il a la meilleure évaluation ce qui n'est pas envisageable. Par conséquent, il est indispensable de définir une méthode qui module le lien entre un modèle et des données en fonction d'une relation de couverture.

Le critère MDL a trois avantages : il évalue des modèles extrêmement variés à condition de définir une fonction qui calcule leur complexité, il ne nécessite pas d'information a priori sur les modèles à extraire ni de mesure de probabilité entre les données et le modèle. La sous-section suivante introduit les bases du principe MDL.

3.2.2 Complexité de Kolmogorov

Supposons que nous voulions décrire un objet donné par une phrase. Parmi toutes les phrases décrivant parfaitement cet objet, on choisit la phrase la plus courte. La taille de cette phrase s'interprète comme la complexité de l'objet en question. Il est naturel d'appeler un objet *simple* s'il possède au moins une courte description, et d'appeler un objet *complexe* si toutes ses descriptions sont longues. La théorie sous-jacente à cette idée est la complexité de Kolmogorov développée par Andrey Kolmogorov, Ray Solomonoff et Gregory Chaitin dans les années 1960 (Li et Vitány, 1997).

Il faut bien noter que la *théorie de l'information* de Shannon (Shannon, 1948; Cover et Thomas, 2006) est différente de la problématique posée par la complexité de Kolmogorov (Grünwald et Vitányi, 2003). Shannon traite du problème de la transmission de données tandis que la complexité de Kolmogorov traite de la quantité d'information contenue dans un objet considéré individuellement. Dans la théorie de l'information, le *sens* du message est ignoré, seul le problème de la communication du message entre un émetteur et un récepteur, sous l'hypothèse que l'ensemble des messages possibles est connu par les deux acteurs de la communication, est traité. On peut voir la complexité de Kolmogorov comme la quantité d'information représentée par un objet. La principale différence est que la complexité de Kolmogorov considère l'objet individuellement alors que la théorie de l'information considère la quantité d'information d'un objet relativement à un ensemble d'objets (et à une distribution de probabilités sur ces objets).

L'intuition tend à nous faire dire que des objets sont complexes et d'autres sont simples. Par exemple, le nombre 2^{1000} est très simple à décrire tandis qu'il existe d'autres nombres composés d'un millier de bits pour lesquels il est difficile d'imaginer une description de taille inférieure à 1000 bits. Ces nombres sont considérés aléatoires. Les deux exemples suivants illustrent le fait que des objets, pourtant a priori vus comme complexes, sont très simples selon leur description :

Exemple 3.10 (Compression de π)
Il existe plusieurs algorithmes générant les n premières décimales de π. De tels algorithmes sont basés sur la décomposition en développement limité de fonctions. Par exemple, la décomposition de $arctan(1) = \pi/4$ introduite James Gregory(1638 - 1675) est :

$$\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

La taille de cet algorithme est constante mis à part l'écriture du nombre n indiquant le nombre de décimales à générer qui s'exprime en $O(log\ n)$ bits. Ainsi, quand n est très grand, la taille du programme générant les n premières décimales est très petite comparée à n : le nombre π est très régulier et est considéré comme simple. ▪

Exemple 3.11 (Génération de Fractales)
Les objets mathématiques comme les fractales (figure 3.12) constituent un bon exemple d'objet simple selon la complexité de Kolmogorov car il existe une fonction permettant de les générer. Celle-ci nécessite des paramètres qui, comme dans l'exemple précédent concernant π, sont codés de façon à ce que la taille réelle de l'image soit bien supérieure à la taille du programme permettant de la générer. Ainsi ce type d'objet est aussi considéré comme simple. ▪

FIG. 3.12: Fractale générée à partir du logiciel Apophysis.

On voit que l'évaluation de la complexité d'un objet nécessite deux mécanismes : un langage de description et une méthode pour retrouver un objet à partir d'une description. Évidemment, une description est utile s'il est possible de reconstruire un seul objet à partir de celle-ci. Cependant un tel langage peut favoriser certaines descriptions plutôt que d'autres et ainsi biaiser la complexité d'un objet (ce point est plus particulièrement discuté dans la sous-section 3.2.4 page 84). Par exemple, il existe des langages de programmation qui favorisent le calcul symbolique tandis que d'autres favorisent le calcul arithmétique alors que ces deux langages sont universels (au sens de Turing). La notion de quantité d'information contenue dans un objet individuel est intéressante si elle est indépendante du langage de description. De manière à éviter cette dépendance, la théorie de la calculabilité, apparue dans les

années 1930, spécifie un langage de description universel au moyen de la machine de Turing. À partir de cette théorie, les trois chercheurs Andrey Kolmogorov, Ray Solomonoff et Gregory Chaitin ont pu prouver dans les années 1960 le théorème suivant : pour deux langages universels de programmation $\mathcal{A}$ et $\mathcal{B}$ et pour toute donnée (tout objet) D, la taille du plus petit programme générant D écrit en $\mathcal{A}$ et la taille du plus petit programme générant D écrit en $\mathcal{B}$ diffèrent d'au plus une constante c qui ne dépend pas de la taille de D. Ce théorème connu sous le nom de *théorème de l'invariance* dit simplement que si la taille des données D est suffisamment importante alors peu importe le langage de programmation utilisé pour la décrire. Dès lors, on peut définir la complexité de Kolmogorov :

Définition 3.13 (Complexité de Kolmogorov)
Soient une machine universelle f et des données D. La quantité d'information $K(D)$ contenue dans les données D est la taille $L(p)$ du plus petit programme p tel que $f(p) = D$:

$$K(D) = min\{L(p)|f(p) = D\}$$.

La complexité de Kolmogorov considère la quantité d'information contenue dans un objet comme la taille de la plus petite description de cet objet. C'est un moyen relativement naturel qui dépend d'une machine universelle capable de reconstruire cet objet à partir de sa description. Un des problèmes fondamentaux de la complexité de Kolmogorov est l'impossibilité de la calculer effectivement. Il est possible de la majorer sans jamais savoir s'il existe une description plus petite. Néanmoins, son approximation la plus célèbre, le principe MDL, s'en inspire largement et propose différents moyens de l'appliquer.

3.2.3 Définitions et propriétés

Comment décider quel est le meilleur modèle M parmi un ensemble de modèles $\mathbb{M}$ généralisant les données D ? C'est le problème de la sélection de modèle auquel appartient la méthode de sélection MDL (Minimum Description Length) (Rissanen, 1978; Rissanen, 1989). Intuitivement, l'idée derrière MDL est que des régularités (au sens large) peuvent être utilisées pour compresser des données et ainsi les décrire avec moins de symboles que pour décrire les données littéralement. Cependant, le langage de description utilisé doit être compréhensible de manière à ce que la description soit analysable par un utilisateur afin qu'il puisse en extraire des connaissances. Autrement dit, plus on peut compresser des données sous une forme compréhensible et plus on en apprend sur elles (Grünwald *et al.*, 2005).

La complexité de Kolmogorov est incalculable ce qui la rend inapplicable. En vue de pallier à cette difficulté, le principe MDL propose de définir un langage de description non universel tel qu'il soit possible de calculer la plus petite description. Cette approximation a pour effet d'introduire un biais constitué par le langage choisi. Néanmoins, de nombreuses applications présentées dans la sous-section 3.2.4 page 84 utilisent avec succès ce principe.

D'après la définition 3.13 page précédente de la complexité de Kolmogorov, le programme p décrivant des données D est nécessaire et suffisant pour retrouver D : il couvre exactement D. Supposons qu'il existe un ensemble de données $\mathbb{D}$ dans lequel D est inclus, que l'on dispose d'un ensemble de modèles $\mathbb{M}$. Un modèle $M \in \mathbb{M}$ couvre un ensemble de données inclus dans $\mathbb{D}$. Si le modèle M a été extrait à partir de D, il y a de fortes chances qu'il ne couvre pas strictement D : d'une part, il ne couvre pas tous les éléments de D et, d'autre part, il couvre d'autres éléments de que ceux de D. Ainsi, il est impossible d'associer le programme p à un modèle M car celui-ci ne couvre pas strictement D. L'intérêt du principe MDL est justement de diviser la description p en deux parties : le modèle et le reste.

Définition 3.14 (Principe MDL - Minimum Description Length)
Soient un ensemble $\mathbb{M}$ de modèles et des données D. Selon le principe MDL, le meilleur modèle $M \in \mathbb{M}$ pour expliquer les données D est celui qui minimise la somme $C(D, M) = L(M) + L(D|M)$ où

- $L(M)$ est la taille, en bits, du modèle M et
- $L(D|M)$ est la taille, en bits, des données D encodées avec l'aide du modèle M. ▪

L'exemple suivant illustre les régularités et le bruit dans des séquences binaires.

Exemple 3.15 (Régularités et bruit)
Soient les 3 séquences suivantes :

$$S_1 = 000100010001000100010001 \ldots 000100010001000100010001$$

$$S_2 = 011010110100111100101001 \ldots 111011100001010111011011$$

$$S_3 = 100010000000011000000010 \ldots 000010100000100001100001$$

La première séquence paraît régulière. Intuitivement, elle obéit à une loi simple qui est la répétition du motif 0001. Il est naturel de dire que les données à venir obéiront à cette loi qui constitue donc le modèle de la séquence. La seconde séquence a été générée par un lancer de pièce. Ce qui veut dire qu'elle est aussi aléatoire que possible et qu'aucune régularité ne peut y être détectée. Son modèle est donc vide et sa complexité correspond à sa taille. La troisième séquence contient approximativement quatre fois plus de 0 que de 1 ce qui peut constituer son modèle. Elle semble plus aléatoire que la première séquence mais plus régulière que la seconde. Sachant qu'il y a plus de 0 que de 1, il est possible d'utiliser cette propriété pour coder le coût d'un 0 et d'un 1 dans cette séquence.

	M	$D\|M$
S_1	répétition de 0001	$\emptyset$
S_2	$\emptyset$	S_2
S_3	4 fois plus de 0 que de 1	coût d'un 0 : $-\frac{4}{5}log(4/5)$ coût d'un 1 : $-\frac{1}{5}log(1/5)$

.

Les propriétés du principe MDL en font un critère privilégié dans l'analyse de données. Malgré son principal défaut qui est la dépendance au langage de description pour enlever le problème d'incalculabilité, l'utilisation d'un tel principe procure plusieurs avantages dont la simplicité des modèles obtenus, le contournement du *surapprentissage* et la généricité des modèles évalués.

Dépendance au langage de description

Dans le cas d'un langage de description universel, on peut montrer qu'il n'existe pas de programme informatique qui prenne en entrée n'importe quel ensemble D de données et retourne le plus court programme qui génère les données D. Si le langage de description est non universel alors il est possible de calculer le plus court programme générant les données D. Cependant la taille de la description dépend du langage utilisé.

Rasoir d'Occam

Le rasoir d'Occam « Entia non sunt multiplicanda praeter necessitatem », attribué au philosophe Guillaume d'Occam (XIV$^{\text{ème}}$ siècle), est souvent considéré comme une des doctrines fondamentales des sciences modernes. Sa traduction : « Les multiples ne doivent pas être utilisés sans nécessité » veut dire qu'il est inutile de formuler des hypothèses supplémentaires lorsque celles dont on dispose sont suffisantes. Étant donné un ensemble déterminé de conclusions, on interprète ce principe comme la préférence pour l'ensemble le plus simple des hypothèses faites pour aboutir à ces conclusions. Le principe MDL utilise implicitement le rasoir d'Occam car il minimise la complexité du modèle et du cheminement pour aboutir aux données.

Cependant, ce principe est à nuancer (Domingos, 1999) et permet de voir les limites de l'approche MDL. Le rasoir d'Occam est une méthodologie efficace pour obtenir une bonne théorie prédictive mais il ne garantit aucunement la justesse d'un modèle explicatif. En effet, soient la séquence 0000000000 et son modèle le plus simple qui est une répétition de 0. Il est possible que cette séquence soit tirée d'un lancer de pièce or le modèle énoncé ne correspond pas à cette explication. Il est donc important de noter que le principe MDL permet de décrire les données mais ne permet pas de décrire le processus qui les a générées.

> *« MDL is a strategy for inferring models from data not a statement about how the world works »* (Grünwald *et al.*, 2005)

Contournement théorique du surapprentissage

Le surapprentissage consiste à choisir un modèle qui est trop proche des données. Autrement dit, le modèle ne généralise pas assez les données. Ce problème est théoriquement contourné par le principe MDL car si le modèle est trop proche des données alors sa taille se rapproche des données et sa complexité augmente. Au

contraire, le principe de MDL favorise les modèles ni trop complexes ni trop simples à partir desquels on peut retrouver les données d'une manière relativement simple.

Généricité

Les modèles de l'ensemble $\mathbb{M}$ sont structurés selon un langage de description. S'il n'est pas universel, le langage peut être aussi vaste que possible à condition de fournir une façon homogène d'évaluer la taille d'un modèle et la taille du codage des données selon le modèle.

La généricité apporte une certaine garantie quant à l'évaluation des modèles. En effet, si l'expression de modèles extrêmement variés (séquences, clustering, arbres de décision...) est possible dans un tel langage et que la fonction d'évaluation est simple et non orientée pour chacun de ces types de modèles alors le biais introduit par le langage tend à diminuer. Le terme « non orienté » signifie qu'il ne faut pas une méthode de calcul propre à chaque type de modèle mais une méthode de calcul globale.

Le principe MDL est une méthode de sélection de modèle basée sur une relaxation de la complexité de Kolmogorov. La sous-section suivante expose quelques applications qui adaptent le principe MDL à leur problématique.

3.2.4 Applications

Il existe de nombreuses applications utilisant le principe MDL. Le but de cette sous-section n'est pas d'en faire un compte rendu exhaustif mais plutôt d'en présenter certaines qui rendent compte des possibilités offertes par MDL. Les trois premières applications concernent des modèles particuliers : arbre de décision, clustering, et sous-graphes. La quatrième application généralise MDL à des modèles plus génériques.

Arbre de décision

Le principe MDL est utilisable dans de nombreuses applications comme la classification supervisée par arbre de décision (Mehta *et al.*, 1996). Soient un ensemble de données $D = \{d_1, \ldots, d_m\}$ avec $d_i \in \mathbb{X}^n$, un ensemble de classes C et la fonction $f_D : D \to C$ qui associe une classe à une donnée. La classification supervisée consiste à extraire un modèle M qui définisse une fonction de classification $f_M : \mathbb{X}^n \to C$ capable de classer des données inconnues. La classification supervisée par arbre de décision définit la fonction f_M comme un arbre de décision. À chaque nœud de l'arbre, les données sont partitionnées en plusieurs groupes par une sélection liée à une variable $j \in 1 \ldots n$. Le partitionnement est poursuivi jusqu'aux feuilles qui sont chacunes associées à une classe. Un arbre de décision a la capacité de faire apparaître les variables les plus discriminantes à proximité de sa racine et les moins discriminantes à proximité de ses feuilles. Lors de la construction d'un tel arbre à partir de

l'ensemble D, un des problèmes est le surapprentissage. En effet, plus l'arbre est profond, plus il est proche des données et plus il surapprend les données. Il y a un choix à faire dans l'élagage de l'arbre. Une réponse peut être apportée par MDL : choisir l'arbre qui minimise la taille de description des données. La taille d'une description est constituée de la taille de l'arbre augmentée du nombre d'erreurs de classification faites sur un ensemble de données de test.

Expérimentalement, les arbres obtenus par cette stratégie ne sont pas aussi prédictifs (leur erreur de généralisation sur des données inconnues est plus importante) que des arbres élagués par d'autres techniques telles que la validation croisée. Cependant, ils ont l'avantage d'être plus petits ce qui rend leur analyse plus simple.

Clustering

Le clustering (regroupement en français) correspond à une classification non supervisée : les données ne sont pas classées a priori i.e l'ensemble C et la fonction $f_D : D \to C$ ne sont pas définis a priori. Le clustering a pour but de regrouper un ensemble de données en différents paquets homogènes, en ce sens que les données de chaque sous-ensemble partagent des caractéristiques communes. Dans le but de définir la proximité des données, il est nécessaire de définir une distance entre elles. En utilisant la complexité de Kolmogorov, il existe une mesure de dissimilarité (une notion proche de la distance) basée sur la compression de données :

$$d(x, y) = \frac{K(x|y) + K(y|x)}{K(xy)} \tag{3.16}$$

où x et y sont deux données et xy correspond à leur concaténation. Si $x = y$ alors $K(x|y = x) = K(y|x = y) = 0$ et ainsi $d(x, y) = 0$. Si x et y sont complètement différentes alors $K(xy) \backsimeq K(x) + K(y)$, $K(x|y) \backsimeq K(x)$, $K(y|x) \backsimeq K(y)$ et ainsi $d(x, y) = 1$. L'incalculabilité de $K(x)$ implique la relaxation d'une telle distance pour son utilisation dans le cadre du clustering de séries temporelles utilisant MDL (Keogh *et al.*, 2004). La mesure de dissimilarité CDM, une version relaxée de (3.16), utilise la taille $C_z(x)$ de la compression des données par un algorithme de compression usuel (gzip, bzip2, ...) :

$$CDM(x, y) = \frac{C_z(x) + C_z(y)}{C_z(xy)}$$

Le principal intérêt de cette méthode de calcul est qu'elle n'impose pas la définition de paramètres par l'utilisateur qui biaiserait le clustering. S'il est mal défini, ce biais peut conduire à des résultats qui surestiment la signification de certains attributs des données. De plus, les types de données acceptés sont très nombreux car les algorithmes de compression nécessitent seulement une description textuelle des données. Les expérimentations montrent que cette méthode est capable de regrouper les séries temporelles relativement similaires dans un ensemble de séries temporelles provenant d'applications très variées. Par exemple, les séries cardiaques sont regroupées et les séries provenant de mécanismes industriels aussi. Par contre dans le cas

de classifications très spécifiques, comme différencier deux rythmes cardiaques différents, cette méthode échoue car la compression ne tient pas compte des légères différences entre séries très proches. MDL est un critère générique globalement satisfaisant mais non optimal.

Découverte de sous-structures dans les graphes

L'identification de sous-structures (sous-graphes) intéressantes et régulières dans un graphe est une composante essentielle de la découverte de connaissances sur des données relationnelles (Cook et Holder, 1994). Le système SUBDUE propose de découvrir itérativement les sous-structures qui compressent les données originales. À une étape i correspond une description composée d'un graphe G^i et d'un ensemble E_i de sous-structures. La qualité d'une sous-structure S correspond à la différence de taille entre (G^i, E_i) et (G^{i+1}, E_{i+1}) où G^{i+1} est le graphe G^i dans lequel toutes les occurences de la sous-structure S ont été remplacées par un nœud et $E_{i+1} = E_i \cup \{S\}$. Plus cette différence est importante plus la sous-structure S compresse le graphe. Le graphe obtenu peut être considéré comme une description hiérarchique des régularités trouvées puisqu'une sous-structure peut contenir des nœuds représentant les occurences d'autres sous-structures.

Dans cette méthode, les auteurs ont inclus une méthode pour moduler le calcul de la taille de la description. L'utilisateur module ce coût en fonction de ses objectifs. Ce point de vue est intéressant car il montre qu'il est possible d'introduire un biais pour mieux utiliser MDL.

Identification de régions intéressantes

Des requêtes sur une base de données retournent tous les éléments satisfaisant la requête. Un élément est un ensemble d'associations attribut-valeur. Dans le cadre d'une analyse de données, même si une réponse normale à une requête peut inclure de nombreux éléments, il peut être intéressant de retourner une description de ces éléments, par exemple sous la forme d'intervalles dans le cas d'attributs numériques ou bien de concepts dans le cas d'attributs catégoriques hiérarchisés. Ainsi au lieu de retourner les éléments $D = \{(a = 1, b = true), (a = 1, b = false), (a = 2, b = true), (a = 2, b = false)\}$ il suffit de décrire D par $D' = \{1 \leq a \leq 2, b \in boolean\}$. Cependant la recherche d'une telle structure peut ne pas aboutir à une description intéressante si la réponse n'est pas complète. Par exemple en enlevant un des éléments de D, on ne peut plus le décrire en utilisant D'. Or, il peut tout de même être intéressant de retourner D' à l'utilisateur et de lui indiquer qu'un des éléments n'est pas présent (Lakshmanan *et al.*, 2002). Dans ce cadre, on voit que le langage de description doit être suffisamment bien étudié pour répondre aux demandes de l'utilisateur.

La quantité d'information d'un objet calculée par la complexité de Kolmogorov repose sur l'existence d'un langage de description universel. Son application, sous la forme du principe MDL relaxe cette condition dans le but de définir une méthode calculable. En contrepartie, le langage utilisé biaise le calcul de la description. Dès lors, la définition du langage doit être la plus générique possible de manière à ne pas favoriser certaines structures de modèle. Le principe MDL est utilisé pour la sélection de modèles. L'évaluation de modèles par le principe MDL permet l'introduction de biais par l'utilisateur en modifiant le calcul de la taille des descriptions.

3.3 Exploration de journaux d'alarmes

Les experts en sécurité des réseaux se préoccupent de l'analyse des journaux d'alarmes. En effet, c'est l'un des seuls moyens d'obtenir des informations sur l'état et les utilisations malveillantes (les attaques) des réseaux. Les alarmes sont générées au moyen de sondes placées sur les équipements réseau et leur regroupement en journaux est présenté à l'utilisateur. Avec les connaissances acquises sur les journaux, l'utilisateur agit sur le réseau pour répondre aux évolutions du matériel surveillé. La figure 3.17 illustre ce processus.

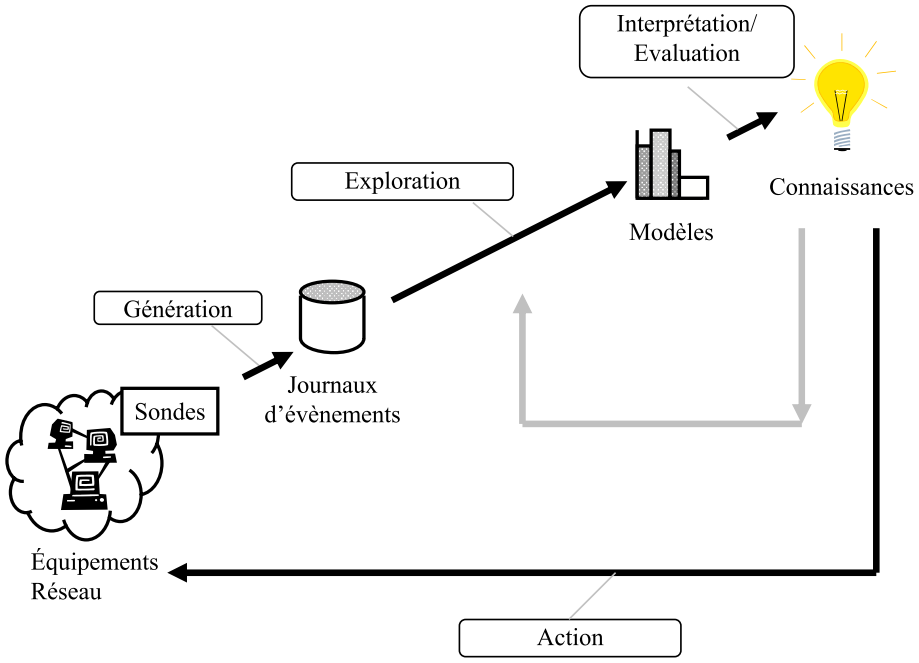


FIG. 3.17: Processus d'exploration et d'action sur les réseaux adapté du processus de fouille de données (figure 3.1 page 67).

Ce domaine a été la motivation de notre travail. Les experts disposent souvent

de peu d'informations sur les phénomènes qui génèrent les journaux d'alarmes qui contiennent une quantité importante d'alarmes.

Même en se concentrant seulement sur les alarmes générées par des équipements réseau, l'hétérogénéité des données explorées et la variété des problématiques considérées débouchent sur des techniques de fouille de données très variées. Ceci est d'autant plus vrai que le principal objectif est souvent de détecter des attaques particulières et ceci de manière très efficace. Or, la contrepartie de l'efficacité est la spécificité des méthodes. Ainsi, on dispose de très nombreuses techniques extrêmement spécifiques.

La spécificité de certaines méthodes implique que l'utilisateur doit posséder des connaissances très pointues. En effet, il n'est pas question d'utiliser un outil spécifique d'exploration d'alarmes sans savoir ce que l'on recherche, visualise, analyse... C'est pourquoi ces méthodes n'entrent pas dans le contexte de cette thèse, elles ne sont donc pas détaillées ici.

L'analyse des méthodes proposées est divisée en deux sous-sections : les caractéristiques des données traitées et les objectifs et moyens des méthodes d'exploration.

3.3.1 Alarmes traitées

Les équipements réseau font circuler des communications à travers les réseaux. Une communication est composée d'un ensemble de *paquets*. En vue d'analyser certains comportements, des *sondes* spécialisées extraient des informations sur les paquets qui transitent pour générer des alarmes. Ces sondes sont logicielles pour les petits débits et matérielles pour les débits plus importants. À titre d'exemple, une sonde Netflow résume les *flux*[4] circulant sur un routeur dont les débits peuvent être de plusieurs térabits par seconde (soit plusieurs milliards de paquets par secondes).

Les structures des alarmes explorées peuvent être regroupées en deux catégories : leur généricité à couvrir la plupart des sondes et l'équipement réseau à protéger.

Plus ou moins de généricité

Une alarme réseau est liée à un niveau d'information. Cela est dû à deux phénomènes : l'organisation sous forme de couches de l'information dans les paquets circulant dans les réseaux et le traitement plus ou moins élaboré des sondes pour générer les alarmes. En extrayant des informations à partir de couches des paquets réseau et en agrégeant ces informations, une sonde génère des alarmes réseau comme l'illustre la figure 3.18 page suivante. Des méthodes tentent de généraliser les alarmes à des structures extrêmement simples ou bien en proposant un format unique de représentation. D'autres méthodes se focalisant sur des applications particulières proposent d'analyser des alarmes ayant des structures spécifiques.

Des méthodes s'intéressent, en vue d'être le plus générique possible, à des alarmes très simples. Une alarme réseau est un triplet composé d'une date d'observation,

[4]Un flux est un ensemble de paquets ayant des caractéristiques identiques en un point d'observation et qui circulent sur les réseaux

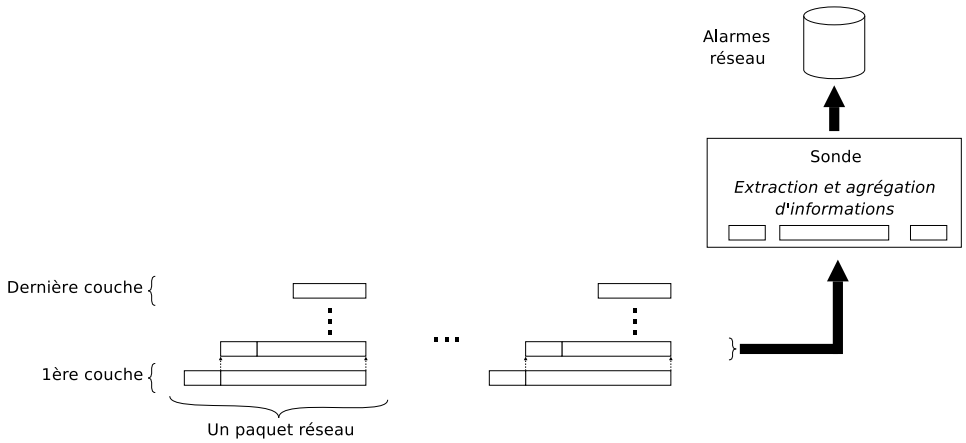


FIG. 3.18: Construction d'alarmes réseau à partir de paquets réseau.

d'un point d'observation et d'un type (Livnat *et al.*, 2005; Klemettinen *et al.*, 1999; Julisch et Dacier, 2002; Lee *et al.*, 1999a; Lee *et al.*, 1999b). Le point d'observation permet de différencier des alarmes provenant de plusieurs sondes. Le type est dépendant de l'application ou du service surveillé par la sonde (Zanero et Savaresi, 2004).

De manière à standardiser la représentation et le traitement des alarmes, l'IDMEF - Intrusion Detection Message Exchange Format (Debar *et al.*, 2007) - a été introduit. Il décrit une alarme comme un arbre dont la structure peut évoluer en fonction des alarmes à traiter. Concrètement, ce format en XML décrit les champs que toute alarme possède (date, source de l'attaque, cible de l'attaque, sévérité,...). Plusieurs méthodes traitent des alarmes dans ce format (Morin et Debar, 2003; Cuppens, 2001; Cuppens et Miège, 2002; Qin et Lee, 2003; Qin et Lee, 2004) et permettent notamment d'élaborer des modèles à partir de sondes de différents types (Valdes et Skinner, 2001).

Beaucoup de méthodes s'intéressent à des alarmes particulières, générées par des applications de surveillance des réseaux. Par exemple, SNORT (Roesch, 1999) génère des alarmes à partir d'une base de données de signatures d'attaque. Si des paquets analysés correspondent à la signature d'une attaque, SNORT émet une alarme. Il existe d'autres applications qui surveillent, par exemple, le débit des communications (Yin *et al.*, 2004; Lakkaraju *et al.*, 2004) à partir d'alarmes Netflow. Ces applications sont très largement répandues pour surveiller les réseaux. Cependant, la multitude des alarmes qu'elles génèrent rend la surveillance inopérante car l'utilisateur est submergé d'informations (Koike et Ohno, 2004). En effet, le but étant de ne rater aucune attaque, ces systèmes génèrent une alarme à la moindre anomalie. Ainsi, beaucoup d'alarmes sont des faux positifs car elles correspondent à des comportements normaux.

Équipement protégé

La surveillance étant la principale préoccupation de l'exploration d'alarmes réseau, les alarmes sont naturellement structurées de façon à surveiller un équipement particulier. La plupart des méthodes proposent de surveiller un réseau contre l'extérieur. D'autres surveillent une application particulière (Teoh *et al.*, 2004) sur internet sans s'intéresser à une cible physique mais ces cas sont rares. La protection d'un réseau inclut le traitement des données structurées sous la forme d'un graphe où les nœuds sont les cibles des attaques ou bien les sources des attaques. La prise en compte de cette structure, très riche, apporte une complexité que peu de méthodes (Lakkaraju *et al.*, 2004; Ball *et al.*, 2004; Cormode et Muthukrishnan, 2005) traitent de façon spécifique.

3.3.2 Moyens et objectifs

Le but de toutes les méthodes inventoriées ici est de faciliter l'extraction de connaissances à partir d'alarmes pour un utilisateur. La généralisation des alarmes est faite soit en appliquant des techniques de fouille de données soit en utilisant la vision humaine.

Les modèles extraits par la fouille de données d'alarmes ont deux finalités : être interprétés par un utilisateur ou prédire des attaques dans de nouveaux journaux. Quand ils sont destinés à un utilisateur (Julisch et Dacier, 2002; Teoh *et al.*, 2004; Klemettinen *et al.*, 1999; Cormode et Muthukrishnan, 2005), ils doivent être compréhensibles et des efforts sont faits pour réduire leur nombre. Ils peuvent représenter des scénarios d'attaques (Lee *et al.*, 1999a; Lee *et al.*, 1999b; Qin et Lee, 2003; Qin et Lee, 2004) notamment sous la forme de chroniques (Morin et Debar, 2003). Une autre utilisation (Julisch, 2003) de la fouille de données consiste à généraliser les alarmes par des modèles ne généralisant que les comportements normaux. En conservant les alarmes non couvertes par ces modèles, l'utilisateur a accès aux alarmes potentiellement intéressantes car elles sont considérées comme inhabituelles.

La vision humaine a cette faculté de pouvoir généraliser très rapidement les représentations graphiques générées à partir de données. Cette faculté est employée de façon à explorer les alarmes réseau. Ainsi, l'utilisateur acquiert des connaissances issues du processus d'extraction qu'il dirige. De plus, la vision humaine génère des connaissances dont la structure est plus variée qu'une méthode de fouille de données particulière. Cependant, l'interface entre les alarmes et l'utilisateur est constituée d'un outil de visualisation (Koike et Ohno, 2004; Lakkaraju *et al.*, 2004; Ball *et al.*, 2004; Yin *et al.*, 2004; Livnat *et al.*, 2005) qui biaise indéniablement l'apprentissage de l'utilisateur. Le paradigme de visualisation doit être étudié pour ne pas orienter l'utilisateur vers des connaissances qui peuvent s'avérer fausses. Enfin, la contrepartie de la souplesse de l'exploration visuelle est évidemment le temps nécessaire à l'utilisateur pour mener une telle exploration.

Enfin la majorité des techniques disponibles proposent d'extraire des modèles très efficaces en vue de repérer des anomalies dans de nouveaux journaux d'alarmes

réseau. Si les modèles sont facilement compréhensibles par un utilisateur alors la technique (Zanero et Savaresi, 2004; Cuppens, 2001; Dain et Cunningham, 2001a; Dain et Cunningham, 2001b; Valdes et Skinner, 2001) permet d'extraire des connaissances. Mais si les modèles sont incompréhensibles alors il n'y a aucune extraction de connaissances possible. L'utilisateur est seulement destiné à recevoir des alarmes filtrées.

L'exploration de journaux d'alarmes est contrainte par la quantité de données à explorer. Ce phénomène tend à croître au vu de l'augmentation des débits réseau. L'efficacité des méthodes est très souvent synonyme d'une spécialisation à des structures d'alarmes particulières. La multiplication des types de sondes, de services réseau, et d'équipements surveillés incite à une généralisation de la structure des alarmes. L'extraction de connaissances a recours à la visualisation et à des techniques de fouille de données. Même si l'objectif de certaines méthodes n'est pas d'aider à explorer des alarmes mais plutôt à protéger un réseau, les modèles générés peuvent tout de même être présentés à un utilisateur.

Conclusion

Résumer l'information est la base de la fouille de données. Les modèles synthétisent l'information contenue dans les données. Les techniques de fouille de données utilisent de nombreux mécanismes afin de produire des modèles intéressants.

Tout d'abord, les connaissances de l'utilisateur souhaitant accéder aux résumés des données peuvent être exploitées. Il dirige l'extraction des modèles en définissant un processus précis ou bien en explorant visuellement les données et en construisant itérativement un processus d'extraction. Si l'utilisateur a un objectif précis, c'est un avantage. En effet, le modèle étant destiné à l'utilisateur, si celui-ci connaît la forme précise de ce qu'il veut obtenir, il est beaucoup plus aisé de le rechercher. Il est important de noter que cela biaise très fortement l'entreprise de découverte de connaissances. Néanmoins, valider les hypothèses de l'utilisateur est une tâche suffisamment importante.

Les statistiques sont mises à contribution en élaborant des techniques de sélection de modèle qui estiment la qualité d'un modèle à résumer correctement les données. Cependant, ces techniques sont toujours orientées selon un certain objectif qu'il est important de maîtriser. De plus des hypothèses sur les données doivent être posées avant d'utiliser de telles techniques.

Enfin, la fouille de données fournit des structures de modèle et des implémentations très efficaces pour extraire les modèles. L'utilisation de ces implémentations d'un domaine à un autre peut se révéler judicieuse. Que ce soient des visualisations selon un certain angle ou bien des structures de modèle particulier, il existe une quantité phénoménale de techniques différentes.

