Université Blaise Pascal
UFR Sciences Exactes et Naturelles
Département de Mathématiques et Informatique
63177 AUBIERE CEDEX

Maîtrise d'Informatique

Fouille de données interactive par navigation

Caroline Noyer

Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes

Responsable du TER : R. Medina

Remerciements

Je tiens à remercier Messieurs Raoul Medina, Olivier Raynaud et Lhouari Nourine, membres de l'équipe Algorithmique, pour m'avoir accueillie au sein de leur équipe et m'avoir permis d'effectuer ce stage dans les meilleures conditions.

Je remercie spécialement Raoul Medina, mon maître de stage, pour le soutien technique qu'il m'a apporté.

Je remercie également mes collègues stagiaires pour m'avoir permis de travailler dans la bonne humeur et donc dans bonnes conditions de travail.

Résumé

L'équipe Algorithme souhaitait disposer d'une bibliothèque de fonctions permettant d'implanter rapidement leurs algorithmes. Lors d'une étude de cas, un ensemble de Types de Données Abstraits (TDA) a été défini. L'objectif de ce TER était d'en réaliser l'implémentation à l'aide de la STL en C++. La bibliothèque de fonctions qui a été réalisée dans ce TER permettra, une fois interfacée avec la plate-forme de fouille de données en cours de développement, de naviguer dans les règles d'association selon un technique mise au point par l'équipe Algorithmique.

Abstract

The Algorithmique team wanted a set of C++ functions in order to ease the development and the implementation of their algorithms. A set of Abstract Data Type (ADT) was first defined during a "Case Study". Aim of the current work was to implement those ADT in C++ using the STL. The obtained functions library, once connected to the ongoing datamining framework, will allow the navigation among association rules using a process developped by the Algorithmique team.

Table des matières

1 Introduction

Aujourd'hui la quantité d'informations stockées dans les bases de données est devenue très importante, il est donc nécessaire d'avoir des outils efficaces pour en extraire des données pertinentes. Il devient utile de développer des outils d'aide à la décision. C'est dans ce domaine que s'inscrit ce travail.

Le data mining correspond à l'ensemble des techniques et des méthodes qui, à partir de gros volumes de données, permettent d'obtenir des connaissances exploitables. Son utilité est grande dès lors qu'une entreprise possède un grand nombre d'informations stockées sous forme de bases de données.

Les applications du data-mining sont multiples :
- la grande distribution
- la vente par correspondance
- les opérateurs de télécommunication
- les banques et les assurances
- la bio-informatique, notamment pour l'étude des génomes en trouvant des gènes dans des séquences d'ADN.

Un problème classique de fouille de données est la recherche de règles d'associations [5] dans une relation binaire.

Une relation binaire peut être vue comme une collection d'ensemble d'items.

Un règle d'association est une expression du type $X \rightarrow Y$, où X et Y sont des ensembles d'items (ou des items). Ce qui signifie intuitivement que si une transaction contient X alors elle contient aussi probablement Y.

Des qualités de mesure sont associées à ces règles d'association :
- le *support* d'une règle définit sa portée, c'est-à-dire la proportion d'objet contenant tous les attributs de la règle. On peut la définit de la façon suivante :

$$Support(I \rightarrow a) = \mid \{ligne \in BD \mid I \subseteq ligne\} \mid$$

- la *confiance* d'une règle définit sa précision, c'est-à-dire la proportion d'objets contenant le conséquent parmi ceux qui contiennent l'antécédent.

Le problème le plus important réside dans le fait que le nombre de règles générées peut être exponentiel par rapport au nombre d'items. Une solution pour réduire ce nombre de règles est de géréner uniquement un sous-ensemble de règles dont on pourra dériver toutes les autres. Un tel ensemble est appelé *base de règles*. Cependant cet ensemble peut lui aussi contenir un nombre exponentiel de règles.

Les méthodes actuelles de génération de règles d'association sont itératives et nécessitent donc une interaction avec l'utilisateur. Les grandes étapes sont les suivantes :
- Générer un ensemble de règles (peut-être exponentiel)
- Parcourir les règles pour trouver des règles d'association intéressantes
- Utiliser ce qui a été trouvé pour affiner la recherche, et recommencer le processus

Cette approche est une approche "aval" : pour pouvoir naviguer dans les règles, celles-ci doivent avoir été calculées a priori. Dans [5], l'approche proposée est une méthode "amont" : les règles sont générées en partie au moment de la navigation. Seules les règles permettant de poursuivre la navigation sont générées. La navigation s'effectuer par spécialisation de règles.

Le nombre de règles est polynomial et le calcul est lui-même polynomial. Cette méthode permet donc une interactivté "temps réel" avec l'utilisateur et celui-ci peut rester derrière son écran durant tout le processus de fouille de données, contrairement aux autres méthodes qui nécessitent toutes des périodes de traitement très long.

L'utilisateur est donc placer au centre de la recherche et pourra ainsi choisir lui-même quelle règle il veut spécialiser suivant ses besoins.

2 Présentation du laboratoire

Le L.I.M.O.S. (Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes) est un laboratoire C.N.R.S. (Centre National de la Recherche Scientifique). Il est au sein de l'I.S.I.M.A. (Institut Supérieur d'Informatique de Modélisations et des Applications).

Ce laboratoire comprend entre autres 39 enseignants-chercheurs, 30 doctorants et 3 post-doctorants. Il centre ses recherches autour de l'étude de modèles et d'outils informatiques pour la conception, la présentation, l'évaluation, la prévision, le contrôle, l'optimisation, de systèmes complexes de nature organisationnelle : systèmes de transport, de télécommunications, de production, écosystèmes...

Les différents programmes de recherche du laboratoire sont groupés en trois axes :
- la recherche opérationnelle et l'aide à la décision :
  - optimisation des réseaux,
  - optimisation combinatoire,
  - intelligence artificielle,
  - programmation logique et solveurs,
  - réseaux neuronaux,
  - applications à la planification de systèmes de production ou de transport, au dimensionnement de réseaux de télécommunications, à la modélisation de systèmes économiques, à la prévision en astronomie,
  - modélisation et simulation d'écosystèmes (algue Caulerpa,...)
  - collaborations avec France TELECOM, la SNCF, MICHELIN, le SPACE EUROPEAN LABORATORY, le CEMAGREF, EDF, l'INRA, incubation de l'entreprise NUMTECH ...
  - algorithmique ;
- les systèmes d'informations et de communication :
  - datamining,
  - bases de données,
  - données semi-structurées et intégration de données issues du web,
  - systèmes d'information,
  - réseaux locaux sans fil,
  - télémédecine,
  - télédomotique,
  - interfaces réseaux/capteurs,
  - collaborations avec France TELECOM, le CHU de Clermont-Fd, la Société Générale, AEGIS, incubation de l'entreprise AUVERTECH ;
- la modélisation, l'organisation, le pilotage des systèmes de production :
  - simulation, modélisation et contrôle d'ateliers automatisés,
  - optimisation de plates-formes logistiques,
  - organisation et design de structures de production,
  - contrôle qualité,
  - collaborations avec MICHELIN, RIVEX, VALEO, les transports NICOLAS, ...

Mon TER s'est effectué au sein de l'équipe Algorithmique, dont les thèmes sont à l'intersection des deux premiers axes du laboratoire. L'un des thèmes de cette équipe, dans lequel s'inscrit mon TER, est l'étude de bases d'implications dont l'une des applications possibles est le Data Mining.

3 Positionnement du problème

3.1 Contexte et objectifs

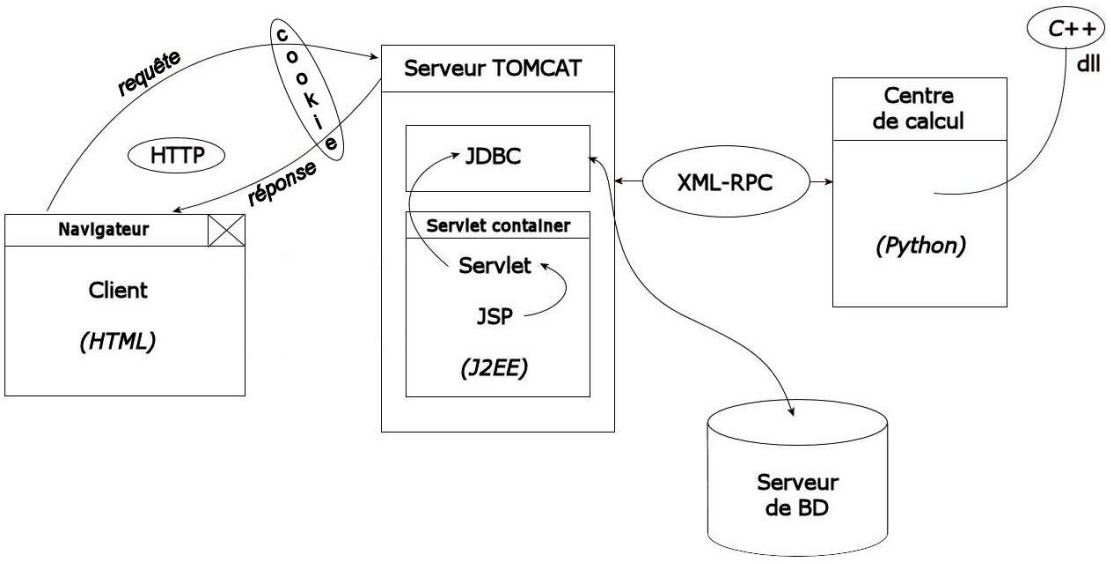
L'équipe Algorithmique souhaite développer une plateforme de fouilles de données implémentant les différents algorithmes et techniques développés par ses membres.

La présente étude intervient dans la génération de règles d'association d'une table binaire de la forme : $P \rightarrow \overline{P} \setminus P$ et avec $\overline{P} \setminus P \neq \emptyset$.

L'objectif est double. Premièrement il est de créer une bibliothèque de fonctions appelables soit de la plateforme (c'est-à-dire à partir du serveur de calcul), soit depuis un programme C++ (en standalone). Le deuxième est de standardiser les échanges de données entre les différents modules, ceci étant possible via l'utilisation de fichiers XML (pour sauvegarder les différentes structures).

Différentes structures doivent être gérées comme une table, un ordre ou un arbre de navigation dans les règles.

Différents travaux ont déjà été réalisés autour de cette application qui s'articule en trois grandes parties. Le client se connecte sur le serveur d'application qui envoie la requête de traitement sur le centre de calcul. Celui-ci renvoie le résultat au client par l'intermédiaire du serveur d'application. En voici donc l'architecture :



Lors d'un projet de génie logiciel de Maîtrise Informatique, les parties *Client* et *Serveur d'application* ont été traitées. Un projet de DESS SIAD a pris en charge la discrétisation des bases de données et vient donc s'inscrire au niveau de *Serveur BD*. Enfin, une étude de cas a permis de définir différents outils qui permettront la mise en place du *Centre de calcul* par l'intermédiaire d'une bibliothèque en C++. Cette étude de cas a essentiellement porté sur la définition de Types de Données Abstraits (TDA) nécessaires

à la navigation dans les règles. Toutefois ces TDA devaient être adaptés à une utilisation dans d'autres algorithmes développés par l'équipe. Le but de ce TER était d'implanter ces TDA en C++ à l'aide de la STL.

Le planning défini pour la réalisation de cette bibliothèque de fonctions a été le suivant :

Semaine 1	Prise en main de la bibliothèque XML (libxml2)
Semaine 2	TDA Table
Semaine 3	TDA SetOfElements
Semaine 4	TDA Order
Semaine 5	Calcul des inf-irréductibles et des sup-irréductibles
Semaine 6	TDA SetOfNodes
Semaine 7	TDA RuleTree
Semaine 8	Spécialisation d'une règle
Semaine 9	Saut dans l'arborescence
Semaine 10	Fonctions utiles à l'application
Semaine 11 & 12	Finalisation, tests et rapport

3.2 Rappels sur les ensembles ordonnés

• Relation d'ordre

Soit E un ensemble d'attributs et $<$ une relation antisymétrique, réflexive et transitive. On appelle alors P = (E,$<$) un *ensemble ordonné*.

On dit que deux éléments a et b sont *comparables* dans P si a$<$b ou b$<$a, sinon on dit qu'ils sont incomparables.

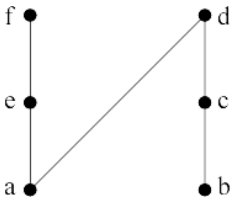
On définit l'ensemble des successeurs de a, *succ(a)*, représentant l'ensemble des éléments supérieurs à a. On a donc : $succ(a) = \{b \in E \mid a < b\}$.

On définit de la même manière l'ensemble des prédecesseurs de a, *pred(a)*, représentant les élements inférieurs à a. On a donc : $pred(a) = \{b \in E \mid b < a\}$.

Un élément a *couvre* un élément b, si et seulement si a$<$b et si il n'existe pas de c tel que a$<$c$<$b, on note $a \prec b$. On définit alors deux notions :

- $ImSucc(a) = \{b \in E \mid b \prec a\}$
- $ImPred(a) = \{b \in E \mid a \prec b\}$

• **Exemple**



succ(a) = {e,f,d}
pred(d) = {a,c,b}
imPred(d) = {a,c}
imSucc(e) = {f}

succ(c) = {d}
pred(a) = ∅

• **Idéal d'un ordre**

Soit P = (E,<) un ordre. I⊆E est un *idéal* si et seulement si, quelque soit a∈I et b∈E, tel que b<a, alors b∈I.

L'*idéal principal* de a, que l'on note $\downarrow a$, est un idéal contenant a, tel que a est le seul élément maximum. On peut écrire : $\downarrow a = a \cup pred(a)$.

• **Filtre d'un ordre**

Soit P = (E,<) un ordre. F⊆E est un *filtre* si et seulement si, quelque soit a∈F et b∈E, tel que a<b, alors b∈F.