En s'aidant de ces deux derniers mécanismes, qui ne nécessitent pas de connaissances particulières de l'utilisateur, accompagnés du mécanisme de spécification pré-

senté aux chapitres précédents, comment peut-on imaginer un système de fouille de données capable de résumer de l'information ? Une réponse motivée par l'analyse d'alarmes réseau est donnée au chapitre suivant.

CHAPITRE 4

Les schémas pour la fouille de données

FACE à une masse de données et *sans information* sur les connaissances qu'il peut espérer en extraire, un utilisateur ne peut envisager la définition d'un processus d'extraction de connaissances précis. Malgré le fait que de nombreux opérateurs[1]de fouille de données existent, il lui est difficile de tous les exécuter sur les données. D'une part cela nécessite souvent d'ajuster les opérateurs aux données et d'autre part cela nécessite d'évaluer les modèles obtenus qui peuvent être très nombreux.

Nous proposons de concevoir une méthode automatique d'exécution d'opérateurs. Cette méthode est rendue possible par l'utilisation des schémas définis dans le chapitre 2. À partir de la spécification des opérateurs par des schémas, il est possible d'envisager l'enchaînement automatique de l'exécution de ces opérateurs sur les données. L'utilisateur doit bien entendu être en mesure de fournir une spécification des données. Les modèles obtenus, par l'exécution de ces opérateurs, sont ensuite évalués par une méthode basée sur le principe MDL.

La première section décrit l'architecture générale sur laquelle repose notre approche. La seconde section présente la construction des schémas opérationnels à partir des schémas abstraits des opérateurs et du schéma spécifiant les données. La troisième section montre la mise en œuvre de MDL pour évaluer les modèles. Cette méthode repose sur la spécification de la relation de couverture. Enfin, des expérimentations apportent des réponses quant à la pertinence d'une telle méthode.

[1]Le terme « opérateur » est introduit en 3.1.1 page 66 et défini en 4.2 page 95

4.1 Architecture

L'objectif de l'architecture à mettre en place est de pouvoir extraire et évaluer automatiquement des descriptions d'un ensemble de données. Plusieurs hypothèses doivent être formulées auparavant.

Hypothèses

Nous supposons que l'utilisateur possède peu de connaissances sur les données mais suffisamment pour qu'il soit capable de spécifier leur représentation élémentaire sous la forme d'un schéma.

L'approche repose sur une hypothèse majeure : l'existence de nombreux opérateurs spécifiés par des schémas. Le fait qu'il existe un grand nombre d'opérateurs et qu'ils génèrent des modèles sous diverses formes (clustering, arbres de décisions, séquences,...) est un fait communément admis. Ce qui l'est moins est l'existence de spécifications de ces opérateurs sous la forme de schémas. Il faut donc compter sur un enrichissement progressif de la base des opérateurs par les utilisateurs d'un tel système. La tâche de spécification revient à la personne qui intègre les opérateurs dans le système de fouille de données décrit ici. Il est important de noter que ce n'est pas forcément l'utilisateur qui spécifie les opérateurs et qui recherche des connaissances dans les données.

La figure 3.1 page 67 présente le processus d'extraction de connaissances comme une succession d'étapes. Nous supposons que toute étape correspond à l'application d'un opérateur. L'enchaînement de ces opérateurs constitue un processus et n'est pas abordé dans notre approche.

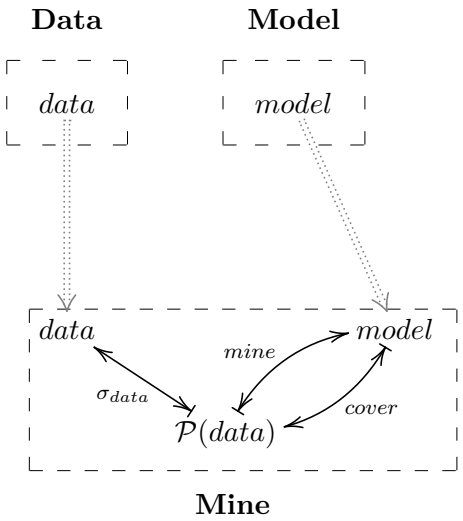
Présentation

De manière à appliquer automatiquement des opérateurs sur des données, il faut que le langage de spécification des opérateurs soit compatible avec le langage de spécification des données. Or, les schémas fournissent un langage qui supporte de telles spécifications. En vue d'abstraire la notion d'opérateurs, nous introduisons le schéma **Mine** (schéma 4.1). Ce schéma est utilisé dans l'architecture (schéma 4.3 page ci-contre) de notre système pour structurer les schémas d'opérateurs.

L'architecture repose sur un ensemble d'opérateurs. Tous ont la même structure décrite par le schéma **Mine**. Les données sont spécifiées par l'utilisateur dans un schéma particulier. L'unification des schémas d'opérateurs et du schéma de données est réalisée automatiquement. Ensuite, l'exécution des opérateurs est lancée et les modèles sont extraits. Enfin, les modèles sont évalués par une méthode décrite dans la section 4.3 page 105.

Schéma 4.1 (Schéma de la fouille de données)
Le schéma **Mine** sert de base à *tous* les opérateurs de fouille de données. Deux schémas sont nécessaires pour construire le schéma de fouille de données : le schéma

représentant les données et le schéma représentant les modèles. Ils sont chacun composés d'un nœud unique, respectivement *data* et *model*.



Tous les opérateurs transforment des données en des modèles.

La relation $mine : \mathcal{P}(data) \nrightarrow model$ abstrait le concept d'opérateur. L'arc *mine* est spécifié comme une relation et non comme une fonction de manière à prendre en compte les algorithmes qui génèrent plusieurs modèles. La relation de couverture $cover : model \nrightarrow \mathcal{P}(data)$ est introduite dans le but d'évaluer les modèles. Comme indiquée dans la sous-section 3.1.2 page 69, un modèle est lié à plusieurs sous-ensembles de données par la relation de couverture[2]. ▪

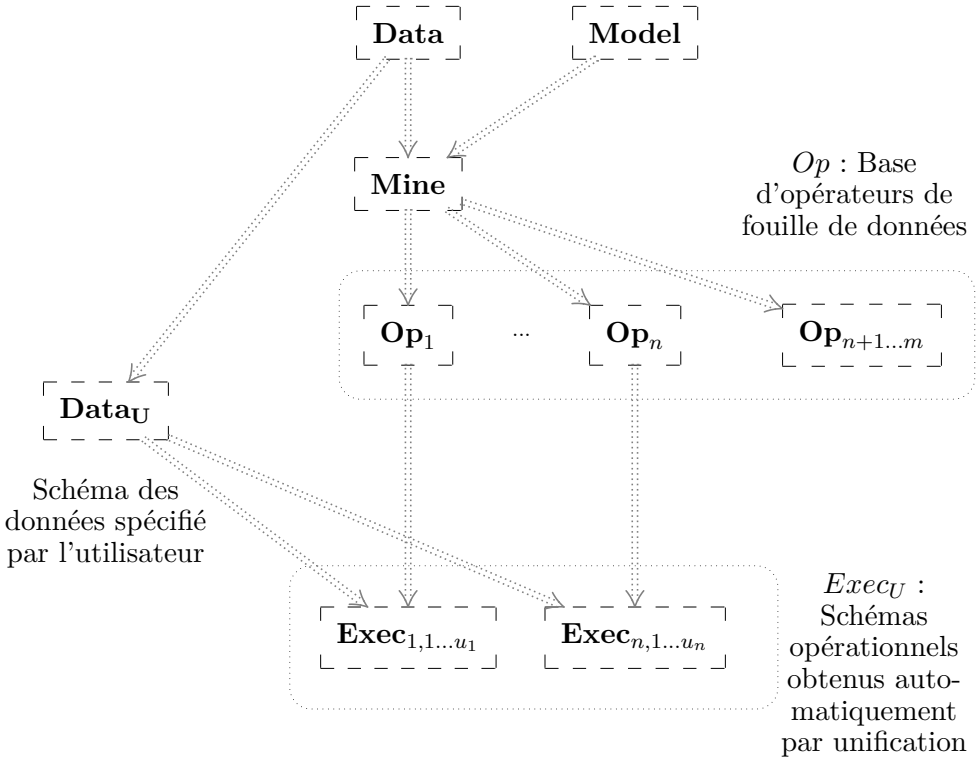
Définition 4.2 (Opérateur de fouille de données)
Un opérateur de fouille de données correspond à une spécialisation du schéma **Mine** pour aboutir à une implémentation de la structure des données, de la structure du modèle, de l'algorithme de fouille de données et de la relation de couverture. Cette spécialisation passe par des morphismes d'esquisse relationnelle et par l'implémentation de la relation $mine : \mathcal{P}(data) \nrightarrow model$. L'exécution d'un opérateur consiste à appliquer la relation *mine* sur des données. Cela correspond à extraire des modèles à partir des données. ▪

Schéma 4.3 (Architecture du système de fouille de données)
Les schémas **Data**, **Model** et **Mine** ont été introduits dans le schéma 4.1 page ci-contre. Le schéma $\mathbf{Data_U}$ représente la spécification des données à abstraire écrite par l'utilisateur. L'ensemble *Op* des schémas $\mathbf{Op}_i$ pour $i = 1 \dots m$ constitue la base d'opérateurs. Un morphisme partant de **Mine** vers chacun des schémas $\mathbf{Op}_i$ est introduit afin de structurer ces schémas. Un morphisme partant de **Data** vers le schéma $\mathbf{Data_U}$ est introduit dans le but de structurer le schéma des données.

[2]L'introduction du chapitre 2 page 39 définit *mine* et *cover* d'une manière simplifiée pour faciliter la compréhension du chapitre 2.

$Exec_U$ représente l'ensemble des schémas opérationnels des opérateurs unifiés avec le schéma des données.

Certains schémas d'opérateurs ($\mathbf{Op}_{n+1\ldots m}$) ne sont pas unifiables au schéma des données. De plus, il peut exister plusieurs unifications entre un opérateur et les données (pour le schéma $\mathbf{Op}_i$ on note u_i le nombre d'unifications avec le schéma $\mathbf{Op}_i$).



Le processus de découverte de descriptions dans le cadre de l'architecture présentée se déroule de la manière suivante. Tout d'abord, l'utilisateur dispose d'un ensemble de données D dont il cherche à extraire des connaissances. La première étape consiste, pour l'utilisateur, à spécifier les données D dans le schéma $\mathbf{Data_U}$. C'est la seule étape manuelle, le reste est exécuté automatiquement. Le calcul des schémas de la base d'opérateurs $Exec_U$ par unification des schémas de la base Op avec le schéma $\mathbf{Data_U}$ est effectué. Puis, les opérations *mine* des schémas opérationnels $Exec_U$ sont exécutées sur les données D pour extraire un ensemble de modèles $\mathbb{M}$. Enfin, les modèles de l'ensemble $\mathbb{M}$ sont évalués relativement aux données et à la relation de couverture. Finalement, l'utilisateur interprète les modèles les mieux évalués pour en extraire des connaissances.

Nous proposons de spécifier des opérateurs existants puis de les unifier automatiquement avec les données pour les exécuter et générer des modèles. Cette proposition est décrite sous la forme d'une architecture composée de schémas et de morphismes d'esquisse relationnelle. L'unification aboutit à la construction des schémas opérationnels que la section suivante présente.

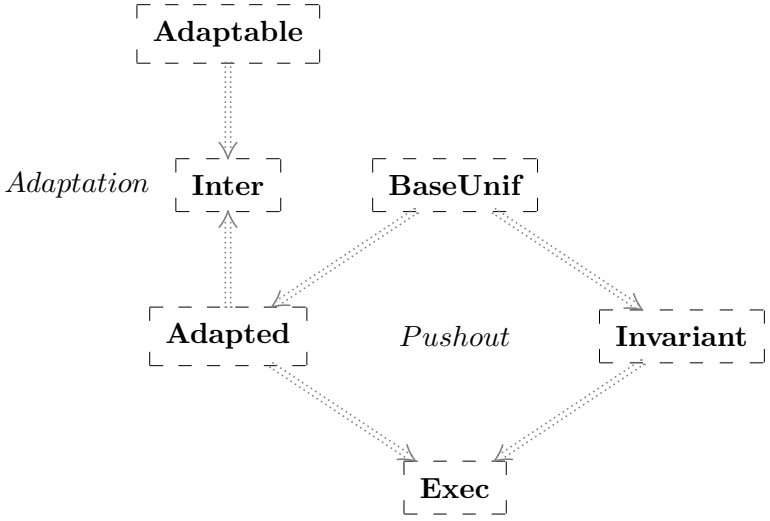
4.2 Construction des schémas opérationnels

La construction des schémas opérationnels consiste à unifier un des schémas spécifiant les opérateurs avec le schéma spécifiant les données. Deux types d'unification sont possibles. Soit conserver la structure des données et adapter l'opérateur, soit conserver la structure de l'opérateur et adapter la structure des données. C'est pourquoi l'unification nécessite le choix de l'invariance d'un des deux schémas (de données ou d'opérateur). La structure du *schéma invariant* est complètement incluse dans le schéma opérationnel produit alors que l'autre schéma, appelé le *schéma adaptable* est modifié pour l'unification. L'architecture 4.4 page suivante décrit les morphismes d'esquisse relationnelle mis en jeu dans l'unification.

Le choix du sens de l'unification est uniquement lié au choix de l'évaluation des modèles extraits. L'évaluation est détaillée dans la partie 4.3.1 page 108. Si le schéma invariant est celui des données, on parle alors d'une *évaluation globale* des modèles. En effet, comme la structure des données est inchangée, le modèle couvre toutes les propriétés des données. De façon opposée, si le schéma invariant est celui de l'opérateur, on parle alors d'une *évaluation locale* des modèles. Dans ce cas, la structure des données peut être élaguée et les modèles ne couvrent qu'une partie des données à abstraire.

Schéma 4.4 (Architecture de l'unification)
On considère deux schémas : un schéma $\mathbf{Op}_s$ représentant un opérateur de la base d'opérateurs et un schéma $\mathbf{Data_U}$ défini par l'utilisateur et représentant la structure des données. À partir de ces deux schémas, deux unifications sont possibles : l'évaluation locale dans laquelle $\mathbf{Adaptable} = \mathbf{Data_U}$ et $\mathbf{Invariant} = \mathbf{Op}_s$ et l'évaluation

globale pour laquelle $\mathbf{Adaptable} = \mathbf{Op}_s$ et $\mathbf{Invariant} = \mathbf{Data}_\mathbf{U}$.



L'unification comporte deux phases. La première phase consiste à adapter le schéma adaptable en analysant le schéma invariant. Cette phase produit le *schéma adapté* et le schéma de base de la phase suivante. La seconde phase consiste à réaliser un pushout (définition 1.43 page 28) entre le schéma adapté obtenu et le schéma invariant en respectant le schéma de base obtenu. Il est important de noter que pour un schéma adaptable et un schéma invariant il existe plusieurs schémas adaptés et que pour un schéma adapté et un schéma invariant il n'existe qu'un seul pushout.

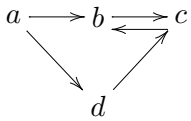
Tout d'abord, nous présentons une construction basée sur un exemple afin d'éclaircir les notions d'adaptation et de pushout. Ensuite, la construction du schéma adapté est détaillée.

4.2.1 Illustration

On s'intéresse à la fouille de graphes où il s'agit d'extraire des graphes généralisant un graphe donné. L'illustration aborde l'unification d'un opérateur de généralisation de graphes avec des alarmes réseau. Dans un premier temps, cet opérateur est présenté. Dans un second temps, l'unification de sa spécification avec le schéma des alarmes réseau (2.44 page 56) est présentée. La première adaptation correspond à une adaptation du schéma de généralisation de graphes au schéma d'alarmes réseau. La seconde adaptation consiste à adapter le schéma des alarmes réseau au schéma de généralisation de graphe.

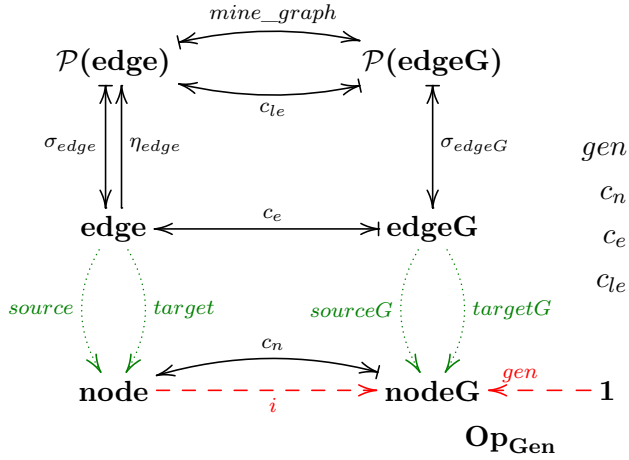
Dans le but de généraliser un graphe, on propose de transformer l'ensemble d'arcs le représentant par un ensemble d'arcs le généralisant. Les sommets de ces arcs peuvent être le nœud $*$ qui remplace n'importe quel nœud. Par exemple le

graphe G :



est généralisé par l'ensemble d'arcs $l_e = \{a \to *, * \to b, * \to c\}$ qui couvre complètement le graphe G. Nous proposons un algorithme qui extrait les k nœuds de plus haut demi-degré extérieur (le nombre d'arcs qui partent du nœud) ou de plus haut demi-degré intérieur (le nombre d'arcs qui arrivent au nœud) et qui définit un graphe généralisé composé uniquement de ces nœuds et du nœud $*$. Dans l'exemple précédent l_e est extrait à partir de G avec $k = 3$. Cet algorithme est spécifié sous la forme d'un opérateur de fouille de données dans le schéma $\mathbf{Op_{Gen}}$ suivant.

Schéma 4.5 (Schéma de l'opérateur de généralisation de graphes)
Le nœud *node* représente le type des nœuds du graphe, le nœud *edge* représente le type des arcs (qui sont des couples de nœuds *node*). Un graphe est un élément de type $\mathcal{P}(edge)$. Les nœuds du graphe généralisé sont composés d'éléments de type *node* et de l'élément $*$, ce qui est traduit par le cocône de sommet *nodeG*. De la même façon qu'un graphe, un graphe généralisé de type $\mathcal{P}(edgeG)$ est un ensemble d'arcs généralisés *edgeG*. L'opérateur est symbolisé par la relation *mine_graph* : $\mathcal{P}(edge) \nmapsto \mathcal{P}(edgeG)$.



$$gen = \{\mathbf{1} \to *\}$$
$$c_n = \langle \Sigma_{node}; id_{node} \rangle$$
$$c_e = \langle c_n \circ sourceG, c_n \circ targetG \rangle$$
$$c_{le} = \eta_{edge} \circ c_e \circ \sigma_{edgeG}$$

La relation de couverture c_{le} est la partie la plus complexe du schéma. La relation c_n : $nodeG \nmapsto node$ représente la relation de couverture d'un nœud généralisant. De la même façon c_e : $edgeG \nmapsto edge$ représente la relation de couverture d'un arc généralisant. Pour l'ensemble des arcs généralisants d'un graphe généralisant, la relation c_{le} : $\mathcal{P}(edgeG) \nmapsto \mathcal{P}(edge)$ représente la relation de couverture composée des relations suivantes :

$\sigma_{edgeG} : \mathcal{P}(edgeG) \nmapsto edgeG$	extrait les arcs généralisés un par un,
$c_e : edgeG \nmapsto edge$	construit la couverture d'un arc généralisé,
$\eta_{edge} : edge \to \mathcal{P}(edge)$	construit un ensemble composé d'un arc.

.

Adaptation de l'opérateur aux données

L'adaptation du schéma $\mathbf{Op_{Gen}}$ au schéma $\mathbf{Data_A}$ (2.44 page 56) consiste intuitivement à ajouter les propriétés manquantes aux arcs (*edge*) du schéma $\mathbf{Op_{Gen}}$. Ces propriétés proviennent des alarmes du schéma $\mathbf{Data_A}$. La figure 4.6 page ci-contre illustre cet ajout. Le schéma adaptable est le schéma $\mathbf{Op_{Gen}}$, le schéma adapté est le schéma $\mathbf{Op''_{Gen}}$. Le schéma intermédiaire $\mathbf{Op'_{Gen}}$ sert intuitivement à récupérer les constructions de $\mathbf{Op_{Gen}}$ pour $\mathbf{Op''_{Gen}}$.

Notre méthode génère automatiquement de nouvelles constructions (cônes, cocônes, objets des parties) dans $\mathbf{Op'_{Gen}}$ basées sur les constructions de $\mathbf{Op_{Gen}}$. Notre technique parcourt tous les nœuds qui structurent les données dans le schéma invariant $\mathbf{Data_A}$. Le même parcours est effectué dans le schéma adaptable $\mathbf{Op_{Gen}}$. Lorsqu'une différence de structure est rencontrée, une nouvelle construction est générée dans le schéma intermédiaire afin de pallier cette différence. Par exemple, les nœuds $date'$ et $severity'$ sont générés dans le schéma intermédiaire $\mathbf{Op'_{Gen}}$ pour qu'ils puissent s'unifier correctement avec les nœuds *date* et *severity* du schéma invariant. Ces constructions entraînent la conversion des relations du schéma $\mathbf{Op_{Gen}}$. Par exemple, la relation c_{le} est convertie en la relation c'_{le} par l'insertion de la relation d'énumération pour tenir compte des nœuds ajoutés à la base du cône :

$$c_{le} = \eta_{edge} \circ c_e \circ \sigma_{edgeG}$$
$$c'_{le} = \eta_{edge'} \circ \langle \Sigma_{date'} \circ \varnothing_{edgeG}, c_e, \Sigma_{severity'} \circ \varnothing_{edgeG} \rangle \circ \sigma_{edgeG}$$

Le pushout présenté dans la figure 4.7 page 102 est déduit à partir des schémas $\mathbf{Op''_{Gen}}$ et $\mathbf{Data_A}$. Plus précisément, le *schéma de base* $\mathbf{BaseUnif_{Gen}}$ est construit d'après les parcours qui ont permis de construire le schéma $\mathbf{Op''_{Gen}}$. Dans la figure 4.7 page 102, le nœud *data* est envoyé sur le nœud $\mathcal{P}(edge)$ par le morphisme F_A et sur $\mathcal{P}(alarm)$ par le morphisme F_D. Sur la figure, les morphismes F_A et F_D sont déduits à partir des noms du schéma de base, par exemple $F_A(node/actor) = node$ et $F_D(node/actor) = actor$. Le pushout des deux morphismes F_A et F_D permet de créer le schéma opérationnel $\mathbf{Exec_{Gen}}$ qui spécifie l'exécution de l'opérateur de généralisation de graphes sur les alarmes. Il est important de noter que l'unification des schémas $\mathbf{Op_{Gen}}$ et $\mathbf{Data_A}$ n'est pas unique. Par exemple, les relations *source* et *target* auraient pu être inversées.

Adaptation des données à l'opérateur

L'adaptation précédemment proposée permet d'adapter le schéma de l'opérateur à celui des données. Ici, nous présentons l'inverse, le schéma invariant est celui de l'opérateur et le schéma adaptable est celui des données. La figure 4.8 page 103 décrit l'adaptation du schéma $\mathbf{Data_A}$ au schéma $\mathbf{Exec_{Gen}}$. Cela consiste intuitivement à faire disparaître les nœuds *severity* et *date*. Le pushout de cet exemple n'est pas illustré. Il est facilement déduit. Les unifications sont : $\mathcal{P}(link)/\mathcal{P}(edge)$, $link/edge$, $\sigma_{link}/\sigma_{edge}$, $actor/node$, $source/source$ et $target/target$.

Une autre adaptation consiste à inverser les nœuds *source* et *target*.

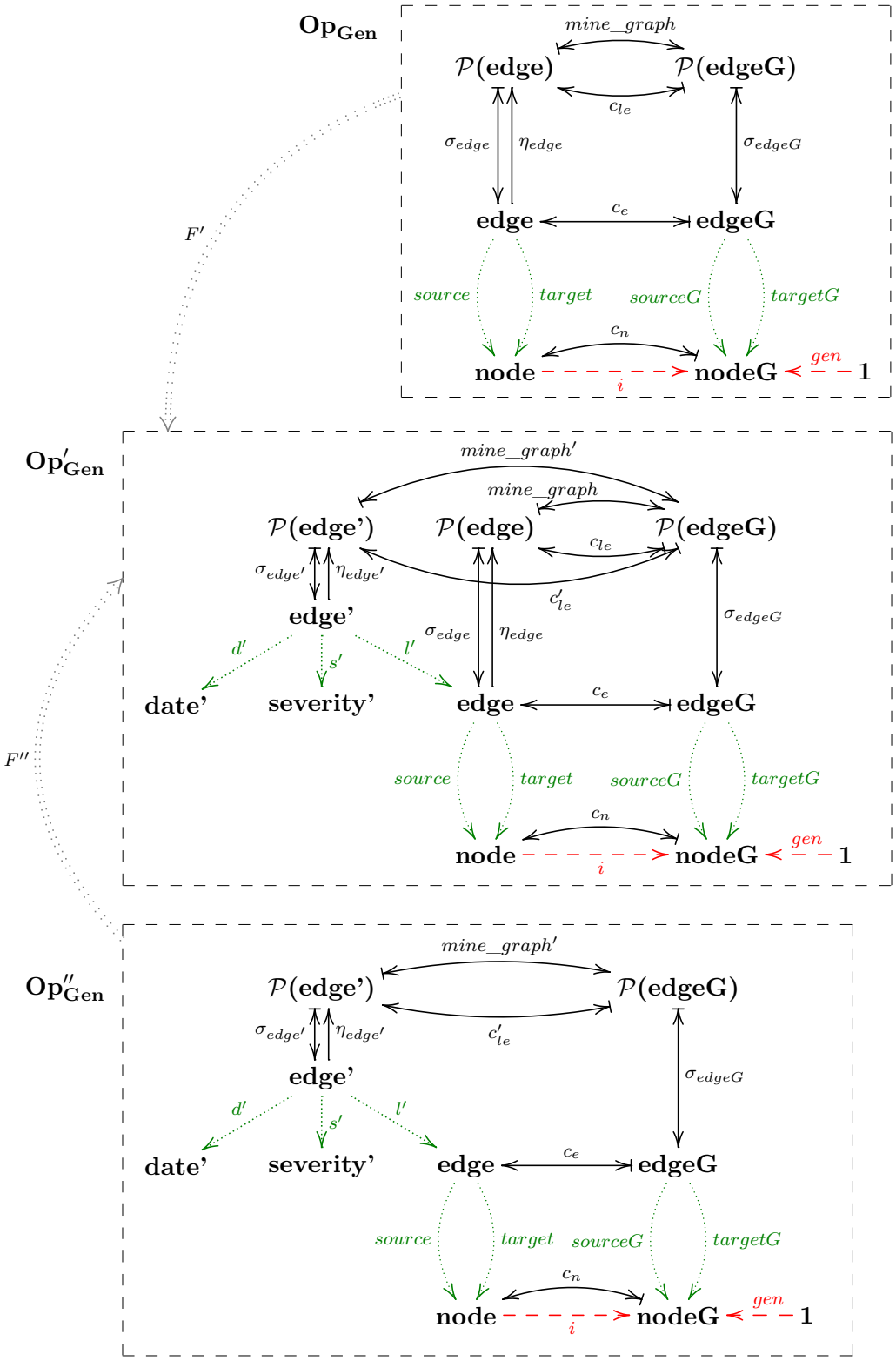


FIG. 4.6: Adaptation du schéma $\mathbf{Op_{Gen}}$ au schéma $\mathbf{Data_A}$.

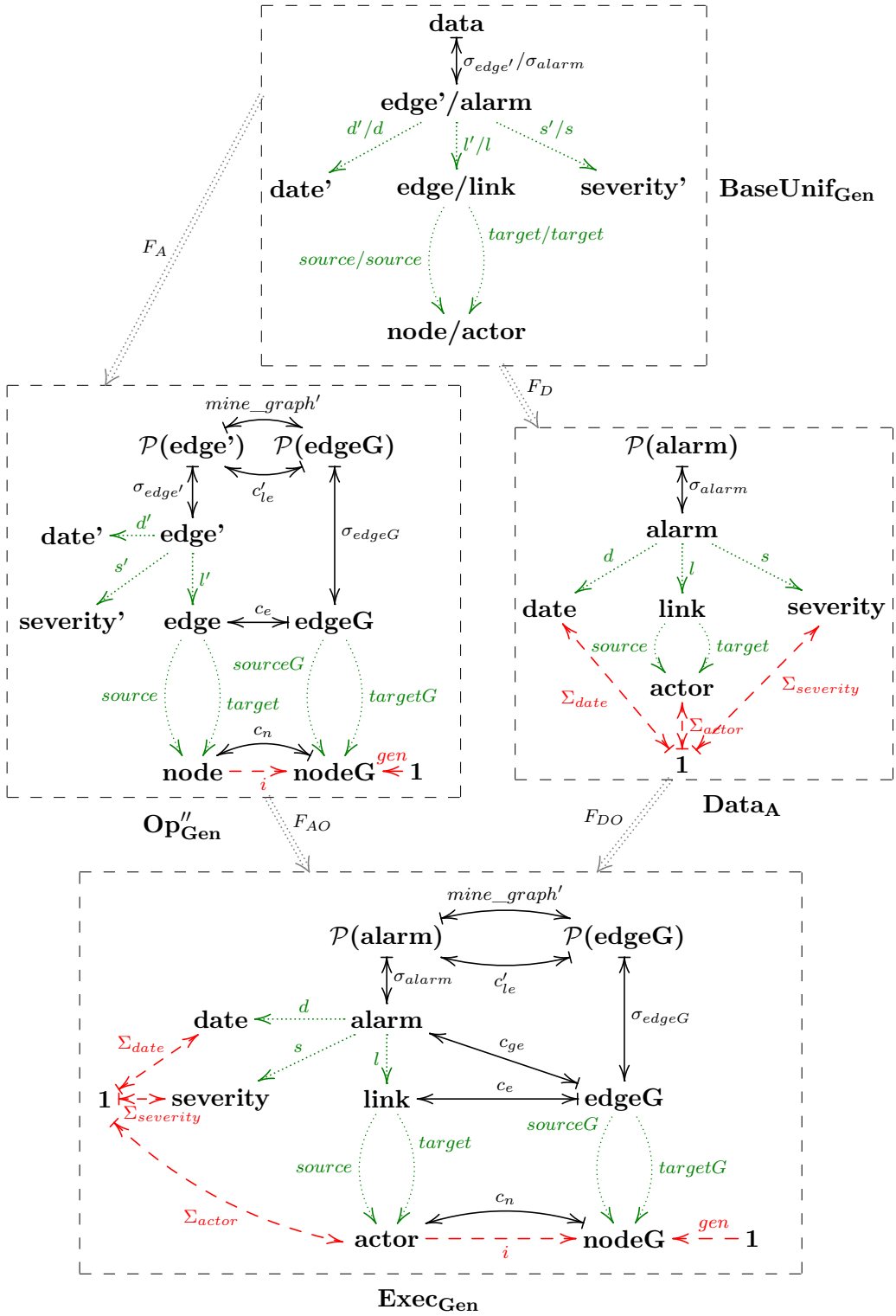


FIG. 4.7: Pushout construisant le schéma opérationnel **Exec$_{\mathbf{Gen}}$** de généralisation de graphes d'alarmes réseau.

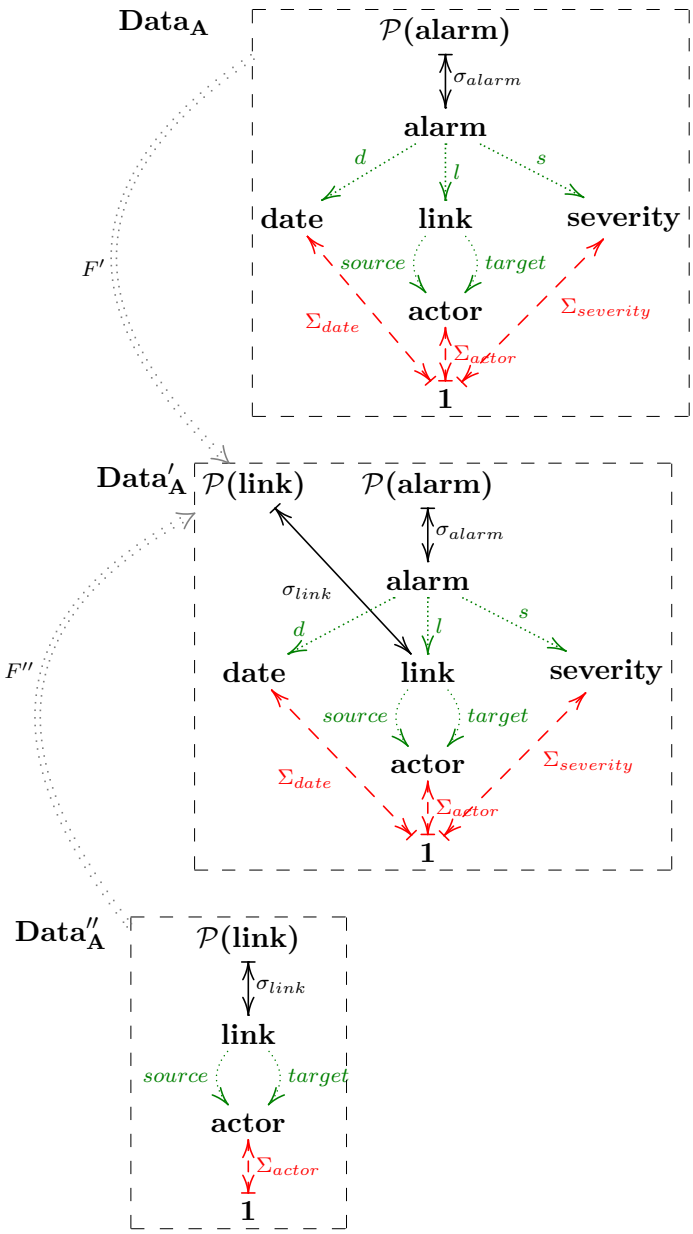


FIG. 4.8: Adaptation du schéma $\mathbf{Data_A}$ au schéma $\mathbf{Op_{Gen}}$.

4.2.2 Unification

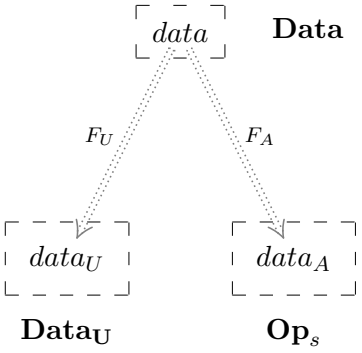
Trois contraintes fortes décrivent la construction de l'unification : l'obtention d'un schéma opérationnel, l'unification des nœuds représentant les données et le non ajout de propriétés aux données.

Obtention d'un schéma opérationnel

Le schéma final doit être opérationnel puisque l'algorithme représenté par la relation $mine : \mathcal{P}(data) \nmapsto model$ doit effectivement être exécuté sur les données à analyser. Plus précisément, toutes les constructions permettant de rendre opérationnelles les constructions issues du schéma **Mine** définies dans les schémas $\mathbf{Data_U}$ et $\mathbf{Op}_s$ doivent figurer dans le schéma opérationnel **Exec**.

Unification des nœuds représentant les données

Le point commun entre tout schéma d'opérateur $\mathbf{Op}_s$ et tout schéma de données $\mathbf{Data_U}$ est le morphisme provenant du schéma **Data** décrit en (4.9).



(4.9)

Tous les « sous-nœuds » (base de cône, base de cocône et objet des parties) de $data_U$ et de $data_A$ doivent être unifiés. Pratiquement, cela n'est pas possible car ces sous-nœuds ont des structures différentes. L'adaptation permet justement de construire de nouvelles structures « recopiées » à partir de structures différentes. Ces recopies sont parfois incomplètes ou au contraire font intervenir de nouveaux nœuds. De plus, elles entrainent la recopie des factorisations et des cofactorisations relationnelles qui doivent être convertie pour prendre en compte les modifications. Cette conversion passe par l'insertion de la relation d'énumération (section 2.4 page 52) dans le cas de l'insertion de nœuds dans la base d'un cône. En effet, cette relation peut être utilisée pour n'importe quel nœud puisque le schéma final **Exec** est opérationnel (définition 2.40 page 53).

Non ajout de propriétés aux données

Les données ne peuvent être « augmentées ». En effet, il n'est pas possible d'ajouter un attribut à des alarmes quand l'utilisateur dispose déjà d'un tas d'alarmes

instanciées. C'est pourquoi lorsque le schéma adaptable est celui des données, notre méthode n'ajoute pas de nœuds à la structure des données.

Nous avons mis en place une méthode permettant de faire correspondre automatiquement les opérateurs avec les données en utilisant les schémas. L'utilisation du pushout et la mise en place de contraintes sur l'unification des schémas a abouti à l'automatisation de l'unification. Afin d'aboutir à un processus réel présentant des modèles à l'utilisateur, il reste à évaluer la qualité des modèles extraits en vue de présenter les meilleurs à l'utilisateur.

4.3 Évaluation générique des modèles

L'exécution automatique d'opérateurs est rendue possible par la manipulation des spécifications de données et d'opérateurs. L'exécution de ces opérateurs génère des modèles. Le nombre de modèles obtenus varie en fonction du nombre d'opérateurs exécutés, de la façon d'adapter les données aux opérateurs et du nombre de modèles extraits par chaque opérateur. Dès lors, le nombre de modèles extraits peut être important. Submerger l'utilisateur par tous ces modèles n'est pas envisageable. Il est important de les évaluer et de ne présenter que ceux qui résument le mieux l'information.

Le principe MDL consiste à sélectionner le modèle qui minimise la taille de la description des données. Pour ce faire, il faut définir un langage de description. Une description est composée d'un modèle et de la description des données sachant le modèle. La mise en œuvre du principe MDL dans le cadre des schémas n'est pas concentrée sur une seule structure de modèle mais sur un ensemble de structures. La généricité de MDL est mise à contribution en vue de comparer des modèles de structures différentes.

La première sous-section décrit, en le décomposant, le calcul de la taille de description des données sachant le modèle en utilisant les schémas. La sous-section suivante détaille comment calculer les éléments d'un nœud et la dernière sous-section explique le calcul des éléments appartenant à l'image d'une relation de couverture pour un modèle donné.

4.3.1 Méthode de calcul du critère d'évaluation

La méthode de calcul du critère d'évaluation est décrite en deux parties. La première partie montre comment tirer parti du pouvoir d'expression des schémas pour décomposer la description des données relativement à un modèle et une relation de couverture. La seconde partie envisage l'évaluation d'un modèle d'une façon locale (en ne se préoccupant que des données couvertes) ou d'une façon globale (en se préoccupant des données couvertes et non couvertes). Ces deux évaluations permettent de faire ressortir différentes caractéristiques des modèles.

Décomposition du critère d'évaluation

Le schéma **Mine** (4.1 page 94) formalise le concept de modèle et de données sous la forme de deux types, respectivement, *model* et $\mathcal{P}(data)$. De plus, il définit la relation de couverture $cover : model \nrightarrow \mathcal{P}(data)$ qui exprime la façon de retrouver des données sachant un modèle. Le schéma **Mine** est abstrait et on a vu qu'un ensemble de schémas opérationnels $Exec_U$ est calculé à partir de pushouts (section 4.2 page 97).

Lors du calcul d'une description, on considère un schéma $\mathbf{Exec}_s \in Exec_U$ muni des nœuds $data'$ et $model'$ et de la relation de couverture $cover' : model' \nrightarrow data'$. Celui-ci est résumé dans la figure 4.10. Dans cette figure, les schémas $\mathbf{Data_U}$, $\mathbf{Op}_s$ et $\mathbf{Exec}_s$ ne sont pas complètement décrits puisque $\mathbf{Op}_s$ est dépendant de la base de schémas *Op* (illustrée dans le schéma 4.3 page 95) et $\mathbf{Data_U}$ est donné par l'utilisateur.

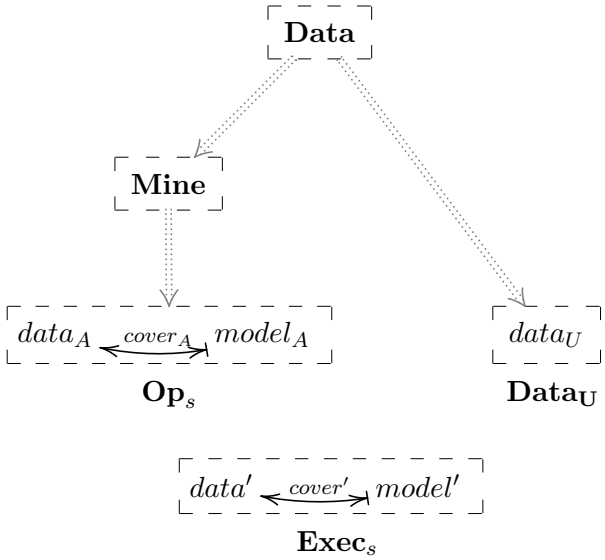


FIG. 4.10: Représentation simplifiée du calcul d'un schéma opérationnel en fouille de données. Un exemple plus complexe est donné dans la figure 4.7 page 102.

En reprenant le principe MDL, et en considérant un modèle $M \in model'$ et des données $D \in \mathcal{P}(data')$, le calcul de la taille $C(D, M) = L(M) + L(D|M)$ de la définition 3.14 page 82 est décomposé à partir du schéma $\mathbf{Exec}_s$. Le modèle M appartient au type (à l'ensemble) $model'$ ce qui est intégré dans la taille de description d'un modèle :

$$L(M) = L(M|model') + L(model') \tag{4.11}$$

où $M|model'$ représente le contenu du modèle M sans la structure exprimée par le nœud $model'$.

La taille $L(D|M)$ de la description des données sachant le modèle est décomposée en introduisant la relation de couverture $cover' : model' \nrightarrow \mathcal{P}(data')$. Les calculs des tailles des données couvertes et des données non couvertes sont différents, il faut décomposer D en deux parties (définition 3.6 page 70) :

$$D = D_{cover'(M)} \cup D_{\overline{cover'}(M)}$$

Ce qui permet de calculer $L(D|M)$:

$$L(D|M) = L(D_{cover'(M)} \cup D_{\overline{cover'}(M)}|cover', M) + L(cover') \tag{4.12}$$

$$= L(D_{cover'(M)}|cover', M) + L(D_{\overline{cover'}(M)}|cover', M) + L(cover') \tag{4.13}$$

Or, comme $cover'$ et M sont inutiles pour calculer $D_{\overline{cover'}(M)}$:

$$L(D_{\overline{cover'}(M)}|cover', M) = L(D_{\overline{cover'}(M)}) \tag{4.14}$$

D'après (4.13) et (4.14), on a :

$$L(D|M) = L(D_{cover'(M)}|cover', M) + L(D_{\overline{cover'}(M)}) + L(cover') \tag{4.15}$$

De la même façon que pour calculer $L(M)$ en (4.11) page précédente, on utilise le type des données pour calculer la taille des données non couvertes:

$$L(D_{\overline{cover'}(M)}) = L(D_{\overline{cover'}(M)}|\mathcal{P}(data')) + L(\mathcal{P}(data')) \tag{4.16}$$

Ainsi, à partir des expressions (4.11) page ci-contre, (4.15) et (4.16), on décompose le calcul de taille de la description des données D :

$$\begin{aligned} C(D, M) = & L(M|model') + L(model') + \\ & L(cover') + \\ & L(D_{cover'(M)}|cover', M) + \\ & L(D_{\overline{cover'}(M)}|\mathcal{P}(data')) + L(\mathcal{P}(data')) \end{aligned}$$

Les tailles des structures représentées par les expressions $L(model')$, $L(cover')$ et $L(\mathcal{P}(data'))$ ne sont pas liées à la taille des données D ni à la taille du modèle M. Autrement dit, pour des données de grandes tailles, ces expressions sont négligeables car constantes. Dans le cas de données de taille moins importante, ces expressions peuvent jouer un rôle. Nous notons la somme des tailles $L(model') + L(cover') + L(\mathcal{P}(data'))$ par $L(struct)$ dont le calcul est explicité en annexe A page 161.

Le programme complet permettant de décrire des données généralisées par un modèle est composé : d'un modèle, d'une façon de retrouver les données couvertes sachant le modèle et la relation de couverture, des données non couvertes par le modèle et de la structure de ces éléments (données et modèle). Formellement, ces éléments permettent de définir la fonction d'évaluation suivante.

Définition 4.17 (Évaluation MDL-Schema)
Soient un schéma $\mathbf{Exec}_s$ structuré exactement comme l'illustre la figure 4.10 page 106, des données D de type $\mathcal{P}(data')$ et un modèle M de type $model'$. La taille de la description des données D relativement au modèle M dans le schéma $\mathbf{Exec}_s$ est :

$$\begin{aligned} C(D,M) = & L(M|model')+ \\ & L(D_{cover'(M)}|cover', M)+ \\ & L(D_{\overline{cover'}(M)}|\mathcal{P}(data'))+ \\ & L(struct) \end{aligned}$$

.

Évaluation locale de la qualité

La définition 4.17 évalue la qualité d'un modèle à résumer globalement un ensemble D de données. En effet, quel que soit le modèle M, $C(D, M)$ représente la taille pour coder tout l'ensemble des données D. Cependant, il peut être intéressant de localiser l'évaluation aux données couvertes dans D par le modèle. Nous différencions ces deux méthodes en utilisant les notions d'évaluation locale et globale.

La première méthode évalue la qualité d'un modèle à résumer globalement un ensemble de données. Prenons le cas d'un modèle et de sa relation de couverture non adaptée aux données. Ce modèle ne couvre que certaines propriétés des données. C'est grâce à l'utilisation de l'adaptation du schéma de l'opérateur et plus précisément à la conversion de la relation de couverture (si elle contient des factorisations ou des cofactorisations) qu'un modèle couvre toutes les propriétés des données. Ainsi l'intégration de la taille de description des propriétés non couvertes dans l'évaluation d'un modèle fournit un critère global de couverture.

La seconde méthode consiste à évaluer la qualité locale d'un modèle. Dans ce cas, on ne s'intéresse qu'aux données couvertes : le modèle est évalué seulement sur les propriétés des données qu'il couvre. La différence par rapport à la méthode globale est de ne pas prendre en compte les données non couvertes et les propriétés non couvertes. La figure 4.17 montre graphiquement cette différence. Dans ce cas c'est le schéma des données qui est adapté au schéma de l'opérateur. Cela consiste simplement à retirer certains nœuds du schéma $\mathbf{Exec}_s$. Ces nœuds spécifient les propriétés des données que la relation de couverture n'utilise pas sans être convertie.

De manière à évaluer la qualité locale d'un modèle et la qualité de sa couverture, nous définissons deux mesures complémentaires l'une à l'autre : *le taux de compression* qui indique la faculté du modèle à compresser les données qu'il couvre et *le taux de couverture* qui indique la quantité de données couvertes relativement à toutes les données à résumer. Plus le taux de compression est faible plus les données sont compressées et plus la qualité du modèle est importante. Plus le taux de couverture est important, plus le modèle rend compte de l'information contenue dans toutes les données.

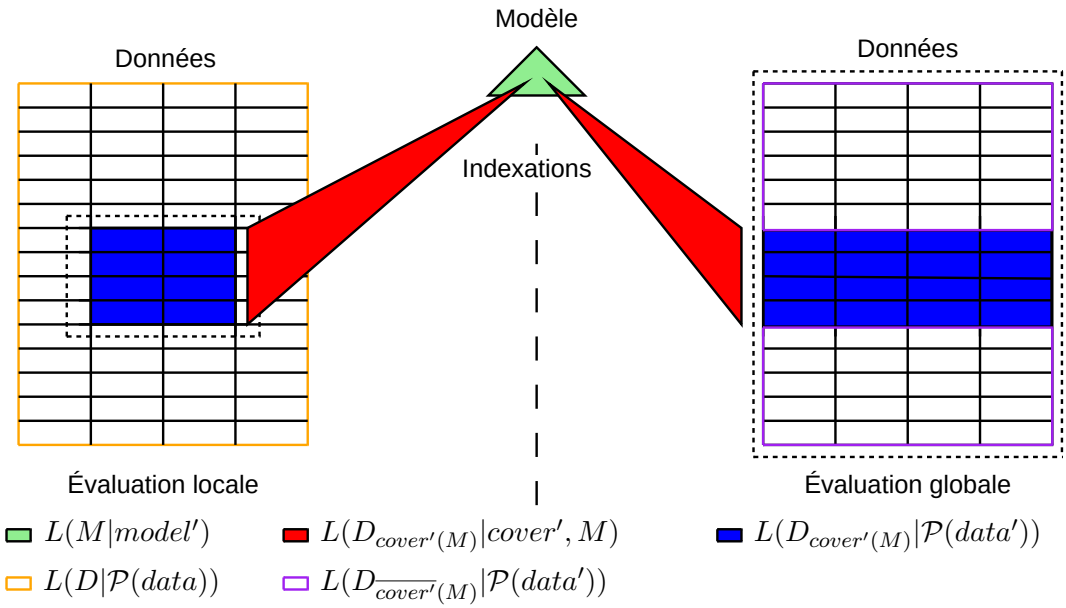


FIG. 4.17: Représentation graphique de la qualité globale et locale d'un modèle. Les données sont considérées comme un tableau. Les propriétés sont en colonne et chaque élément des données est une ligne. Les cadres pointillés indiquent les données prises en compte dans l'évaluation. Les données en bleu foncé représentent les données couvertes par le modèle.

Définition 4.18 (Taux de compression)
Soient un schéma $\mathbf{Exec}_s$ structuré exactement comme l'illustre la figure 4.10 page 106, des données D de type $\mathcal{P}(data')$ et un modèle M de type $model'$. Le rapport entre la taille pour retrouver les données D couvertes par M en sachant M et en ne sachant pas M est le taux de compression de M relativement à D :

$$\alpha(D, M) = \frac{L(M|model') + L(D_{cover'(M)}|cover', M) + L(struct)}{L(D_{cover'(M)}|\mathcal{P}(data')) + L(struct)} \quad .$$

Définition 4.19 (Taux de couverture)
Soient un schéma $\mathbf{Exec}_s$ structuré exactement comme l'illustre la figure 4.10 page 106, des données D de type $\mathcal{P}(data')$ et un modèle M de type $model'$. Le rapport entre la taille des données D couvertes par M et la taille de D est le taux de couverture de M relativement à D :

$$\beta(D, M) = \frac{L(D_{cover'(M)}|\mathcal{P}(data')) + L(struct)}{L(D|\mathcal{P}(data)) + L(struct)} \quad .$$

Les définitions précédentes imposent le calcul de tailles pour différents types d'objets : nœud, chemin, élément typé et élément couvert. La taille d'un objet correspond à la taille de la chaîne binaire qui le représente. Or, il existe plusieurs manières de représenter de tels objets. De manière à choisir une méthode de codage, nous faisons l'hypothèse, que du point de vue de l'utilisateur, la complexité d'un objet qu'il visualise correspond au nombre de bits nécessaires pour coder naturellement l'objet. Même si cette hypothèse peut être fausse dans de nombreux cas, elle est celle qui suppose le moins de connaissances de la part de l'utilisateur.

Le calcul de ces tailles est présenté en annexe et dans les sous-sections suivantes. Le calcul de la taille de $L(struct)$ qui peut se résumer au calcul de la taille de description des nœuds et des chemins d'un schéma est défini en annexe A. Le calcul des tailles $L(M|model')$ et $L(D_{\overline{cover'}(M)}|\mathcal{P}(data'))$ qui sont des tailles d'éléments en fonction de leur type est décrit dans la sous-section 4.3.2. La taille $L(D_{cover'(M)}|cover', M)$ est la taille de la description des données couvertes par un modèle dont le calcul est décrit dans la sous-section 4.3.3.

4.3.2 Taille de description d'un élément relativement à un nœud

Lorsque l'on connaît le nœud (le type) N d'un élément alors il est possible de déterminer la taille de description $L(E|N)$ de cet élément. En fonction de ce nœud, on exprime la taille de représentation de l'élément dans l'algorithme 4.20 page ci-contre. On différencie les trois cas où l'élément E appartient à un nœud N qui est un cône, un cocône ou l'objet des parties. Dans un cas différent de ceux-ci, l'algorithme n'est pas capable de déterminer la taille de l'élément E (ligne 7). Si E est un cône (ligne 1) alors pour chacune de ses composantes, le même algorithme est appelé. Si E est un sous-ensemble (ligne 2) alors on utilise la forme auto-délimitante (ligne 3) pour sauvegarder la taille de l'ensemble. La forme auto-délimitante est détaillée dans l'annexe A.1.2 page 162. Elle permet de calculer la taille de représentation d'une information dont on ne connaît pas a priori la taille maximum. Pour chacun des éléments de l'ensemble, le même algorithme est appelé. Dans le cas d'un cocône (ligne 4), la fonction $trouve\_inclusion(N, E)$ retourne l'inclusion utilisée dans le cocône N pour représenter l'élément E. Deux situations sont différenciées. Si l'inclusion vient du nœud **1** (ligne 5) alors la taille de l'élément correspond au *log* de la taille de l'image de la relation d'inclusion. Si l'inclusion vient d'un autre nœud (ligne 6) alors la taille de l'élément est la taille de son antécédent par la fonction d'inclusion ($antécédent(f, e)$ retourne l'antécédent de e par la fonction bijective f).

4.3.3 Taille de description des données couvertes par un modèle

Il existe plusieurs façons d'encoder des données D couvertes par un modèle M relativement à une relation de couverture $cover'$. Tout d'abord, nous introduisons la notion d'*indexation* dans le but de formaliser ces méthodes d'encodage. Une indexation est construite à partir d'un chemin qui représente la relation de couverture

Algorithme 4.20 (Taille de la description d'un élément typé *taille*$(E|N)$)

Entrées : Un élément E et son nœud associé N : $E|N$
Sorties : La taille de la description de l'élément : l
suivant *la structure de* N **faire**
1 **cas où** N *est un cône*
$l = 0$
pour chaque $p \in projections(N)$ **faire**
$l = l + taille(p(E))$
2 **cas où** N *est un objet des parties*
3 $l = taille\_auto\_delim(log(|E|))$
pour chaque $e \in E$ **faire**
$l = l + taille(e)$
4 **cas où** N *est un cocône*
$Inclusion = trouve\_inclusion(N, E)$
$(Domaine, Codomaine) = définition(Inclusion)$
5 **si** $Domaine = \mathbf{1}$ **alors**
$l = log(|Inclusion(1)|)$
sinon
6 $l = taille(antécédent(Inclusion, E))$
7 **autres cas** *exception*(« impossible de déterminer la taille d'un élément de type abstrait »)
retourne(l)

cover′. Enfin, un algorithme de recherche de l'indexation minimisant la taille de la description est introduit.

Indexation

La problématique de l'indexation est de définir un moyen de retrouver des données $D \in \mathcal{P}(data')$ dans l'ensemble des données couvertes par un modèle $M \in model'$ à partir d'une relation de couverture $cover' : model' \nrightarrow data'$ (spécifiée dans le schéma $\mathbf{Exec}_s$ de la figure 4.10) en sachant que (d'après les définitions 3.6 page 70 et 4.17 page 108) :

$$D'_c = \{d | d \in cover'(M) \wedge d \subseteq D\}$$

Sachant que les éléments de D'_c sont suffisants pour retrouver D, une des façons les plus simples pour retrouver D est d'indexer le sous-ensemble D'_c dans l'ensemble $cover'(M)$. L'indexation correspond à retrouver un sous-ensemble de taille $n = |D'_c|$ dans un ensemble de taille $m = cover'(M)$. En calculant le nombre de sous-ensembles de taille n dans l'ensemble m, le nombre de combinaison $\binom{m}{n}$ permet de connaître la place nécessaire pour retrouver D'_c dans $cover'(M)$. La définition 4.21 page suivante définit le mécanisme de calcul de la taille de description d'un sous-ensemble.

Définition 4.21 (Taille de la description d'un sous-ensemble)
Soient E un ensemble et un de ses sous-ensembles $E_s \subseteq E$. La taille de la description de E_s sachant E, notée $L_c(E_s|E)$, est définie par :

$$L_c(E_s|E) = log(|E|) + log(\binom{|E|}{|E_s|})$$

L'expression $log(|E|)$ correspond à la place nécessaire pour le codage du nombre d'éléments du sous-ensemble de taille maximale $|E|$ et $log(\binom{|E|}{|E_s|})$ correspond à la place nécessaire pour le codage du sous-ensemble à retrouver parmi tous les sous-ensembles de taille n. Par abus de notation, on pourra écrire $L_c(n|m)$ avec $m = |E|$ et $n = |E_s|$. ▪

Il est possible de généraliser la façon d'indexer illustrée précédemment. L'indexation d'éléments intervient dans l'image du modèle par la relation de couverture. Cependant, l'indexation peut aussi intervenir plus en amont. Par exemple, considérons une relation de couverture qui est une composition $s \circ r$, il peut être intéressant de retenir seulement l'ensemble des éléments de $r(M)$ nécessaires pour calculer D'_c. L'exemple 4.22 illustre ce point précis. L'exemple illustre un arbre pour chaque proposition d'indexation. Si l'image d'une relation est indexée alors le nœud représentant l'image est encadré dans les représentations graphiques de l'arbre.

Exemple 4.22 (Description de données couvertes par une composition)
On définit une relation $c : int \nmapsto int$ qui est la composition de deux relations $c = s \circ r$. Intuitivement, la relation r construit un ensemble d'intervalles à partir d'un entier et la relation s associe un intervalle à tous les entiers compris dans celui-ci. Dans cet exemple, un modèle est un nombre.

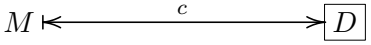
- $r : int \nmapsto int \times int$ telle que $r(x) = \{(x, x), (x + 1, x + 2), (x + 3, x + 6), (x + 7, x + 14), (x + 15, x + 30)\}$ et
- $s : int \times int \nmapsto int$ telle que $s(x_1, x_2) = \{y | x_1 \leq y \leq x_2\}$

Quelle est la taille l de la description de $D = \{1, 2, 7, 8, 13\}$ sachant $M = 0$ et la relation c ?

1. Il est possible de ne pas utiliser c et M. Ainsi, on écrit D relativement au type int (codé pour l'exemple sur 8 bits). Cela correspond au calcul de taille de description d'éléments typés expliqué dans la sous-section 4.3.2 page 110.

$$\begin{aligned} l_1 &= taille(D|int) \\ &= taille\_auto\_delim(log(|D|)) + |D| \times log(|int|) \\ &= taille\_auto\_delim(log(5)) + 5log(256) \\ &= 8 + 5 \times 8 \\ &= 48\ bits \end{aligned}$$

2. Il est possible de retrouver D dans $c(M)$. $c(M)$ couvre les 31 éléments compris entre 0 et 30.



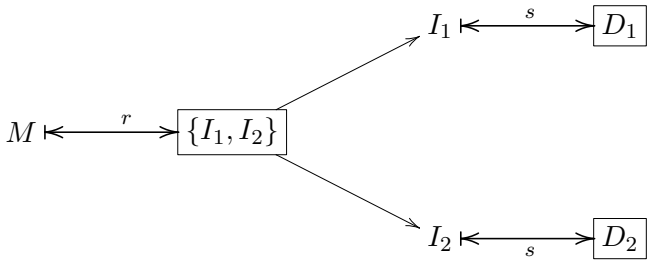
$$
\begin{aligned}
l_2 &= L_c(D|c(M)) \\
&= log(|c(M)|) + log(\binom{|c(M)|}{|D|}) \\
&= log(31) + log(\binom{31}{5}) \\
&= 5 + 18 \\
&= 23 \ bits
\end{aligned}
$$

3. Il est possible de retrouver les intervalles $I = \{(1,2),(7,14)\}$ dans $r(M)$. Ces éléments sont suffisants pour retrouver D en utilisant s. Ensuite, il faut retrouver D dans $s(I)$ soit :

$$M \overset{r}{\longmapsto} \boxed{I} \overset{s}{\longmapsto} \boxed{D}$$

$$
\begin{aligned}
l_3 &= L_c(I|r(M)) + L_c(D|s(I)) \\
&= L_c(2|5) + L_c(5|10) \\
&= 3 + 4 + 4 + 8 \\
&= 19 \ bits
\end{aligned}
$$

4. Comme pour la précédente proposition, on décompose la relation c. Cependant, on calcule l'image des intervalles $I_1 = (1,2)$ et $I_2 = (7,14)$ par s. Ensuite, il faut retrouver $D_1 = \{1,2\}$ dans $s(I_1)$ et $D_2 = \{7,8,13\}$ dans $s(I_2)$ soit :



$$
\begin{aligned}
l_4 &= L_c(\{I_1,I_2\}|r(M)) + L_c(D_1|s(I_1)) + L_c(D_2|s(I_2)) \\
&= L_c(2|5) + L_c(2|2) + L_c(3|8) \\
&= log(5) + log(\binom{5}{2}) + log(2) + log(\binom{2}{2}) + log(8) + log(\binom{8}{3}) \\
&= 3 + 4 + 2 + 0 + 3 + 6 \\
&= 18 \ bits
\end{aligned}
$$

Dans ce cas précis, la meilleure solution (l_4) consiste à décomposer complètement la relation c. Il est important de noter que pour un autre ensemble de données $D' = \{30\}$ réduit à un singleton, un rapide calcul montre que parmi les quatre propositions de calcul précédentes, la meilleure solution consiste à ne pas décomposer la relation c ($l'_1 = 16$, $l'_2 = 5{+}5 = 10$, $l'_3 = 3{+}3{+}4{+}4 = 14$ et $l'_4 = 3{+}3{+}4{+}4 = 14$).■

Deux possibilités sont considérées pour retrouver les éléments dans l'image d'une relation : dissocier ou associer les éléments indexés. Dissocier revient à considérer les éléments *individuellement* (cas 3 de l'exemple 4.22 page 112) et associer revient à considérer les éléments *globalement* (cas 4 de l'exemple 4.22 page 112). L'indexation d'un sous-ensemble est toujours calculée relativement à l'image d'une relation, c'est pourquoi si on considère une relation r, on représente les deux types d'indexation par les deux symboles d'indexation ${}^{g}r$ et ${}^{i}r$ (respectivement pour indexation globale et individuelle) situés derrière la relation r.

L'indexation est relative aux relations et plus généralement aux chemins. Or, un chemin contient aussi des compositions, des factorisations et des cofactorisations. Une composition $r \circ s$ implique deux chemins alors qu'une factorisation et une cofactorisation en impliquent plusieurs en fonction, respectivement, du nombre de projections et d'inclusions. L'indexation de tous ces chemins est fait de la même manière que pour une composition. Par exemple, soit la factorisation $\langle r_1, r_2 \rangle$, plusieurs façons de l'indexer peuvent être proposées :

- ${}^{g}\langle r_1, r_2 \rangle$: indexation dans l'image de la factorisation.
- ${}^{g}\langle {}^{g}r_1, r_2 \rangle$: indexation dans les images de r_1 et de la factorisation.
- ${}^{g}\langle {}^{g}r_1, {}^{g}r_2 \rangle$: indexation dans les images de r_1, de r_2 et de la factorisation.

La définition suivante formalise les indexations et les deux exemples qui la suivent illustrent plusieurs indexations.

Définition 4.23 (Indexation d'un chemin)
Soit c un chemin composé de compositions, de factorisations, de cofactorisations et de relations. L'ensemble $I(c)$ des indexations du chemin c est défini tel que :

- si c est une composition $r \circ s$ alors $I(c) = \{r' \circ s' | r' \in I(r), s' \in I(s)\}$
- si c est une factorisation $\langle r_1, \ldots, r_n \rangle$ alors $I(c) = \{{}^{g}c', {}^{i}c', c' | c' \in I_s\}$ tel que $I_s = \{\langle r'_1, \ldots, r'_n \rangle | \forall i \in 1 \ldots n, r'_i \in I(r_i)\}$
- si c est une cofactorisation $\langle r_1; \ldots; r_n \rangle$ alors $I(c) = \{{}^{g}c', {}^{i}c', c' | c' \in I_s\}$ tel que $I_s = \{\langle r'_1; \ldots; r'_n \rangle | \forall i \in 1 \ldots n, r'_i \in I(r_i)\}$
- si c est une relation alors $I(c) = \{c, {}^{g}c, {}^{i}c\}$ ▪

Exemple 4.24 (Formalisation des indexations de l'exemple 4.22)
On utilise la formalisation de l'indexation pour montrer comment les différentes propositions de l'exemple 4.22 page 112 sont écrites simplement :

1. La non utilisation de la relation de couverture n'est pas formalisée.
2. Indexation de c dans sa globalité : ${}^{g}c = {}^{g}s \circ r$.
3. Indexation individuelle des éléments nécessaires de l'image du modèle par r : ${}^{g}s \circ {}^{i}r$.
4. Indexation globale des éléments nécessaires de l'image du modèle par r : ${}^{g}s \circ {}^{g}r$. ▪

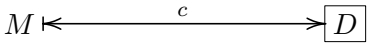
Exemple 4.25 (Indexation d'une factorisation)
On définit une relation $c : int \mapsto int$ telle que $c = s \circ \langle r_1, r_2 \rangle$. La relation $c(x)$ énumère tous les nombres pouvant être énumérés avec deux symboles en base x (ces

éléments vont de 0 jusqu'à $x^2 - 1$). La relation s correspond à la somme. La relation r_1 (resp. r_2) calcule les valeurs de la position 1 (resp. 2) du nombre.

- $r_1 : int \nmapsto int$ telle que $r_1(x) = \{n | n = 0 \dots x - 1\}$,
- $r_2 : int \nmapsto int$ telle que $r_2(x) = \{n \times x | n = 0 \dots x - 1\}$ et
- $s : int \times int \nmapsto int$ telle que $s(x, y) = x + y$.

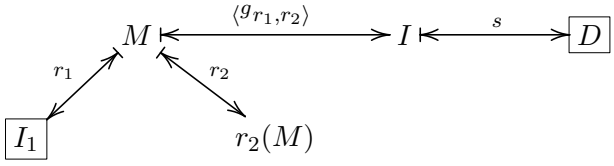
On propose quelques manières d'indexer la composition c pour retrouver les données $D = \{23, 26, 28, 51, 53, 57\}$ à partir du modèle $M = 10$.

- ${}^g c \circ \langle r_1, r_2 \rangle$: Comme $M = 10$, $c(M)$ couvre 100 éléments et D est composé de 6 éléments.



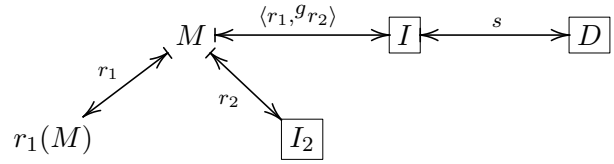
$$\begin{aligned} l_1 &= L_c(D|c(M)) \\ &= L_c(6|100) \\ &= 38 \ \textit{bits} \end{aligned}$$

- ${}^g s \circ \langle {}^g r_1, r_2 \rangle$: Les éléments de $r_1(M)$ (de taille 10) suffisants pour calculer D sont $I_1 = \{1, 3, 6, 7, 8\}$. On obtient l'ensemble intermédiaire I en calculant le produit cartésien $I = I_1 \times r_2(M)$ tel que $r_2(M) = \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$. La taille de $s(I)$ est de $|I_1| \times |r_2(M)| = 5 \times 10 = 50$.



$$\begin{aligned} l_2 &= L_c(I_1|r_1(M)) + L_c(D|s(I)) \\ &= L_c(5|10) + L_c(6|50) \\ &= 12 + 30 \\ &= 42 \ \textit{bits} \end{aligned}$$

- ${}^g s \circ {}^g \langle r_1, {}^g r_2 \rangle$: les éléments de $r_2(M)$ suffisants pour calculer D sont $I_2 = \{20, 50\}$. Les éléments retenus dans I sont $\{(20, 3), (20, 6), (20, 8), (50, 1), (50, 3), (50, 7)\}$.



$$\begin{aligned} l_3 &= L_c(I_2|r_2(M)) + L_c(I|r_1(M) \times I_2) + L_c(D|s(I)) \\ &= L_c(2|10) + L_c(6|10 \times 2) + L_c(6|6) \\ &= 10 + 21 + 3 \\ &= 34 \ \textit{bits} \end{aligned}$$

La meilleure indexation est la troisième. Cela est dû au fait que l'indexation ${}^g r_2$ réduit de manière importante l'ensemble $r_1(M) \times I_2$ à partir duquel I est retrouvé.■

Nous avons défini un algorithme *taille_indexation*(D, M, e) à l'aide de la définition d'une indexation qui calcule la taille de l'indexation $e \in I(c)$ pour représenter les données D sachant le modèle M. Cet algorithme est présenté en annexe A.2 page 163. Intuitivement, il extrait les instances d'un modèle dans les données, construit un arbre d'instances (identique aux arbres présentés dans les exemples précédents) et calcule la taille de la représentation de cet arbre.

Il est important de noter qu'il existe plusieurs indexations d'un même chemin et que ces indexations fournissent des tailles d'indexation différentes. Or, le principe MDL préconise de choisir l'indexation qui minimise la taille de la description. La partie suivante propose cette minimisation à partir de la génération aléatoire d'indexations.

Minimisation de la taille d'indexation

L'algorithme *taille_indexation*(D, M, e) calcule la taille d'indexation de données à partir d'un modèle et d'une indexation. Il est nécessaire de calculer une indexation à partir de la relation de couverture avant de la soumettre à l'algorithme *taille_indexation*. Or, l'ensemble $I(c)$ des indexations possibles d'une relation c est relativement volumineux. Il n'est pas envisageable d'énumérer toutes les indexations possibles. De ce fait un algorithme de minimisation aléatoire d'indexation a été élaboré. Il consiste à générer aléatoirement plusieurs indexations puis à calculer les tailles de la description des données selon ces indexations. La plus petite valeur d'indexation est retenue.