Le *filtre principal* de a, que l'on note $\uparrow a$, est un filtre contenant a, tel que a est le seul élément minimum. On peut écrire : $\uparrow a = a \cup succ(a)$.

• **Treillis**

Soit P = (E,<) un ordre non vide. Si a∨b ∈ E et a∧b ∈ E, ∀a, b∈E, alors P est un *treillis*.

L'ensemble des idéaux d'un ordre P ordonnés par inclusion forme un treillis que l'on appelle *treillis des idéaux*.

Un *inf-irréductible* est un élément du treillis qui a au plus un seul successeur.

Un *sup-irréductible* est un élément du treillis admettant au plus un seul prédécesseur.

• **Définition sur les fermés (par rapport à une relation binaire)**

La *fermeture d'un idéal* est calculée par l'intersection des objets contenant l'idéal. Si la fermeture est égale à l'idéal lui-même, on dit que celui-ci est un *fermé*.

On note la fermeture de I : $\overline{I}$

• **Régle d'association**

Nous nous intéresserons ici aux règles d'association de la forme suivante : $X \rightarrow x$ avec avec $x \in \overline{X} \setminus X$. En conséquence, on s'intéresse aux règles dotn la partie gauche (antécédent) n'est pas un fermé. On considère aussi uniquement les règles exactes (c'est-à-dire celle dont la confiance est égale à 1). On ne prend pas en compte le support des règles.

Prenons un exemple :
Si $X = \{a_1, a_3, a_5, a_6\}$
et son fermé : $\overline{X} = \{a_1, a_3, a_5, a_6, a_7, a_8, a_{12}\}$
On obtiendra alors la règle : $\{a_1, a_3, a_5, a_6\} \rightarrow \{a_7, a_8, a_{12}\}$
On peut alors décomposer cette règle en trois règles :
$\{a_1, a_3, a_5, a_6\} \rightarrow \{a_7\}$, $\{a_1, a_3, a_5, a_6\} \rightarrow \{a_8\}$ et $\{a_1, a_3, a_5, a_6\} \rightarrow \{a_{12}\}$

La partie gauche de la règle est appelée *l'antécédent*, quant à la partie droite : *le conséquent*.

3.3 Principe de navigation

Le principe de navigation est détaillé dans [5].

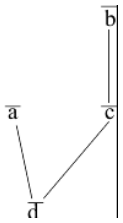
Dans cet article, les auteurs montrent que pour toute base de règles Σ d'une relation binaire, on peut se ramener à une base $\Sigma' = \Sigma_{trivial} \cup \Sigma_{\downarrow}$, où :
- $\Sigma_{trivial}$ correspond aux règles dont la partie gauche ne contient qu'un seul item
- $\Sigma_{\downarrow}$ correspond aux règles de la forme $I \rightarrow \overline{I} \setminus I$ avec I un idéal non fermé.

$\Sigma_{trivial}$ s'obtient facilement (en temps polynomial) à partir de la relation binaire. En effet ces règls correspondent aux arcs de couverture de l'ordre d'inclusion des fermés de chacun des items.

Prenons l'exemple suivant :

	a	b	c	d
1	1	0	1	1
2	0	1	1	1
3	0	0	1	1
4	1	1	1	1
5	1	0	0	1

$$\overline{a} = ad$$
$$\overline{b} = bcd$$
$$\overline{c} = cd$$
$$\overline{d} = d$$

Avec l'ordre d'inclusion : 

Ainsi on a : $\Sigma_{trivial} = \{b \rightarrow c, a \rightarrow d, c \rightarrow d\}$.

Concernant $\Sigma_{\downarrow}$, on sait qu'il s'agit de règles dont l'antécédent est un idéal non fermé. Le nombre de règles de $\Sigma_{\downarrow}$ peut potentiellement être exponentiel dans le nombre d'items. Plutôt que de chercher à générer la totalité des règles possibles et laisser l'utilisateur choisir parmi celles-ci celles qui l'intéressent, l'approche proposé dans [5] ets de ne générer qu'une partie des règles, à la demande de l'utilisateur. Cette navigation prend comme point de départ les règles de la forme $A \rightarrow x$ avec $|A|$ maximum. Puis l'utilisateur peut choisir de spécialiser une règle, c'est-à-dire de chercher à retirer des éléments de A de manière à ce que x soit toujours impliqué.

Comment fonctionne cette spécialisation ?
Tout d'abord, rappelons que dans $\Sigma_{\downarrow}$, si $I \rightarrow x$ alors I est un idéal non fermé tel que $x \in \overline{I}$.

Pour un x donné, le treillis des idéaux peut être décomposé en deux sous-treillis : les idéaux contenant x ($\mathcal{I}_x$) et ceux qui ne contiennent pas x ($\mathcal{I}_{\not x}$).

Clairement I ne peut pas appartenir à $\mathcal{I}_x$. En revanche, $\overline{I}$ doit appartenir à $\mathcal{I}_x$. Le bottom de $\mathcal{I}_x$ est l'idéal $\downarrow x$. Le sous-treillis $\mathcal{I}_{\not x}$ admet un élément maximum : $J \setminus \uparrow x$ (où J est l'ordre définit par $\Sigma_{trivial}$).

$J \setminus \uparrow x$ est un inf-irréductible du treillis des idéaux : il n'admet qu'un seul successeur et celui-ci appartient à $\mathcal{I}_x$.

Dans [5], les auteurs montrent que :

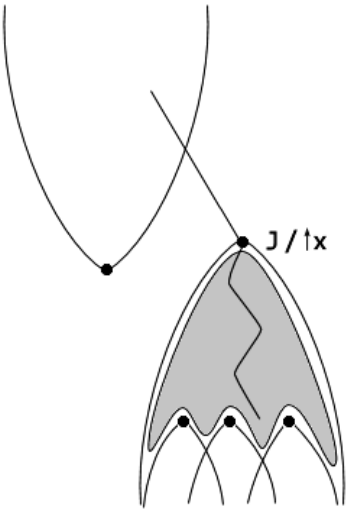
Il existe une règle $A \rightarrow x$ dans $\Sigma_{\downarrow}$ si et seulement si :
$J \setminus \uparrow x$ n'est pas fermé.

En effet, dans ce cas-là, $\overline{J \setminus \uparrow x}$ appartient obligatoirement à $\mathcal{I}_x$. Dans ce cas,
$J \setminus \uparrow x \rightarrow x$ sera la règle impliquant x avec le plus gros antécédent. On dira que cette règle est la plus générale pour x.

Prenons maintenant un idéal I non fermé appartenant à $\mathcal{I}_{\not x}$. Dans quel cas aura-t-on $x \in \overline{I}$? si $\overline{I} \in \mathcal{I}_x$. Et $\overline{I} \in \mathcal{I}_x$ si il n'existe pas un fermé $F \in \mathcal{I}_{\not x}$ tel que $I \subset F$.

Autrement dit, tout idéal I inclus dans un fermé maximal de $\mathcal{I}_{\not x}$ ne peut pas impliquer x. Notons que ces fermés maximaux situés sous $J \setminus \uparrow x$ sont des inf-irréductibles du treillis des fermés [5].

Prenons le treillis des idéaux suivant :



Le règle la plus générale est : $J\setminus \uparrow x$ à condition que $J\setminus \uparrow x \neq \overline{J\setminus \uparrow x}$ (c'est-à-dire que $J\setminus \uparrow x$ ne doit pas être un fermé)

Les règles qui nous intéresse se situent dans la partie grisée.

En dessous de cette partie, les trois points représentent les inf-maximaux. On sait alors que n'importe quelle règle se situant sous l'un de ces inf ne pourra pas impliquer x puisque leur fermé se situera dans cette partie du treillis plutôt que dans celle contenant x.

Le nombre de règles les plus générales est polynomial dans le nombre d'items. A partir d'une règle générale, on peut spécialiser en considérant tous les prédécesseurs immédiats de $J\setminus \uparrow x$ dans le treillis des idéaux tels qu'ils impliquent x. Et ainsi de suite jusqu'à ce qu'on ne puisse plus spécialiser (c'est-à-dire qu'on quitte la partie grisée). Si on spécialise une seule règles à chaque étape, le nombre de nouvelles règles reste polynomial.

L'idée est donc de laisser l'utilisateur choisir à chaque étape la règle qui l'intéresse pour spécialisation, et de ne lui proposer qu'un petit nombre de nouvelles règles à analyser.

En général l'utilisateur va chercher à obetnir des règles avec la plus petite partie gauche possible (à la limite basse de la partie grisée du dessin).

Dans [5], les auteurs montrent que de telles règles sont obligatoirement situées SOUS les successeurs immédiats des inf maximaux. Ils proposent donc un opérateur de saut permettant d'atteindre directement ces successeurs immédiats, et de proposer les règles correspondantes à l'utilisateur. Le nombre de règles obtenues reste polynomial. On pourrait atteindre ces éléments par plusieurs spécialisations consécutives, mais le saut présente l'avantage de les atteindre en un seul coup. Ces règles constituent alors le point de départ pour la suite de la navigation.

4 Les TDA nécessaires pour la navigation

Comme il l'avait été défini lors de l'étude de cas, huit TDA (Types de Données Abstraits) seront nécessaires pour cette partie de l'application :

- **Itemset** sur lequel on a défini les opérations concernant un ensemble d'entiers. Au niveau du projet il pourra gérer entre autres la notion d'ensemble d'items.

- **Table** permet de gérer les données relatives à une relation. On peut ainsi stocker et manipuler les données de la table de départ en ce qui concerne le projet global.

- **Element** est défini afin de manipuler un ensemble (pour le projet ce sera des items) et un numéro qui lui est associé.

- **SetOfElements** qui, comme son nom l'indique, est un ensemble d'objets du TDA Element. Il comporte donc toutes les opérations de suppression, d'ajout... qui sont susceptibles d'être utilisées sur un ensemble d'éléments.

- **Order** qui regroupe toutes les informations relatives à un ordre. En fait, à partir de la table de départ on peut calculer par exemple l'ordre des sup-irréductibles *(cf. partie Navigation)* et donc ce TDA permet de le stocker et de le manipuler.

- **Node** permet de représenter une règle, c'est-à-dire un nœud de l'arborescence.

- **SetOfNodes** manipule un ensemble de règles.

- **RuleTree** est défini afin de stocker et de manipuler l'arborescence des règles.

On suppose que les différentes opérations : $\in$, $\cup$, $\cap$, $\setminus$ sont définies (par exemple par l'utilisation de la STL puisque la plupart des traitements s'effectuent sur des ensembles).

4.1 Notation

La notation pour chacun des TDA sera la suivante :

Pour les attributs :
- **nom de l'attribut** descriptif

Pour les méthodes :
nom de la méthode :
en entrée :
en sortie :
descriptif :

Pour ce qui est des références : - à un fichier XML : `reference`
- à un TDA défini précédemment : *reference*

4.2 Itemset

Ce TDA permet de définir toutes les fonctions qui pourront être appliquées sur un ensemble d'entiers.

4.2.1 Attributs

- **intSet** un ensemble d'entiers

4.2.2 Méthodes

setIntSet :
entrée : l'ensemble avec lequel on initialise
sortie : /
descriptif : initialise l'attribut *intSet*

getIntSet :
entrée : /
sortie : un ensemble d'items
descriptif : renvoie l'ensemble *intSet*

add :
entrée : l'item à ajouter
sortie : /
descriptif : ajoute un item à l'ensemble *intSet*

remove :
entrée : l'item à retirer
sortie : /
descriptif : retire un item à l'ensemble *intSet*

RQ : Nous aurons aussi besoin d'un itérateur sur un objet de type *Itemset*.

4.3 Table

Table est le TDA qui va définir tout ce qui concerne le stockage, la manipulation... de la table de départ.

Trois attributs on été rajoutés par rapport à l'étude de cas : *xmlReader*, *currentTuple* et *fic* qui serviront pour la lecture de la table, puisque celle-ci doit pouvoir être lue en séquentiel.

4.3.1 Attributs

- **name** : nom de la table (fichier XML)
- **nbTuples** : le nombre de tuples dans la table
- **nbItems** : le nombre d'items
- **mapping** $\left\{ \begin{array}{l} item \rightarrow nom \\ nom \rightarrow item \end{array} \right\}$ = tableau des correspondances
- **xmlReader** : un pointeur permettant de garder la position du parser XML
- **currentTuple** : le numéro du tuple courant
- **fic** : un pointeur vers le fichier ouvert

4.3.2 Méthodes

setName :
entrée : le nom du fichier XML contenant la table
sortie : /
descriptif : initialise la valeur de l'attribut *name*

getName :
entrée : /
sortie : le nom du fichier XML contenant la table
descriptif : retourne la valeur courante de *name*

setNbTuples :
entrée : un entier correspondant au nombre de tuples
sortie : /
descriptif : initialise la valeur de l'attribut *nbTuples*

getNbTuples :
entrée : /
sortie : un entier, correspondant au nombre de tuples dans la table
descriptif : retourne la valeur courante de *nbTuples*

setNbItems :
entrée : un entier correspondant au nombre d'items
sortie : /
descriptif : initialise la valeur de l'attribut *nbItems*

getNbItems :
entrée : /
sortie : un entier, le nombre d'items
descriptif : retourne la valeur courante de *nbItems*

setMapping :
entrée : une table des correspondances
sortie : /
descriptif : initialise la valeur de l'attribut *mapping*

getMapping :
entrée : /
sortie : la table des correspondances (item,nom)
descriptif : retourne la valeur courante de *mapping*

setXmlReader :
entrée : un pointeur représentant le parser XML
sortie : /
descriptif : initialise la valeur de l'attribut *xmlReader*

getXmlReader :
entrée : /
sortie : le parser XML
descriptif : retourne la valeur courante de *xmlReader*

setCurrentTuple :
entrée : un entier, le numéro du tuple courant
sortie : /
descriptif : initialise la valeur de l'attribut *currentTuple*

getCurrentTuple :
entrée : /
sortie : le numéro du tuple courant
descriptif : retourne la valeur courante de *currentTuple*

rewind :
entrée : le nom du fichier XML_TABLE
sortie : `vrai` = ok ; `faux` = erreur
descriptif : ouvre le fichier XML_TABLE, puis utilise la fonction *readHeader* et positionne le curseur fichier sur le premier tuple

readHeader :
entrée : /
sortie : /
descriptif : lit les en-têtes c'est-à-dire le nom de la table `name`, le nombre de tuples dans la table `nbTuples` et le nombre d'items `nbItems` et parcourt les informations contenues entre `<listItems ...>` et `</listItems>` et retourne la liste (appelle la fonction