Au cours de nos expérimentations, nous avons remarqué que l'exécution de l'algorithme *taille_indexation* était plus ou moins longue suivant l'indexation. Plus son exécution est longue, plus la taille de la description est importante. D'après cette heuristique, une contrainte de temps a été adjointe à l'algorithme 4.26 page suivante de minimisation d'indexation.

L'algorithme 4.26 page ci-contre recherche une approximation de la plus petite taille d'indexation. Il est basé sur les méthodes de Monte Carlo (Krauth, 1996) et d'« Iterative deepening » (Reinefeld et Marsland, 1994). L'algorithme génère aléatoirement n indexations (ligne 1) et autorise un temps maximum t pour le calcul de la taille d'une indexation. Le temps maximum au lancement de l'algorithme est court et s'accroît en fonction du nombre de calculs d'indexation non achevés pour prendre en compte l'heuristique de limite de temps.

Pour ce faire, la fonction *indexation_aléatoire*(c) extrait aléatoirement une indexation dans l'ensemble $I(c)$ et la taille de cette indexation est calculée par l'algorithme *taille_indexation* (ligne 2). La réponse de celui-ci est attendue pendant un temps maximum t. Au début ce temps est fixé par le paramètre t_{min}. Si la réponse n'est pas rendue au bout du temps t alors l'algorithme génère aléatoirement une nouvelle indexation et le processus recommence. Quand le nombre d'essais infructueux dépasse le seuil m_{max} alors le temps alloué est multiplié par deux (ligne 3).

Algorithme 4.26 (Indexation minimisée $min\_index(n, t_{min}, m_{max}, D, M, c)$)

Entrées :
Nombre d'indexations aléatoire à générer : n
Durée minimum de calcul : t_{min}
Nombre de tentatives de calcul de même durée : m_{max}
Un ensemble de données : $D \in \mathcal{P}(data)$
Un modèle : $M \in model$
Une relation de couverture $c : model \nrightarrow \mathcal{P}(data)$
Sorties : Taille minimale d'indexation calculée : l_{min}
$l_{min} = \infty$
1 **pour tous les** $i \in 1 \dots n$ **faire**
  $m = m_{max}$
  $t = t_{min}$
  $e = indexation\_aléatoire(c)$
2 **tant que** $l = taille\_indexation(D, M, e)$ *ne se finit pas en une durée* t **faire**
    $e = indexation\_aléatoire(c)$
    $m = m - 1$
    si $m = 0$ **alors**
3 $t = t \times 2$
      $m = m_{max}$
  $l_{min} = min(l_{min}, l)$
retourne(l_{min})

Le critère MDL est mis en œuvre dans le but d'évaluer la qualité d'un modèle pour résumer des données. La description des données est décomposée à partir des constructions fournies par les schémas et de la relation de couverture qui relie le modèle aux données. Un mécanisme d'indexation a été introduit pour indexer des éléments appartenant à l'image d'un chemin. Ce mécanisme utilise la génération aléatoire d'indexation dans le but de minimiser la taille de la description. Cet algorithme est évalué expérimentalement dans la section suivante.

4.4 Expérimentations

Les expérimentations sur des données synthétiques permettent de montrer la validité de la méthode. Le chapitre 5 page 129 montre son intérêt dans le cas de données réelles. Il est nécessaire de valider plusieurs points. Dans un premier temps, l'écriture d'opérateurs sous la forme d'un schéma est abordée. Il permet d'appréhender le langage mis en place pour définir les différents concepts théoriques vus précédemment. Dans un second temps, nous validons l'évaluation générique aux moyens de données synthétiques.

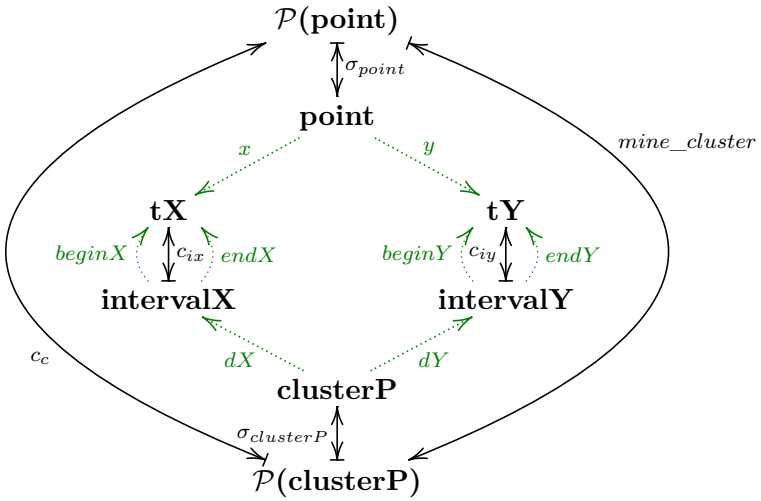
4.4.1 Spécification de la fouille de données par clustering

Les méthodes de clustering (sous-section 3.2.4 page 85) varient dans la façon de regrouper les données mais aussi dans la façon de représenter les regroupements. Si aucune généralisation du regroupement n'est effectuée alors c'est l'énumération des points du regroupement qui constitue la description du regroupement. Il est aussi possible de représenter un regroupement par son centroïde (centre de gravité). Dans ce cas, la classe d'un point particulier est définie par le centroïde le plus proche (relativement à une distance). Dans le cas présenté ici, nous considérons qu'un cluster est représenté par un ensemble d'intervalles, un par dimension. Pour chaque dimension d'un cluster, on calcule le plus petit intervalle encadrant toutes les valeurs des points du cluster dans cette dimension.

Nous avons choisi d'illustrer l'approche sur l'algorithme de clustering k-means (Anderberg, 1973). Il existe de nombreuses implémentations de cet opérateur. Weka (Witten et Frank, 2005) fournit un ensemble d'outils pour la fouille de données implémentés en Java . Il aurait été possible d'implémenter directement cet opérateur en Prolog mais le but de cette expérimentation est de montrer qu'il est possible de lier une implémentation dans un langage différent des schémas. La version graphique (schéma 4.27) et la version texte (programme 4.28) du schéma sont données. La version graphique est partielle puisque la version texte introduit les définitions de deux morphismes permettant de spécifier les intervalles $intervalX$ et $intervalY$ à partir du schéma B.5 page 169. L'opérateur k-means est appelé à la ligne 43 par la méthode statique buildClusters2 de la classe k-means. Le code de cette classe est donné dans le programme B.4 page 168. Il montre, entre autre, que les relations définies dans un schéma peuvent être utilisées en Java. De ce fait, il est aisé d'utiliser la structure mise en place dans les schémas en vue de convertir les données au format de Weka.

Schéma 4.27 (Schéma de k-means)
Les données à abstraire sont un ensemble de points ($\mathcal{P}(point)$). Un point est un couple (x, y) avec x de type tX et y de type tY. L'opérateur $mine\_cluster$: $\mathcal{P}(point) \rightarrow \mathcal{P}(clusterP)$ transforme un ensemble de points en un ensemble de clusters ($\mathcal{P}(clusterP)$). Un cluster est un couple (dX, dY) où dX et dY sont des intervalles encadrant des éléments de type respectif tX et tY. La relation de couverture

$c_c : \mathcal{P}(clusterP) \rightarrow \mathcal{P}(point)$ lie le modèle aux données.



$$c_c = \eta_{point} \circ \langle c_{ix} \circ dX, c_{iy} \circ dY \rangle \circ \sigma_{clusterP}$$

.

Programme 4.28 (Schéma de k-means)

```
:− begin_schema(k_means).

:− set_def_sort(tX).
:− set_def_sort(tY).
:− product(point,[(x,tX),(y,tY)]).
:− part(point).

% foncteur pour definir les intervalles
:− begin_functor(interval_F_X,interval_abs_int).
  :− functor_sort(int,tX).
  :− functor_sort(interval_int,intervalX).
  :− functor_rel(begin_int,beginX).
  :− functor_rel(end_int,endX).
  :− functor_rel(couv_interval_int,c_ix).
:− end_functor(interval_F_X).
:− begin_functor(interval_F_Y,interval_abs_int).
  :− functor_sort(int,tY).
  :− functor_sort(interval_int,intervalY).
  :− functor_rel(begin_int,beginY).
  :− functor_rel(end_int,endY).
  :− functor_rel(couv_interval_int,c_iy).
:− end_functor(interval_F_Y).
:− product(clusterP,[(dX,intervalX),(dY,intervalY)]).
:− part(clusterP).
```

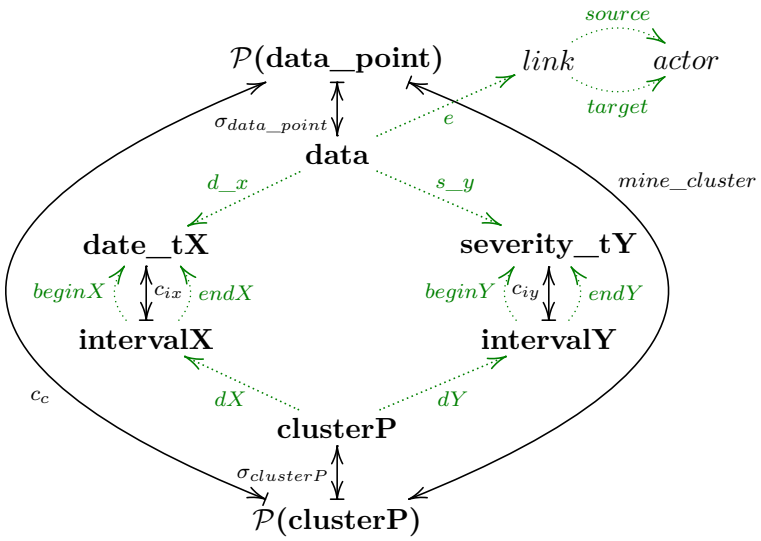
```
% foncteur du schema mine
:- begin_functor(sgc,mineSchema).
  :- functor_sort(data,point).
  :- functor_sort(model,p_clusterP).
  :- functor_rel(cover,c_c).
  :- functor_rel(mine,mine_cluster).
:- end_functor(sgc).

% relation de couverture
:- set_path(comp(eta_point,comp(fact([(x,comp(c_ix,dX)),
                                      (y,comp(c_iy,dY))]),
                        det_clusterP)),
            c_c).

% implementation de l'operateur
mine_cluster(Points @ p_point, ClusterP @ p_clusterP) :-
    between(2,10,N),
    call_java_relation('k_means',
                       'buildClusters2',
                       [N,Points @ p_point], ClusterP @ p_clusterP).
:- end_schema(k_means).
```

Il est important de noter que l'opérateur k-means nécessite de donner en paramètre le nombre de clusters à générer. Or, les paramètres des opérateurs ne sont pas traités dans l'approche proposée. En effet, il n'est pas possible de demander à un utilisateur de les définir alors que l'on suppose qu'il ne connaît même pas les techniques de fouille de données employées. Nous générons les clusterings de 2 à 10 clusters dans le but de prendre en compte ce problème. De ce fait, nous nous appuyons sur l'évaluation de la complexité en vue de classer ces clusterings. Il est évident que cette méthode n'est pas efficace : faire une étude paramétrique dans le cas de techniques nécessitant de nombreux paramètres est très coûteux. Il est important d'envisager d'autres solutions qui sont évoquées dans les perspectives abordées dans la conclusion de cette thèse.

Le schéma de k-means et le schéma 2.44 page 56 des alarmes réseau sont unifiés automatiquement. Deux schémas sont générés. L'un d'eux est représenté graphiquement (schéma 4.29) et textuellement (programme 4.30 page ci-contre). La composante x des points est unifiée avec la date et la composante y avec la sévérité.

Les deux morphismes ayant pour extrémité le schéma général de l'unification suffisent à définir le programme. Il est aussi intéressant de remarquer que la réécriture de la relation de couverture $c_c : \mathcal{P}(clusterP) \nrightarrow \mathcal{P}(data\_point)$ qui n'est pas définie dans le programme 4.30 page suivante est automatique (section 4.2 page 97).

Schéma 4.29 (Schéma k-means adapté aux alarmes réseau)



$$c_c = \eta_{alarm} \circ \langle c_{ix} \circ dX, c_{iy} \circ dY, \Sigma_{link} \circ \varnothing_{clusterP} \rangle \circ \sigma_{clusterP}$$

.

Programme 4.30 (Schéma k-means adapté aux alarmes réseau)

```
:- begin_schema(u_alarm_data_cluster2_1).

% Foncteur du schema des alarmes reseau
:- begin_functor(f_mineA_4_P,alarm_data).
    :- functor_sort(1,1).
    :- functor_sort(alarm,data).
    :- functor_sort(p_alarm,p_alarm_p_point).
    :- functor_sort(severity,severity_tY).
    :- functor_sort(date,date_tX).
    :- functor_rel((s,alarm,severity),s_y).
    :- functor_rel((eta_alarm,alarm,p_alarm),eta_alarm_eta_point).
    :- functor_rel((det_alarm,p_alarm,alarm),det_alarm_det_point).
    :- functor_rel((d,alarm,date),d_x).
:- end_functor(f_mineA_4_P).

% Foncteur du schema k-means
:- begin_functor(f_mineB_4_P,cluster2).
    :- functor_sort(1,1).
    :- functor_sort(point,data).
    :- functor_sort(p_point,p_alarm_p_point).
    :- functor_sort(tY,severity_tY).
    :- functor_sort(tX,date_tX).
    :- functor_rel((y,point,tY),s_y).
    :- functor_rel((eta_point,point,p_point),eta_alarm_eta_point).
```

```
    :- functor_rel((det_point,p_point,point),det_alarm_det_point).
    :- functor_rel((x,point,tX),d_x).
:- end_functor(f_mineB_4_P).
:- end_schema(u_alarm_data_cluster2_1).
```

Les schémas présentés sont relativement simples. Ils montrent que les schémas bénéficient d'une souplesse d'utilisation permettant de satisfaire deux objectifs de notre problématique. D'une part, la spécification des entrées et sorties des opérateurs est complètement réalisée en utilisant les schémas. Ces spécifications sont le point central de l'unification. D'autre part, l'implémentation des opérateurs est directement réalisée dans un langage approprié afin de conserver l'efficacité de l'opérateur. De ce fait la réutilisation des opérateurs s'en trouve améliorée jusqu'à permettre l'automatisation de leur adaptation à différentes structures de données.

4.4.2 Évaluation générique

Les expérimentations suivantes portent sur des données synthétiques générées à partir de modèles et de bruit. Deux aspects sont considérés dans cette étude. Le premier concerne la qualité des résultats obtenus et le second concerne l'efficacité de la méthode. La qualité des résultats fait l'objet d'une expérimentation qui valide l'approche heuristique par limite de temps de l'algorithme 4.26 page 117. L'efficacité de la méthode fait l'objet de deux expérimentations. La première étudie l'influence du paramètre m_{max} sur le temps d'exécution et la seconde étudie l'influence de la taille des données sur le temps d'exécution. Les trois expérimentations sont présentées dans les trois parties suivantes.

Expérimentation qualitative de *min_index*

L'expérimentation comporte trois étapes. La première étape consiste à générer différents triplets (D, M, c) composés de données, d'un modèle et d'une relation de couverture. La génération se déroule de la façon suivante. Un modèle est généré selon une structure particulière (nous avons choisi les schémas des clusterings et des ensembles de séquences adaptés au schéma des alarmes réseau). Ensuite, les données sont générées en fonction du modèle de manière à ce qu'il couvre toutes les données. La seconde étape consiste à exécuter, pour chacun des triplets (D, M, c), l'algorithme *min_index* en faisant varier le paramètre n de 1 à 50. Les autres paramètres sont fixés de manière arbitraire : à 1 seconde pour t_{min} et 10 secondes pour m_{max}. La qualité d'une minimisation est représentée par le rapport de la taille retournée sur la taille minimum retournée pour plusieurs expérimentations. La fonction suivante résume ce calcul :

$$q(n, D, M, c) = \frac{min\_index(n, 1, 10, D, M, c)}{min\{min\_index(o, 1, 10, D, M, c) | o \in 1 \dots 50\}}$$

Pour un quadruplet (n, D, M, c) donné, plus $q(n, D, M, c)$ est proche de 1, plus l'évaluation minimise la taille de description. Enfin la troisième étape consiste à représenter graphiquement les valeurs de $q(n, D, M, c)$ en fonction de n. Au lieu de représenter seulement les moyennes, nous représentons la distribution par quartile en fonction de n : 50% des résultats sont inférieurs à la médiane, 75% des résultats sont inférieurs au troisième quartile et 100% des résultats sont inférieurs à la valeur maximum. Les résultats sont présentés dans les graphiques des figures 4.31 et 4.32 page suivante.

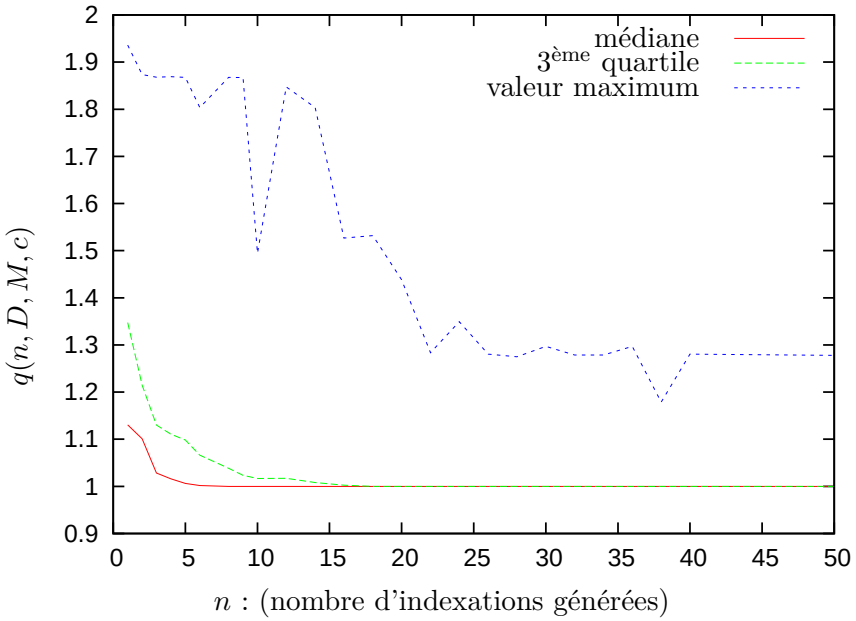


FIG. 4.31: Graphique de l'influence du nombres de tirages aléatoires sur l'indexation dans le cas d'un clustering. le profil de $q(n, D, M, c)$ est représenté pour tous les triplets (D, M, c) générés.

Dans les pires des cas (courbe du maximum), la minimisation devient acceptable pour au moins 5 itérations dans le cas des séquences et 20 dans le cas des clusterings. Cette expérimentation prouve que l'heuristique de la limite de temps est valide dans le cas des opérateurs évalués et plus précisément dans le cas des relations de couverture analysées. Cependant, on ne prouve pas que l'heuristique est valide pour toute relation de couverture. Néanmoins, on peut remarquer que dans le pire des cas ($n = 1$), l'ordre de grandeur des résultats obtenus n'est pas éloigné du meilleur des cas ($n = 50$). Cela est dû à cette heuristique. Ces deux expérimentations montrent que les tailles calculées ont un ordre de grandeur proche du meilleur résultat. Or ces tailles dépendent de la recherche d'indexations calculables rapidement. Les deux expérimentations suivantes montrent les problèmes d'efficacité que cela implique.

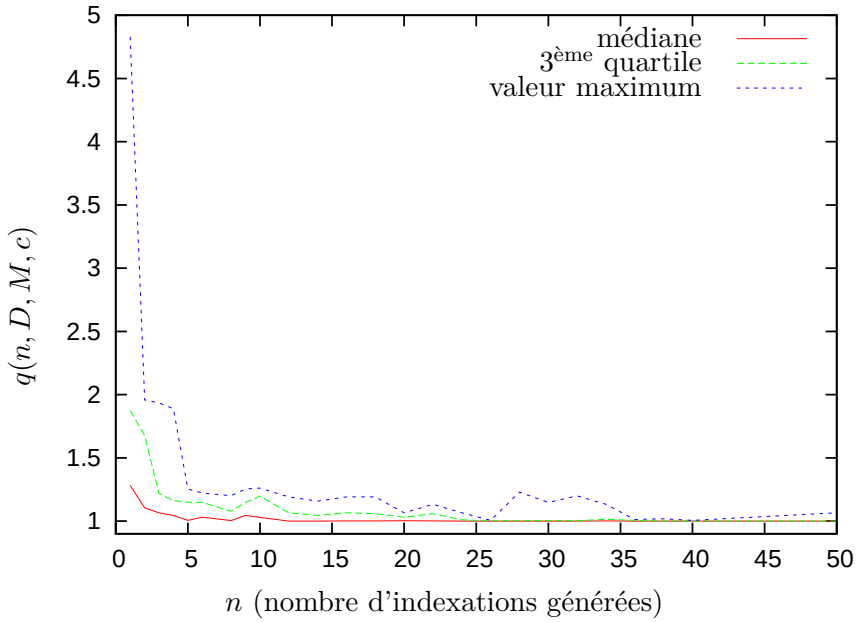


FIG. 4.32: Influence du nombre de tirages aléatoires sur l'indexation dans le cas de séquences.

Influence de m_{max} sur l'efficacité temporelle de *min_index*

La seconde expérimentation consiste à étudier l'influence du paramètre m_{max} sur la durée d'exécution de l'algorithme *min_index*. Pour ce faire, le protocole retenu est identique à l'expérimentation précédente mis à part que la variable de sortie retenue est la durée d'exécution de l'algorithme. La fonction suivante résume ce calcul :

$$q2(m_{max}, D, M, c) = \frac{time(min\_index(1, 1, m_{max}, D, M, c))}{min\{time(min\_index(1, 1, o, D, M, c))|o \in 3 \ldots 50\}}$$

Le nombre n d'indexations est fixé arbitrairement à 1. Il faut noter que la durée d'exécution de *min_index* est proportionnelle à ce paramètre. Or comme ce paramètre est constant lors de l'expérimentation, il n'a pas d'influence sur les résultats. Le graphique de la figure 4.33 page ci-contre montre l'évolution du temps en fonction du paramètre m_{max}

L'approximation linéaire par parties réalisée dans la figure 4.33 page suivante montre que l'on observe deux parties dans l'évolution de la durée d'exécution. La première partie est due au fait que plus le nombre m_{max} est faible plus l'algorithme augmente souvent la durée maximum autorisée (et ceci de manière exponentielle). La seconde partie montre que plus le nombre m_{max} est important, plus la durée est importante. Cela est dû au fait que l'algorithme ne parvient pas à trouver une indexation qui s'exécute dans le temps maximum imparti. Plus le nombre d'essais infructueux est important, plus la durée de l'exécution est importante. Il y a donc un compromis à faire entre ces deux phénomènes. Dans le cadre cette expérimen-

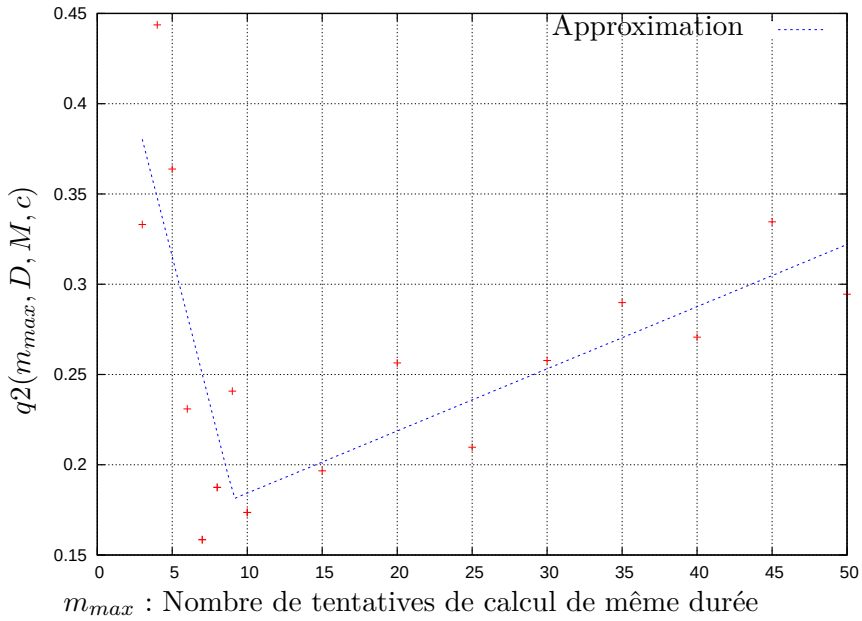


FIG. 4.33: Graphique de l'influence de la taille des données sur le temps de calcul d'une indexation d'une durée contrainte.

tation l'optimum est aux alentours de $m_{max} = 10$. Il faut noter que pour d'autres opérateurs, cette valeur optimale peut être différente.

Influence de la quantité de données sur l'efficacité temporelle de *min_index*

La troisième expérimentation consiste à examiner l'efficacité en temps de l'algorithme *min_index* en fonction de la taille des données à traiter. L'algorithme *min_index* appelle l'algorithme A.8 page 166 *taille_indexation* qui comporte trois phases : l'extraction des instances, la construction de l'arbre d'instances et le calcul de la taille de description de l'arbre des instances. La première phase d'extraction des instances est relative à l'implémentation de la relation de couverture. Or son implémentation (sous-section A.2 page 163) est dépendante du schéma. Ainsi, l'efficacité de l'extraction des instances n'est pas analysée.

L'expérimentation conduite a été faite sur des modèles en forme de cluster. Ce choix est arbitraire et ne reflète que la complexité de la relation de couverture du clustering. L'expérimentation a consisté à générer un clustering, à générer des données couvertes par ce clustering puis à rechercher la taille des données sachant le clustering en utilisant l'algorithme *min_index*. La taille des données générées varient entre 0 et 140000 bits. Ces tailles ont été calculées avec l'algorithme *taille*. Le graphique de la figure 4.34 page suivante illustre l'efficacité de la construction de l'arbre des instances. La courbe suggère un complexité en $O(n^2)$.

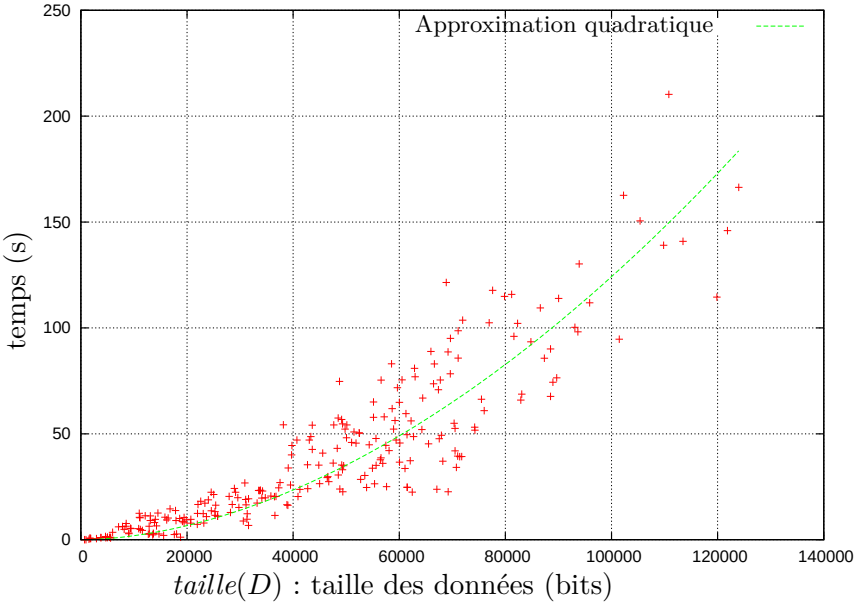


FIG. 4.34: Influence de la taille des données sur la construction de l'arbre des instances.

Le calcul de la taille d'indexation par l'algorithme *taille_indexation* est représenté dans la figure 4.35 page suivante. Il faut noter que plusieurs appels à *taille_indexation* sont effectués dans l'algorithme *min_index*. Certains de ces appels n'aboutissent pas à un calcul de la taille d'indexation car ils nécessitent plus de temps que le temps imparti maximum. Aussi, le graphique 4.35 page ci-contre ne représente que les temps des calculs de taille d'indexation ayant abouti. On remarque que les points sont répartis selon trois distributions : deux linéaires et une quadratique. Cette figure indique donc que certaines indexations sont calculables en un temps linéaire et d'autres en un temps quadratique. Cette différence est due aux différentes relations implémentées dans le schéma et appelées par l'indexation. Pour chaque arc de l'arbre des instances (définition A.7 page 166), la taille de l'image $c_a(source_a)$ est calculée (où c_a est une relation implémentée dans le schéma). Or le seul moyen de calculer cette taille est d'énumérer l'ensemble des éléments de l'image. Pour certaines relations, cette énumération est quadratique et pour d'autres elle est linéaire.

Enfin les résultats de l'efficacité de l'algorithme *min_index* sont représentés comme les temps d'exécution en fonction de la taille des données. La figure 4.36 page ci-contre montre l'efficacité avec utilisation de l'évaluation globale.

Dans le cas de l'évaluation globale, il est difficile de trouver une loi modélisant la répartition des points. On peut seulement remarquer que pour des données de grande taille, le temps d'exécution de l'algorithme est plus long. Alors que cet algorithme appelle des algorithmes de complexité au pire quadratique, aucune loi ne s'en

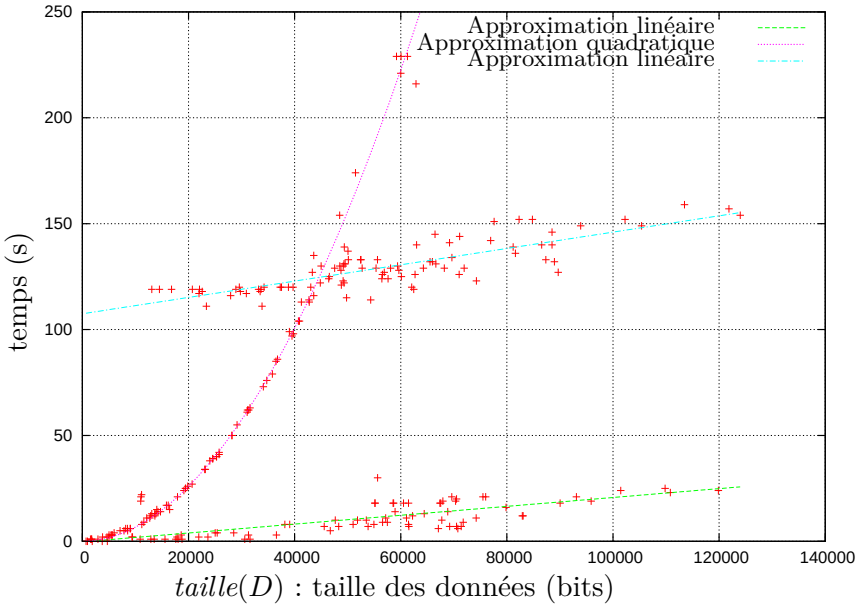


FIG. 4.35: Influence de la taille des données sur le temps de calcul de l'indexation aboutissant à un résultat.

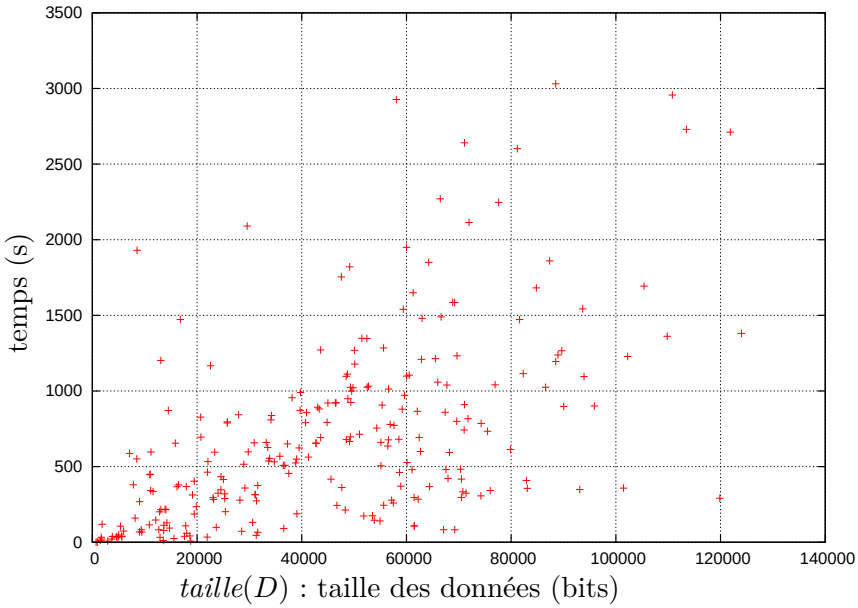


FIG. 4.36: Influence de la taille des données sur le temps de calcul de la minimisation de l'indexation (algorithme *min_index*) avec évaluation globale.

dégage. Cela est dû au fait que *min_index* teste de nombreuses indexations avant de trouver l'indexation qui se calcule en un temps limité. Ainsi, ce graphique nous apprend que trouver une indexation calculable en un temps maximum n'est pas aisé. La méthode de Monte Carlo est une solution simple mais qui montre ici ses limites. L'un des problèmes majeurs de cette approche est l'utilisation de l'évaluation globale qui introduit des relations d'énumération dans la relation de couverture. Les relations d'énumération couvre tous les éléments d'un type. De ce fait l'évaluation de la taille de l'ensemble des éléments à couvrir peut être très importante. Comme il a été remarqué à la fin de la sous-section 4.3.1 page 105, utiliser l'évaluation globale permet de rechercher des liens entre les propriétés des données réellement couvertes par le modèle et les propriétés des données non couvertes par le modèle. Cette expérimentation montre qu'il est difficile d'extraire ce lien.

Dans le cas de l'évaluation locale, la distribution des points est semblable. Cependant, l'exécution est plus rapide : jusqu'à une taille de 250000 bits, la durée maximale est inférieure à 700 secondes.

Les expérimentations sur la spécification d'opérateurs montre combien il est aisé de réutiliser des opérateurs existants. L'interface entre les schémas et l'implémentation est facilitée par divers outils implémentés dans les langages Prolog et Java. De la même façon la structure et les données sont facilement spécifiées par les schémas. La génération de schémas par l'unification est faite automatiquement pour adapter les opérateurs aux données.

L'évaluation générique nécessite la recherche d'indexations pour minimiser la taille de la description des données. L'évaluation de certaines indexations sont au pire quadratiques. Cependant, pour certaines d'entre elles, il est difficile de connaître à l'avance le temps nécessaire pour les exécuter. Ainsi l'évaluation aléatoire a des limites mais qui permettent néanmoins d'évaluer la qualité d'un modèle pour résumer des données.

Conclusion

Dans ce chapitre, nous avons montré que les schémas permettent de structurer les opérateurs de fouille de données (Vautier *et al.*, 2007). La souplesse d'utilisation des schémas autorise soit l'implémentation soit la spécification des opérateurs de fouille de données. Dans un premier temps, cela permet d'extraire automatiquement des modèles à partir d'une masse de données en unifiant les spécifications. Dans un second temps, cela permet d'exécuter les opérateurs de fouille de données existants déjà. Et dans un troisième temps, cela permet d'évaluer la qualité des modèles à résumer les données.

Il reste à expérimenter une telle méthode sur des données réelles. Le chapitre suivant présente une série d'expérimentations sur des données issues de journaux d'alarmes de réseaux de télécommunications.

Chapitre 5

Application à l'analyse de journaux d'alarmes réseau

La profusion des alarmes produites par les systèmes informatiques rend l'analyse des journaux d'alarmes de plus en plus difficile pour l'utilisateur. Les dispositifs qui génèrent les journaux sont, par exemple, des composants réseau, des dispositifs de détection d'intrusions ou le système d'exploitation. Ils sont complexes et difficiles à appréhender complètement par un utilisateur qui n'a, en général, qu'une connaissance partielle de certains d'entre eux. Confronté à cette masse de données, l'utilisateur est, la plupart du temps, contraint de négliger bon nombre des alarmes et de ne s'intéresser qu'à certaines qu'il sélectionne de façon plus ou moins pertinente. Comme on l'a vu dans la section 3.3 page 87, peu d'outils performants permettent de le seconder dans cette tâche risquée. L'augmentation grandissante des échanges d'informations, d'une part, et de la demande en sécurité, d'autre part, réclame la conception et la mise en œuvre de tels outils.

La première section montre comment l'utilisation des schémas peut se révéler judicieuse dans l'analyse de journaux de connexions VPN. La seconde partie illustre l'exploration d'alarmes DDoS en tirant parti des schémas.

5.1 Alarmes faiblement structurées

La plupart des techniques actuelles traitent des données structurées. Or, il reste encore de nombreuses applications où un journal est simplement un texte faiblement structuré. L'utilisateur peut souvent paramétrer, au moins partiellement, les systèmes de génération de journal mais il en a rarement la maîtrise totale. Cet état de fait est en train de changer, au moins pour les applications critiques, en particulier grâce à l'utilisation de XML. De plus, c'est souvent à la suite d'un incident

que l'utilisateur se penche sur un journal. Dans ce cas là, il est trop tard pour paramétrer le système et il lui faut faire avec ce qui existe. Les alarmes ainsi produites sont souvent faiblement structurées. Elles sont théoriquement composées de champs mais les valeurs de ces champs sont souvent simplement copiées dans un fichier sous forme de chaînes de caractères sans marqueur syntaxique. On peut conjecturer que cela est dû en grande partie au fait que l'analyse automatique de ces alarmes n'a pas été prévue lors de leur génération.

Notre approche est appliquée aux alarmes produites par un concentrateur VPN (Virtual Private Network) de marque Cisco dans le réseau de France-Télécom. Un concentrateur est un matériel réseau acceptant un grand nombre de connexions VPN simultanées. Tout d'abord, les alarmes sont présentées et un prétraitement permettant de les structurer en extrayant leurs attributs est introduit dans la sous-section 5.1.1. Le prétraitement aboutit à la génération d'alarmes normalisées que notre plate-forme peut gérer. La sous-section 5.1.2 détaille l'opérateur supplémentaire intégré dans la plate-forme et l'ensemble des modèles générés par celle-ci à partir de tous les opérateurs présents dans la plate-forme.

5.1.1 Prétraitement

Présentation des alarmes

Le journal étudié contient 1.262.117 alarmes issues du concentrateur VPN pendant environ 1 mois. Elles ont le format suivant : *date, nom de concentrateur VPN, type d'alarme, message* ainsi que les champs optionnels *adresse IP du client* et *groupe du client*. Il est aisé d'extraire les attributs de tous les champs, sauf du champ *message* qui n'est pas structuré et qui est pourtant le plus riche. Ces messages sont très variés, ils ne dépendent pas seulement du type de l'alarme associée mais aussi de l'état du concentrateur quand il génère l'alarme. Ils contiennent des attributs de différents types (adresse IP, nombres sous forme hexadécimale ou décimale, nom d'utilisateur, etc.). Le tableau 5.1 page suivante présente quelques exemples d'alarmes du journal. Par exemple, la première indique qu'un incident de type L2TP/46 a eu lieu le 9 mai 2004 à 23h11min2.75s indiquant que le tunnel du client 82.83.84.85[1] a été fermé car il ne répondait plus.

Normalisation des alarmes

Chaque alarme est convertie en une *alarme normalisée* composée d'une date, d'un type et d'un ensemble d'attributs. Cette normalisation est divisée en deux phases. La première phase consiste à extraire les attributs et la seconde phase consiste à faire correspondre les attributs d'alarmes de même type.

Certains attributs d'alarmes sont extraits à partir du message non structuré de l'alarme en s'appuyant sur des expressions régulières définies par l'utilisateur. Comme l'utilisateur n'a pas forcément la connaissance pour définir directement les

[1]Les noms d'utilisateurs et les adresses IP apparaissant dans les exemples sont fictifs.

Date	VPN	Type	Client	Groupe
Message				
05/09/2004 23:11:02.750	VPN-1	L2TP/46	82.83.84.85	?
Tunnel to peer 82.83.84.85 closed, reason: Peer no longer responding				
05/09/2004 23:12:44.530	VPN-1	IKE/24	80.13.14.15	Group [Group1]
Received local Proxy Host data in ID Payload: Address 85.75.65.55, Protocol 17, Port 0				
05/10/2004 07:46:46.950	VPN-2	AUTH/37	20.21.22.23	
User [Unknown] Protocol [SNMP] attempted ADMIN logon.. Status: <ACCESS GRANTED> !				
06/02/2004 22:37:04.510	VPN-1	IKE/41	?	?
IKE Initiator: Rekeying Phase 2, Intf 2, IKE Peer 81.71.61.51 local Proxy Address 85.75.65.55, remote Proxy Address 81.71.61.51, SA (WindowsServer1)				
06/06/2004 15:13:45.640	VPN-1	IKE/41	81.251.55.54	Group [Group1]
IKE Initiator: Rekeying Phase 1, Intf 2, IKE Peer 81.251.55.54 local Proxy Address N/A, remote Proxy Address N/A, SA (N/A)				

Tab. 5.1: Exemples d'alarmes produites par un concentrateur VPN.

bonnes expressions, la définition peut être affinée en plusieurs itérations. Les attributs sont générés à la suite de plusieurs interactions entre l'utilisateur et un processus d'extraction automatique. À partir des résultats d'une extraction automatique, l'utilisateur affine la définition des expressions régulières et le processus est réitéré.

Dans un premier temps, l'utilisateur établit un ensemble $\mathcal{A}$ d'expressions régulières qui représentent la forme générale des attributs à extraire. Par exemple, la figure 5.2 montre trois expressions régulières[2] permettant de rechercher des adresses IP (constituées de 4 groupes de 2 à 3 chiffres), des nombres (hexadécimaux d'au plus 3 chiffres) et des noms d'utilisateurs (« User » suivi du nom de l'utilisateur).

```
- (\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}
        (?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b)(?:$|\W)
- (?:^|\W|0x)([a-fA-F0-9]{3,})(?:$|\W)
- User\s\[(\w*)\]
```

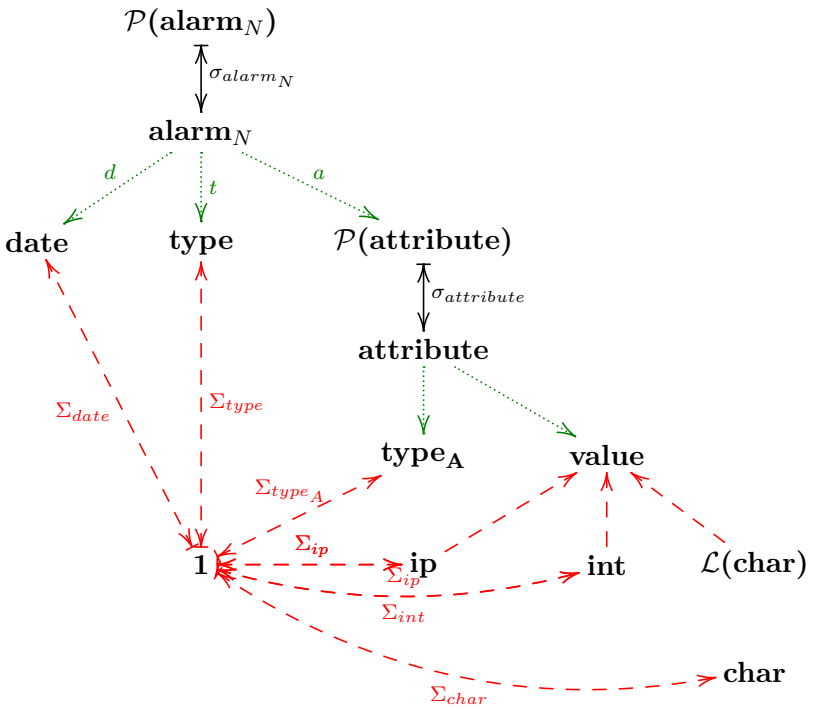
Fig. 5.2: Un ensemble $\mathcal{A}$ de trois expressions régulières.

Le schéma 5.3 formalise la structure des alarmes normalisées. La date et le type sont extraits des champs date et type de l'alarme. Les attributs des champs adresse IP du client, groupe du client et message sont extraits en utilisant $\mathcal{A}$. De cette façon, l'utilisateur exprime approximativement ses connaissances dans $\mathcal{A}$, qu'il va pouvoir ensuite affiner au vu des résultats de l'extraction automatique.

Schéma 5.3 (Schéma d'alarmes normalisées)
Une alarme normalisée ($alarm_N$) est un triplet composé d'une date, d'un type et d'un ensemble d'attributs. Un attribut est composé d'une valeur et de son type.

[2]Le format des expressions régulières est celui du package java.util.regex qui utilise la même syntaxe que Perl.

Le type d'une alarme est un ensemble de codes propres au concentrateur Cisco. La date est codée sur 32 bits au format Unix. Le nœud $\mathcal{L}(char)$ est interprété comme l'ensemble des listes composées d'éléments du nœud *char*.



$$\Sigma_{date} = \mathbf{1} \mapsto \{1, \ldots, 2^{32}\}$$
$$\Sigma_{type} = \mathbf{1} \mapsto \{IKE/41, \ldots, L2TP/46\}$$
$$\Sigma_{type_A} = \mathbf{1} \mapsto \{ip, int, string\}$$
$$\Sigma_{ip} = \mathbf{1} \mapsto \{?, 0.0.0.0, \ldots, 255.255.255.255\}$$
$$\Sigma_{int} = \mathbf{1} \mapsto \{1, \ldots, 2^{32}\}$$
$$\Sigma_{char} = \mathbf{1} \mapsto \{\text{'a'}, \ldots, \text{'z'}, \text{'0'}, \ldots, \text{'9'}, \text{' '}\}$$

L'extraction des attributs n'est pas suffisante pour structurer les alarmes. En effet, on peut trouver les mêmes informations dans deux alarmes de même type sans pour autant que ces informations soient structurées de la même manière. C'est pourquoi, nous introduisons la notion de *signature d'alarmes* qui décrit comment les attributs d'un type d'alarme ont été extraits.

Définition 5.4 (Signature d'alarme)
Une signature est un triplet (t, l, e) où t est un type d'alarme, l une liste de types d'attribut et e une liste de doublets (r, f) formé d'une expression régulière r et d'une fonction $f : l \rightarrow \{0, 1\}$. Les expressions régulières r sont appliquées sur la concaténation des champs adresse IP du client, groupe du client et message pour extraire les attributs. La première expression régulière r de la liste e aboutissant à une extraction est utilisée. $\forall x \in l$ si $f(x) = 1$ alors l'attribut correspondant au

type x est présent dans l'expression r, sinon $f(x) = 0$ et cet attribut est absent de l'expression r. ▪

Par exemple, la signature du tableau 5.5 est générée par l'extraction automatique. Dans cet exemple, la liste l d'une signature est décrite par la colonne « Attributs », ici quatre attributs de type IP. Les lignes de la matrice décrivent les éléments (r, f) de la liste e.

Expression régulière	Attributs			
(IP) remplace une expression régulière	*ip*	*ip*	*ip*	*ip*
_ IKE Initiator: Rekeying Phase 2, Intf 2, IKE Peer (IP) local Proxy Address (IP), remote Proxy Address (IP), SA \(WindowsServer1\)	0	1	1	1
(IP) Group \[Group1\] IKE Initiator: Rekeying Phase 1, Intf 2, IKE Peer (IP) local Proxy Address N/A, remote Proxy Address N/A, SA \(N/A\)	1	0	0	0

TAB. 5.5: Signature (t, l, e) générée pour les alarmes de type t = IKE/41, $l = [ip, ip, ip, ip]$. Les éléments de e sont représentés en ligne dans le tableau.

Une signature d'alarme a deux utilités. D'une part elle synthétise la façon dont les attributs ont été automatiquement extraits pour un type d'alarme donné. D'autre part elle peut être utilisée pour extraire les attributs de ce même type d'alarme lors de l'extraction automatique. Après une extraction automatique, l'utilisateur examine les signatures générées et évalue si l'ensemble $\mathcal{A}$ est suffisant. Si tel n'est pas le cas pour un type d'alarme donné, il peut soit modifier l'ensemble $\mathcal{A}$, en ajoutant ou modifiant des expressions régulières, soit modifier la signature de ce type d'alarme dans $\mathcal{S}$. Ensuite, l'utilisateur relance une extraction automatique basée, pour chaque type d'alarme, soit sur la signature extraite pour S si l'utilisateur l'a décidé, soit sur l'ensemble $\mathcal{A}$ des formes d'attributs.

Le tableau 5.6 page suivante montre les alarmes normalisées construites à partir des alarmes du tableau 5.1 et de l'ensemble $\mathcal{A}$ du tableau 5.2 page 131. Notons que l'attribut 82.83.84.85 est présent dans le champ client et le champ message de l'alarme de type L2TP/46 mais un tel attribut n'est représenté qu'une fois dans l'alarme normalisée correspondante.

L'exemple suivant montre comment l'utilisateur peut interagir avec l'extraction automatique. Les deux alarmes du type IKE/41 du tableau 5.1 page 131 sont normalisées en deux alarmes de même type (voir tableau 5.6 page suivante) dont les listes de types d'attributs sont différentes. Pourtant, toutes les alarmes d'un même type doivent avoir la même liste de types d'attributs. L'examen, par l'utilisateur, de la signature 5.5 lui permet de s'apercevoir que les alarmes de type IKE/41 ont en fait trois attributs (et non un ou quatre). Il peut alors modifier cette signature comme le montre le tableau 5.7 page suivante.

Date	Type	Attributs
5/09/2004 23:11:02.750	L2TP/46	82.83.84.85
05/09/2004 23:12:44.530	IKE/24	80.13.14.15, 85.75.65.55
05/10/2004 07:46:46.950	AUTH/37	20.21.22.23
06/02/2004 22:37:04.510	IKE/41	81.71.61.51,85.75.65.55
06/06/2004 15:13:45.640	IKE/41	81.251.55.54

TAB. 5.6: Alarmes du tableau 5.1 converties en alarmes normalisées.

Plusieurs interactions entre l'utilisateur et l'extraction automatique peuvent être nécessaires pour générer des alarmes normalisées utilisables dans l'étape de construction de transactions. Ceci permet à l'utilisateur d'introduire, de façon aisée et incrémentale, ses connaissances sur le journal et d'améliorer l'abstraction du journal en vue de la phase suivante de construction des transactions.

Expression régulière *(IP) remplace une expression régulière*	Attributs *ip*	*ip*	*ip*
_ IKE Initiator: Rekeying Phase 2, Intf 2, IKE Peer (IP) local Proxy Address (IP), remote Proxy Address (IP), SA \(WindowsServer1\)	1	1	1
(IP) Group \[Group1\] IKE Initiator: Rekeying Phase 1, Intf 2, IKE Peer (IP) local Proxy Address N/A, remote Proxy Address N/A, SA \(N/A\)	1	0	0

TAB. 5.7: Signature modifiée par l'utilisateur pour les alarmes de type IKE/41 : $l = [ip, ip, ip]$.

Les 1.262.117 alarmes du journal du concentrateur VPN ont été entièrement normalisées par l'extraction des attributs. Trois passages de l'extraction automatique ont été effectués. 37 signatures générées automatiquement ont dû être modifiées et 4 formes d'attributs ont été ajoutées aux expressions régulières du tableau 5.2 page 131. Les signatures sont visibles en annexe C page 171.

5.1.2 Intégration dans la plate-forme

Les alarmes VPN normalisées ont été traitées par notre plate-forme. Le schéma des données VPN correspond au schéma 5.3 page 131. La base des opérateurs est composée des schémas suivants (les paramètres nécessaires à ces opérateurs sont aussi introduits) :

1. *Schéma du clustering en deux dimension.* Le schéma génère des clusterings à deux dimensions en utilisant l'algorithme k-means. Cet opérateur nécessite le nombre de clusters k à extraire.

2. *Schéma du clustering en une dimension.* Ce schéma correspond au schéma 4.27 page 118 sans la dimension Y. Cet opérateur nécessite le nombre de clusters k à extraire.
3. *Schéma des séquences.* Les séquences générées ne contiennent que deux évènements. Les données nécessaires à cet algorithme sont un ensemble d'éléments datés et typés. Cet opérateur construit toutes les séquences de deux évènements qui ont au moins une occurrence dans l'ensemble d'évènements. La différence maximum de temps entre les deux dates des deux évènements d'une occurrence est fixée à l'aide d'un paramètre *maxGap*. Ensuite, l'opérateur ordonne les séquences de la plus fréquente à la moins fréquente. Cet opérateur retourne les k séquences les plus fréquentes. La valeur de k est donné en paramètre.
4. *Schéma de généralisation de graphes.* Le schéma 4.5 page 99 extrait les nœuds ayant les plus hauts degrés. Il retourne les k nœuds de plus haut degré avec k donnée en paramètre.

L'analyse des schémas opérationnels obtenus en unifiant ces schémas avec le schéma spécifiant les données VPN a fait apparaître que la liste des attributs n'était jamais utilisée. C'est pourquoi, nous avons décidé de concevoir un nouvel algorithme qui gère une telle liste d'attributs pour mettre en relation les alarmes partageant des attributs communs sous forme de transactions. Le fonctionnement de cet algorithme et l'opérateur associé sont présentés dans la partie suivante.

Construction de transactions

Après avoir analysé manuellement le journal d'alarmes normalisées, nous avons remarqué que des alarmes normalisées se caractérisent par une proximité temporelle et l'égalité des valeurs de certains attributs communs. En effet, une connexion VPN entre un client et un concentrateur se compose de *transactions* réseaux qui provoquent des rafales d'alarmes.

Les alarmes agrégées dans une transaction liée à un client partagent des attributs propres à ce client. Or, a priori, l'utilisateur ne connaît pas ces attributs. C'est pourquoi, dans un premier temps *tous* les attributs sont considérés de manière à obtenir des transactions primitives. Celles-ci vont ensuite servir de base à la construction, assistée par l'utilisateur, des transactions.

Les alarmes normalisées sont regroupées. Les alarmes sont corrélées temporellement, en se basant sur la date des alarmes, et rationnellement, en se basant sur la similarité entre attributs autres que la date. On obtient ainsi une partition du journal. L'objectif de cette construction est de réduire les informations présentes dans la synthèse présentée à l'utilisateur.

Partition du journal en transactions primitives À partir de l'ensemble des alarmes normalisées extraites du journal, les transactions primitives sont extraites. Elles respectent les définitions suivantes :

Définition 5.8 (α-transaction)
- Un ensemble $\{x\}$ composé d'une alarme normalisée x est une α-transaction.
- Soient T une α-transaction et x une alarme. l'ensemble $T \cup \{x\}$ est une α-transaction si et seulement si il existe une alarme y de T qui possède au moins n_a attributs en commun avec l'alarme x et si l'intervalle de temps qui les sépare est inférieur à *maxGap* (l'écart temporel maximum). ▪

Définition 5.9 (Transaction primitive)
Une transaction primitive est une α-transaction maximale (il n'existe pas d'autre α-transaction qui la contienne). ▪

Les paramètres n_a et *maxGap* sont fixés arbitrairement par l'utilisateur. Le tableau 5.13 page ci-contre montre un exemple de découpage du journal en deux transactions primitives. Les attributs partagés qui permettent la construction des transactions primitives sont repérés par une marque semblable en exposant ($\diamond$,* ,$^+$).

Proposition 5.10 (Unicité d'un partionnement en transactions)
Pour tout journal d'alarmes normalisées et des paramètres n_a et maxGap donnés, il existe un unique partitionnement composé d'un ensemble de transactions primitives représentant ce journal. ▪

Preuve. Raisonnement par l'absurde.
Supposons qu'il existe deux partitionnements P_1 et P_2 différents :

$$\forall t \in P_i, \text{t est une transaction primitive (avec i = 1,2)} \tag{5.11}$$

Si P_1 et P_2 sont différents alors il existe une alarme a appartenant à une transaction primitive $t_1 \in P_1$ et à une transaction primitive différente $t_2 \in P_2$:

$$\exists a \in t_1, a \in t_2 \text{ tels que } t_1 \in P_1, t_2 \in P_2 \text{ et } t_1 \neq t_2 \tag{5.12}$$

D'après (5.12) et la définition 5.8 page précédente

$$\forall b \in t_2 \setminus t_1, [t_1, b] \text{ est une } \alpha\text{-transaction}$$

Ce qui implique que l'union de t_1 et t_2 est une α-transaction.
Donc t_1, t_2 ne sont pas des transactions primitives (α-transaction maximale) ce qui contredit (5.11). □

Le partitionnement dépend fortement des paramètres n_a et *maxGap*. La méthode peut générer un partitionnement comprenant autant de transactions primitives que d'alarmes normalisées jusqu'à un partitionnement constitué d'une seule transaction primitive couvrant tout le journal. Il faut trouver un juste milieu en vue d'extraire des transactions qui ne soient ni trop proches des données ni trop générales. C'est le rôle de la phase suivante qui permet de construire des modèles de transactions à partir des transactions primitives.

Date	Type	Attributs
05/13/2004 14:15:50.910	IKE/25	82.81.80.79*
05/13/2004 14:15:50.910	IKE/24	82.81.80.79*, 85.75.65.55
05/13/2004 14:15:50.910	IKE/66	82.81.80.79*
05/13/2004 14:15:50.910	IKE/75	82.81.80.79*, 3600
05/13/2004 14:15:50.960	IKE/49	82.81.80.79*, 53e9fac2
05/13/2004 14:15:50.960	IKE/120	82.81.80.79*, 5da6fd05
05/13/2004 14:15:53.960	IKE/170	82.81.80.79*, d81626c
05/13/2004 14:16:18.140	IKEDBG/64	217.128.126.9*
05/13/2004 14:16:18.450	IKE/172	217.128.126.9*
05/13/2004 14:16:18.460	AUTH/12	483$^\diamond$
05/13/2004 14:16:18.560	AUTH/41	217.128.126.9*, 483$^\diamond$
05/13/2004 14:16:18.560	AUTH/13	483$^\diamond$
05/13/2004 14:16:18.630	IKE/79	217.128.126.9*, 6cf24368000000000f41
05/13/2004 14:16:23.320	AUTH/12	484$^+$
05/13/2004 14:16:23.430	AUTH/4	217.128.126.9*, 10.169.25.80, 484$^+$, user2

TAB. 5.13: Deux transactions primitives partitionnant une partie du journal.

Construction des modèles de transactions Le but de cette étape est d'abstraire les attributs des alarmes de manière à regrouper des transactions primitives impliquant la même liste de types d'alarmes. Avant de présenter la construction des modèles de transactions, les définitions de modèle d'alarme et de modèle de transaction sont présentées.

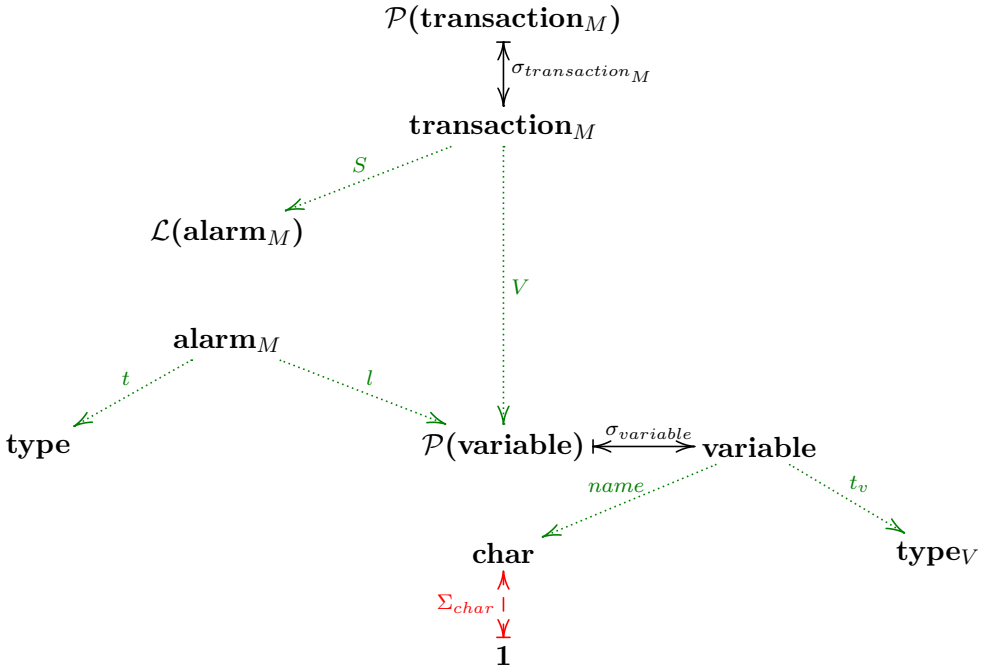
Définition 5.14 (Modèle d'alarme)
Un modèle d'alarme est un couple (t, l) où t est un type d'alarme et l est un ensemble de variables. Une variable $v \in l$ possède un type t_v. ▪

Il faut noter que le temps n'est pas représenté explicitement dans un modèle d'alarme. Le temps est considéré comme un élément permettant d'ordonner les alarmes au niveau d'un modèle de transaction. Un modèle d'alarme couvre des alarmes appelées instances de ce modèle d'alarme. Par exemple, l'alarme (date = 05/13/2004 14:19:46.940, type = PPPDECODE/16, attributs = [80.14.52.129, user1, 85.75.65.55]) est une instance du modèle d'alarme (t = PPPDECODE/16, $l = [X, Y, Z]$) où X et Z sont des variables de type adresse IP qui correspondent respectivement aux adresses 80.14.52.129 et 85.75.65.55 et Y est une variable de type chaîne de caractères qui correspond à l'attribut user1. L'ensemble des valeurs que peuvent prendre les variables est restreint par leur domaine associé. La définition d'un modèle de transaction est donnée dans la définition suivante et illustrée dans le schéma 5.16 page suivante.

Définition 5.15 (Modèle de transaction)
Un modèle de transaction est un doublet (S,V) où S est une liste de modèles d'alarmes et V est l'ensemble des variables apparaissant dans les modèles d'alarmes

de la transaction. Un modèle de transaction couvre des transactions appelées instances de transaction. ▪

Schéma 5.16 (Schéma de modèle de transaction)
Un modèle de transaction de type $transaction_M$ est une liste de modèles d'alarmes ($alarm_M$) associée à un ensemble de variables. Une liste de modèles d'alarmes est représenté par le nœud $\mathcal{L}(alarm_M)$. Une variable ($variable$) est composée d'un type ($type_V$) et d'un nom ($char$).



▪

La deuxième transaction du tableau 5.13 page précédente est une instance du modèle de transaction du tableau 5.17 page ci-contre. Les lignes du tableau sont les éléments de S et donc des modèles d'alarmes. La dernière ligne correspond à V et décrit les variables et leur domaine. Ce dernier comporte une variable X partagée par 5 modèles d'alarmes. Une telle variable apparaissant dans plusieurs modèles d'alarmes de la transaction contraint les attributs correspondant des instances à posséder un type (domaine) identique.

La complexité de la relation de couverture et la taille du schéma font que le schéma du nouvel opérateur introduit n'est pas décrit ici. Cet opérateur génère des transactions à partir d'un ensemble d'évènements où un évènement est composé d'une date, d'un type et d'une liste d'attributs. Le partitionnement du journal est effectué à partir de deux paramètres $maxGap$ et n_a qui représentent les conditions sous lesquels deux alarmes sont unies dans une même transaction. Enfin l'opérateur génère des modèles de transactions décrits dans le schéma 5.16.

Type	Variables
IKEDBG/64	X
IKE/172	X
AUTH/12	Z
AUTH/41	X, Z
AUTH/13	Z,
IKE/79	X, W
AUTH/12	A
AUTH/4	X, B, A, C
$X, B \in ip, Z, A \in int, C \in string$	

TAB. 5.17: Modèle de transaction généralisant la seconde transaction du tableau 5.13.

Extraction de modèles

L'utilisation de la plate-forme pour résumer des données est composé de trois étapes : (1) analyser les schémas opérationnels obtenus par unification automatique du schéma des données avec les opérateurs de la plate-forme complétés de l'opérateur de génération de transactions, (2) exécuter ces opérateurs sur les données en définissant un ensemble de paramètres pour chacun des opérateurs, (3) représenter les qualités des modèles extraits d'une façon synthétique.

Schémas opérationnels Les schémas de la base sont décrits au début de la sous section 5.1.2 page 134. Ils sont complétés par le schéma de la construction des transactions. Ces schémas ont pu s'adapter automatiquement avec le schéma des alarmes normalisées pour construire quatre schémas opérationnels décrits ici :

- *Séquence.* L'adaptation du schéma des séquences à deux évènements avec les alarmes normalisées a généré deux schémas opérationnels. Le premier [1] considère le type des évènements comme le type des alarmes. Le second [2] considère le type des évènements comme l'ensemble des attributs de l'alarme. Cette dernière adaptation a généré des modèles dont la qualité n'était pas intéressante (leur taux de compression était supérieur à 1) et qui ne sont pas représentés dans la suite.
- *Clustering en une dimension* [3]. Le schéma du clustering en une seule dimension a été adapté pour regrouper les alarmes selon leur date.
- *Transaction* [4]. L'adaptation du schéma des transactions est unique et est relativement simple puisque l'opérateur utilise les trois attributs des alarmes normalisées : date, type et ensemble d'attributs.

Les autres schémas (clustering en deux dimensions et généralisation de graphe) n'ont pas pu s'unifier. Le clustering en deux dimensions nécessite deux attributs numériques et la généralisation de graphes par extraction de nœuds de haut degré

nécessite que la notion d'un arc soit définie. Pour chacun d'eux, la spécification des alarmes normalisées ne répond pas à ces exigences.

Paramètres et exécution L'extraction de modèles sur ces données a été réalisée sur les 1000 premières alarmes du journal. Cette contrainte sur le nombre d'alarmes vient des limites de temps et d'espace mémoire pour évaluer les modèles. Pour chaque schéma opérationnel, les paramètres des opérateurs ont été fixés sans connaissance particulière mais dans le but d'obtenir des résultats. En effet, pour des raisons d'efficacité, nous avons sélectionné les paramètres qui permettaient à l'évaluation de tenir en mémoire.

- *Séquence.* Les paramètres des séquences sont $k = 15$ et $maxGap = 20$.
- *Clustering en une dimension.* Le paramètre k du nombre de clusters a été fixé à plusieurs valeurs : $\{3, 4, 5, 10, 20, 30, 40, 50\}$.
- *Transaction.* Enfin les transactions ont été générées pour toutes les combinaisons $n_a = \{1, 2\}$ et $maxGap = \{20, 30, 50, 100, 1000\}$.

De plus, pour des raisons d'efficacité dues aux problèmes de l'efficacité temporelle de l'évaluation globale (sous-section 4.4.2 page 122), l'évaluation locale a été utilisée. De cette façon la qualité des modèles correspond à leur taux de compression et leur taux de couverture définis dans la sous-section 4.3.1 page 105.

Les trois structures de modèle proposées sont des ensembles d'éléments. C'est pourquoi, nous avons évalué chacun de ces éléments en plus d'évaluer les ensembles pour mieux analyser les modèles obtenus.

Analyse des modèles Les résultats sont résumés dans la figure 5.18 page suivante et le tableau 5.19 page 142. Des modèles ont été obtenus avec un taux de compression supérieur à 1. Autrement dit, ces modèles sont plus complexes que les données qu'ils couvrent, ils ne sont donc pas intéressants. Le point optimal de ce graphique est le point $p = (0, 1)$ puisqu'il correspond à une couverture totale et à une compression totale. Plus les caractéristiques d'un modèle sont *proches* de p plus le modèle est intéressant. Cette notion de distance peut être modulée en fonction de ce que l'utilisateur recherche : des modèles qui compressent très bien une petite partie des données ou des modèles qui compressent moyennement mais une grande partie des données. Le point $q = (0, 0)$ est aussi intéressant puisqu'il symbolise les modèles qui couvrent très peu d'informations mais qui les compressent parfaitement. Compresser un ensemble relativement grand de données redondantes est plus simple que de résumer un ensemble plus petit de données, qu'elles soient redondantes ou non.

Globalement, on voit que les modèles sous forme de transactions 4 résument mieux le journal (sont plus proches de p) que des clustering 3 ou des séquences 2 puisque leur taux de compression est plus faible et leur taux de couverture est plus important. L'ensemble des transactions situé au point $(0.26, 0.99)$ couvre quasiment tout le journal mais a un taux de compression plus important que la meilleure des transactions $(0.06, 0.51)$. On voit aussi que tous les clusterings 1 couvrent 37 % des

données. En effet, d'après la technique employée, tous les points utilisés pour l'apprentissage sont couverts par un cluster généré. Cela veut tout simplement dire que l'attribut date des données représente 37 % de la quantité d'informations présente dans les données. De plus, on voit que le clustering avec $k = 3$ semble suffisant pour regrouper l'information. Enfin l'ensemble des séquences semble intéressant à visualiser puisqu'il couvre 25% des données avec un taux de compression inférieur à 40 %.