InitCorrespond). Puis se place sur le premier tuple, c'est-à-dire la première balise `<tuple ...>`.

initCorrespond :
entrée : /
sortie : le tableau de correspondances
descriptif : cette fonction parcourt le fichier XML entre les balise `<listitem>` et `</listitem>` et initialise le tableau des correspondances *mapping* en entrant le nom de l'item et en lui associant un numéro

readTuple :
entrée : /
sortie : un objet de type *Itemset*
descriptif : lit un tuple dans le fichier XML_TABLE. Parcourt toutes les informations contenues entre `<tuple ...>` et `</tuple>` et initialise les différents attributs de l'objet à retourner

close :
entrée : /
sortie : /
descriptif : ferme le fichier XML_TABLE en cours de traitement

writeHeader :
entrée : /
sortie : /
descriptif : initialise un nouveau fichier XML_TABLE de type table, c'est-à-dire écrit les en-têtes de ce fichier, ainsi que la liste des items de la table

writeTuple :
entrée : un objet de type *Itemset*
sortie : /
descriptif : écrit un tuple dans le fichier XML_TABLE les informations qui seront comprises entre les balises `<tuple ...>` et `</tuple>` avec la conversion item-nom

writeEnd :
entrée : /
sortie : /
descriptif : écrit la fin du fichier XML_TABLE, c'est-à-dire la balise `</Table>`, et ferme le fichier

getListItems :
entrée : /
sortie : un ensemble d'items
descriptif : retourne un *Itemset* contenant tous les items de la table

getItemByName :
entrée : le nom de l'item
sortie : le numéro de l'item
descriptif : récupère l'item par son nom, c'est-à-dire recherche le numéro associé à un nom d'item dans le tableau des correspondances

setItemName :
entrée : le numéro de l'item
sortie : le nom de l'item
descriptif : initialise le nom de l'item par rapport à son numéro. Fonction utilisée lorque l'on souhaite écrire un fichier XML_TABLE et que l'on ne dispose que des numéros des items. Il faut alors pouvoir initialiser les noms correspondants en utilisant la fonction *getNameOfItem*

getNameOfItem :
entrée : le numéro de l'item
sortie : le nom de l'item
descriptif : recherche le nom d'un item par son numéro associé à l'aide du tableau des correspondances

4.3.3 DTD associée : Table.dtd

```
<!------------------------------------------>
<!-- Definition de la structure de la table -->
<!------------------------------------------>

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- definition des elements -->
    <!-- une table est composee d'une liste d'items
    et de tuples -->
    <!ELEMENT Table (ListItems+, Tuple+) >
    <!-- la liste d'items contient des items -->
    <!ELEMENT ListItems (Item+) >
    <!-- un tuple contient des items -->
    <!ELEMENT Tuple (Item*) >
    <!ELEMENT Item (#PCDATA) >

<!-- definition des attributs -->
    <!-- attributs de table -->
    <!ATTLIST Table name CDATA #REQUIRED >
    <!ATTLIST Table nbTuples CDATA #REQUIRED >
    <!ATTLIST Table nbItems CDATA #REQUIRED >
```

4.3.4 Exemple de fichier XML_TABLE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ruletree SYSTEM "table.dtd">

<Table name="essaiBD" nbTuples="6" nbItems="5">
    <ListItems>
        <Item>1</Item>
        <Item>2</Item>
        <Item>3</Item>
        <Item>4</Item>
        <Item>5</Item>
    </ListItems>
    <Tuple>
        <Item>1</Item>
        <Item>3</Item>
        <Item>4</Item>
    </Tuple>
    <Tuple>
        <Item>1</Item>
        <Item>2</Item>
        <Item>3</Item>
        <Item>5</Item>
    </Tuple>
    <Tuple>
        <Item>2</Item>
        <Item>3</Item>
        <Item>5</Item>
    </Tuple>
    <Tuple>
        <Item>2</Item>
        <Item>5</Item>
    </Tuple>
    <Tuple>
        <Item>1</Item>
        <Item>2</Item>
        <Item>3</Item>
        <Item>5</Item>
    </Tuple>
    <Tuple>
        <Item>2</Item>
        <Item>3</Item>
        <Item>5</Item>
    </Tuple>
</Table>
```

4.4 Element

Il s'agit d'un élément de l'ordre. On stockera le numéro et l'ensemble associé à cet élément.

4.4.1 Attributs

Un element est :

- **number** : un numéro (l'item s'il s'agit d'un sup-irréductible)
- **elementSet** : un ensemble associé à l'élément (le fermé de l'item quand il s'agit d'un sup-irréductible)

4.4.2 Méthodes

setNumber :
entrée : le numéro de l'élément
sortie : /
descriptif : initialise la valeur de l'attribut *number*

getNumber :
entrée : /
sortie : le numéro de l'élément
descriptif : retourne la valeur courante de *number*

setElementSet :
entrée : un ensemble d'entiers
sortie : /
descriptif : initialise la valeur de l'attribut *elementSet*

getElementSet :
entrée : /
sortie : l'ensemble associé à l'élément
descriptif : retourne la valeur courante de *elementSet*

4.5 SetOfElements

Il s'agit d'un ensemble d'éléments (au sens du TDA *Element*).

On ne sauvegardera plus l'ordre dans un fichier XML mais la collection des éléments de cet ordre. Pour cela on doit disposer dans ce TDA de deux nouveaux attributs : *name*, le nom du fichier XML et *table*, le nom du fichier contenant la table correspondante. Ainsi que des méthodes de sauvegarde et de chargement de la collection.

4.5.1 Attributs

- **collection** : un ensemble d'éléments
- **nbElements** : le nombre d'éléments dans l'ensemble
- **name** : le nom du fichier XML contenant la collection
- **table** : le nom du fichier XML contenant la table correspondante

4.5.2 Méthodes

setCollection :
entrée : un ensemble d'objets de type *Element*
sortie : /
descriptif : initialise la valeur de l'attribut *collection*

getCollection :
entrée : /
sortie : l'ensemble d'*Element*
descriptif : retourne la valeur courante de *collection*

setNbElements :
entrée : un entier, le nombre d'éléments de la collection
sortie : /
descriptif : initialise la valeur de l'attribut *nbElements*

getNbElements :
entrée : /
sortie : le nombre d'éléments de la collection
descriptif : retourne la valeur courante de *nbElements*

setName :
entrée : le nom du fichier XML contenant la collection
sortie : /
descriptif : initialise la valeur de l'attribut *name*

getName :
entrée : /
sortie : le nom du fichier XML contenant la collection
descriptif : retourne la valeur courante de *name*

setTable :
entrée : le nom du fichier XML contenant la table correspondante à la collection
sortie : /
descriptif : initialise la valeur de l'attribut *table*

getTable :
entrée : /
sortie : le nom du fichier XML contenant la table correspondante
descriptif : retourne la valeur courante de *table*

load :
entrée : /
sortie : /
descriptif : charge la collection à partir du fichier XML_COLLECTION pointé par *name*

save :
entrée : /
sortie : /
descriptif : sauvegarde la collection dans le fichier XML_COLLECTION pointé par *name*

add :
entrée : un objet de type *Element*, celui à insérer
sortie : /
descriptif : ajoute un élément à la collection

remove :
entrée : un objet de type *Element*, celui à supprimer
sortie : /
descriptif : supprime un élément de la collection

getNumberBySet :
entrée : l'ensemble associé à l'élément que l'on souhaite récupérer
sortie : la valeur de l'attribut *number* correspondant
descriptif : retourne le numéro d'un élément par son ensemble associé

getSetByNumber :
entrée : le numéro de l'élément que l'on souhaite récupérer
sortie : l'ensemble *set* associé à l'élément correspondant
descriptif : retourne l'ensemble associé à un élément par son numéro

getNumbers :
entrée : /
sortie : un ensemble d'entiers de type *Itemset*
descriptif : renvoie un ensemble d'entiers correspondant à la liste des *number* des *Element*

getSize :
entrée : /
sortie : un entier représentant le nombre d'éléments de la collection
descriptif : retourne le nombre d'éléments de l'ensemble

closureAll :
entrée : le nom du fichier XML_TABLE correspondant
sortie : /
descriptif : calcule la fermeture de tous les *elementSet* des éléments. *(cf. algo)*

4.5.3 Algorithmes

Calcul de la fermeture de tous les itemset des éléments

Algorithme 1: closureAll

```
Données : Le nom de la table tableName
Résultat :
début
    new Table(tableName)
    Table.rewind()
    pour chaque element faire
        // itérateur du TDA setOfElements
        element.setSet(Table.listItems)
    pour chaque tuple T de Table faire
        // itérateur du TDA table
        pour chaque element faire
            // iterateur du TDA setOfElements
            si element.getNumber() ∈ T alors
                element.setSet(element.getSet() ∩ T)
fin
```

4.5.4 DTD associée : collection.dtd

Afin de stocker les propriétés d'un ensemble d'éléments on crée un fichier XML qui sera XML_COLLECTION. Il s'agit d'un nouveau fichier XML puisque l'ordre n'est plus sauvegarder entièrement mais juste sa collection d'élément qui le compose. Ces fichiers devront suivre la DTD suivante :

```
1   <!------------------------------------------------>
    <!-- Definition de la structure de la collection -->
    <!------------------------------------------------>

    <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!-- definition des elements -->
    <!-- la collection est composee d'elements -->
    <!ELEMENT Collection (Element+)>
    <!-- les elements sont composes d'un numero et
    d'un ensemble -->
    <!ELEMENT Element (Number+,Set+)>
    <!ELEMENT Number (#PCDATA)>
    <!-- un ensemble est compose d'items -->
    <!ELEMENT Set (Item+)>
    <!ELEMENT Item (#PCDATA)>

<!-- definition des attributs -->
    <!ATTLIST Collection name CDATA #REQUIRED>
    <!ATTLIST Collection nbElements CDATA #REQUIRED>
    <!ATTLIST Collection tableName CDATA #REQUIRED>
```

4.5.5 Exemple de fichier XML_COLLECTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ruletree SYSTEM "collection.dtd">

<Collection name="collectionGolf.xml" nbElements="5"
tableName="golf.xml">
    <Element>
        <Number>1</Number>
        <Set>
            <Item>T.basse</Item>
            <Item>T.haute</Item>
            <Item>T.moyenne</Item>
        </Set>
    </Element>
    <Element>
        <Number>2</Number>
        <Set>
            <Item>Temps.rain</Item>
            <Item>Temps.overcast</Item>
        </Set>
    </Element>
</Collection>
```

4.6 Order

Ce TDA pourra être utilisé aussi bien en ce qui concerne l'ordre des sup-irréductibles que celui des inf-irréductibles.

A noter : le fichier XML_ORDER disparait et est remplacé par XML_COLLECTION.

4.6.1 Attributs

Un ordre est :

- **J** : un ensemble d'éléments
- **imPred** : un vecteur indicé par des entiers (*number* d'un élément) correspondant à des éléments prédécesseurs de l'élément indiçant l'entrée
- **imSucc** : un vecteur indicé par des entiers (*number* d'un élément) correspondant à des éléments successeurs de l'élément indiçant l'entrée
- **name** : le nom du fichier XML contenant l'ordre
- **table** : le nom du fichier XML contenant la table correspondante

4.6.2 Méthodes

setJ :
entrée : un ensemble d'éléments, *SetOfElements*
sortie : /
descriptif : initialise la valeur de l'attribut *J*

getJ :
entrée : /
sortie : l'ensemble d'éléments de l'ordre
descriptif : retourne la valeur courante de *J*

setImPred :
entrée : un *Element* `elt` et l'ensemble de ses prédecesseurs de type *SetOfElements*
sortie : /
descriptif : initialise la valeur de *imPred* pour `elt`

getImPred :
entrée : un *Element* `elt`
sortie : l'ensemble de ses prédecesseurs immédiats
descriptif : récupère l'ensemble des prédecesseurs immédiats de `elt`

setImSucc :
entrée : un *Element* `elt` et l'ensemble de ses successeurs de type *SetOfElements*
sortie : /
descriptif : initialise la valeur de *imSucc* pour `elt`

getImSucc :
entrée : un *Element* `elt`
sortie : l'ensemble de ses successeurs immédiats
descriptif : récupère l'ensemble des successeurs immédiats de `elt`

setName :
entrée : le nom du fichier XML contenant la collection des éléments de l'ordre
sortie : /
descriptif : initialise la valeur de l'attribut *name*

getName :
entrée : /
sortie : le nom du fichier XML contenant la collection de l'ordre
descriptif : retourne la valeur courante de *name*

setTable :
entrée : le nom du fichier XML contenant la table correspondante à l'ordre
sortie : /
descriptif : initialise la valeur de l'attribut *table*

getTable :
entrée : /
sortie : le nom du fichier XML contenant la table correspondante
descriptif : retourne la valeur courante de *table*

load :
entrée : /
sortie : /
descriptif : charge les éléments de l'ordre à partir du fichier XML_COLLECTION

save :
entrée : /
sortie : /
descriptif : sauvegarde les éléments de l'ordre dans un fichier XML_COLLECTION

getJnumbers :
entrée : /
sortie : un ensemble d'entiers de type *Itemset*
descriptif : retourne J sous la forme d'un ensemble d'entiers

insert :
entrée : l'*Element* à insérer
sortie : /
descriptif : insère un élément dans l'ordre. *(cf. algo)*

remove :
entrée : l'*Element* à supprimer
sortie : /
descriptif : supprime un élément dans l'ordre. *(cf. algo)*

max :
entrée : /
sortie : les maximaux de l'ordre sous forme de *SetOfElements*
descriptif : Retourne les éléments maximaux de l'ordre (c'est-à-dire les sommets qui n'ont pas de successeur).On va regarder pour chaque élément de l'ordre s'il a un successeur ou pas. S'il n'a pas de successeur alors on ajoute cet élément à l'ensemble des maximaux

min :
entrée : /
sortie : les minimaux de l'ordre sous forme de *SetOfElements*
descriptif : Retourne les éléments minimaux de l'ordre (c'est-à-dire les sommets qui n'ont pas de prédécesseur).On va regarder pour chaque élément de l'ordre s'il a un prédecesseur ou pas. S'il n'a pas de prédecesseur alors on ajoute cet élément à l'ensemble des minimaux

duplicate :
entrée : /
sortie : ordre2, similaire à l'ordre
descriptif : on parcourt l'ordre et on recopie toutes les informations dans ordre2

getIdeal :
entrée : un ensemble d'éléments (*SetOfElements*)
sortie : l'ordre sur I
descriptif : renvoie l'ordre qui correspond à l'idéal de l'ensemble donné en entrée. *(cf. algo)*

removeFilter :
entrée : un élément x
sortie : J \ ↑x
descriptif : enlève ↑x de l'ordre. *(cf. algo)*

imSuccIdeal :
entrée : un ensemble d'éléments (*SetOfElements*)
sortie : les successeurs immédiats de cet ensemble (un *SetOfElements*)
descriptif : renvoie les successeurs immédiats d'un idéal donné dans le treillis des idéaux (ImSucc = { I ∪ x tq x∈Min(J\I) }). *(cf. algo)*

imPredIdeal :
entrée : un ensemble d'éléments (*SetOfElements*)
sortie : les prédecesseurs immédiats de cet ensemble (un *SetOfElements*)
descriptif : renvoie les prédecesseurs immédiats d'un idéal donné dans le treillis des

idéaux (ImPred = { {x} tq x∈Max(I) }). *(cf. algo)*

4.6.3 Algorithmes

Ajout d'un élément à l'ordre

Cet algorithme est basé sur la réduction transitive.

Algorithme 2: insert

```
Données : Un element N
Résultat : l'ordre avec N ajouté
début
    pour chaque element M ∈ J.collection faire
        // itérateur du TDA setOfElements
        arc = 0
        si N.getSet() ⊆ M.getSet() alors
            // M imSucc de N ?
            pour M' ∈ imPred[M.getNumber()] faire
                // itérateur du TDA setOfElements
                si N.getSet() ⊆ M'.getSet() alors
                    arc = 1
            si arc ≠ 1 alors
                // il n'existe pas de M' tel que N ∈ M' ∈ M
                imSucc[N.getNumber()].add(M)
                pour M' ∈ imPred[M.getNumber()] faire
                    // suppression des arcs de transitivité
                    si M'.getSet() ⊆ N.getSet() alors
                        imPred[M.getNumber()].remove(M')
                imPred[M.getNumber()].add(N) // N imPred de M
        si M.getSet() ⊆ N.getSet() alors
            // M imPred de N ?
            pour M' ∈ imSucc[M.getNumber()] faire
                si M'.getSet() ⊆ N.getSet() alors
                    arc = 1
            si arc ≠ 1 alors
                // M est imPred de N
                imPred[N.getNumber()].add(M)
                pour M' ∈ imSucc[M.getNumber()] faire
                    // suppression des arcs de transitivité
                    si N.getSet() ⊆ M'.getSet() alors
                        imSucc[M.getNumber()].remove(M')
                imSucc[M.getNumber()].add(N) // N imSucc de M
fin
J.add(N) // on peut donc ajouter le nœud
```

Suppression d'un élément de l'ordre

Cet algorithme est basé sur la réduction transitive.

Algorithme 3: remove

```
Données : Un element N
Résultat : L'ordre avec N supprimé
début
    pour chaque K ∈ imSucc[N.getNumber()] faire
        imPred[K.getNumber()].remove(N)
    pour chaque K ∈ imPred[N.getNumber()] faire
        imSucc[K.getNumber()].remove(N)
    pour chaque K ∈ imSucc[N.getNumber()] faire
        pour chaque I ∈ imPred[N.getNumber()] faire
            si I.getSet() ⊂ K.getSet() alors
                // I imPred potentiel de K
                arc = 0
                pour chaque L ∈ imPred[K.getNumber()] faire
                    si I.getSet() ⊂ L.getSet() alors
                        // arc de transitivité existant
                        arc = 1
                si arc = 0 alors
                    // pas d'arc de transitivité
                    imPred[K.getNumber()].add(I)
                    imSucc[I.getNumber()].add(K)
    J.collection.remove(N)
fin
```

Calcul des éléments maximaux de l'ordre

Idée : x est un puit de l'ordre ssi : $ImSucc(x) = \emptyset$

Algorithme 4: max

```
Données :
Résultat : Les éléments maximaux de l'ordre max
début
    max = ∅
    pour chaque element de J.collection faire
        si imSucc[element.getNumber()].getSize() = 0 alors
            max.add(element)
    retourner max
fin
```

Calcul des éléments minimaux de l'ordre

Idée : x est une source de l'ordre ssi : $ImPred(x) = \emptyset$

Algorithme 5: min

```
Données :
Résultat : Les éléments minimaux de l'ordre min
début
    min = ∅
    pour chaque element de J.collection faire
        si imPred[element.getNumber()].getSize() = 0 alors
            min.add(element)
    retourner min
fin
```

Calcul de l'ordre correspondant à un idéal

Algorithme 6: getIdeal

Données : Un ensemble d'elements I
Résultat : L'ordre correspondant à cet ensemble I
début
  Traite = $\emptyset$
  tant que $Traite \neq I$ **faire**
    Choisir $a \in I \setminus Traite$
    Traite = Traite ∪ a
    I = I ∪ ImPred(a)
  02 = duplicate()
  aretirer = J\ I
  pour $i \in aretirer$ **faire**
    O2.remove(i)
  retourner O2
fin

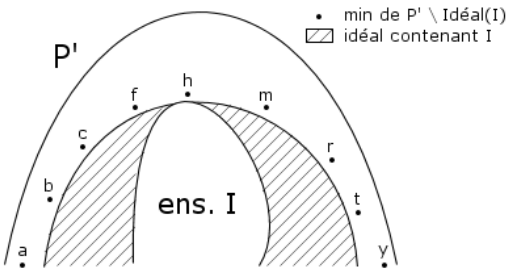
Calcul de J \ ↑x

Algorithme 7: removeFilter

Données : Un élément x
Résultat : L'ensemble J privé de ↑x
début
  I = x
  Traite = $\emptyset$
  tant que $Traite \neq I$ **faire**
    Choisir $a \in I \setminus Traite$
    Traite = Traite ∪ a
    I = I ∪ ImSucc(a)
  02 = duplicate()
  pour $i \in I$ **faire**
    O2.remove(i)
fin

Calcul des successeurs immédiats d'un idéal

Idée : $I' \in ImSucc(I)$ dans le treillis des idéaux ssi $I' = I \cup \{x\}$ avec $x \in Source(P \setminus I)$



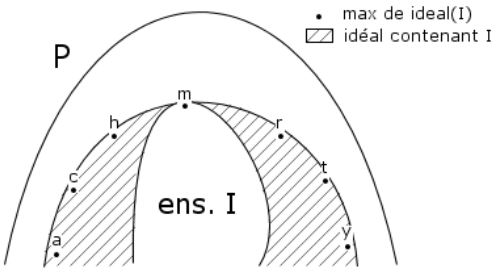
La partie hachurée représente l'idéal contenant I, appelons-le Id. Chaque imSucc de cet idéal s'obtient en ajoutant un élément successeur, c'est-à-dire les éléments minimaux de l'ordre $P' \setminus Id$. Dans l'exemple ci-dessus, l'ensemble des imSucc est : Id $\cup$ {a}, Id $\cup$ {b}, ..., Id $\cup$ {y}.

Algorithme 8: imSuccIdeal

```
Données : Un ensemble d'elements I
Résultat : Les successeurs immédiats de I ImSucc
début
    ImSucc = ∅
    Id = getIdeal(I) // on veut un idéal
    O2 = duplicate()
    pour i ∈ Id faire
        O2.remove(i)
    pour i ∈ O2.min() faire
        num = i.getNumber()
        ens = Id.getJNumbers()
        ens.add(num)
        Insérer l'élément (num,ens) dans ImSucc
    retourner ImSucc
fin
```

Calcul des prédecesseurs immédiats d'un idéal

Idée : $I' \in ImPred(I)$ dans le treillis des idéaux ssi $I' = I \setminus \{x\}$ avec $x \in Puit(I)$



La partie hachurée représente l'idéal contenant I, appelons-le Id. Chaque imPred de cet idéal s'obtient en retirant un élément maximal de cet idéal. Dans l'exemple ci-dessus, l'ensemble des imPred est : Id \ {a}, Id \ {b}, ..., Id \ {y}.

Algorithme 9: imPredIdeal

Données : Un ensemble d'elements I
Résultat : Les prédecesseurs immédiats de I ImPred
début
  ImPred = ∅
  Id = getIdeal(I) *// on veut un idéal*
  pour $i \in Id.max()$ **faire**
    num = i.getNumber()
    ens = Id.getJNumbers()
    ens.add(num)
    Insérer l'élément (num,ens) dans ImPred
  retourner ImPred
fin

4.7 Node

Ce TDA va permettre de représenter un nœud de l'arborescence.

4.7.1 Attributs

∘ **number** : un numéro
∘ **itemset** : représente les numéros (associés aux différents attributs) composant la partie gauche de la règle
∘ **closure** : représente le fermé de itemset
∘ **support** : correspond au nombre de fois que itemset est présent dans la table
∘ **confidence** : représente un pourcentage (confiance de la règle)
∘ **specializable** : spécifie si la règle est spécialisable ou non
∘ **processed** : booléen indiquant si le nœud a été traité (seuls les nœuds traités seront affichés)

4.7.2 Méthodes

setNumber :
entrée : le numéro du nœud
sortie : /
descriptif : initialise la valeur de l'attribut *number*

getNumber :
entrée : /
sortie : le numéro du nœud
descriptif : retourne la valeur courante de *number*

setItemset :
entrée : un objet de type *Itemset*
sortie : /
descriptif : initialise la valeur de l'attribut *itemset* (la partie gauche du nœud)

getItemset :
entrée : /
sortie : un objet de type *Itemset*
descriptif : retourne la valeur courante de *itemset*

setClosure :
entrée : un objet de type *Itemset*
sortie : /
descriptif : initialise la valeur de l'attribut *closure* (le fermé du nœud)

getClosure :
entrée : /
sortie : un objet de type *Itemset*
descriptif : retourne la valeur courante de *closure*

setSupport :
entrée : un entier, le support de la règle
sortie : /
descriptif : initialise la valeur de l'attribut *support*

getSupport :
entrée : /
sortie : un entier, le support de la règle
descriptif : retourne la valeur courante de *support*

setConfidence :
entrée : un réel, la confiance de la règle
sortie : /
descriptif : initialise la valeur de l'attribut *confidence*

getConfidence :
entrée : /
sortie : un réel, la confiance de la règle
descriptif : retourne la valeur courante de *confidence*

setSpecializable :
entrée : un booléen, définissant si la règle est spécialisable ou non
sortie : /
descriptif : initialise la valeur de l'attribut *specializable*

getSpecializable :
entrée : /
sortie : un booléen, définissant si la règle est spécialisable ou non
descriptif : retourne la valeur courante de *specializable*

setProcessed :
entrée : un booléen, spécifiant si le nœud a été traité ou non
sortie : /
descriptif : initialise la valeur de l'attribut *processed*

getProcessed :
entrée : /
sortie : un booléen, spécifiant si le nœud a été traité ou non
descriptif : retourne la valeur courante de *processed*

4.8 SetOfNodes

4.8.1 Attributs

- **collection** : un ensemble d' objets de type *Node*

4.8.2 Méthodes

setCollection :
entrée : un ensemble de nœuds de type *Node*
sortie : /
descriptif : initialise la valeur de l'attribut *collection*

getCollection :
entrée : /
sortie : un ensemble de nœuds de type *Node*
descriptif : retourne la valeur courante de *collection*

add :
en entrée : un objet de type *Node*
en sortie : /
descriptif : ajoute un noeud à la collection

remove :
en entrée : un objet de type *Node*
en sortie : /
descriptif : retire un noeud de la famille

getNodeByItemset :
en entrée : un ensemble de type *Itemset*
en sortie : un objet de type *Node*
descriptif : récupère un nœud associé à un ensemble, c'est-à-dire recherche dans la famille un nœud ayant `set` pour *itemset*, puis retourne ce nœud avec les informations correspondantes

getNodeByNumber :
entrée : le numéro du nœud
sortie : le nœud
descriptif : récupère les caractéristiques associées à un nœud par le numéro de ce nœud

getItemsetByNumber :
entrée : le numéro du nœud
sortie : un objet de type *Itemset*
descriptif : retourne l'*itemset* associé à un nœud par l'intermédiaire de son numéro

updateClosureAll :
en entrée : un objet de type *itemset*
en sortie : /
descriptif : met à jour l'attribut *closure* en faisant l'intersection avec *itemset* pour

tous les nœuds. *(cf. algo)*

Nous aurons aussi besoin d'un itérateur pour parcourir la collection.

4.8.3 Algorithmes

Mise à jour de la fermeture

Algorithme 10: updateClosureAll

Données : Un itemset T
Résultat : Le fermé mis à jour
début
  pour chaque $N \in Collection$ **faire**
    // itérateur setOfNodes
    si $N.getItemset() \subseteq T$ **alors**
      N.setClosure(N.getClosure()∩T)
      N.setSupport(N.getSupport()+1)
fin

4.9 RuleTree

Ce TDA permet de gérer le stockage et les opérations au niveau de l'arborescence.