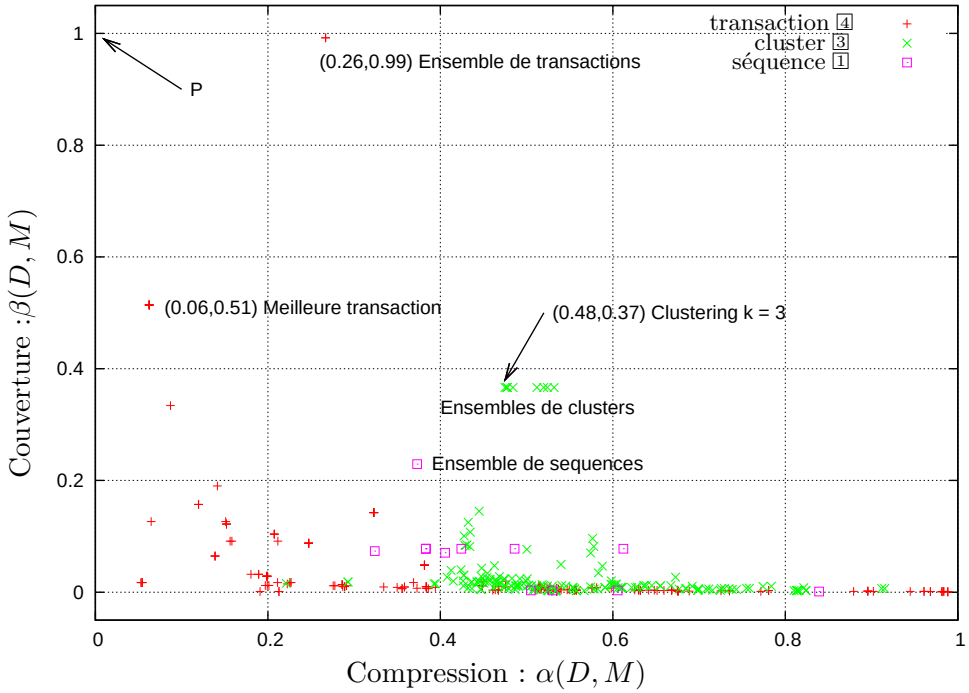


FIG. 5.18: Compression et couverture des modèles générés à partir de 1000 alarmes VPN.

Les expérimentations de prétraitement sur l'extraction d'attributs montrent que les modifications de signatures font apparaître qu'un dispositif d'inférence grammaticale (Cicchello et Kremer, 2003) pourrait être utile. En effet, en effectuant une corrélation entre les expressions régulières d'une même signature, il serait possible d'associer automatiquement les attributs correspondants aux expressions régulières.

Les expérimentations sur la plate-forme montrent que le graphique représentant les modèles en fonction de leur compression et de leur couverture constitue un outil s'avérant extrêmement utile pour l'utilisateur dans le but de choisir quel modèle analyser manuellement.

Type de modèle	Paramètres	Compression	Couverture
Ensemble de transactions	$maxGap = 20, n_a = 1$	0.26	0.99
Transaction	$maxGap = 20, n_a = 1$	0.06	0.51
Transaction	$maxGap = 20, n_a = 1$	0.09	0.33
Ensemble de clusters	$k = 3$	0.48	0.37
Ensemble de clusters	$k = 4$	0.48	0.37
Ensemble de clusters	$k = 5$	0.481	0.37
Ensemble de clusters	$k = 10$	0.48	0.37
Ensemble de clusters	$k = 20$	0.52	0.37
Ensemble de clusters	$k = 30$	0.51	0.37
Ensemble de clusters	$k = 40$	0.52	0.37
Ensemble de clusters	$k = 50$	0.53	0.37
Cluster	$k = 3$	0.44	0.15
Ensemble de séquences	$k = 15, maxGap = 20$	0.37	0.23
Séquence	$k = 15, maxGap = 20$	0.32	0.07

TAB. 5.19: Compression et couverture des « meilleurs » modèles générés à partir de 1000 alarmes VPN.

5.2 Alarmes DDoS

Internet a été développé dans un but fonctionnel et non de sécurité. Il offre à ses utilisateurs des mécanismes de communication rapides et à faible coût, des protocoles de haut niveau qui assurent une distribution des messages en un certain temps et une qualité de service minimale garantie. Son organisation distribuée implique qu'aucune politique commune ne peut être imposée entre ses participants. Par conséquent, cette organisation ouvre la voie à des attaques de type « déni de service distribué » (DDoS - Distributed Denial of Service) (Mirkovic et Reiher, 2004).

Une attaque DDoS consiste à submerger un équipement informatique cible (le plus souvent un serveur) avec un grand volume de données à partir de plusieurs sources. Ce genre d'attaques est en constante augmentation et leur détection devient de plus en plus difficile pour plusieurs raisons : premièrement, une attaque DDoS doit être découverte en ligne et deuxièmement la distinction entre une attaque DDoS et un trafic normal mais très volumineux est difficile à établir.

Pour ces raisons, les experts réseau ont besoin d'une bonne compréhension des flux de données sur internet pour identifier les attaques. Les informations de flux sont générés par des sondes Netflow (voir sous-section 3.3.1 page 88) qui agrègent les informations sur la quantité de données transmises dans les flux circulants en un point d'observation. Lorsqu'elles sont connectées à des routeurs importants d'internet, ces sondes génèrent énormément de données.

La structure d'internet sous la forme d'un graphe et la distribution sur ce graphe des flux à analyser implique la prise en compte de la notion de graphe lors de l'analyse. C'est pourquoi, nous avons mis en place un algorithme simple généralisant

un graphe étiqueté.

Tout d'abord nous présentons les attaques DDoS et les données fournies par France-Télécom issues des sondes Netflow. Puis, nous détaillons comment ces alarmes ont pu être gérées par notre plate-forme.

5.2.1 Présentation des attaques DDoS et des données

Stratégies des attaques DDoS

Du point de vue de l'attaquant, la stratégie générale d'une attaque DDoS (figure 5.20 page suivante) est tout d'abord de recruter des machines appelées « esclaves » ou « zombies ». Ce processus est généralement réalisé automatiquement à travers un balayage des trous de sécurité de machines distantes. Les machines vulnérables repérées sont alors infectées avec le code correspondant à l'attaque et deviennent ainsi des zombies. Un zombie peut à son tour balayer des machines distantes afin de les infecter à leur tour. La seconde étape consiste à générer l'attaque proprement dite. Tous les zombies envoient des paquets réseau à la cible. La cible, submergée par une quantité énorme de paquets, ne peut plus répondre aux connections normales des utilisateurs. C'est ce qu'on appelle un déni de service.

Variations de l'attaque L'attaquant dispose d'un ensemble de zombies qui constitue un réseau capable d'attaquer une cible. Si les zombies sont repérés alors l'attaque s'arrête car la cible peut facilement filtrer les paquets provenant des zombies et de plus l'attaquant risque de perdre une partie de ses ressources d'attaque. Pour rendre plus difficile cette détection, plusieurs techniques d'attaque sont possibles. Elles consistent à faire varier le débit de l'attaque, à masquer les zombies en utilisant des serveurs intermédiaires ou encore à varier la structure des paquets envoyés à la cible.

Le débit d'une attaque classique augmente très rapidement jusqu'à ce que les services de la cible s'interrompent. Une autre technique est d'augmenter progressivement le débit dans le but de dégrader progressivement la qualité des services. Enfin, la plus élaborée des techniques consiste à faire varier le débit en fonction du comportement de la cible. Cela peut consister à générer des attaques par impulsions qui impliquent des coupures momentanées de service. Si ces impulsions sont réparties entre plusieurs groupes de zombies alors la cible est continuellement submergée par différents groupes de zombies et connaît un déni service.

Une technique avancée de DDoS est le *DRDoS* (Distributed Reflection Denial of Service). Elle consiste, pour les zombies, à envoyer des demandes de connections à des serveurs importants d'internet, dits *réflecteurs*, en usurpant l'adresse de la cible. Ainsi, les serveurs renvoient une réponse à la demande de connexion à la cible, qui de ce fait est submergé. La figure 5.20 page suivante illustre ce procédé. Pour la cible, il est d'autant plus difficile d'identifier les zombies que pour retrouver leurs adresses il faut revenir sur les cheminements parcourus par les messages. La reconstitution de ce cheminement est relativement complexe.

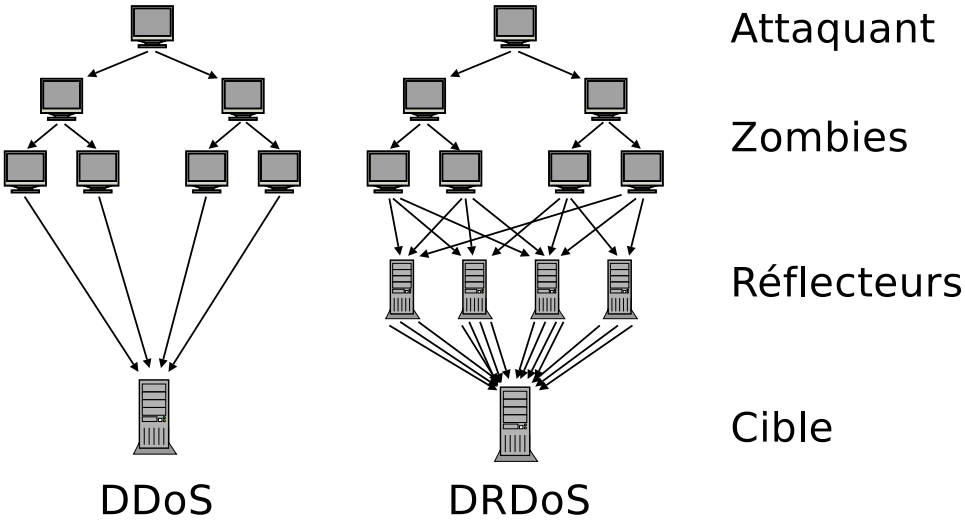


FIG. 5.20: Stratégies d'attaque DDoS et DRDoS.

Les paquets réseau envoyés sont structurés en vue de communiquer entre deux points d'internet. Ils ont été découpés en couche et en service. Par exemple, la commande ping envoie une requête ICMP_ECHO. Si la machine distante ne répond pas alors il se peut que la communication ne soit pas possible. Une attaque peut être constituée par l'envoi de beaucoup de requêtes ICMP_ECHO des zombies vers la cible. Cependant, la détection d'une telle attaque est relativement triviale. De manière à échapper à la détection, les zombies peuvent envoyer d'autres types de paquets. Il en existe énormément mais ceux qui sont le plus utilisés sont liés aux couches les plus basses du réseau : TCP, IP et ICMP. Par exemple, les envois de paquets TCP mal structurés appelés TCP_NULL ou encore des demandes de connexions par paquets SYN sont souvent constatés.

Les attaques DDoS et DRDoS utilisent l'architecture distribuée d'internet pour submerger une cible de paquets réseau. Des techniques de plus en plus évoluées alimentent ce procédé et procurent à l'attaquant un arsenal capable d'empêcher la détection de l'attaque. Cependant beaucoup d'informations sur ces attaques circulent par les routeurs d'internet. Une analyse des flux circulant sur ces routeurs permet de mieux comprendre les mécanismes et d'appréhender plus facilement de nouvelles attaques. La partie suivante présente les données accessibles à partir de ces routeurs.

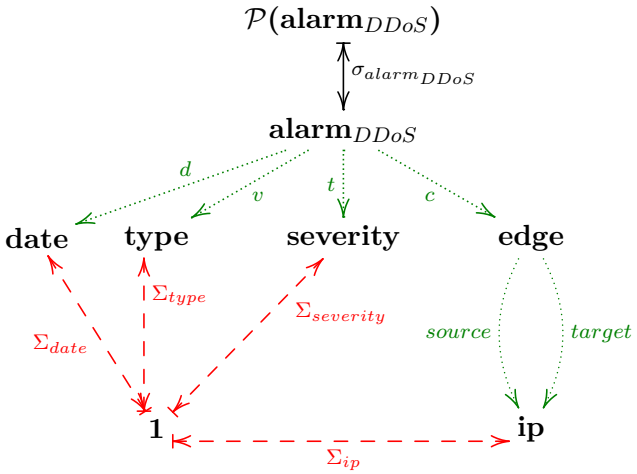
Alarmes Netflow agrégées

Le trafic normal et les attaques génèrent des flux qu'il est possible d'analyser en plaçant des sondes Netflow sur les routeurs constituant l'« épine dorsale » d'internet.

Cependant, la quantité de flux transitant sur ces routeurs est gigantesque. Il n'est pas question de sauvegarder les alarmes Netflow qui occuperaient un volume beaucoup trop important. C'est pourquoi, une première étape consiste à agréger ces alarmes en vue de simplifier l'information. La seconde étape est leur analyse introduite dans la sous-section suivante.

L'agrégation des alarmes Netflow est directement implémentée dans le système gérant les sondes. Les alarmes agrégées, appelées *alarmes DDoS*, sont ensuite stockées. Leur structure est représentée dans le schéma 5.21. Une alarme DDoS résume un flux important de paquets entre une source et une cible. Une alarme DDoS est un quadruplet composé d'une date, d'un type, d'une sévérité et d'un arc composé d'une source et d'une cible. Les sources et les cibles sont des adresses IP. Le type d'une alarme DDoS est lié à la structure des paquets constituants le flux d'informations. Dans un souci d'efficacité de l'agrégation des alarmes Netflow, ces types sont restreints à SYN, TCP_NULL et ICMP_ECHO. La sévérité d'une alarme est liée au débit du flux : plus le débit est élevé plus la sévérité est importante.

Schéma 5.21 (Schéma des alarmes DDoS)



$$\Sigma_{date} = \mathbf{1} \mapsto \{1, \ldots, 2^{32}\}$$
$$\Sigma_{type} = \mathbf{1} \mapsto \{SYN, TCP\_NULL, ICMP\_ECHO\}$$
$$\Sigma_{severity} = \mathbf{1} \mapsto \{1, 2, 3\}$$
$$\Sigma_{ip} = \mathbf{1} \mapsto \{0.0.0.0, \ldots, 255.255.255.255\}$$

.

Les attaques DDoS sont des phénomènes complexes à détecter. D'une part, la quantité de données à analyser est gigantesque et d'autre part, la connaissance des mécanismes impliqués est incertaine puisque les attaquants se préoccupent de dissimuler ces mécanismes. Ce constat implique qu'une meilleure connaissance des flux réseau circulant sur internet est indispensable afin de pouvoir mieux cibler les analyses à effectuer. La sous-section suivante se propose de définir un méthode permettant de simplifier des flux réseau.

5.2.2 Intégration dans la plate-forme

Les alarmes DDoS intègrent trois notions particulières : la notion de graphe (*edge*), la notion temporelle (*date*) et la notion de caractéristiques (*severity* et *type*). La prise en compte de toutes ces notions dans un même modèle est difficile à mettre en œuvre. En effet, la plupart des méthodes s'intéressent à un de ces aspects mais négligent souvent les autres. Par exemple, les opérateurs de la plate-forme présentés au début de la sous-section 5.1.2 page 134 ne se préoccupent que d'un ou deux attributs. De la même façon que pour VPN, nous avons défini un nouvel algorithme spécifié sous la forme d'un opérateur et ajouté dans la base des opérateurs pour faciliter le résumé des alarmes DDoS. Cet algorithme est basé sur une simplification de graphe avec étiquette sur les arcs.

Dans une première partie, nous introduisons un algorithme relativement simple qui généralise un graphe. Ensuite, nous présentons quelques résultats sur l'extraction de connaissances possible à partir de ces données grâce à la plate-forme.

Simplification de graphe

Nous avons visualisé les alarmes DDoS sous la forme d'un graphe (figures 5.22 et 5.23 page ci-contre) où un nœud représente une adresse IP et un arc une alarme DDoS. Nous avons remarqué que des nœuds pouvaient être regroupés pour plusieurs raisons : ils sont la source d'alarmes vers une cible identique au même instant ou au contraire ils sont la cible d'alarmes de plusieurs sources au même instant. C'est pourquoi, nous proposons un algorithme qui regroupe les nœuds ayant des caractéristiques communes relativement à leurs arcs.

Nous considérons que les données sont constituées en graphe dont les arcs sont étiquetés de manière à généraliser la structure sur laquelle l'algorithme présenté est exécuté. Le schéma 5.29 page 149 décrit la structure de ce graphe. La figure 5.25 page 148 présente un exemple de simplification de graphe avec prise en compte de l'étiquette des arcs.

Schéma 5.24 (Graphe étiqueté)
Le nœud *ledge* représente un arc étiqueté et le nœud *edge* représente un arc non étiqueté. Un arc est composé d'un nœud source représenté par la projection s et un

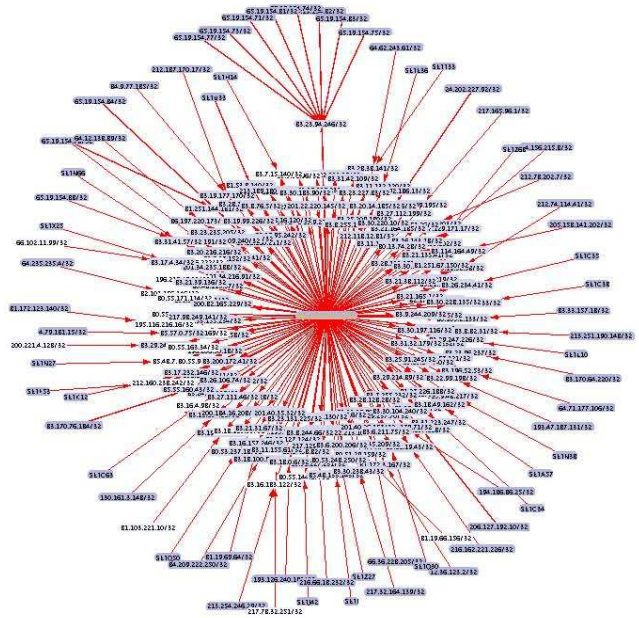


FIG. 5.22: Visualisation d'alarmes DDoS. Une adresse IP est la cible de plusieurs alarmes.

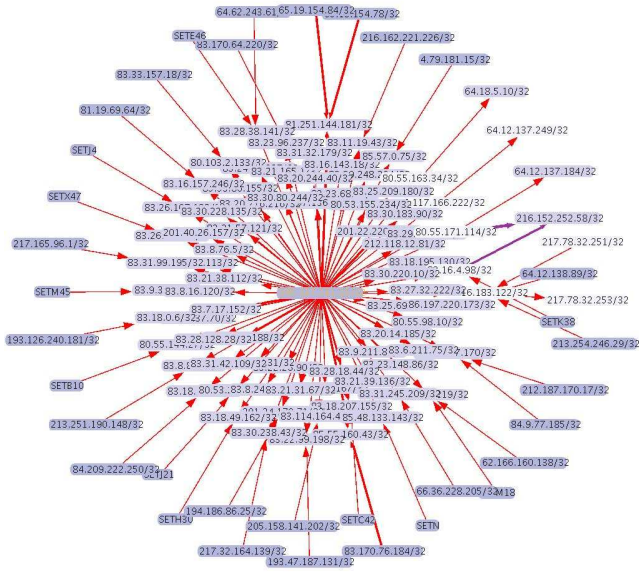


FIG. 5.23: Visualisation d'alarmes DDoS. Une adresse IP est la source de plusieurs alarmes.

nœud cible représenté par la projection t.

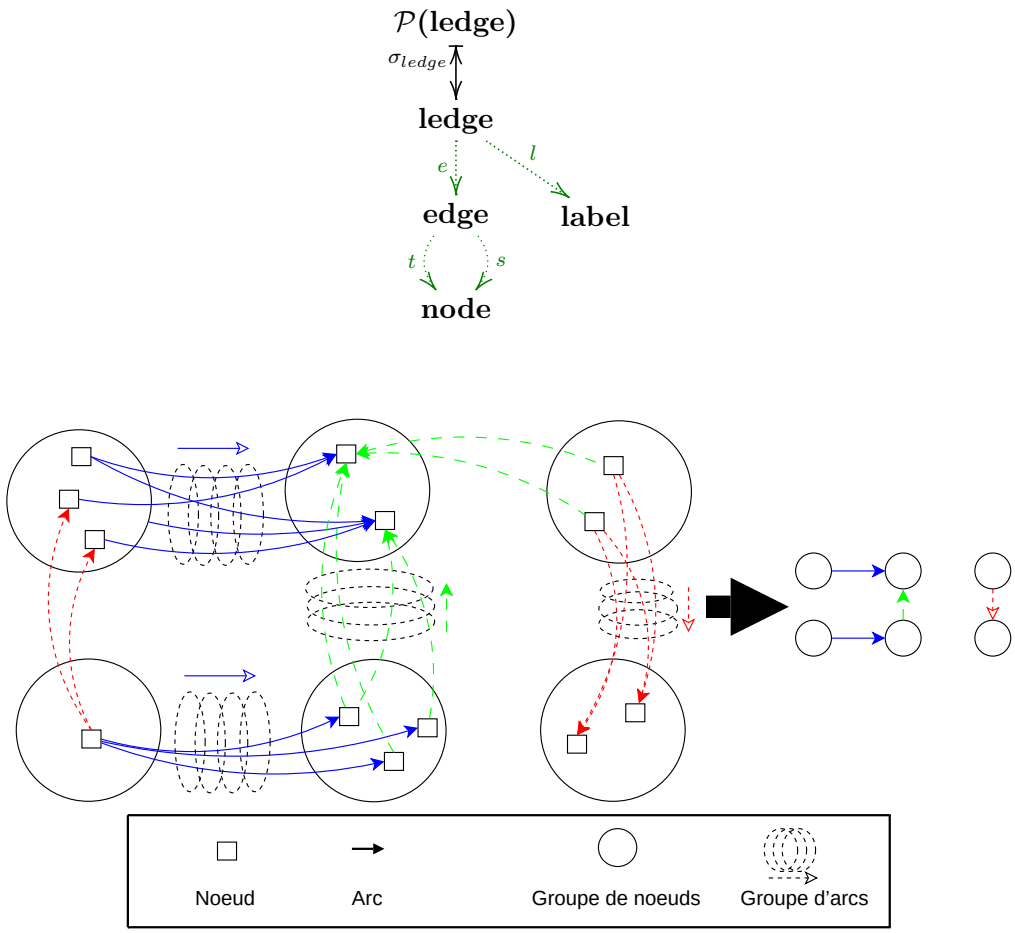


FIG. 5.25: Simplification d'un graphe avec en paramètres : $n_m = 2$ et $a_m = 3$. L'étiquette d'un arc est représenté par sa couleur.

La façon de définir un groupe de nœuds est similaire à la façon de définir une transaction (voir les définitions 5.8 et 5.9 page 136). Cette méthode permet de regrouper les nœuds proches en fonction de leurs arcs. L'ensemble des arcs d'un nœud est défini tout d'abord. Ensuite la définition du graphe simplifié est présentée.

Définition 5.26 (Ensemble des arcs étiquetés d'un nœud)
Soient un graphe G et un nœud n. L'ensemble des arcs $EA(n)$ du nœud n est l'ensemble de tous les triplets (A, m, l) tels que si $A = s$, m est le nœud source d'un arc étiqueté l de cible n et si $A = t$, m est le nœud cible d'un arc étiqueté l de source n dans le graphe G.

Définition 5.27 (α-groupe de nœuds)
Soit un graphe $G = (G_0, G_1)$ (voir la définition de graphe 1.1 page 11)

- Un ensemble $\{x\}$ composé d'un nœud est un α-groupe de nœuds.
- Soient M un α-groupe et x un nœud. L'ensemble $M \cup \{x\}$ est un α-groupe si et seulement si il existe un nœud y de M qui possède au moins n_m arcs en commun avec le nœud x :

$$y \in M, |EA(x) \cap EA(y)| \geq n_m \quad .$$

Nous ne le prouvons pas formellement, mais un raisonnement identique à la preuve de la proposition 5.10 page 136 montre que pour un ensemble G_0, il existe un unique partitionnement $p(G_0, n_m) \in \mathcal{P}(G_0)$ des nœuds en groupes maximaux (aucun α-groupe ne les contient). À partir de cette définition, le graphe simplifié d'un graphe de base est défini.

Définition 5.28 (Graphe simplifié)
Soient $G = (G_0, G_1)$ un graphe, n_m le nombre d'arcs minimum pour regrouper les nœuds et a_m le nombre d'arcs minimum pour générer un arc dans le graphe simplifié. Le graphe simplifié $G' = (G'_0, G'_1)$ du graphe de base G en fonction des paramètres n_m et a_m est défini tel que :

- G'_0 correspond au regroupement en groupes maximaux de nœuds :

$$p(G_0, n_m)$$

- G'_1 correspond aux arcs étiquetés du graphe simplifié. Si il existe au moins a_m arcs dans G_1 avec la même étiquette l entre les nœuds de deux groupes M et N de G'_0 alors il existe un arc de M vers N étiqueté l :

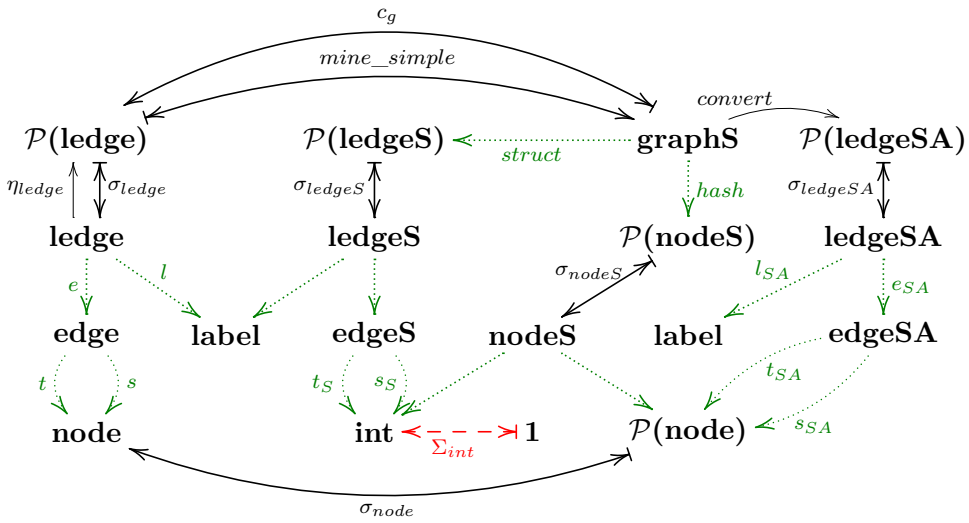
$$G'_1 = \{(l, (M, N)) | M, N \in G'_0, a_m \leq |\{(l, (m, n)) \in G_1 | m \in M, n \in m\}\} \quad .$$

L'opérateur qui transforme un graphe en un graphe simplifié est présenté dans le schéma 5.29.

Schéma 5.29 (Schéma de simplification de graphe)
Les données à abstraire sont représentées par le nœud $\mathcal{P}(edge)$ qui représente un graphe dont les arcs sont étiquetés. Le graphe simplifié est représenté par le nœud $graphS$. Il est composé d'un ensemble d'arcs étiquetés ($\mathcal{P}(ledgeS)$) représenté par la projection *struct* et d'un ensemble de nœuds ($\mathcal{P}(nodeS)$) représentant une fonction de hachage des ensembles de nœuds regroupés. À un entier int est associé un ensemble de nœuds $\mathcal{P}(node)$. De cette façon, un groupe de nœuds n'est pas répété pour chaque arc dont il est la source ou la cible.

La relation $mine\_simple : \mathcal{P}(edge) \rightarrow graphS$ construit le graphe simplifié à partir du graphe de base. La relation de couverture $c_g : graphS \nrightarrow \mathcal{P}(edge)$ utilise la structure $\mathcal{P}(ledge_{SA})$ pour convertir un graphe simplifié en un graphe dont les

nœuds sont des ensembles de nœuds de base.



$$\Sigma_{int} = \{\mathbf{1} \to 0, \ldots, \mathbf{1} \to 2^{32}\}$$

$$c_g = \eta_{ledge} \circ \langle l_{SA}, \langle \sigma_{node} \circ s_{SA}, \sigma_{node} \circ t_{SA} \rangle \circ e_{SA} \rangle \circ \sigma_{ledgeSA} \circ convert$$

.

L'algorithme présenté permet de regrouper les arcs et les nœuds d'un graphe en fonction du nombre de points commun entre deux nœuds et du nombre d'arcs entre deux groupes de nœuds. Si deux nœuds ont plus de n_m arcs avec la même étiquette et la même source (ou cible) alors ces deux nœuds sont regroupés. Si deux groupes de nœuds ont au moins a_m arcs entre eux alors un regroupement d'arcs est généré.

Cette méthode permet de simplifier le graphe avec une certaine perte d'informations. En effet, à partir d'un graphe simplifié, tous les arcs ne sont pas couverts et des arcs non présents dans le graphe de base le sont. La sous-section suivante se propose d'évaluer cette méthode de résumé sur des données réelles par rapport à d'autres méthodes de résumé classiques en fouille de données.

Extraction de modèles

Les expérimentations portent sur l'acquisition de modèles à partir d'un journal d'alarmes DDoS fourni par France-Télécom. De la même façon que pour VPN, nous détaillons les schémas opérationnels obtenus, puis nous expliquons comment ces schémas opérateurs ont été exécutés. Enfin, nous fournissons une analyse des modèles extraits.

Schémas opérationnels Les schémas de la base présentés au début de la sous section 5.1.2 page 134 sont complétés par le schéma de simplification de graphe. Ces schémas ont pu s'adapter automatiquement avec le schéma des alarmes DDoS (5.21 page 145) pour obtenir neuf schémas opérationnels décrits ici :

- *Clustering en deux dimensions* 1. Il a été adapté sur la sévérité et la date.
- *Clustering en une dimension.* Il a été adapté de deux façons : sur la date 2 et la sévérité 3.
- *Séquences.* Le schéma des séquences a été adapté. Deux adaptations ont été réalisées sur les types des évènements : le type DDoS 4 et la sévérité 5 d'une alarme.
- *Généralisation de graphes* 6. Ce schéma a été adapté sur l'arc représentant la source et la cible de l'attaque.
- *Simplification de graphes par étiquette.* Le schéma 5.29 page 149 de la simplification de graphes a été adapté de trois façons : l'étiquette est soit la sévérité 7, le type 8 ou encore la date 9 d'une alarme DDoS.

Paramètres et exécution Les problèmes d'efficacité de l'évaluation générique ont influencé largement les résultats obtenus. Afin de réduire ces effets (temps d'exécution et mémoire occupée) plusieurs mesures ont été prises : réduction du nombre d'alarmes analysées à 1500, évaluation locale, évaluation seulement des éléments d'un modèle quand celui-ci est un ensemble. Les paramètres des algorithmes sont donnés dans la liste suivante :

- *Clustering en deux dimensions.* Le paramètre k du nombre de clusters a été fixé à plusieurs valeurs : $\{3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30\}$.
- *Clustering en une dimension.* Le paramètre k prend les mêmes valeurs que le clustering en deux dimensions.
- *Séquences.* Les $k = 15$ séquences les plus fréquentes sont retournées.
- *Généralisation de graphe.* L'algorithme retourne les 30 nœuds ayant le plus d'arcs.
- *Simplification de graphe par étiquette.* L'algorithme a été testé avec les valeurs $a_m = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30\}$ et $n_m = \{1, 2\}$.

Analyse des modèles Les caractéristiques des modèles extraits sont représentées dans la figure 5.30 page suivante. Il faut bien entendu tenir compte des évaluations qui ont été effectuées. Pour la structure de graphe simplifié 7 8 9, les modèles entiers ont été évalués. Pour tous les autres types de modèles : clusters 1 2 3, séquences 4 5 et graphes généralisés 6, seuls les éléments de ces modèles ont été évalués. Cela explique que les modèles de simplification de graphes par étiquette ont globalement un taux de couverture plus important que les clusters, les séquences ou les graphes généralisés.

Les modèles les plus proches du point optimal $(0, 1)$ sont des simplifications de graphe sur la date 9. On peut remarquer que l'étiquette est importante puisque les simplifications de graphe utilisant la sévérité 7 ou le type d'alarmes 8 résument moins bien l'information.

Un des points qui n'est pas abordé ici est le recouvrement entre modèles. En effet, deux modèles peuvent couvrir la même information. Dans ce cas, l'utilisateur devrait regarder celui qui compresse le mieux les données.

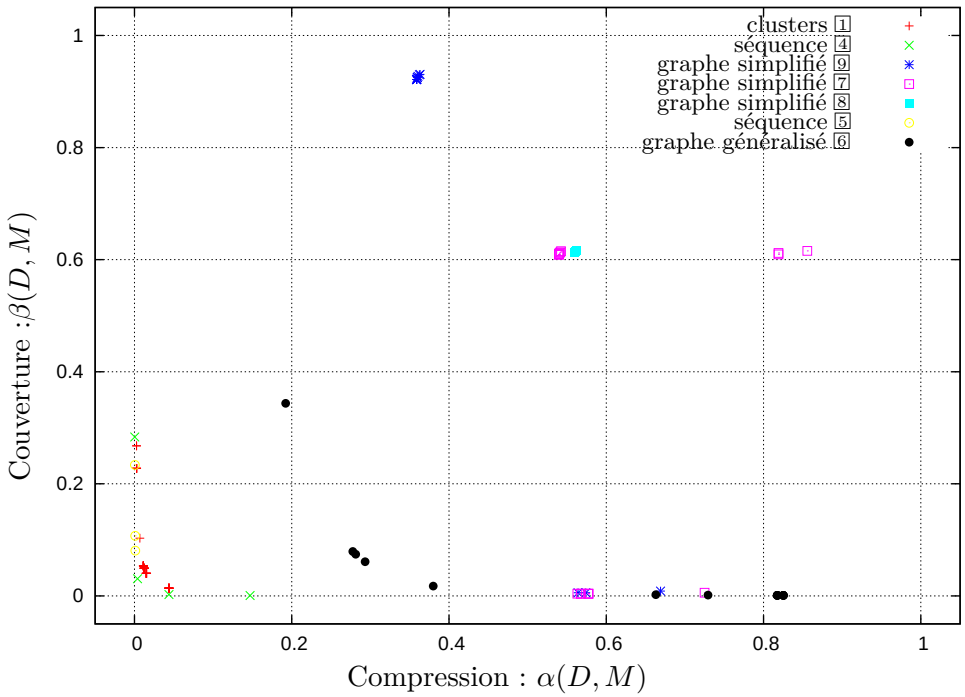


FIG. 5.30: Compression et couverture des modèles générés à partir de 1500 alarmes DDoS.

Conclusion

Ce chapitre décrit les expérimentations réalisées sur des données réelles issues de journaux d'alarmes de télécommunication. Les faibles connaissances sur les phénomènes liés à ces journaux et la quantité de données accumulées motivent la recherche de résumés analysables par un utilisateur.

Les outils catégoriques utilisés ont permis de décrire des opérateurs de fouille de données complexes applicables sur les données VPN et DDoS. La variété des opérateurs utilisés pour extraire des séquences, des transactions, des graphes... montrent l'intérêt d'utiliser un outil catégorique. De plus, dans un tel formalisme, les données sont décrites d'une façon élémentaire à la portée de tout utilisateur en informatique.

Sur les données VPN, la principale difficulté de l'analyse est le manque de structure explicite et de renseignements sur les attributs des alarmes. L'utilisateur interagit avec les différents modules de notre approche dans le but d'élaborer une synthèse du journal et d'acquérir de nouvelles connaissances sur celui-ci (Vautier *et al.*, 2006a). L'analyse effectuée n'est pas propre aux journaux de concentrateurs VPN mais est généralisable à d'autres journaux faiblement structurés : syslog, traces de programme, etc. Les techniques utilisées, l'extraction des attributs à partir de leur forme, l'agrégation en transactions primitives composées d'alarmes corrélées relationnellement et la construction des modèles de transactions, sont toutes

généralisables à d'autres problèmes.

Sur les données DDoS, la principale difficulté est la structure en graphe très riche mais très complexe à gérer. En effet, la structure distribuée d'internet multiplie les combinaisons de flux. La simplification de ces flux sous la forme d'un graphe étiqueté est une réponse simple (Vautier *et al.*, 2006b) ne nécessitant pas de connaissances sur le graphe de flux pour être exécutée. Le graphe obtenu permet d'avoir une vue globale des phénomènes.