4.9.1 Attributs

- **listNodesItem** : un vecteur indicé par des entiers (*Node Number*) indiquant l'item impliqué par le nœud (partie droite de la règle)
- **listNodes** : un ensemble de nœuds
- **parent** : un vecteur indicé par des entiers correspondant à des numéros de nœuds
- **children** : un vecteur indicé par des entiers correspondant à des numéros de nœuds
- **name** : le nom du fichier XML contenant l'arborescence
- **table** : le nom de la table XML associée
- **orderSup** : ordre des sup-irréductibles
- **orderInf** : ordre des inf-irréductibles
- **lastNumber** : dernier numéro attribué à un nœud

4.9.2 Méthodes

setListNodesItem :
entrée : la liste des nœuds et l'item qu'ils impliquent
sortie : /
descriptif : initialise la valeur de l'attribut *listNodesItem*

getListNodesItem :
entrée : /
sortie : la liste des nœuds et l'item qu'ils impliquent
descriptif : retourne la valeur courante de *listNodesItem*

setListNodes :
entrée : la liste des nœuds de l'arborescence
sortie : /
descriptif : initialise la valeur de l'attribut *listNodes*

getListNodes :
entrée : /
sortie : la liste des nœuds de l'arborescence
descriptif : retourne la valeur courante de *listNodes*

setParent :
entrée : un *Node* `nd` et le père de ce nœud (*Node*)
sortie : /
descriptif : initialise la valeur de l'attribut *parent* pour `nd`

getParent :
entrée : un *Node* `nd`
sortie : le père de ce nœud
descriptif : retourne la valeur courante de *parent* pour `nd`

setChildren :
entrée : un *Node* `nd` et l'ensemble de ses fils (*Itemset*)
sortie : /
descriptif : initialise la valeur de l'attribut *children* pour `nd`

getChildren :
entrée : un *Node* `nd`
sortie : la liste des fils de ce nœud
descriptif : retourne la valeur courante de *children* pour `nd`

setName :
entrée : le nom du fichier XML contenant l'arborescence
sortie : /
descriptif : initialise la valeur de l'attribut *name*

getName :
entrée : /
sortie : le nom du fichier *xml* contenant l'arborescence
descriptif : retourne la valeur courante de *name*

setTable :
entrée : le nom du fichier XML contenant la table correspondante
sortie : /
descriptif : initialise la valeur de l'attribut *table*

getTable :
entrée : /
sortie : le nom du fichier XML contenant la table correspondante
descriptif : retourne la valeur courante de *table*

setOrderSup :
entrée : l'ordre des sup-irréductibles correspondant à cette arborescence
sortie : /
descriptif : initialise la valeur de l'attribut *orderSup*

getOrderSup :
entrée : /
sortie : l'ordre des sup-irréductibles correspondant à cette arborescence
descriptif : retourne la valeur courante de *orderSup*

setOrderInf :
entrée : l'ordre des inf-irréductibles correspondant à cette arborescence
sortie : /
descriptif : initialise la valeur de l'attribut *orderInf*

getOrderInf :
entrée : /
sortie : l'ordre des inf-irréductibles correspondant à cette arborescence
descriptif : retourne la valeur courante de *orderInf*

setLastNumber :
entrée : un entier, le prochain numéro à attribuer
sortie : /
descriptif : initialise la valeur de l'attribut *lastNumber*

getLastNumber :
entrée : /
sortie : un entier, le prochain numéro à attribuer
descriptif : retourne la valeur courante de *lastNumber*

getItemByNode :
entrée : un *Node* de l'arborescence
sortie : un entier représentant l'item impliqué par ce nœud
descriptif : retourne l'item impliqué par ce nœud à l'aide de l'attribut *listNodesItem*

getNodeByNumber :
entrée : un entier, correspondant à un numéro de nœud
sortie : le *Node*
descriptif : retourne le nœud par son numéro (lu dans *lisNodes)*

load :
entrée : le nom du fichier XML
sortie : `vrai` = ok ; `faux` = erreur
descriptif : ouvre le fichier XML_A, le lit et charge l'arborescence en mémoire, ainsi que l'ordre des sup-irréductibles et l'ordre des inf-irréductibles

saveRule :
entrée : le fichier (dans lequel on sauvegarde) ouvert, un entier correspondant à un numéro de nœud, et la table correspondant à cette arborescence
sortie : /
descriptif : sauvegarde une règle dans le fichier XML, c'est-à-dire un nœud (avec toutes ses caractéristiques)

save :
entrée : le nom du fichier XML
sortie : /
descriptif : sauvegarde la structure. Si le fichier n'existe pas on le remplit directement,

sinon on efface avant (le remplissage dépend aussi de la structure du fichier)

addChild :
entrée : le nœud auquel on doit ajouter un fils, et ce fils (*Node*)
sortie : /
descriptif : ajoute un fils à un nœud de l'arborescence

insert :
entrée : un ensemble d'entiers et un nœud représentant le père
sortie : /
descriptif : insère un nœud dans l'arborescence. *(cf. algo)*

remove :
entrée : le nœud à supprimer
sortie : /
descriptif : supprime un nœud (c'est-à-dire ce nœud et l'ensemble de tous ses fils). *(cf. algo)*

specialize :
entrée : le nœud à spécialiser
sortie : /
descriptif : spécialise un nœud de l'arborescence. *(cf. algo)*

jump :
entrée : un nœud
sortie : `vrai` = le saut est réalisable, `faux` = sinon
descriptif : effectue un saut dans l'arborescence des règles d'association. *(cf. algo)*

4.9.3 Algorithmes

Insertion d'un nœud de l'arborescence

Algorithme 11: insert

Données : Un ensemble d'entiers E et un nœud pere
Résultat : Un entier correspondant au numéro attribué
début
    // attribution du numéro pour ce nœud
    this.getLastNumber() = this.getLastNumber() + 1
    new node n(this.getLastNumber(),E)
    this.setParent(n,pere)
    si *pere* $\neq$ *null* **alors**
        this.addChild(pere,n)
    children[n.getNumber()] = $\emptyset$
    listNodes.add(n)
    si *pere* $\neq$ *null* **alors**
        listNodesItem[n.getNumber()] = listNodesItem[pere.getNumber()]
    retourner n.getNumber()
fin

Suppression d'un nœud de l'arborescence

Algorithme 12: remove

Données : Un nœud N
Résultat :
début
    pour chaque $M \in N.getChildren()$ **faire**
        // itérateur setOfNodes
        this.remove(M) *// on supprime d'abord les fils récursivement*
    children[N.getParent().getNumer()].remove(N)
    children[N.getNumber()] = $\emptyset$ *// libération de la mémoire*
    parent[N.getNumber()] = $\emptyset$ *// libération de la mémoire*
    listNodes.remove(N) *// libération de la mémoire*
    listNodesItem[N.getNumber()] = 0 *// libération de la mémoire*
fin

Spécialisation d'un nœud de l'arborescence

Algorithme 13: specialize

```
Données : Un nœud N
Résultat :
début
    // on suppose que les fils de N sont déjà présents dans l'arborescence
    // on vérifie simplement que ceux-ci sont déjà spécialisables ou pas et on
    // les marque comme traités (donc affichables)
    si N.specializable() alors
        // Calcul des ImPred et insertion dans l'arbre
        pour chaque F ∈ children[N.getNumber()] faire
            ensItem = listNodes.getItemsetByNumber(F)
            ensElt = itemsetToSetOfElements(ensItem)
            nd = listNodes.getNodeByNumber(F)
            L = orderSup.getImpPredIdeal(ensElt)
            pour chaque l ∈ L faire
                insert(l,nd)
        // Calcul des fermés
        pour chaque tuple T ∈ Table faire
            listNodes.updateClosureAll(T)
        pour chaque F ∈ children[N.getNumber()] faire
            nd = listNodes.getNodeByNumber(F)
            nd.setProcessed(true)
            item = getItemByNode(nd)
            pour chaque K ∈ children[F] faire
                ndBis = listNodes.getNodeByNumber(K)
                rightSide = ndBis.getClosure() \ ndBis.getItemset()
                si item ∈ rightSide alors
                    nd.setSpecializable = true
                sinon
                    remove(ndBis)
                // On retire les règles invalides
                si RightSide = ∅ alors
                    remove(ndBis)
fin
```

Saut dans l'arborescence

Algorithme 14: jump

```
Données : Un nœud N
Résultat : Un booléen qui est true si le saut est possible, false sinon
début
    si (getParent(N) = ∅) et (N.specializable()) alors
        // le saut est possible
        // on commence par supprimer les fils existants
        pour K ∈ children[N.getNumber()] faire
            remove(K)
        // calcul des infs maximaux
        O2 = orderInf.duplicate()
        pour a ∈ O2.collection faire
            si a.getSet() ∉ N.getItemSet() alors
                O2.remove(a)
        maxIdeaux = ∅
        pour K ∈ O2.max() faire
            // parcours des maximaux
            maxIdeaux = maxIdeaux ∪ oredreSup.getImSuccIdeal(K.getSet())
        // insertion des maxIdeaux et de leurs predecesseurs dans arbo
        pour K ∈ maxIdeaux faire
            insert(K.getElementSet(),N)
            pour I ∈ orderSup.getImPredIdeal(K.getElementSet()) faire
                insert(I.getElementSet(),K)
        // calcul des fermes
        pour chaque tuple T ∈ Table faire
            listNodes.updateClosureAll(T)
        item = listNodesItem[N.getNumber()]
        pour K ∈ children[N.getNumber()] faire
            pour I ∈ children[K.getNumber()] faire
                si item ∈ I.getClosure() \ I.getItemset() alors
                    K.setSpecializable(true)
                sinon
                    remove(K)
                si I.getClosure() \ I.getItemset() = ∅ alors
                    remove(I)
            K.setProcessed(true)
        si item ∈ F.getClosure() \ F.getItemset() alors
            remove(F)
        retourner true
    sinon
        retourner false
fin
```

4.9.4 DTD associée : ruletree.dtd

Dans la définition de cette DTD :
- `LeftSide` : contient *itemset* de *node*
- `RightSide` : contient *closure* \ *itemset* de *node*
- `Item` : item impliqué par le nœud racine de cette arborescence (nœud racine = règle la plus générale)

```
<!-------------------------------------------------->
<!-- Definition de la structure d'un arbre de regles -->
<!-------------------------------------------------->

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- definition des elements -->
    <!-- un arbre de regles est compose d'au moins 1 regle -->
    <!ELEMENT RuleTree (Rule+) >
    <!-- une regle contient au moins 1 partie gauche
    et eventuellement d'autres regles -->
    <!ELEMENT Rule (LeftSide+, Item+, RightSide+, Rule*) >
    <!-- contient itemset de Node -->
    <!ELEMENT LeftSide (Item+) >
    <!-- contient Closure\Itemset de Node -->
    <!ELEMENT RightSide (Item+) >
    <!-- item implique par le noeud racine de cette arborescence
    (noeud racine = regle la plus generale) -->
    <!ELEMENT PrincipalItem (#PCDATA) >
    <!-- item composant un itemset (de LeftSide et RightSide) -->
    <!ELEMENT Item (#PCDATA) >

<!-- definition des attributs -->
    <!-- attributs de ruletree -->
    <!ATTLIST RuleTree table CDATA #REQUIRED>
    <!ATTLIST RuleTree supOrder CDATA #REQUIRED>
    <!ATTLIST RuleTree infOrder CDATA #REQUIRED>

    <!-- attributs de rule -->
    <!ATTLIST Rule support CDATA #REQUIRED>
    <!ATTLIST Rule confidence CDATA #REQUIRED>
    <!ATTLIST Rule specializable ("yes"|"no") #REQUIRED>
    <!ATTLIST Rule processed ("yes"|"no") #REQUIRED>
```

Au chargement, nœud.closure = `LeftSide` + `RightSide`

Remarques :
- La dtd, et par la même le fichier xml, associés à l'arborescnces ont été modifiés par

rapport à la version donnée dans le rapport du projet de génie logiciel, ceci pour des raisons de compréhension et de facilité de manipulation. En effet, on a ajouté l'élément `RightSide` qui contiendra la fermeture du nœud. Pour ce qui est des attributs de l'arborescence, on stockera le nom des fichiers associés à la table (`Table`), à l'ordre des sup-irréductibles (`supOrder`) et à l'ordre des inf-irréductibles (`infOrder`). On a ajouté comme attribut d'une règle `processed` qui permet de savoir si la règle a été traitée ou pas.
- Ces modifications n'auront pas de conséquences sur l'existant.
- En théorie seuls les nœuds traités (ayant `processed`=yes) devraient être affichés mais en pratique tous les nœuds le seront.