Enfin les modèles extraits et leur qualité représentée sur les graphiques compression/couverture constituent un bon moyen pour l'utilisateur de choisir quelles représentations analyser. Il manque la mise en œuvre d'un processus itératif basé sur cet outil pour résumer complètement le journal. Cet aspect est plus longuement discuté dans les perspectives.

Conclusion et perspectives

Nous nous sommes intéressés dans cette thèse à la problématique de l'extraction de connaissances dans les données lorsque l'utilisateur qui fouille les données n'a pas d'information sur la structure de la connaissance à extraire. Nous rappelons les principales contributions de ce travail et proposons quelques perspectives de recherche.

Contributions

Cette thèse comporte deux parties : une partie concerne la mise en place d'un outil de spécification particulier et une seconde partie propose une architecture basée sur ce concept de spécification dans le but de découvrir des connaissances dans les données.

Concernant l'outil de spécification, nous avons utilisé la théorie des catégories afin de définir les *esquisses relationnelles*. Le cadre théorique fourni par les catégories sert de support, non seulement aux constructions existantes qui sont réutilisées pour la définition des esquisses mais aussi pour formaliser les nouvelles constructions introduites. Un soin particulier a été apporté à la définition et la formalisation des esquisses relationnelles. Plus précisément, les esquisses relationnelles constituent un enrichissement des *esquisses finies et discrètes* auxquelles elles apportent les notions de relation et d'objet des parties. L'introduction de ces notions impose la définition d'une *catégorie relationnelle*. La catégorie des ensembles et des *relations orientées* est une catégorie relationnelle. Nous avons prouvé que le produit relationnel est le produit cartésien et que la somme relationnelle est l'union disjointe dans cette catégorie.

Par ailleurs, nous utilisons le concept de *schéma* qui est une implémentation d'une esquisse relationnelle pour définir des opérateurs de fouille de données. Dans notre cadre, Prolog, un langage de programmation logique, a été utilisé pour implémenter les esquisses relationnelles. De cette manière, les schémas permettent de définir avec souplesse des algorithmes et les structures avec lesquelles ils interagissent.

Concernant *l'architecture de découverte de connaissances*, nous nous sommes intéressés à la définition d'un paradigme de fouille de données prenant en charge un

utilisateur ayant peu ou pas d'information sur la structure des connaissances à extraire. À partir de cette problématique, les schémas procurent un moyen d'organiser une architecture pour spécifier les structures des *modèles* devant être extraits des données par des opérateurs (algorithmes) de fouille de données. Un modèle est une généralisation partielle ou complète d'un ensemble de données, plus « simple » que l'énumération des données qu'il couvre.

Dans notre approche, nous supposons qu'il existe une base d'opérateurs de fouille de données spécifiés sous la forme d'esquisses relationnelles et implémentés. De plus, l'utilisateur fournit une spécification des données à analyser. Nous avons défini une opération d'*unification* entre schémas qui, dans le cadre qui nous intéresse, correspond à une adaptation d'un opérateur de fouille de données aux données à analyser. L'unification repose sur des heuristiques dont le but est de construire des adaptations valides pour l'extraction des modèles. Ainsi, tout opérateur définit une fonction d'extraction automatique de modèles qui est exécutée sur les données pour chacune des adaptations.

Les modèles extraits doivent être classés selon leur qualité avant d'être présentés à l'utilisateur. Nous avons élaboré une méthode d'évaluation générique basée sur la *complexité de Kolmogorov*. Cette évaluation utilise la définition de la relation de couverture définie dans tout schéma d'opérateur. La relation de couverture explicite exactement le lien qui relie un modèle aux données qu'il couvre. Elle est composée de constructions catégoriques. Deux méthodes d'évaluation ont été mises en œuvre : la première évalue la qualité d'un modèle à résumer globalement toutes les données, la seconde évalue la qualité d'un modèle à résumer localement les données qu'il couvre ainsi que l'étendue de ces données relativement à l'ensemble des données à analyser.

Des expérimentations sur des données synthétiques et réelles ont permis d'évaluer les performances et la qualité de la méthode proposée. L'évaluation générique a été testée sur des données synthétiques et a mis en avant les difficultés pour minimiser la taille de description des données couvertes par un modèle à partir de la relation de couverture. Néanmoins, l'algorithme de minimisation mis en place obtient des résultats satisfaisants. Des expérimentations sur des données réelles ont été conduites afin de montrer que l'intégration d'algorithmes dans la base d'opérateurs de fouille de données est simple à effectuer. L'évaluation empirique de la méthode exposée montre la pertinence des choix effectués : la spécification des données par l'utilisateur est relativement simple; l'unification remplit son rôle pour adapter les opérateurs de la base aux données; l'évaluation des modèles extraits fournit un résultat synthétique et pertinent à l'utilisateur. Par la suite, celui-ci a la possibilité d'analyser les modèles qui résument le mieux les données afin d'en extraire des connaissances. Cela montre que le concept de l'architecture mise en œuvre est valide pour l'extraction de connaissances quand l'utilisateur dispose de peu d'informations sur leur structure.

Perspectives

Les perspectives sont présentées selon trois axes de recherche complémentaires. Le premier concerne l'efficacité de l'évaluation générique. Ensuite, la mise en œuvre d'un processus itératif de fouille de données est envisagée. Enfin, l'aspect catégorique de l'approche est discuté.

Efficacité

On a vu dans le chapitre 4 que le temps d'exécution de l'évaluation posait problème. Tout d'abord, il faudrait intégrer un dispositif permettant à un concepteur de schémas de définir des moyens de calculer simplement la taille de l'image d'un chemin. De plus, une meilleure recherche des indexations et une approximation de leur taille de description amélioreraient nettement l'efficacité de la méthode.

Les schémas intègrent des concepts de spécification et d'implémentation. L'implémentation réalisée en Prolog n'est qu'un prototype qu'il est difficile de faire évoluer. Dans le but de rendre réalisable la construction de schémas dans divers langages de programmation, il serait intéressant de définir un système général dans lequel ces implémentations cohabiteraient. De plus, la mise en œuvre d'une interface graphique faciliterait sérieusement la définition de schémas dont la représentation sous forme de graphes est nettement plus simple à appréhender que la représentation textuelle.

Processus itératif

L'approche proposée dans cette thèse souffre de quelques inconvénients du point de vue de l'intégration des connaissances de l'utilisateur dans le processus d'extraction de connaissances. En effet, dans un processus itératif classique, l'utilisateur modifie peu à peu les éléments utilisés lors des étapes d'extraction au fur et à mesure que ses connaissances sur les données progressent : il se focalise sur certaines propriétés des données, favorise certains algorithmes et certains paramètres de ces algorithmes... Il serait intéressant de mettre en œuvre un tel processus en se basant sur notre approche.

Une des manières de répondre à cette problématique est d'intégrer un processus itératif s'appuyant sur des distributions de probabilités relatives aux éléments de l'architecture. Au lieu d'exécuter *tous* les opérateurs de toutes les façons possibles sur les données, un échantillonnage de ces combinaisons possibles est effectué. Les éléments utilisés sur lesquels les distributions sont définies sont : les propriétés des données, les opérateurs de fouille de données, des adaptations particulières et les valeurs des paramètres des opérateurs. Ces distributions représentent les *hypothèses a priori* sur l'extraction des modèles à effectuer dans le but d'obtenir les meilleurs modèles. Au début de l'exploration, ces distributions sont équitablement réparties. Le système échantillonne selon ces distributions des schémas opérationnels et les exécute. Ensuite, suivant la qualité des modèles obtenus et les objectifs de l'utilisateur qui s'affinent, les distributions sont recalculées (en suivant, par exemple, une approche bayésienne) et le processus est relancé.

Cette approche permet de répondre à plusieurs problèmes. Dans le cas de données possédant beaucoup de propriétés (comme une base de données réelle), la combinatoire liée aux adaptations des opérateurs de fouille de données peut être considérable. De plus, en fouille de données, des successions d'opérateurs sont effectués mais dans notre approche, la complexité exponentielle de cette proposition est un frein. En effet, il existe un nombre exponentiel de possibilités. Avec le processus décrit précédemment, ce problème est contourné. Enfin, de la même façon que dans un processus d'aide à la fouille de données (Bernstein *et al.*, 2005), il est possible d'intégrer des connaissances issues d'heuristiques sur les techniques à exécuter en fonction de la proximité des données avec les données sur lesquelles des processus de fouille de données ont réussi. Par exemple, si les données ne peuvent pas être stockées en mémoire alors les opérateurs doivent tenir compte de cette caractéristique ou bien ne pas être exécutés.

Aspect catégorique

Dans l'approche proposée, pour chaque opérateur il faut choisir une spécification ni trop proche de l'implémentation pour permettre une unification automatique ni trop abstraite pour décrire précisément la relation de couverture. Par exemple, plus l'esquisse relationnelle d'un opérateur de fouille de données est précise, plus il est possible de spécifier la relation de couverture d'une façon détaillée. Plus la relation de couverture est détaillée plus le calcul de la complexité pour retrouver les données sachant le modèle est précis. Or ce niveau de détail s'oppose au fait que la spécification doit suffisamment abstraire les structures utilisées (graphe, arbre, intervalles,...) pour permettre une unification entre esquisses. Par exemple, un graphe peut être représenté par un graphe abstrait ou plus précisément par une matrice d'adjacence ou une liste d'arcs. Une relation de couverture qui couvre ces structures est exprimée plus ou moins précisément pour ces trois spécifications : d'une façon simpliste pour le graphe abstrait et d'une façon plus détaillée pour la liste d'arcs ou la matrice d'adjacence. Or, un graphe abstrait s'unifie avec la plupart des représentations de graphes alors que la liste d'arcs et la matrice d'adjacence s'unifie avec des structures qui utilisent les mêmes implémentations. Il serait donc intéressant d'étudier cette contrainte de précision de spécification dans le but de pouvoir décrire précisément la relation de couverture dans une esquisse la plus abstraite possible.

Une autre proposition est d'intégrer les propriétés des relations dans l'esquisse relationnelle. Par exemple, les propriétés d'injectivité et de surjectivité des relations permettent de faciliter la recherche des images d'une relation. De ce fait, la spécification de ces propriétés améliore l'efficacité du calcul de la complexité pour retrouver des données sachant un modèle et la relation de couverture.

La nouvelle vision qui émerge de ce travail est un outil d'extraction de connaissances intégrant de nombreux opérateurs de fouille de données. La fouille de données n'est plus vue comme un processus itératif dirigé seulement à partir des connaissances d'un utilisateur mais aussi à partir d'une multitude de facteurs tels que les

opérateurs de fouille de données disponibles, la proximité du processus avec d'autres processus réussis, la qualité des modèles extraits, les caractéristiques des opérateurs et des données et d'autres facteurs qu'il reste à définir.

ANNEXE A

Calcul de la taille des constructions d'un schéma

L'évaluation générique repose sur plusieurs calculs. Les principaux ont été présenté dans la section 4.3 page 105. Nous ajoutons la présentation du calcul des éléments d'un schéma dans la première section et la manière d'implémenter le calcul de la taille d'indexation de données couvertes par un modèle dans la seconde section.

A.1 Variantes du langage de descriptions

La taille de la structure représenté par $L(struct)$ peut être importante pour des données de petite taille. Cette structure est composée de la taille de la relation de couverture et de la taille des nœuds qui structure les modèles et les données. Nous en expliquons ici le calcul.

Plusieurs façons d'envisager le calcul de la taille de $L(struct)$ sont possibles. *struct* symbolise essentiellement le schéma opérationnel obtenu par unification. Il faut le schéma des données $\mathbf{Data_U}$ et le schéma d'opérateur $\mathbf{Op}_s$ contenu dans la base *Op* des schémas des opérateurs (illustrée dans le schéma 4.3 page 95) pour retrouver *struct*. Or la définition de la description de *struct* est relative à la machine qui l'interprète (définition 3.13 page 81). Plus le cheminement de la description vers les données est important, plus la machine f doit intégrer des algorithmes longs et complexes. Comme, nous identifions cette machine à l'utilisateur, plus le codage est complexe plus l'utilisateur a de connaissances. Nous faisons donc les hypothèses suivantes : l'utilisateur ne connaît pas la base *Op* des schémas d'opérateur mais il connaît le schéma $\mathbf{Data_U}$ des données. Ainsi, le schéma d'opérateur $\mathbf{Op}_s$ n'est pas dans la machine f mais le schéma $\mathbf{Data_U}$ y est. Dans ce cas il est nécessaire de décrire les éléments du schéma $\mathbf{Op}_s$ (figure 4.10) utiles à l'unification, et aux calculs

de $L(model')$, $L(cover')$ et $L(\mathcal{P}(data'))$. Ces éléments sont $model_A$, $data_A$ et $cover_A$:

$$L(struct) = L(model_A) + L(data_A) + L(cover_A)$$

Ainsi, il faut décrire le calcul de la taille de description d'un nœud (sous-section A.1.1) et d'un chemin (sous-section A.1.2).

A.1.1 Taille de description d'un nœud

Intuitivement, un nœud est transformé en une chaîne binaire (composée de 0 et de 1). Un nœud est un produit, une somme, un objet des parties, le nœud terminal ou un nœud abstrait. Si un nœud est un produit ou une somme, plusieurs *sous-nœuds* interviennent dans sa définition. En effet, les projections indiquent les nœuds qui composent le produit et les inclusions indiquent les nœuds qui composent la somme. L'algorithme A.1 page ci-contre détaille le calcul de la complexité d'un nœud. Il utilise la fonction *définition*(r) qui retourne le domaine et le codomaine de la relation r et les fonctions *projections* et *inclusions* qui retournent les relations représentant respectivement les projections d'un produit et les inclusions d'une somme. La fonction *sous_nœud* retourne les types des éléments d'un nœud représentant le type liste. Chaque élément de la chaîne binaire représentant un nœud est de taille $log(6)$ car il existe cinq types de nœud et un marqueur de fin. La ligne 1 inclut la taille nécessaire pour indiquer le type du nœud. Le marqueur de fin est utilisé aux lignes 2 et 3. Il permet d'indiquer qu'un ensemble est terminé.

A.1.2 Taille de description d'un chemin

Un chemin est une composition de relations (définition 1.4 page 11 et section 2.2.1 page 42). Une relation est une factorisation, une cofactorisation ou une relation (implémentée ou non). Nous utilisons une version récursive de la notion de chemin pour simplifier le calcul de la taille d'un chemin. Un chemin est :

- une composition de deux chemins,
- une factorisation de plusieurs chemins,
- une cofactorisation de plusieurs chemins ou
- une relation.

L'algorithme A.2 page 164 détaille le calcul de la taille d'un chemin pour extraire les chemins impliqués dans les factorisations et les cofactorisations (deux fonctions *factorisation* et *cofactorisation* sont utilisées). La fonction *taille_relation* (ligne 4) calcule la taille de représentation d'une relation implémentée ou non par l'utilisateur. Le contenu de cette représentation étant inconnue, il est impossible d'utiliser un marqueur de fin pour indiquer qu'elle est terminée. De plus, la taille maximale d'une relation implémentée n'est pas connue non plus. C'est pourquoi, on utilise la forme auto-délimitante d'une chaîne binaire (Li et Vitány, 1997). En effet, une relation implémentée dans n'importe quel langage a une représentation binaire x. La description auto-délimitante d'une chaîne binaire x est $x' = 1^{|x|}0x$ où $1^{|x|}$ représente une chaîne de 1 de taille $|x|$. Une autre description auto-délimitante de

Algorithme A.1 (Taille de la description d'un nœud *taille_nœud*(N))

Entrées : Un nœud N
Sorties : La taille de la description du nœud : l
1 $l = log(6)$
suivant *le type de* N **faire**
  cas où N *est un produit*
    pour chaque $p \in projections(N)$ **faire**
      $(Domaine, Codomaine) = définition(p)$
      $l = l + taille\_nœud(Codomaine)$
2   $l+ = log(6)$
  cas où T *est une somme*
    pour chaque $i \in inclusions(N)$ **faire**
      $(Domaine, Codomaine) = définition(i)$
      $l = l + taille\_nœud(Domaine)$
3   $l+ = log(6)$
  cas où N *est un objet des parties*
    $E = sous\_nœud(N)$
    $l = l + taille\_nœud(E)$
  cas où $T = \mathbf{1}$ *ou* T *est abstrait*
    « rien à faire de plus »
retourne(l)

x est $x'' = 1^{log(|x|)}0|x|x$, où $|x|$ représente la chaîne décrivant la taille de x. La taille auto-délimittante utilisée pour calculer l'espace nécessaire pour sauvegarder une information de taille l sans connaître sa taille maximum est :

$$taille\_auto\_delim(l) = l + 2log(l) + 1$$

À la ligne 5, on utilise cette formule pour calculer la taille de la description d'une relation implémentée dans un chemin. Aux quatre possibilités d'un chemin (composition, factorisation, cofactorisation et relation) s'ajoutent un marqueur de fin (utilisé aux lignes 2 et 3). Ce qui donne cinq possibilités pour coder un élément (ligne 1).

A.2 Construction de l'arbre des instances

Nous détaillons la transformation d'une indexation e, d'un ensemble de données D et d'un modèle M en une unique structure : un *arbre d'instances*. Intuitivement, une instance structure les valeurs intermédiaires calculées par l'indexation e pour établir le lien entre le modèle M et des données appartenant à D. L'arbre d'instances agrège toutes les instances et sert de support au calcul de la taille de description.

Sur les exemples 4.22 page 112 et 4.25 page 114, le calcul des ensembles intermédiaires et suffisants pour calculer les données est relativement simple. Dans des cas

Algorithme A.2 (Taille de la description d'un chemin *taille_chemin*(C))

Entrées : un chemin C
Sorties : La taille de la description du chemin : l

```
1  l = log(5)
   suivant le type de N faire
       cas où C est une composition
           r ∘ s = C
           l = l + taille_chemin(r)
           l = l + taille_chemin(s)
       cas où C est une factorisation⟨..., ...⟩
           pour chaque f ∈ factorisations(C) faire
               l = l + taille_chemin(f)
2          l = l + log(5)
       cas où C est une cofactorisation⟨...; ...⟩
           pour chaque cf ∈ cofactorisations(C) faire
               l = l + taille_chemin(cf)
3          l = l + log(5)
       cas où C est une relation
4          l_r = taille_relation(C)
5          l = taille_auto_delim(l_r)
   retourne(l)
```

concrets, ce calcul peut être complexe. De manière à généraliser un moyen de calculer ces ensembles intermédiaires, nous définissons la notion d'*instance* qui permet de récupérer les valeurs intermédiaires calculées par l'application d'une relation de couverture *cover* indexée sur un modèle M et des données D. Ensuite, les instances sont agrégées en un arbre d'instances à partir duquel on calcule la taille de la description. La définition A.3 formalise la notion d'instance et l'exemple A.4 en illustre un exemple très simple.

Définition A.3 (Instance)
Soient un modèle $M \in model$, une relation de couverture $c : model \nrightarrow \mathcal{P}(data)$, des données $D \in \mathcal{P}(data)$ couvertes par M selon c, une indexation $e \in I(c)$. L'ensemble des instances $T(M, D, e)$ est définit par :

$$T(M, D, e) = \{i | \exists d \in \mathcal{P}(data), d \in c(M), d \subseteq D, i = values(e(M) = d)\}$$

où la fonction *values* transforme l'expression d'une indexation en insérant les valeurs intermédiaires calculées au niveau des codomaines des relations indexées. ▪

Exemple A.4 (Instance)
Soient le modèle $M = 1$, l'ensemble de données $D = \{2, 4\}$ et le chemin $cover = inc\_dec \circ inc \circ inc$ où inc représente l'incrémentation et inc_dec représente une

relation qui incrémente ou décrémente. Le chemin *cover* est indexé de cette façon : $e = {}^{i}inc\_dec \circ inc \circ {}^{i}inc$. L'application du modèle $M = 1$ à *cover* génère les valeurs intermédiaires $inc(1) = 2, inc(2) = 3$ et $inc\_dec(3) = 4$ pour la donnée 4 et $inc(1) = 2, inc(2) = 3$ et $inc\_dec(3) = 2$ pour la donnée 2. Ces valeurs sont structurées sous une forme en relation avec l'indexation e dans l'ensemble

$$T(M, D, e) =$$
$$\{(4\ inc\_dec\ \circ inc\ \circ 2\ inc\ 1),$$
$$(2\ inc\_dec\ \circ\ inc\ \circ 2\ inc\ 1)\}$$

L'ensemble $T(M, D, e)$ constitue les instances entre le modèle M et les données D selon l'indexation e. ▪

Deux possibilités permettent de calculer les instances. Une première méthode consiste à analyser automatiquement la relation de couverture et à générer une méthode de calcul. Cette méthode a été développée et a rencontré quelques limites du point de vue de son efficacité. C'est pourquoi, nous avons introduit un second moyen qui renvoie le calcul à une fonction attachée à la relation de couverture. Autrement dit, une fonction définie dans un schéma $\mathbf{Op}_i$ (de la base d'opérateurs) détaille le calcul des instances d'une relation de couverture. Cette solution s'avère avantageuse dans de nombreux cas où l'efficacité de l'automatisation n'est pas comparable à l'expertise humaine. L'exemple A.5 illustre parfaitement l'intérêt de cette méthode.

Exemple A.5 (CRS : une relation de couverture)
CRS est un système de reconnaissance de chroniques dans un ensemble d'évènements. Le modèle est constitué d'un ensemble de chroniques et les données sont constituées d'un ensemble d'évènements. Il est possible de formaliser le lien qui existe entre les chroniques et les évènements sous la forme d'un chemin. Ce chemin constitue la relation de couverture entre les chroniques et les évènements. L'extraction automatique des instances de ce chemin relativement à un ensemble de chroniques données et un ensemble d'évènements donnés est exactement le calcul effectué par CRS. L'utilisation de cet algorithme déjà mis en place est sans l'ombre d'un doute plus efficace que l'extraction automatique d'instances calculée à partir d'une relation de couverture.▪

Le calcul de la taille de représentation des données est effectué à partir de l'ensemble des instances d'une relation de couverture entre un modèle et des données et d'une indexation. Ce calcul passe par la construction d'un *arbre d'instances* (des arbres illustrent chacun des cas analysés dans les exemples 4.22 page 112 et 4.25 page 114). À partir d'un arbre, la taille de description des données est égale à la somme des tailles de description des arcs.

Définition A.6 (Arbre d'instances)
Un arbre d'instances est un arbre tel que :

- Chaque nœud représente un ensemble indexé ou non. Un ensemble indexé est représenté graphiquement par un nœud encadré $\boxed{N}$.

- Chaque arc a a une cible $cible_a$, une source $source_a$ et est étiqueté par un chemin c_a. ▪

Un exemple d'arbre est donné dans les exemples 4.22 page 112 et 4.25 page 114.

Définition A.7 (Taille de la description d'un arbre d'instances)
La taille de description (relativement à une indexation et un modèle) d'un arbre d'instances A est définie par :

$$L(A) = \sum_{a \in arcs(A)} L(a) \text{ Si } cible_a \text{ est indexé alors } L(a) = L_c(cible_a | c_a(source_a))$$
$$\text{sinon } L(a) = 0 \quad \blacksquare$$

L'algorithme A.8 calcule la taille d'indexation de données à partir d'un modèle et d'une relation de couverture. L'efficacité de cet algorithme est dépendant des relations implémentées dans le schéma $\mathbf{Exec}_s$. En effet, une implémentation de la relation de couverture est appelée (ligne 1). Or celle-ci peut être implémentée automatiquement, ce qui n'est pas efficace (exemple A.5 page précédente), ou bien implémentée par le concepteur du schéma d'opérateur correspondant. De plus le calcul de la taille de description (ligne 2) d'un arbre d'instance nécessite le calcul de la taille de l'image d'une relation (définition A.7). Or le calcul de cette taille passe par l'appel des relations correspondantes définies dans le schéma $\mathbf{Exec}_s$. Plus ces relations sont efficaces plus le calcul de la taille de description de l'arbre des instances est efficace.

Algorithme A.8 (Taille d'une indexation $taille\_indexation(D, M, e)$)

Entrées :
Un ensemble d'éléments : $D \in \mathcal{P}(data)$
Un modèle : $M \in model$
Une indexation : $e \in I(c)$ telle que $c : model \nmapsto \mathcal{P}(data)$
Sorties :
La taille de la description des données D relativement à l'indexation i et au modèle M : l
1 Calcul de l'ensemble des instances $T(M, D, e)$
Construction de l'arbre d'instances A à partir de $T(M, D, e)$
2 Calcul de la taille l de la description de A
retourne(l)

ANNEXE B

Schémas de base

Programme B.1 (Booléens)

```
:- begin_schema(boolean).
    % definition d'une somme
    :- sum(boolean,[(true,1),(false ,1)]).
:- end_schema(boolean).
```

Programme B.2 (Ensemble ordonné)

```
:- begin_schema(ordered_set).

    % inclusion d'un schema sans renommage
    :- declare_functor(boolean).

    :- set_def_sort(o_element).
    :- product(eXe,[(e1,o_element),(e2,o_element)]).
    :- set_definitions(inf,eXe,boolean).

:- end_schema(ordered_set).
```

Programme B.3 (Entiers)

```
:- begin_schema(abstractInt).

    :- begin_functor(ordered_set_to_int,ordered_set).
        :- functor_sort(o_element,int).
        :- functor_sort(eXe,intXint).
```

```
    :- end_functor(ordered_set_to_int).

    :- set_definitions(plus,intXint,int).
    :- set_def_relation_type((plus,intXint,int),function).

    % implementation de l'addition
    plus(TxT @ intXint, T @ int) :-
        e1(TxT @ intXint, T1 @ int),
        e2(TxT @ intXint, T2 @ int),
        T is T1 + T2.

:- end_schema(abstractInt).
```

Programme B.4 (Classe en Java couplant le schéma k-means et Weka)

```
public class Kmeans {

public static Term buildClusters2(jpl.Integer n, Term dataIn) throws Exception
{
int dim = 2;
// Recupere le schema courant
RunSchema sr = new RunSchema(new Schema("cluster2"),
                             Schema.currentSchema());

// Recupere les relations et les sorts
Sort elementX = sr.getSort("tX");
Sort elementY = sr.getSort("tY");
Sort finalSort = sr.getSort("p_clusterP");
Relation rX = sr.getRelation("x");
Relation rY = sr.getRelation("y");
// recuperation des constructeurs lies aux produits
SortConstructor dcluster = sr.getSortConstructor("clusterP","dX","dY");
SortConstructor dintX = sr.getSortConstructor("intervalX","beginX","endX");
SortConstructor dintY = sr.getSortConstructor("intervalY","beginY","endY");

// Donnees a traiter
Term [] tabPoint = Tools.getData(dataIn).toTermArray();

// Conversion des donnees vers le format de weka
 ...
Instances ins = new Instances("dataset",tabPoint.length);

for(int i = 0; i < tabPoint.length; i++)
{
    ins.add(new Instance(2,new double[]{
            ((jpl.Integer)Tools.getData(rX.call(tabPoint[i ]))).intValue(),
            ((jpl.Integer)Tools.getData(rY.call(tabPoint[i ]))).intValue()}));
}
```

```
    //Execution de l'algorithme
    SimpleKMeans km = new SimpleKMeans();
    km.setNumClusters(n.intValue());
38  km.buildClusterer(ins);

    //Recuperation des clusters
    int [][][]  intervals  = new int[km.numberOfClusters()][dim][2];

43  ...

    Term [] tabCluster = new Term[intervals.length];

    for(int i = 0; i < intervals.length ; i ++)
48  {
        Term intervalDim [] = new Term [dim];
        ...
        intervalDim[0] = dintX.construct(
                            Tools.setType(new jpl.Integer(intervals[i][j][0]),  elementX),
53                          Tools.setType(new jpl.Integer(intervals[i][j][1]),  elementX)
                            );
        ...
        tabCluster[i] = dcluster.construct(intervalDim);
    }
58
    Term res = Util.termArrayToList(tabCluster);
    Term rest = Tools.setType(res,finalSort);
    return rest;
    }
63  }
```

Programme B.5
Programme du schéma d'un intervalle d'entiers

```
    :- begin_schema(interval_abs_int).
2
        :- begin_functor(interval_for_int,interval).
            :- functor_sort(o_element,int).
            :- functor_sort(interval,interval_int).
            :- functor_rel(begin,begin_int).
7           :- functor_rel(end,end_int).
            :- functor_rel(couv_interval,couv_interval_int).
            :- functor_sort(eXe,intXint).
        :- end_functor(interval_for_int).

12      :- begin_functor(abstractIntInterval,abstractInt).
            :- functor_sort(int,int).
            :- functor_sort(boolean,boolean).
            :- functor_sort(intXint,intXint).
```

```
    :- end_functor(abstractIntInterval).
17
      couv_interval_int( Interval @ interval_int, I @ int) :-
          begin_int(Interval @ interval_int, B @ int),
          end_int(Interval @ interval_int, E @ int),
          between(B,E,I).
22
    :-  set_def_relation_type((couv_interval_int,interval_int,int),overlap).

:- end_schema(interval_abs_int).
```

ANNEXE C

Signatures VPN

STR =.*
IP =(?:_|(?:N/A))|(?:\b(?:(?:25[0−5]|2[0−4][0−9]|[01]?[0−9][0−9]?)\.)
{3}(?:25[0−5]|2[0−4][0−9]|[01]?[0−9][0−9]?)\b)
NUM =[a−fA−F_0−9]*

alarmType =IKE/24
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group1\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group1\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group2\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group2\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 0, Port (**NUM**)
classOk =Y,Y,Y,Y
regExp =(**IP**) Group \[VPNC_Base_Group\] Received local Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
}

alarmType =IKE/41
classType =visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataIP
checked =Y
{
regExp =_ IKE Initiator: Rekeying Phase 2, Intf 2, IKE Peer (**IP**) local Proxy Address (**IP**), remote Proxy Address (**IP**), SA \(WindowsArchimede\)
classOk =N,Y,Y,Y
regExp =(**IP**) Group \[Group1\] IKE Initiator: Rekeying Phase 1, Intf 2, IKE Peer (**IP**) local Proxy Address N/A, remote Proxy Address N/A, SA \(N/A\)
classOk =Y,Y,N,N

regExp =(**IP**) Group \[Group2\] IKE Initiator: Rekeying Phase 1, Intf 2, IKE Peer (**IP**) local Proxy Address N/A, remote Proxy Address N/A, SA \(N/A\)
classOk =Y,Y,N,N
}

alarmType =IKE/137
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group1\] Reaper overriding refCnt \[1\] and tunnelCnt \[0\] −− deleting SA!
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Reaper overriding refCnt \[1\] and tunnelCnt \[0\] −− deleting SA!
classOk =Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Reaper overriding refCnt \[0\] and tunnelCnt \[1\] −− deleting SA!
classOk =Y,Y
}

alarmType =IKE/194
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Sending IKE Delete With Reason message: No Reason Provided\.
classOk =Y,Y
regExp =(**IP**) Sending IKE Delete With Reason message: No Reason Provided\.
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Sending IKE Delete With Reason message: Connectivity to Client Lost\.
classOk =Y,Y
regExp =(**IP**) Group \[Group2\] Sending IKE Delete With Reason message: No Reason Provided\.
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Sending IKE Delete With Reason message: Maximum Configured SA Lifetime Exceeded\.
classOk =Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Sending IKE Delete With Reason message: No Reason Provided\.
classOk =Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Sending IKE Delete With Reason message: Maximum Idle Time for Session Exceeded\.
classOk =Y,Y
regExp =(**IP**) Group \[Group1\] Sending IKE Delete With Reason message: No Reason Provided\.
classOk =Y,N
}

alarmType =AUTH/13
classType =visionair.data.DataLong
checked =Y
{
regExp =_ Authentication session closed: handle = (**NUM**)
classOk =Y
}

alarmType =IKE/199
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Reaper corrected an SA that has not decremented the concurrent IKE negotiations counter \(0\)!
classOk =Y,Y
regExp =(**IP**) Group \[Group2\] Reaper corrected an SA that has not decremented the concurrent IKE negotiations counter \(4\)!
classOk =Y,N
}

alarmType =PPP/9
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[.*] User \[(**STR**)\] disconnected\.\. failed authentication \(MD5CHAP \)
classOk =Y,Y
regExp =(**IP**) User \[(**STR**)\] disconnected\.\. failed authentication \(MD5CHAP \)
classOk =Y,Y
}

alarmType =IKE/136
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group1\] IKE session establishment timed out \[MM_WAIT_DELETE\], aborting!
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] IKE session establishment timed out \[MM_WAIT_DELETE
\], aborting!
classOk =Y,Y
}

alarmType =IKE/48
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Error processing payload: Payload ID: 9
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Error processing payload: Payload ID: 14
classOk =Y,Y
regExp =(**IP**) Group \[Group\d\] Error processing payload: Payload ID: .*
classOk =Y,N
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Error processing payload: Payload ID: 14
classOk =Y,Y
regExp =(**IP**) Error processing payload: Payload ID: 1
classOk =Y,N
}

alarmType =PPPDECODE/21
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataIP,
visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) User \[(**STR**)\] IPCP Received CFG−REQ : \(**IP**−ADDR=(**IP**)\)\(PRIM−DNS=(**IP**)\)\(
PRIM_WINS=(**IP**)\)\(SEC−DNS=(**IP**)\)\(SEC−WINS=(**IP**)\)
classOk =Y,Y,Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\] IPCP Received CFG−REQ : \(**IP**−ADDR=(**IP**)\)
classOk =Y,Y,Y,N,N,N,N
regExp =(**IP**) User \[(**STR**)\] IPCP Received CFG−REQ :
classOk =Y,Y,N,N,N,N,N
}

alarmType =PPP/8
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) User \[(**STR**)\] Authenticated successfully with PAP
classOk =Y,Y
regExp =(**IP**) Group \[.*\] User \[(**STR**)\] Authenticated successfully with PAP
classOk =Y,Y
}

alarmType =PPPDECODE/7

classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataLong,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−REQ : \(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)\(ADDR/CTRL−COMP\)\(CALLBACK\)\(UNKNOWN\)\(UNKNOWN\)
classOk =Y,Y,N,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−REQ : \(MRU=(**NUM**)\)\(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)\(ADDR/CTRL−COMP\)\(CALLBACK\)\(UNKNOWN\)\(UNKNOWN\)
classOk =Y,Y,Y,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−REQ : \(MRU=(**NUM**)\)\(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)\(ADDR/CTRL−COMP\)
classOk =Y,Y,Y,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−REQ : \(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)\(ADDR/CTRL−COMP\)
classOk =Y,Y,N,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−REQ : \(MRU=(**NUM**)\)\(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)\(ADDR/CTRL−COMP\)\(CALLBACK\)
classOk =Y,Y,Y,Y
}

alarmType =IKE/167
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Remote peer has failed user authentication − check configured username and password
classOk =Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Remote peer has failed user authentication − check configured username and password
classOk =Y,Y
}