4.9.5 Exemple de fichier XML_A

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE RuleTree SYSTEM "Arbo.dtd">

<RuleTree table="essai.xml" supOrder="supEssai.xml"
infOrder="infEssai.xml">
    <Rule support="3" confidence="100" specializable="yes"
    processed="yes">
        <LeftSide>
            <Item>a1</Item>
            <Item>a2</Item>
            <Item>a3</Item>
            <Item>a4</Item>
        </LeftSide>
        <PrincipalItem>F</PrincipalItem>
        <RightSide>
            <Item>a2</Item>
            <Item>a4</Item>
        </RightSide>
        <Rule support="2" confidence="100" specializable="yes"
        processed="yes">
            <LeftSide>
                <Item>a1</Item>
                <Item>a2</Item>
                <Item>a4</Item>
            </LeftSide>
            <PrincipalItem>F</PrincipalItem>
            <RightSide>
                <Item>a1</Item>
                <Item>a4</Item>
            </RightSide>
            <Rule support="1" confidence="100" specializable="yes"
            processed="no">
                <LeftSide>
                    <Item>a1</Item>
                    <Item>a3</Item>
```

```
                    </LeftSide>
                    <PrincipalItem>F</PrincipalItem>
                    <RightSide>
                        <Item>a1</Item>
                    </RightSide>
                </Rule>
                <Rule support="1" confidence="100" specializable="yes"
                processed="no">
                    <LeftSide>
                        <Item>a3</Item>
                        <Item>a2</Item>
                    </LeftSide>
                    <PrincipalItem>F</PrincipalItem>
                    <RightSide>
                        <Item>a2</Item>
                    </RightSide>
                </Rule>
            </Rule>
        </Rule>
    </RuleTree>
```

5 Implantation des TDA

5.1 Lecture/Ecriture XML

5.1.1 Présentation de la libxml

Le langage XML (e**X**tensible **M**arkup **L**anguage) a été créé pour mettre en place et définir de façon structurée et hierarchisée les données pour, entre autre, faciliter les échanges entre apllications. Un document XML commence toujours par un nœud racine qui lui-même peut contenir d'autres nœuds.

Il existe trois manières principales de parcourir un fichier XML :
- **DOM** : consiste à charger toutes les informations en mémoire. Chaque donnée est assimilée à un nœud d'une arborescence qui pourrait être parcourue. Cette technique est plus simple à programmer que SAX mais elle utilise plus de mémoire et est plus lente.
- **SAX** : parcourt le fichier en le survolant pour trouver les données recherchées.
- **Xpath** : est un standard proposé par le World Wide Web Consortum (W3C) permettant la localisation de nœuds dans l'arbre d'un document XML. Dans ce but, il définit une syntaxe pour décrire un chemin et un jeu de fonctions. XPath n'est pas destiné à être utilisé seul, mais avec XSLT ou XPointer qui en font un usage intensif et l'étendent pour leurs propres besoins par de nouvelles fonctions et de nouveaux types de base.

Afin de parser les fichiers XML il nous faut utiliser la bibliothèque libxml2 [6] compatible avec le langage C++. L'interface qui sera utilisée en particulier est la classe `xmlReader`. En effet, la plupart des fonctions de cette bibliothèque sont basées sur la notion d'arbre, c'est-à -dire à l'opération de "parsing" résulte d'un document entièrement chargé en mémoire, ce qui est relativement simple et puissant mais qui présente des limitations de la mémoire disponible.

Pour réaliser ceci il faut inclure le fichier : `libxml/xmlreader.h`
On créé un xmlreader sur le fichier pointé par filename :

```
xmlTextReaderPtr reader = NULL;
reader = xmlNewTextReaderFilename(filename);
```

Ensuite on avance dans le fichier par la ligne de commande :

```
ret = xmlTextReaderRead(reader);
```

Cette fonction permet aussi de tester si la lecture s'effectue sans erreur, en retournant `1`. Lorsqu'elle renvoie `0` cela signifie que le parser est en fin de fichier.

Ainsi on est sur le nœud racine du document.

Un nœud XML a plusieurs propriétés : *NodeType*, *Name*, *Depth*, *HasAttributes*, *HasValue*, *Value*, ...

Notons quelques types de nœuds :
- **1** : pour les "start element", par exemple `<Balise>`
- **15** : pour les "end of element", par exemple `</Balise>`
- **2** : pour les attributs du nœud

– **3** : pour les nœuds de texte (c'est-à-dire contenu entre deux balises : `<Balise>` ... `</Balise>`)
– ...

Pour tester le type d'un nœud on dispose de la fonction `xmlTextReaderNodeType( reader)` qui retourne un entier.

Les autres fonctions disponibles pour traiter les nœuds sont les suivantes :

Si l'on souhaite récupérer la valeur d'un attribut, on pourra appeler la fonction `xmlTextReaderGetAttribute`, qui prend en paramètre deux objets de type `xmlChar *` et retourne un objet du même type. Prenons un exemple : `<Balise nom="test">`. Pour atteindre la valeur de l'attribut il va donc falloir convertir la chaîne `nom` avec `xmlCharStrdup` qui convertit un `char *` en `xmlChar *` (pour faire la conversion inverse un cast sera suffisant).
Ainsi il suffira d'écrire :

```
xmlChar * attrName = xmlCharStrdup("name") ;
tmpName = xmlTextReaderGetAttribute(reader, attrName) ;
```

Si on veut à présent vérifier que la le nœud sur lequel on se trouve est bien celui que l'on souhaite on peut récupérer son nom et le comparer de la manière suivante :

```
xmlStrcmp(xmlTextReaderName(reader), xmlCharStrdup("Balise")) == 0
```

Enfin pour obtenir la valeur d'un nœud, c'est-à-dire l'information contenue entre deux balises :

```
xmlTextReaderValue(reader)
```

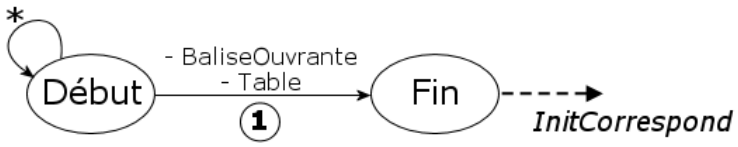
5.1.2 Automates pour parser les fichiers XML

• TDA Table :

Quatre fonctions sont concernées :

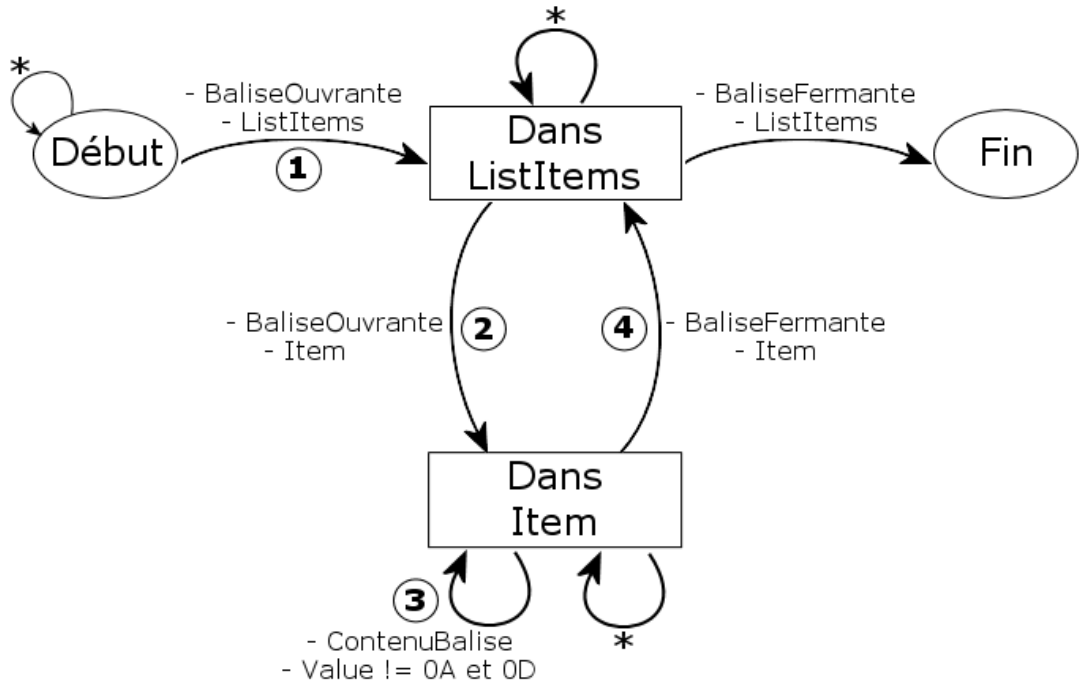
⋆ Rewind :
- Ouverture du fichier
- Appel de la fonction *readHeader*
- Initialisation : `tupleCourant = 0`

⋆ ReadHeader :



[**1**] : - Récupération des attributs de la balise `<Table...>`
- Initialisation des champs : `name`, `nbTuples`, `nbItems`
- Appel de la fonction *initCorrespond*

⋆ InitCorrespond :



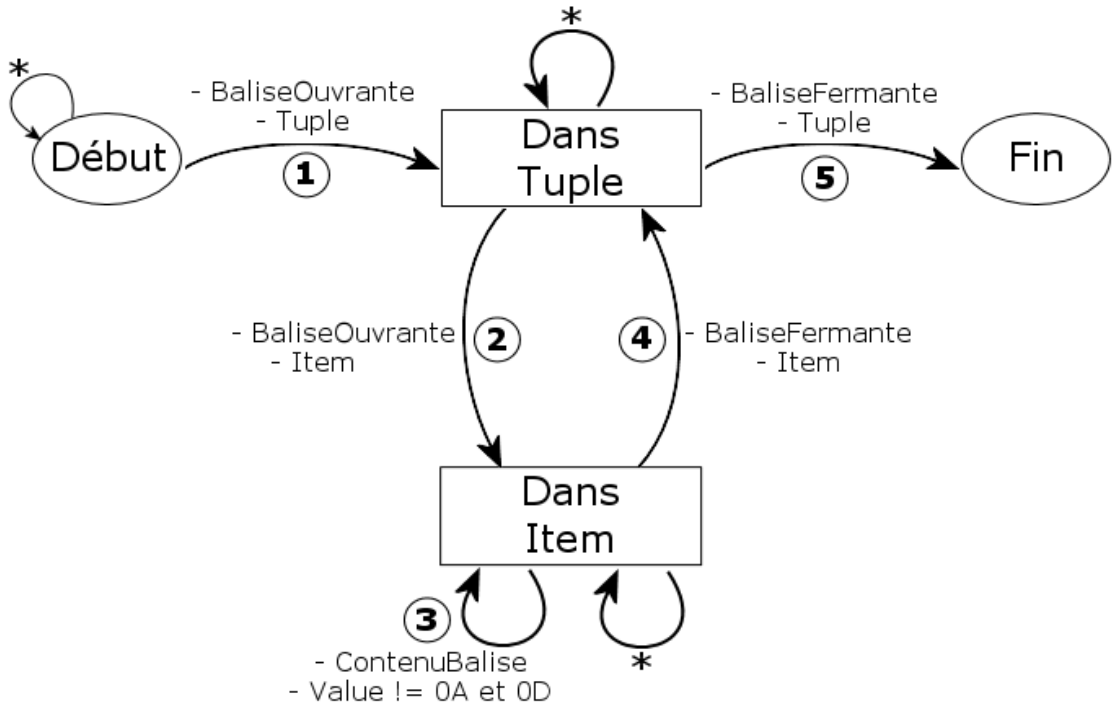
[**1**] : - Mise à zéro de l'attribut `mapping`
- Initialisation : `nbItemsDansMapping = 0`
- Passer au noeud suivant

[**2**] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

[**3**] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- `nbItemsDansMapping + 1`
- Passer au nœud suivant

[**4**] : - Insérer le couple `(nomItem,nbItemsDansMapping)` dans `mapping`

⋆ ReadTuple :



[**1**] : - Initialisation : `itemset = ∅`
- Passer au nœud suivant

[**2**] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

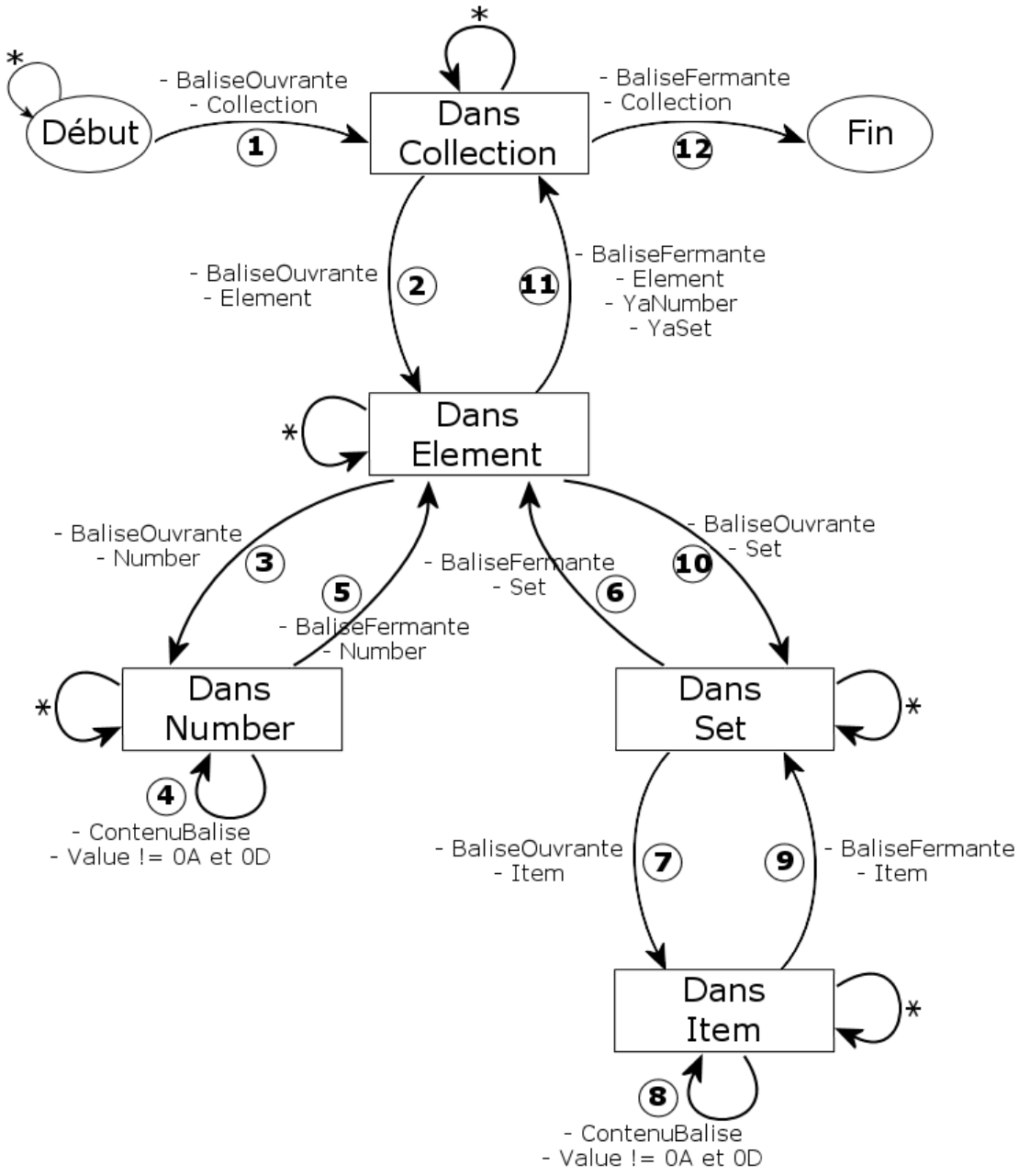
[**3**] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[**4**] : - `itemset = itemset` ∪ *getItemByName*(`nomItem`)

[**5**] : - `tupleCourant + 1`
- Si `tupleCourant = nbTuples` alors fermer le fichier
- Retourner `itemset`

- TDA SetOfElements :

  ⋆ Load :



[**1**] : - Récupération des attributs du nœud `<Collection ...>`
- Initialisation des champs : `nbElements`, `name`, `table`
- Passer au nœud suivant

[**2**] : - Initialisation : `yaNumber = FAUX`
- Initialisation : `yaSet = FAUX`
- Passer au nœud suivant

[3] : - Initialisation : `number = -1`
- Passer au nœud suivant

[4] : - `number =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[5] : - `yaNumber = VRAI`
- Passer au nœud suivant

[6] : - Initialisation : `itemset` = $\emptyset$
- Passer au nœud suivant

[7] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

[8] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[9] : - Insertion de *getItemByName(* `nomItem` *)* dans `itemset`
- Passer au nœud suivant

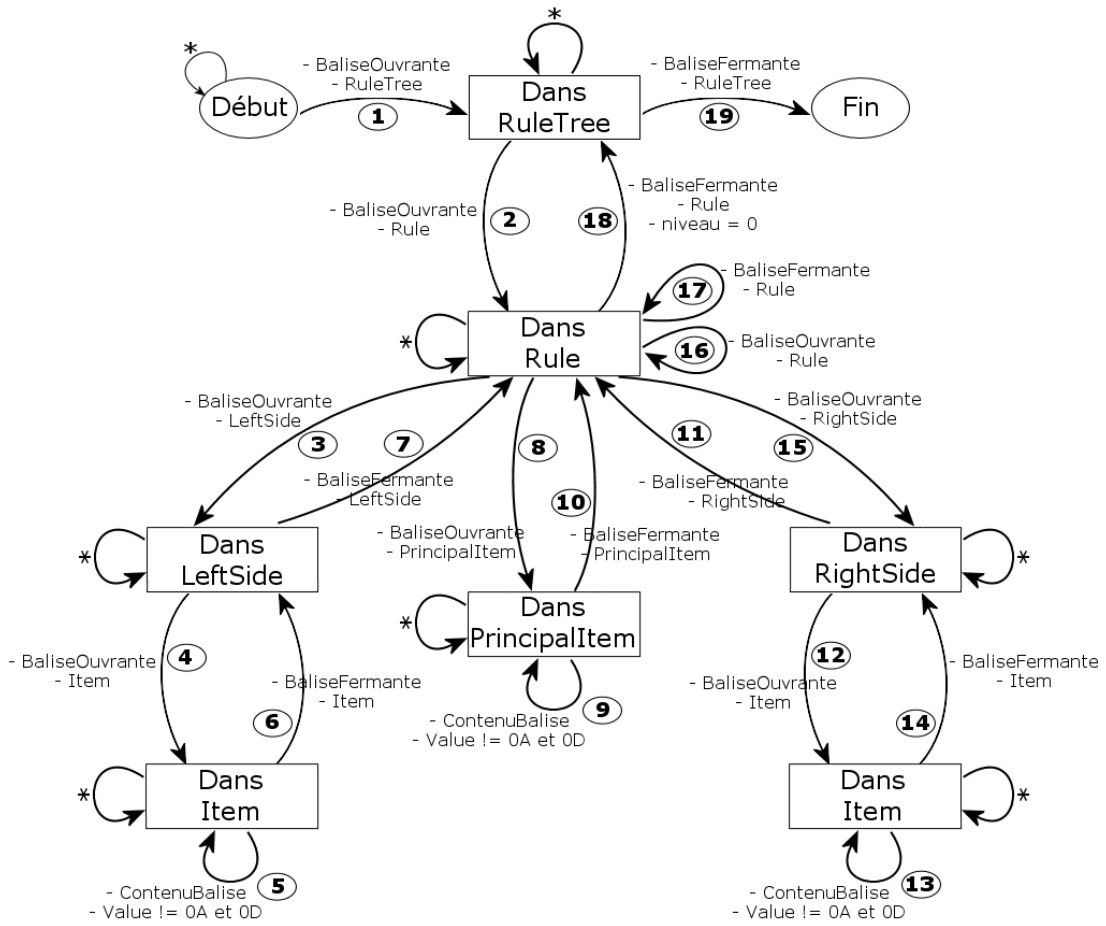
[10] : - `yaSet = VRAI`
- Passer au nœud suivant

[11] : - Insertion de `(number,itemset)` dans la Collection
- Passer au nœud suivant

[12] : - Fermeture du fichier XML

- TDA RuleTree :

  ⋆ LOAD : (utilisation d'un automate à pile)



[**1**] : - Récupération des attributs du nœud `<RuleTree ...>`
- Initialisation des champs : `table` et `name`
- Pour `orderSup` et `orderInf`, initialisation avec appel des fonctions : *setName*, *setTable* et *load*
- Initialisation : `numero = 0` (lastNumber)
- Initialisation d'un objet pile de Node (`pile`)
- Passer au nœud suivant

[**2**] : - Récupération des attributs du nœud `<Rule ...>`
- Initialisation d'un objet Node (`number`, `support`, `confidence`, `specializable`, `processed`)
- Empilement du nœud dans `pile`
- Initialisation : `niveau = 0`
- Passer au nœud suivant

[**3**] : - Initialisation d'un objet de type Itemset : `ensItems` = ∅
- Passer au nœud suivant

[**4**] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

[5] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[6] : - Insertion de *getItemByName(*`nomItem`*)* dans `ensItems`
- Passer au nœud suivant

[7] : - Initialisation de l'attribut du nœud : `itemset = ensItems`
- Modification du nœud en sommet de `pile`
- Passer au nœud suivant

[8] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

[9] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[10] : - Insertion de *getItemByName(*`nomItem`*)* dans `listNodesItem` = sommet de `pile`
- Passer au nœud suivant

[11] : - Initialisation d'un objet de type Itemset : `ensItems =` $\emptyset$
- Passer au nœud suivant

[12] : - Initialisation : `nomItem = ""`
- Passer au nœud suivant

[13] : - `nomItem =` récupérer la valeur contenue entre les balises
- Eventuellement retirer 0A et 0D de la fin de chaîne
- Passer au nœud suivant

[14] : - Insertion de *getItemByName(*`nomItem`*)* dans `ensItems`
- Passer au nœud suivant

[15] : - Initialisation de l'attribut `closure` d'un nœud = $LeftSide \cup RightSide$
- Passer au nœud suivant

[16] : - Récupération des attributs du nœud `<Rule ...>`
- Initialisation d'un nouvel objet Node (avec `parent` = sommet de `pile`)
- Ajout de ce nœud aux `children` du sommet de `pile`
- `niveau + 1`
- Passer au nœud suivant

[17] : - `niveau - 1`
- Dépiler le sommet de `pile`
- Ajout de ce nœud à `listNodes`
- Passer au nœud suivant

[18] : - Fermer le fichier XML

5.2 Utilisation de la STL

5.2.1 Set

Un set permet de représenter une collection ordonnée d'éléments sans doublon.

On dispose donc de toutes les opérations de "base" comme les constructeurs et destructeurs, ainsi que d'opération de comparaison :

```
c.size()  c.empty()  c.max_size()
c1 == c2  c1 != c2   c1 < c2
c1 > c2   c1 <= c2   c1 >= c2
```

On notera que l'insertion, la suppression et le test pour l'inclusion sont efficaces (puisque s'exécute en temps logarithmique).

```
c.insert(elem)  c.insert(pos,elem)  c.insert(beg,end)
c.erase(elem)   c.erase(pos)        c.erase(beg,end)
```

D'autres fonctions utiles :

```
c.count(elem)  c.find(elem)  c.clear()
```

Cette classe permet aussi de gérer les opérations ensemblistes :

```
set_difference()  set_intersection()  set_union()
```

Pour ce qui est de l'utilisation de ces fonctions on insistera sur les suivantes :

⋆ Includes :
Retourne vrai si le second ensemble est inclus dans le premier : $E2 \subseteq E1$?
RQ : On devra alors utiliser les itérateurs sur cette classe.

```
bool includes(first1, last1, first2, last2) ;
```

⋆ Set_Difference :
Fait la différence entre le premier ensemble et le second : $E1 \setminus E2$
On doit tout d'abord définir un `insert_iterator`, et `res` l'objet dans lequel on met le résultat : `insert_iterator<set<int, less<int> > > ii (res, res.begin()) ;`
Ensuite on appelle la fonction en passant à la fin en paramètre le `insert_iterator` :

```
set_difference(first1, last1, first2, last2, ii) ;
```

⋆ Set_Intersection :
Effectue l'intersection : $E1 \cap E2$
S'utilise de la même façon que `set_difference()` :

```
insert_iterator<set<int, less<int> > > ii (res, res.begin()) ;
set_intersection(first1, last1, first2, last2, ii) ;
```

⋆ Set_Union :
Effectue l'union : $E1 \cup E2$
S'utilise elle aussi comme `set_difference()` :

```
insert_iterator<set<int, less<int> > > ii (res, res.begin()) ;
set_union(first1, last1, first2, last2, ii) ;
```

5.2.2 Map

Un map permet de représenter une collection de paires de la forme (clé,valeur) sans doublon.

Les opérateurs de comparaison doivent exister sur le type de la clé. Les valeurs peuvent être quelconques.

```
c.size()  c.empty()  c.max_size()
c1 == c2  c1 != c2   c1 < c2
c1 > c2   c1 <= c2   c1 >= c2
```

On notera que l'insertion et la suppresion sont efficaces.

```
c.insert(elem)  c.insert(pos,elem)  c.insert(beg,end)
c.erase(elem)   c.erase(pos)        c.erase(beg,end)
```

D'autres fonctions utiles :

```
c.count(elem)  c.find(elem)  c.clear()
```

On dispose également d'un accès direct aux éléments du map : `m[key]`, retourne la valeur de l'élément dont la clé est `key`, ou insère un élément avec `key` si elle n'existe pas déjà.

Enfin pour les itérateurs, l'accès à la clé se fait par : `it -> first`, tandis que l'accès à la valeur par `it -> second`.

5.2.3 Stack

Ceci représente une pile, c'est-à-dire un objet utilisant le protocole LIFO (Last In First Out).

Les opérateurs de comparaison sont définis sur ce type (on notera que deux objets sont égaux si ils ont le même nombre d'éléments, et si ils contiennent les mêmes éléments dans le même ordre) :

```
c.size()  c.empty()  c.max_size()
c1 == c2  c1 != c2   c1 < c2
c1 > c2   c1 <= c2   c1 >= c2
```

Pour ce qui est des opérations élémentaires sur les piles :

```
c.push(elem)  c.top()  c.pop()
```

`push` : empile l'élément passé en paramètre.
`top` : retourne la valeur en sommet de pile.
`pop` : dépile le sommet.

D'autres fonctions utiles :

```
c.count(elem)  c.find(elem)
```

5.3 TDA et STL

Dans la plupart des cas, on stockera les ensembles à l'aide des `set` de la STL puisqu'il simplieront toutes les opérations d'intersection, d'union ou les tests d'inclusion. De plus, on préferera utiliser des `map` plutôt que des `vector` lorsque l'on a un élément qui est stocké par rapport à une clé (un entier en général), ainsi les "trous" dans la numérotation ne perturberont rien (ce qui aurait été le cas avec des vecteurs). Enfin l'objet `stack` nous sera utile pour implanter l'automate à pile, c'est-à-dire celui permettant de charger l'arborescence des règles.

5.3.1 Itemset

```
set<int> intSet
```

Remarques :
Un *Itemset* vide correspond à :
`intSet` = $\emptyset$.
L'intersection de deux *Itemset* est en fait l'intersection des deux `intSet`. De même pour l'union et l'inclusion.

5.3.2 Table

```
char * name
int nbTuples
int nbItems
map<char *,int> mapping
xmlTextReaderPtr xmlReader
int currentTuple
ifstream fic
```

Remarques :
Une *Table* vide correspond à :

```
name = ""
nbTuples = 0
nbItems = 0
mapping = ∅
currentTuple = 0
```

5.3.3 Element

```
int number
Itemset elementSet
```

Remarques :
Un *Element* vide correspond à :

```
number = 0
elementSet = ∅
```

5.3.4 SetOfElements

```
set<Element> collection
int nbElements
char * name
char * table
```

Remarques :
Un *Element* vide correspond à :

```
collection = ∅
nbElements = 0
```

L'intersection de deux *SetOfElements* se fait par rapport aux `number` des éléments formant cet ensemble. De même pour l'union ou l'inclusion. On a cependant redéfini une fonction (*unionSets*) qui fait l'union de deux *SetOfElements* par rapport aux `elementSet` des éléments.

5.3.5 Order

```
SetOfElements J
map<int, SetOfElements> imPred
map<int, SetOfElements> imSucc
char * name
char * table
```

Remarques :
Un *Element* vide correspond à :

```
J = ∅
imPred = ∅
imSucc = ∅
name = ""
table = ""
```

5.3.6 Node

```
int number
Itemset itemset
Itemset closure
int support
float confidence
bool specializable
bool processed
```

Remarques :
Un *Element* vide correspond à :

```
number = -1
itemset = ∅
closure = ∅
support = 0
```

```
confidence = 100
specializable = false
processed = false
```

5.3.7 SetOfNodes

```
set<Node> collection
```

Remarques :
Un *Element* vide correspond à :