alarmType =IKE/100
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group2\] Received encrypted Oakley Informational packet with invalid payloads, MsgId \(0x(**NUM**)\)
classOk =Y,N,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Received encrypted Oakley Transactional packet with invalid payloads, MsgId \(0x(**NUM**)\)
classOk =Y,Y,Y
regExp =(**IP**) Group \[Group1\] Received encrypted Oakley Informational packet with invalid payloads, MsgId \(0x(**NUM**)\)
classOk =Y,N,Y
regExp =(**IP**) Group \[Group1\] Received encrypted Oakley Quick Mode packet with invalid payloads, MsgId \(0x(**NUM**)\)
classOk =Y,N,Y
regExp =(**IP**) Received encrypted Oakley Informational packet with invalid payloads, MsgId \(0x(**NUM**)\)
classOk =Y,N,Y
}

alarmType =IPSEC/0
classType =visionair.data.DataLong,visionair.data.DataLong
checked =Y
{
regExp =_ (**NUM**): (**NUM**) (**NUM**) (**NUM**) .{12}
classOk =Y,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N

regExp =_ (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**) .{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**)
.{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**) .{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**)
.{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**) .{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**)
.{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**) .{16} (**NUM**): (**NUM**) (**NUM**) (**NUM**) (**NUM**)
.{16}
classOk =Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y
}

alarmType =IKE/120
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group1\] PHASE 2 COMPLETED \(msgid=(**NUM**)\)
classOk =Y,N,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] PHASE 2 COMPLETED \(msgid=(**NUM**)\)
classOk =Y,Y,Y
}

alarmType =IKE/25
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group1\] Received remote Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Received remote Proxy Host data in ID Payload: Address (**IP**), Protocol 0, Port (**NUM**)
classOk =Y,Y,Y,Y
regExp =(**IP**) Group \[Group2\] Received remote Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Received remote Proxy Host data in ID Payload: Address (**IP**), Protocol 0, Port (**NUM**)
classOk =Y,Y,Y,Y
regExp =(**IP**) Group \[VPNC_Base_Group\] Received remote Proxy Host data in ID Payload: Address (**IP**), Protocol 17, Port (**NUM**)
classOk =Y,N,Y,Y
}

alarmType =AUTH/5
classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataIP,visionair.data.DataString, visionair.data.DataString
checked =Y
{
regExp =(**IP**) Authentication rejected: Reason = Unspecified handle = (**NUM**), server = (**IP**), user = (**STR**), domain = (**STR**)<not specified>
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) Authentication rejected: Reason = Unspecified handle = (**NUM**), server = .*\(none\), user = (**STR**), domain = (**STR**)<not specified>
classOk =Y,Y,N,Y,Y
regExp =(**IP**) Authentication rejected: Reason = Unspecified handle = (**NUM**), server = .*\(none\), user = (**STR**), domain = (**STR**)<not specified>
classOk =Y,Y,N,Y,Y
regExp =(**IP**) Authentication rejected: Reason = Unspecified handle = (**NUM**), server = (**IP**), user = (**STR**), domain = (**STR**)
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) Authentication rejected: Reason = Simultaneous logins exceeded for user handle = (**NUM**), server = (**IP**), user = (**STR**), domain = (**STR**)<not specified>
classOk =Y,Y,Y,Y,Y
}

alarmType =IKE/119
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] PHASE 1 COMPLETED

classOk =Y,Y
regExp =(**IP**) Group \[Group1\] PHASE 1 COMPLETED
classOk =Y,N
regExp =(**IP**) Group \[Group2\] PHASE 1 COMPLETED
classOk =Y,N
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] PHASE 1 COMPLETED
classOk =Y,Y
regExp =(**IP**) Group \[VPNC_Base_Group\] PHASE 1 COMPLETED
classOk =Y,N
}

alarmType =IKE/50
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Connection terminated for peer .*Remote Proxy (**IP**), Local Proxy (**IP**)
classOk =Y,Y,Y,Y
regExp =(**IP**) Group \[Group1\] Connection terminated for peer .*Remote Proxy (**IP**), Local Proxy (**IP**)
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group2\] Connection terminated for peer .*Remote Proxy (**IP**), Local Proxy (**IP**)
classOk =Y,N,Y,Y
regExp =(**IP**) Connection terminated for peer .*Remote Proxy (**IP**), Local Proxy (**IP**)
classOk =Y,N,Y,Y
}

alarmType =AUTH/12
classType =visionair.data.DataLong
checked =Y
{
regExp =_ Authentication session opened: handle = (**NUM**)
classOk =Y
}

alarmType =AUTH/28
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataString,visionair.data.DataLong,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) User \[(**STR**)\], Group \[Group2\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: User Requested
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\], Group \[Group1\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: User Requested
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\], Group \[Group2\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: Lost Service
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\], Group \[Group1\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: Unknown
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\], Group \[Group2\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: Idle Timeout
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\], Group \[Group2\] disconnected: Duration: (**STR**) Bytes xmt: (**NUM**) Bytes rcv: (**NUM**) Reason: Idle Timeout
classOk =Y,Y,Y,Y,Y
}

alarmType =IKE/1
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataLong,visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) Received Invalid Cookie notify − retrying last packet
classOk =Y,N,N,N,N,N

regExp =(**IP**) Group \[Group1\] Failed IKE phase 2 source proxy selector negotiation for L2TP/IPSec Expected Selectors: Type 1, Proto 17, Port (**NUM**), Addr (**IP**) Received Selectors: Type 1, Proto 17, Port (**NUM**), Addr (**IP**)
classOk =Y,N,Y,Y,Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Received invalid phase 2 L2TP/IPSec Responder ID payload Expected ID: Type 1, Proto 17, Port (**NUM**), Addr (**IP**) Received ID: Type 4, Proto 0, Port (**NUM**), Addr (**IP**)
classOk =Y,Y,Y,Y,Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Received invalid phase 2 L2TP/IPSec Responder ID payload Expected ID: Type 1, Proto 17, Port (**NUM**), Addr (**IP**) Received ID: Type 4, Proto 0, Port (**NUM**), Addr (**IP**)
classOk =Y,Y,Y,Y,Y,Y
regExp =(**IP**) Group \[Group1\] User \[(**STR**)\] Received invalid phase 2 L2TP/IPSec Responder ID payload Expected ID: Type 1, Proto 17, Port (**NUM**), Addr (**IP**) Received ID: Type 1, Proto 0, Port (**NUM**), Addr (**IP**)
classOk =Y,Y,Y,Y,Y,Y
}

alarmType =AUTH/9
classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Authentication failed: Reason = Unspecified handle = (**NUM**), server = (**IP**), user = (**STR**)
classOk =Y,Y,Y,Y
regExp =_ Authentication failed: Reason = Unspecified handle = (**NUM**), server = (**IP**), user = (**STR**)
classOk =N,Y,Y,Y
regExp =_ Authentication failed: Reason = No response from server handle = (**NUM**), server = (**IP**), user = (**STR**)
classOk =N,Y,Y,Y
}

alarmType =AUTH/4
classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataIP,visionair.data.DataString, visionair.data.DataString
checked =N
{
regExp =(**IP**) Authentication successful: handle = (**NUM**), server = (**IP**), user = (**STR**)
classOk =Y,Y,Y,Y,N
regExp =_ Authentication successful: handle = (**NUM**), server = (**STR**), user = (**STR**)
classOk =N,Y,N,Y,Y
}

alarmType =IKE/75
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataLong,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group1\] Overriding Initiator's IPSec rekeying duration from (**NUM**) to (**NUM**) seconds
classOk =Y,N,Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Overriding Initiator's IPSec rekeying duration from (**NUM**) to (**NUM**) seconds
classOk =Y,Y,Y,Y
}

alarmType =L2TP/47
classType =visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 1, local (**NUM**), serial 0\), reason: Call disconnected for administrative reasons
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 1, local (**NUM**), serial 0\), reason: L2TP peer terminated connection
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 1, local (**NUM**), serial 0\), reason:
classOk =Y,Y,Y

regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 1, local (**NUM**), serial 0\), reason: L2TP peer terminated connection
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 2, local (**NUM**), serial 0\), reason: Insufficient resources to handle this operation now
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 2, local (**NUM**), serial 0\), reason:
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 2, local (**NUM**), serial 0\), reason: L2TP peer terminated connection
classOk =Y,Y,Y
regExp =(**IP**) Session closed on tunnel (**IP**) \(peer 2, local (**NUM**), serial 0\), reason: Call disconnected for administrative reasons
classOk =Y,Y,Y
}

alarmType =IKE/173
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] NAT−Traversal successfully negotiated! IPSec traffic will be encapsulated to pass through NAT devices\.
classOk =Y,Y
regExp =(**IP**) Group \[Group1\] NAT−Traversal successfully negotiated! IPSec traffic will be encapsulated to pass through NAT devices\.
classOk =Y,N
}

alarmType =PPPDECODE/19
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataIP, visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) User \[(**STR**)\] IPCP Sending CFG−REJ : \(**IP**−ADDR=(**IP**)\)
classOk =Y,Y,Y,N,N,N,N
regExp =(**IP**) User \[(**STR**)\] IPCP Sending CFG−REJ : \(**IP**−ADDR=(**IP**)\)\(PRIM−DNS=(**IP**)\)\(PRIM_WINS=(**IP**)\)\(SEC−DNS=(**IP**)\)\(SEC−WINS=(**IP**)\)
classOk =Y,Y,Y,Y,Y,Y,Y
}

alarmType =AUTH/41
classType =visionair.data.DataIP,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Authentication successful: handle = (**NUM**), server = Internal, group = Group2
classOk =Y,Y
regExp =(**IP**) Authentication successful: handle = (**NUM**), server = Internal, group = Group1
classOk =Y,Y
regExp =(**IP**) Authentication successful: handle = (**NUM**), server = Internal, group = VPNC_Base_Group
classOk =Y,Y
}

alarmType =PPPDECODE/17
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP,visionair.data.DataIP, visionair.data.DataIP,visionair.data.DataIP,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) User \[(**STR**)\] IPCP Sending CFG−ACK : \(**IP**−ADDR=(**IP**)\)\(PRIM−DNS=(**IP**)\)\(PRIM_WINS=(**IP**)\)\(SEC−DNS=(**IP**)\)\(SEC−WINS=(**IP**)\)
classOk =Y,Y,Y,Y,Y,Y,Y
regExp =(**IP**) User \[(**STR**)\] IPCP Sending CFG−ACK :
classOk =Y,Y,N,N,N,N,N
regExp =(**IP**) User \[(**STR**)\] IPCP Sending CFG−ACK : \(**IP**−ADDR=(**IP**)\)
classOk =Y,Y,Y,N,N,N,N
}

alarmType =PPPDECODE/2
classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataLong,visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Conn ID (**NUM**) : LCP Sending CFG−ACK : \(MAGIC **NUM**=(**NUM**)\)\(PROT−COMP\)
\(ADDR/CTRL−COMP\)
classOk =Y,Y,N,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Sending CFG−ACK : \(MRU=(**NUM**)\)\(MAGIC **NUM**=(**NUM**)\)
\(PROT−COMP\)\(ADDR/CTRL−COMP\)
classOk =Y,Y,Y,Y
}

alarmType =IKE/22
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataIP
checked =Y
{
regExp =(**IP**) No Group found matching (**STR**) for Pre−shared key peer (**IP**)
classOk =Y,Y,Y
}

alarmType =IKE/49
classType =visionair.data.DataIP,visionair.data.DataString,visionair.data.DataString,visionair .data.DataLong,
visionair.data.DataLong
checked =Y
{
regExp =(**IP**) Group \[Group1\] Security negotiation complete for User \(.∗\) Responder, Inbound SPI = 0x(
NUM), Outbound SPI = 0x(**NUM**)
classOk =Y,N,N,Y,Y
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] Security negotiation complete for User \((**STR**)\) Responder,
Inbound SPI = 0x(**NUM**), Outbound SPI = 0x(**NUM**)
classOk =Y,Y,Y,Y,Y
regExp =(**IP**) Group \[Group1\] Security negotiation complete for User \((**STR**)\) Initiator, Inbound SPI = 0
x(**NUM**), Outbound SPI = 0x(**NUM**)
classOk =Y,N,Y,Y,Y
regExp =(**IP**) Group \[Group1\] Security negotiation complete for User \(.∗\) Initiator, Inbound SPI = 0x(
NUM), Outbound SPI = 0x(**NUM**)
classOk =Y,N,N,Y,Y
}

alarmType =IKE/0
classType =visionair.data.DataIP,visionair.data.DataLong
checked =Y
{
regExp =_ Could not find centry for IPSec SA delete message
classOk =N,N
regExp =(**IP**) Group \[Group1\] received an unencrypted packet when crypto active!! Dropping packet\.
classOk =Y,N
regExp =(**IP**) received an unencrypted packet when crypto active!! Dropping packet\.
classOk =Y,N
regExp =(**IP**) Group \[Group1\] Phase 2 creation failed, found existing entry \(msg id 0x(**NUM**)\)
classOk =Y,Y
regExp =(**IP**) Group \[Group1\] Can’t create conn entry!
classOk =Y,N
regExp =(**IP**) Group \[Group1\] All IPSec SA proposals found unacceptable!
classOk =Y,N
regExp =(**IP**) Group \[VPNC_Base_Group\] All IPSec SA proposals found unacceptable!
classOk =Y,N
}

alarmType =PPPDECODE/9
classType =visionair.data.DataIP,visionair.data.DataLong,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−NAK : \(AUTH=MSCHAP−V2\)

classOk =Y,Y,N
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−NAK : \(AUTH = (**STR**)\)
classOk =Y,Y,Y
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−NAK :
classOk =Y,Y,N
regExp =(**IP**) Conn ID (**NUM**) : LCP Received CFG−NAK : \(AUTH=MSCHAP−V1\)
classOk =Y,Y,N
}

alarmType =IKE/66
classType =visionair.data.DataIP,visionair.data.DataString
checked =Y
{
regExp =(**IP**) Group \[Group1\] IKE Remote Peer configured for SA: WindowsArchimede
classOk =Y,N
regExp =(**IP**) Group \[Group2\] User \[(**STR**)\] IKE Remote Peer configured for SA: CiscoArchimede
classOk =Y,Y
regExp =(**IP**) Group \[Group1\] IKE Remote Peer configured for S: WindowsArchimede
classOk =Y,N
regExp =(**IP**) Group \[VPNC_Base_Group\] IKE Remote Peer configured for SA: CiscoArchimede
classOk =Y,N
}

Bibliographie

Akaike H. (1973). Information theory as an extension of the maximum likelihood principle. *Dans ISIT'73: 2nd International Symposium of Information Theory*, pages 267–281. (cité aux pages 76 et 77)

Anderberg M. R. (1973). *Cluster Analysis for Applications*. Monographs and Textbooks on Probability and Mathematical Statistics. Academic Press, Inc. (cité à la page 118)

Ball R., Fink G. A. et North C. (2004). Home-centric visualization of network traffic for security administration. *Dans VizSEC/DMSEC'04: Workshop on Visualization and Data Mining for Computer Security*, pages 55–64. ACM. (cité à la page 90)

Barr M. et Wells C. (1990). *Category Theory for Computing Science*. Prentice-Hall. (cité aux pages 7, 8 et 33)

Bernstein A., Provost F. et Hill S. (2005). Towards intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *Transactions on Knowledge and Data Engineering*, 17(4):503–518. (cité aux pages 72 et 158)

Bert D., Echahed R., Jacquet P., Potet M.-L. et Reynaud J.-C. (1995). Spécification, généricité, prototypage: Aspects du langage LPG. *Technique et Science Informatiques*, 14:1097–1129. (cité aux pages 8 et 33)

Borceux F. (1994). *Handbook of Categorical Algebra*. Numéro 50 de Encyclopedia of Mathematics and its Applications. Cambridge University Press. Trois volumes. (cité à la page 7)

Boulicaut J.-F., Klemettinen M. et Mannila H. (1999). Modeling KDD processes within the inductive database framework. *Dans DaWaK'99: First International Conference on Data Warehousing and Knowledge Discovery*, pages 293–302. Springer-Verlag. (cité à la page 72)

Calders T., Lakshmanan L. V. S., Ng R. T. et Paredaens J. (2006). Expressive power of an algebra for data mining. *Transactions on Database Systems*, 31(4): 1169–1214. (cité à la page 72)

Carboni A., Lack S. et Walters R. (1993). Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158. (cité à la page 26)

Cicchello O. et Kremer S. C. (2003). Inducing grammars from sparse data sets: a survey of algorithms and results. *Journal of Machine Learning Research*, 4:603–632. (cité à la page 141)

Colmerauer A. et Roussel P. (1996). Naissance de Prolog. *History of Programming Languages*, pages 37–52. (cité à la page 55)

Cook D. J. et Holder L. B. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255. (cité à la page 86)

Cormode G. et Muthukrishnan S. (2005). Space efficient mining of multigraph streams. *Dans PODS'05: 24th SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 271–282. ACM. (cité à la page 90)

Corradini A., Montanari U., Rossi F., Ehrig H., Heckel R. et Löwe M. (1997). *Algebraic approaches to graph transformation. Part I: basic concepts and double pushout approach*, pages 163–245. World Scientific Publishing Co. (cité à la page 10)

Cover T. M. et Thomas J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience. (cité à la page 79)

Cuppens F. (2001). Managing alerts in a multi-intrusion detection environment. *Dans ACSAC'01: 17th Annual Computer Security Applications Conference*, pages 22–31. IEEE Computer Society. (cité aux pages 89 et 91)

Cuppens F. et Miège A. (2002). Alert correlation in a cooperative intrusion detection framework. *Dans SP'02: Symposium on Security and Privacy*, pages 202–215. IEEE Computer Society. (cité à la page 89)

Dain O. et Cunningham R. (2001a). Building scenarios from a heterogeneous alert stream. *Dans SMC-IAW'01: Systems, Man and Cybernetics-Information Assurance Workshop*, pages 231–235. IEEE Computer Society. (cité à la page 91)

Dain O. et Cunningham R. (2001b). Fusing a heterogeneous alert stream into scenarios. *Dans DMSEC'01: Workshop on Data Mining for Security Applications*, pages 1–13. ACM. (cité à la page 91)

Dampney C. N. G., Johnson M. S. J. et Monro G. P. (1992). An Illustrated Mathematical Foundation for ERA. *The Unified Computation Laboratory*, pages 77–83. (cité à la page 8)

Debar H., Curry D. et Feinstein B. (2007). RFC 4765: The intrusion detection message exchange format (IDMEF). (cité à la page 89)

Demsar J., Zupan B., Leban G. et Curk T. (2004). Orange: from experimental machine learning to interactive data mining. *Dans PKDD'04: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 537–539. Springer-Verlag. (cité aux pages 1 et 74)

Domingos P. (1999). The role of occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425. (cité à la page 83)

Dumas J.-G. et Duval D. (2006). Vers une modélisation diagrammatique de la bibliothèque c++ d'algèbre linéaire linbox. *Dans LMO'06: conférence Langages et Modèles à Objet*, pages 117–132. (cité à la page 9)

Duval D. (2003). Diagrammatic specifications. *Mathematical Structures in Computer Science*, 13(6):857–890. (cité à la page 9)

Duval D. et Reynaud J.-C. (2006). Diagrammatic logic and exceptions: an introduction. *Mathematics, Algorithms, Proofs*. (cité aux pages 9 et 26)

Ehresmann C. (1965). *Catégories et structures*. Dunod. (cité à la page 7)

Ehrig H. et Mahr B. (1985). *Fundamentals of Algebraic Specification I*. Springer-Verlag. (cité aux pages 8 et 33)

Fayyad U., Piatetsky-Shapiro G. et Smyth P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54. (cité à la page 66)

Frawley W. J., Piatetsky-Shapiro G. et Matheus C. J. (1992). Knowledge discovery in databases: an overview. *AI Magazine*, 13(3):57–70. (cité à la page 67)

Fromont E., Blockeel H. et Struyf J. (2006). Integrating decision tree learning into inductive databases. *Dans KDID'06: Knowledge Discovery in Inductive Databases*, volume 4747 de *LNCS*, pages 81–96. Springer-Verlag. (cité à la page 72)

Goguen J. (1991a). Types as theories. *Dans Topology and Category Theory in Computer Science*, pages 357–390. Oxford. (cité à la page 8)

Goguen J. A. (1991b). A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67. (cité à la page 7)

Goguen J. A. et Burstall R. M. (1992). Institutions: abstract model theory for specification and programming. *Journal ACM*, 39(1):95–146. (cité à la page 9)

Grünwald P. D., Myung I. J. et Pitt M. A. (2005). *Advances in Minimum Description Length: Theory and Applications (Neural Information Processing)*. The MIT Press. (cité aux pages 81 et 83)

Grünwald P. D. et Vitányi P. M. B. (2003). Kolmogorov complexity and information theory. *Journal of Logic, Language and Information*, 12(4):497–529. (cité à la page 79)

Gutierrez-Pena E. et Walker S. (2001). A bayesian predictive approach to model selection. *Journal of Statistical Planning and Inference*, 93:259–276. (cité à la page 78)

Han J., Fu Y., Wang W., Koperski K. et Zaiane O. (1996). DMQL: A data mining query language for relational databases. *Dans SIG-MOD'96 DKMD - Workshop on Research Issues in Data Mining and Knowledge Discovery*. (cité à la page 72)

Hansen M. H. et Yu B. (2001). Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454): 746–774. (cité à la page 77)

Henkel J. et Diwan A. (2003). Discovering algebraic specifications from java classes. *Dans ECOOP'03 : 17th European Conference of Object-Oriented Programming*, pages 157–177. Springer-Verlag. (cité à la page 8)

Imielinski T. et Mannila H. (1996). A database perspective on knowledge discovery. *Communications of the ACM*, 39:58–64. (cité aux pages 1 et 72)

Jannink J., Pichai S., Verheijen D. et Wiederhold G. (1998). Encapsulation and composition of ontologies. *Dans AAAI'98: 15th National Conference on Artificial Intelligence - Workshop on AI and Information Integration*, pages 157–164. (cité à la page 10)

Johnson M. et Dampney C. N. G. (2001). On category theory as a (meta) ontology for information systems research. *Dans FOIS'01: Formal Ontology in Information Systems*, pages 59–69. ACM. (cité à la page 10)

Johnson T., Lakshmanan L. V. S. et Ng R. T. (2000). The 3w model and algebra for unified data mining. *Dans VLDB'00: 26th International Conference on Very Large Databases*, pages 21–32. Morgan Kaufmann Publishers. (cité aux pages 1 et 72)

Julisch K. (2003). Clustering intrusion detection alarms to support root cause analysis. *Transactions on Information and System Security*, 6(4):443–471. (cité à la page 90)

Julisch K. et Dacier M. (2002). Mining intrusion detection alarms for actionable knowledge. *Dans KDD'02: eighth International Conference on Knowledge Discovery and Data Mining*, pages 366–375. ACM. (cité aux pages 89 et 90)

Karabatsos G. (2006). bayesian nonparametric model selection and model testing. *Journal of Mathematical Psychology*, 50:123–148. (cité à la page 78)

Keogh E., Lonardi S. et Ratanamahatana C. A. (2004). Towards parameter-free data mining. *Dans KDD'04: tenth International Conference on Knowledge Discovery and Data Mining*, pages 206–215. ACM. (cité à la page 85)

Klemettinen M., Mannila H. et Toivonen H. (1999). Rule discovery in telecommunication alarm data. *Dans Journal of Network and Systems Management*, volume 7, pages 395–423. (cité aux pages 89 et 90)

Kohavi R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Dans IJCAI'95: International Joint Conferences on Artificial Intelligence*, pages 1137–1145. (cité à la page 78)

Koike H. et Ohno K. (2004). Snortview: visualization system of snort logs. *Dans VizSEC/DMSEC'04: Workshop on Visualization and Data Mining for Computer Security*, pages 143–147. ACM. (cité aux pages 89 et 90)

Kramer S., De Raedt L. et Helma C. (2001). Molecular feature mining in HIV data. *Dans KDD'01: eighth International Conference on Knowledge Discovery and Data Mining*, pages 136–143. ACM. (cité à la page 72)

Krauth W. (1996). Introduction to monte carlo algorithms. Rapport technique, Laboratoire de Physique Statistique, Ecole Normale Supérieure de Paris. (cité à la page 116)

Krötzsch M., Hitzler P., Ehrig M. et Sure Y. (2005). Category theory in ontology research: Concrete gain from an abstract approach. Rapport technique, AIFB, Universität Karlsruhe. (cité à la page 10)

Kverh B. et Leonardis A. (2004). A generalisation of model selection criteria. *Pattern Analysis & Applications*, 7(1):51–65. (cité à la page 76)

Lakkaraju K., Yurcik W. et Lee A. J. (2004). Nvisionip: netflow visualizations of system state for security situational awareness. *Dans VizSEC/DMSEC'04: Workshop on Visualization and Data Mining for Computer Security*, pages 65–72. ACM. (cité aux pages 89 et 90)

Lakshmanan L. V., Ng R. T., Wang C. X., Zhou X. et Johnson T. (2002). The generalized MDL approach for summarization. *Dans VLDB'02 : 28th International Conference on Very Large Databases*, pages 766–777. (cité à la page 86)

Lebarbier E. et Mary-Huard T. (2006). Une introduction au critère bic : Fondements théoriques et interprétation. *Journal de la société française de statistique*, pages 39–57. (cité à la page 77)

Lee S. D. et De Raedt L. (2004). *Database Support for Data Mining Applications*, volume 2682 de *LNCS*, chapitre Constraint Based Mining of First Order Sequences in SeqLog, pages 154–173. Springer-Verlag. (cité à la page 72)

Lee W., Stolfo S. J. et Mok K. W. (1999a). A data mining framework for building intrusion detection models. *Dans SP'99 Symposium on Security and Privacy*, pages 120–132. IEEE Computer Society. (cité aux pages 89 et 90)

Lee W., Stolfo S. J. et Mok K. W. (1999b). Mining in a data-flow environment: experience in network intrusion detection. *Dans KDD'99: fifth International Conference on Knowledge Discovery and Data Mining*, pages 114–124. ACM. (cité aux pages 89 et 90)

Li M. et Vitány P. (1997). *Introduction to Kolmogorov complexity and its applications*. Springer-Varlag. (cité aux pages 79 et 162)

Li R. et Pereira L. M. (1995). Application of category theory in model-based diagnostic reasoning. *Dans FLAIRS'95 : Florida Artificial Intelligence Research Society*, pages 123–127. (cité aux pages 8 et 9)

Livnat Y., Agutter J., Moon S., Erbacher R. F. et Foresti S. (2005). A visualization paradigm for network intrusion detection. *Dans SMC-IAW'05: Systems, Man and Cybernetics-Information Assurance Workshop*, pages 92–99. IEEE Computer Society. (cité aux pages 89 et 90)

Mac Lane S. (1971). *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer-Verlag. (cité aux pages 7 et 51)

Mannila H., Toivonen H. et Verkamo A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289. (cité à la page 74)

Mehta M., Agrawal R. et Rissanen J. (1996). Sliq: A fast scalable classifier for data mining. *Dans EDBT'96: Proceedings of the 5th International Conference*

on Extending Database Technology, pages 18–32. Springer-Verlag. (cité à la page 84)

Mierswa I., Wurst M., Klinkenberg R., Scholz M. et Euler T. (2006). Yale: Rapid prototyping for complex data mining tasks. *Dans KDD'06: International Conference on Knowledge Discovery and Data Mining*, pages 935–940. ACM. (cité aux pages 1 et 74)

Mirkovic J. et Reiher P. (2004). A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Computer Communication Review*, 34(2):39–53. (cité à la page 142)

Morik K. (2000). The representation race - preprocessing for handling time phenomena. *Dans ECML'00: 11th European Conference on Machine Learning*, pages 4–19. Springer-Verlag. (cité à la page 74)

Morin B. et Debar H. (2003). Correlation of intrusion symptoms: an application of chronicles. *Dans RAID'03: 6th International Symposium on Recent Advances in Intrusion Detection*, volume 2820 de *Lecture Notes in Computer Science*, pages 97–112. Springer-Varlag. (cité aux pages 89 et 90)

Qin X. et Lee W. (2003). Statistical causality analysis of infosec alert data. *Dans RAID'03: 6th International Symposium on Recent Advances in Intrusion Detection*, volume 2820 de *LNCS*, pages 73–93. Springer-Verlag. (cité aux pages 89 et 90)

Qin X. et Lee W. (2004). Discovering novel attack strategies from infosec alerts. *Dans ESORICS'04: European Symposium on Research in Computer Security*, volume 3193 de *LNCS*, pages 439–456. Springer-Verlag. (cité aux pages 89 et 90)

Reinefeld A. et Marsland T. A. (1994). Enhanced iterative-deepening search. *Transactions on Pattern Analysis and Machine Intelligence*, 16(7):701–710. (cité à la page 116)

Rissanen J. (1978). Modelling by the shortest data description. *Automatica*, 14:465–471. (cité aux pages 76, 77 et 81)

Rissanen J. (1989). *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co. Inc. (cité à la page 81)

Roesch M. (1999). Snort - lightweight intrusion detection for networks. *Dans LISA'99: 13th USENIX Conference on System administration*, pages 229–238, Berkeley, CA, USA. USENIX Association. (cité à la page 89)

Rosebrugh R. et Wood R. (1992). Relational databases and indexed categories. *Dans CMS'92 : Canadian Mathematical Society Conference*, pages 391–407. (cité aux pages 2 et 8)

Schwarz G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464. (cité aux pages 76 et 77)

Shannon C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423. (cité à la page 79)

Srinivas Y. V. et Jüllig R. (1995). Specware: Formal support for composing software. *Dans MPC'95: Mathematics of Program Construction*, pages 399–422. Springer-Verlag. (cité à la page 9)

Teoh S. T., Zhang K., Tseng S.-M., Ma K.-L. et Wu S. F. (2004). Combining visual and automated data mining for near-real-time anomaly detection and analysis in bgp. *Dans VizSEC/DMSEC'04: Workshop on Visualization and Data Mining for Computer Security*, pages 35–44. ACM. (cité à la page 90)

Valdes A. et Skinner K. (2001). Probabilistic alert correlation. *Dans RAID'01: 4th International Symposium on Recent Advances in Intrusion Detection*, volume 2212 de *LNCS*, pages 54–68. Springer-Verlag. (cité aux pages 89 et 91)

Vautier A., Cordier M.-O. et Quiniou R. (2005a). Extension des bases de données inductives pour la découverte de chroniques avec contraintes temporelles. *Dans EGC'05: Extraction et Gestion des Connaissances*, pages 421–432. (cité à la page 72)

Vautier A., Cordier M.-O. et Quiniou R. (2005b). An inductive database for mining temporal patterns in event sequences (short version). *Dans IJCAI'05: International Joint Conferences on Artificial Intelligence*, pages 1640–1641. (cité à la page 72)

Vautier A., Cordier M.-O. et Quiniou R. (2007). Towards data mining without information on knowledge structure. *Dans PKDD'07: 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 300–311. (cité à la page 128)

Vautier A., Cordier M.-O., Quiniou R. et Ducassé M. (2006a). Agrégation d'alarmes faiblement structurées. *Dans EGC'06: Extraction et Gestion des Connaissances - Atelier Fouille de Données Temporelles*, pages 421–432. (cité à la page 152)

Vautier A., Cordier M.-O., Quiniou R. et Ducassé M. (2006b). Visualization of internet flow records. Rapport technique, Irisa. (cité à la page 153)

Williamson K., Healy M. et Barker R. A. (2001). Industrial applications of software synthesis via category theory-case studies using specware. *Automated Software Engineering*, 8(1):7–30. (cité à la page 9)

Witten I. H. et Frank E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann. (cité aux pages 1 et 118)

Yin X., Yurcik W., Treaster M., Li Y. et Lakkaraju K. (2004). Visflowconnect: Netflow visualizations of link relationships for security situational awareness. *Dans VizSEC/DMSEC'04: Workshop on Visualization and Data Mining for Computer Security*, pages 45–54. ACM. (cité aux pages 89 et 90)

Zaki M. J., Parimi N., De N., Gao F., Phoophakdee B., Urban J., Chaoji V., Hasan M. A. et Salem S. (2005). Towards generic pattern mining. *Dans PReMI'2005: Pattern Recognition and Machine Intelligence*, pages 91–97. Springer-Verlag. (cité à la page 74)

Zanero S. et Savaresi S. M. (2004). Unsupervised learning techniques for an intrusion detection system. *Dans SAC'04: Symposium on Applied Computing*, pages 412–419. ACM. (cité aux pages 89 et 91)

Zúniga G. L. (2001). Ontology: its transformation from philosophy to information systems. *Dans FOIS'01: International Conference on Formal Ontology in Information Systems*, pages 187–197. ACM. (cité à la page 9)

Index

Résumé

Les travaux de recherche présentés dans cette thèse ont pour objectif de proposer un cadre à la fouille de données pour la découverte de connaissances lorsque l'on n'a pas d'information a priori sur la structure des connaissances. Le cadre proposé est générique et s'appuie sur la théorie des catégories, plus précisément sur les esquisses qui sont un outil de spécification. Nous proposons le concept d'esquisse relationnelle qui enrichit les esquisses par les notions d'ensemble des parties et de relation. De plus, nous associons une esquisse relationnelle à une implémentation, ce qui constitue un schéma. Ce cadre permet de spécifier des données de natures diverses et des opérateurs de fouille de données variés. L'exécution des opérateurs de fouille de données pour extraire des modèles est rendue possible grâce à l'unification de la spécification des opérateurs avec la spécification des données. Une méthode générique, basée sur la complexité de Kolmogorov, est proposée pour évaluer la qualité des modèles et leur capacité à résumer les données. Elle s'appuie notamment sur la relation de couverture qui lie les modèles aux données.

L'application ayant motivé ces travaux est l'analyse de journaux d'alarmes réseau dans le but d'aider des utilisateurs à résumer ces journaux pour extraire des connaissances. Les applications de ce travail ont été effectuées dans le cadre d'un CRE (contrat de recherche externalisée) avec France-Télécom R&D Lannion. La première application porte sur le résumé d'alarmes VPN non structurées. La seconde application concerne l'analyse des flux réseau importants sur « l'épine dorsale » d'internet pour la détection d'attaques DDoS (Distributed Denial Of Service).

Abstract

The aim of this thesis is to propose a data mining framework for discovering knowledge when the user has no a priori information about the knowledge structure. The proposed framework is generic and based on the category theory, more precisely on the sketches that are a specification tool. We propose the concept of relational sketches that enhances the sketches with the concepts of power set and relation. Moreover we associate a relational sketch to an implementation, which defines a schema. This framework enables the specification of various data types and various data mining algorithms. The execution of data mining algorithms for model extraction is enabled by the unification of algorithm specifiations with the data specification. A generic methodology, based on the Kolmogorov complexity, is proposed to evaluate the model quality and their ability to summarize the data. The evaluation essentially relies on the covering relation that links the model and data.

The application which motivated this work is an analysis of network alarm logs. The aim is to help users to summarize these logs to extract knowledge. The applications of this work have been carried out within a CRE (contract research outsourced) with France Télécom R&D Lannion. The first application focuses on the summarization of unstructured VPN alarms. The second application concerns the analysis of network flows from the internet "backbone" to detect DDoS attacks (Distributed Denial Of Service).