```
collection = ∅
```

L'intersection de deux *SetOfNodes* se fait par rapport aux `number` des éléments formant cet ensemble. De même pour l'union ou l'inclusion.

5.3.8 RuleTree

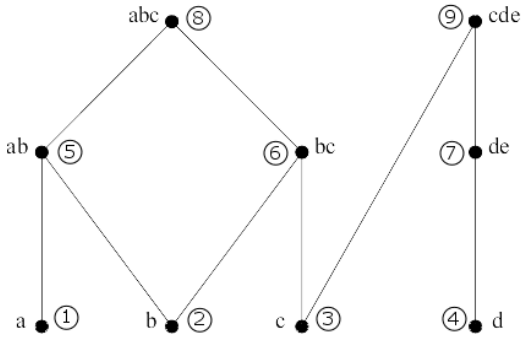
```
map<Node, int> listNodesItem
SetOfNodes listNodes
map<int, Node> parent
map<int, Itemset> children
char * name
char * table
Order orderSup
Order orderInf
int lastNumber
Itemset listItems
```

Remarques :
Un *RuleTree* vide correspond à :

```
listNodesItem = ∅
listNodes = ∅
parent = ∅
children = ∅
name = ""
table = ""
orderSup = ∅
orderInf = ∅
lastNumber = 0
```

5.4 Les tests

5.4.1 Order



Avec la table des correspondances suivantes :

nom	a	b	c	d	e
item	1	2	3	4	5

Voici l'ordre tel qu'il est stocké en mémoire :

```
Ensemble J :
nombre d'elements : 9
1 = ( - 1 - )
2 = ( - 2 - )
3 = ( - 3 - )
4 = ( - 4 - )
5 = ( - 1 -  - 2 - )
6 = ( - 2 -  - 3 - )
7 = ( - 4 -  - 5 - )
8 = ( - 1 -  - 2 -  - 3 - )
9 = ( - 3 -  - 4 -  - 5 - )

Ensemble des ImSucc :
[1] =     5 = ( - 1 -  - 2 - )

[2] =     5 = ( - 1 -  - 2 - )
          6 = ( - 2 -  - 3 - )

[3] =     6 = ( - 2 -  - 3 - )
          9 = ( - 3 -  - 4 -  - 5 - )

[4] =     7 = ( - 4 -  - 5 - )

[5] =     8 = ( - 1 -  - 2 -  - 3 - )

[6] =     8 = ( - 1 -  - 2 -  - 3 - )

[7] =     9 = ( - 3 -  - 4 -  - 5 - )
```

```
Ensemble des ImPred :
[5] =     1 = ( - 1 - )
          2 = ( - 2 - )

[6] =     2 = ( - 2 - )
          3 = ( - 3 - )

[7] =     4 = ( - 4 - )

[8] =     5 = ( - 1 -  - 2 - )
          6 = ( - 2 -  - 3 - )

[9] =     3 = ( - 3 - )
          7 = ( - 4 -  - 5 - )
```

• Max :

Recherche des éléments maximaux de l'ordre :

```
nombre d'elements : 2
8 = ( - 1 -  - 2 -  - 3 - )
9 = ( - 3 -  - 4 -  - 5 - )
```

• Min :

Recherche des éléments minimaux de l'ordre :

```
nombre d'elements : 4
1 = ( - 1 - )
2 = ( - 2 - )
3 = ( - 3 - )
4 = ( - 4 - )
```

• GetIdeal :

Test de cette fonction sur l'ensemble `6,9` (c'est-à-dire bc,cde)

```
Ensemble J :
nombre d'elements : 6
2 = ( - 2 - )
3 = ( - 3 - )
4 = ( - 4 - )
6 = ( - 2 -  - 3 - )
7 = ( - 4 -  - 5 - )
9 = ( - 3 -  - 4 -  - 5 - )

Ensemble des ImSucc :
[2] =     6 = ( - 2 -  - 3 - )

[3] =     6 = ( - 2 -  - 3 - )
```

```
          9 = ( - 3 -  - 4 -  - 5 - )

[4] =     7 = ( - 4 -  - 5 - )

[6] =

[7] =     9 = ( - 3 -  - 4 -  - 5 - )

Ensemble des ImPred :
[6] =     2 = ( - 2 - )
          3 = ( - 3 - )

[7] =     4 = ( - 4 - )

[9] =     3 = ( - 3 - )
          7 = ( - 4 -  - 5 - )
```

• RemoveFilter :

Test de cette fonction sur l'ensemble `3` (c'est-à-dire c)

```
Ensemble J :
nombre d'elements : 5
1 = ( - 1 - )
2 = ( - 2 - )
4 = ( - 4 - )
5 = ( - 1 -  - 2 - )
7 = ( - 4 -  - 5 - )

Ensemble des ImSucc :
[1] =     5 = ( - 1 -  - 2 - )

[2] =     5 = ( - 1 -  - 2 - )

[4] =     7 = ( - 4 -  - 5 - )

[5] =

[7] =

Ensemble des ImPred :
[5] =     1 = ( - 1 - )
          2 = ( - 2 - )

[7] =     4 = ( - 4 - )
```

• ImSuccIdeal : Test de cette fonction sur l'ensemble `2,6`

```
nombre d'elements : 2
```

```
1 = ( - 1 -  - 2 -  - 3 -  - 6 - )
4 = ( - 2 -  - 3 -  - 4 -  - 6 - )
```

• ImPredIdeal :
Test de cette fonction sur l'ensemble `1,2,3,4,5,6,7,8,9`

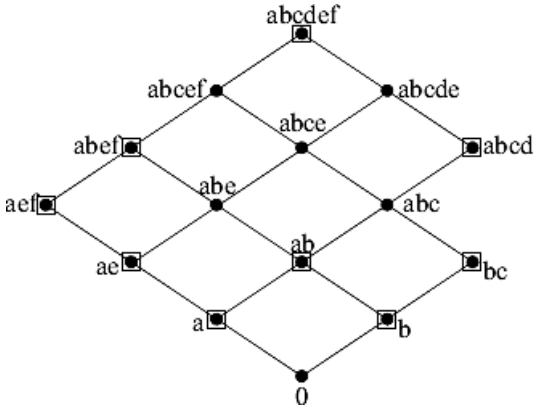
```
nombre d'elements : 2
8 = ( - 1 -  - 2 -  - 3 -  - 4 -  - 5 -  - 6 -  - 7 -  - 9 - )
9 = ( - 1 -  - 2 -  - 3 -  - 4 -  - 5 -  - 6 -  - 7 -  - 8 - )
```

5.4.2 RuleTree

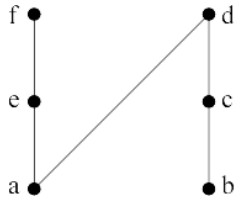
• Specialize :
Prenons la relation binaire suivante :

a	b	c	d	e	f
1	0	0	0	0	0
0	1	0	0	0	0
1	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	1	0
1	1	1	1	0	0
1	0	0	0	1	1
1	1	0	0	1	1

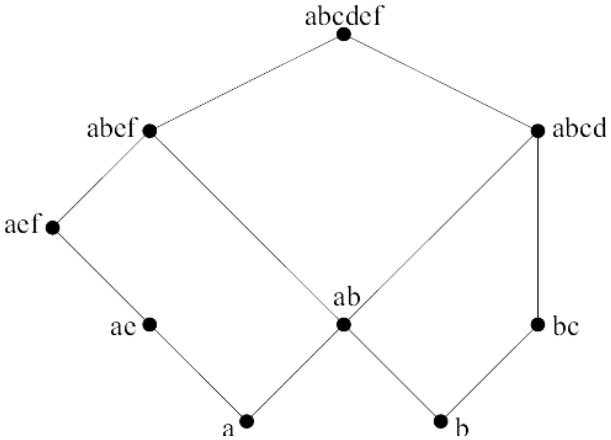
Le treillis des idéaux :



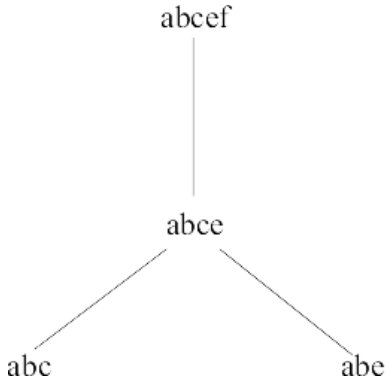
L'ordre des sup-irréductibles :



L'ordre des fermés :



Après spécialisation, on a :



On souhaite spécialiser le nœud `abcef`, soit le `0`.

```
number : 0
itemset : (0 - 1 - 2 - 4 - 5)
closure : (0 - 1 - 2 - 3 - 4 - 5)
support : 1
confidence : 100
specializable : true
processed : false
```

On obtient :

```
listNodesItem :
0 => 3
1 => 3

listNodes :  - 0 -  - 1 -

LeftSide de chaque noeud :
0 : ( - 0 -  - 1 -  - 2 -  - 4 -  - 5 - )
1 : ( - 0 -  - 1 -  - 2 -  - 4 - )
```

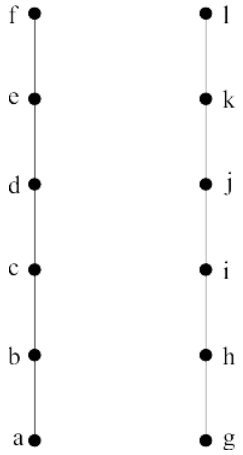
```
parent :
[1] = {0}

children :
[0] = { - 1 - }

table : tableNavig.xml
```

- Jump :

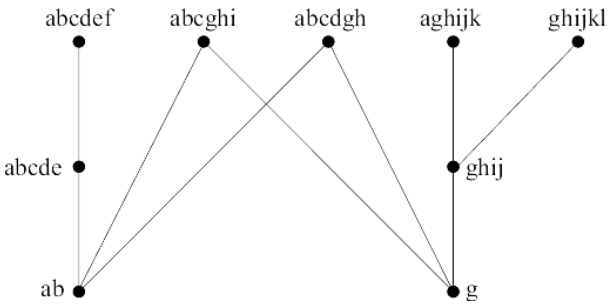
Prenons la relation binaire suivante :

a	b	c	d	e	f	g	h	i	j	k	l
1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	0
1	1	1	0	0	0	1	1	1	0	0	0
1	1	1	1	0	0	1	1	0	0	0	0

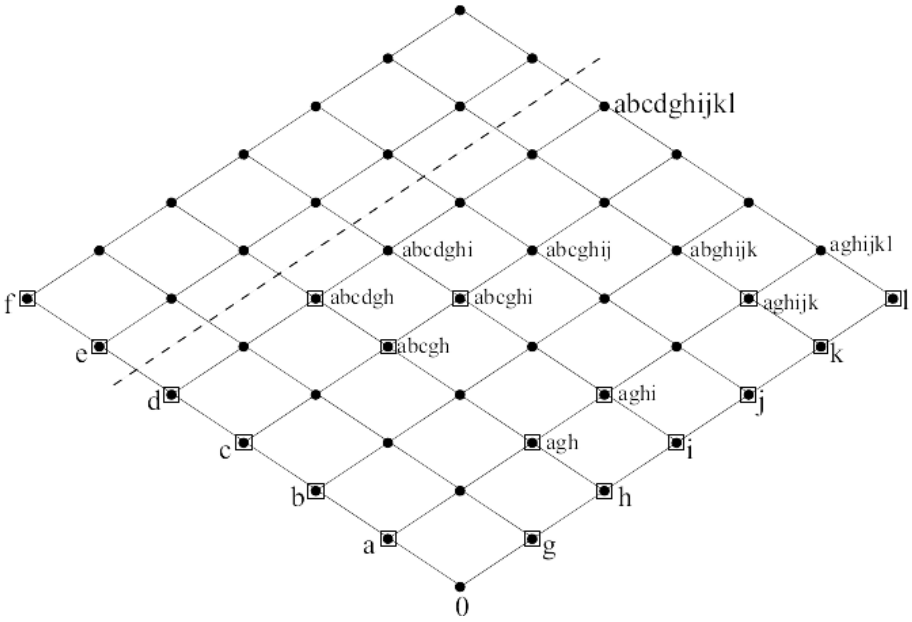
L'ordre des sup-irréductibles correspondant :



L'ordre des inf-irréductibles :



On a donc le treillis des idéaux suivant :



On souhaite donc effectuer un saut dans l'arborescence des règles d'association. On part de abcdghijkl (c'est-à-dire $J \uparrow e$) qui représente la règle : abcdghijkl → e.

Les inf maximaux situés sous abcdghijkl sont : abcdgh, abcghhi, aghijk et ghijkl.
A partir de ceux-ci on obtient (en cherchant leurs successeurs immédiats) :

- abcdgh ⇒ abcdghi
- abcghi ⇒ abcdghi et abcghij
- aghijk ⇒ abghijk et aghijkl
- ghijkl ⇒ aghijkl

On doit alors générer les règles suivantes : - abcdghi → e *(non spécialisable)*
- abcghij → e *(spécialisable)*
- abghijk → e *(spécialisable)*
- aghijkl → e *(non spécialisable)*

Si l'on exécute le programme on aura :

Avant le saut, l'arborescence a été chargée en mémoire ainsi que l'ordre des inf et l'ordre des sup associés.

```
listNodesItem :
0 => 4

listNodes :  - 0 -

LeftSide de chaque noeud :
0 : ( - 0 -  - 1 -  - 2 -  - 3 -  - 6 -  - 7 -  - 8 -  - 9 -  - 10 -
     - 11 - )
```

```
parent :

children :

table : tableJump.xml

lastNumber : 1
```

Après le saut effectué sur le noeud 0.

```
number : 0
itemset : (0 - 1 - 2 - 3 - 6 - 7 - 8 - 9 - 10 - 11)
closure : (0 - 1 - 2 - 3 - 4 - 6 - 7 - 8 - 9 - 10 - 11)
support : 1
confidence : 100
specializable : true
processed : false


listNodesItem :
0 => 4
1 => 4
4 => 4
6 => 4
7 => 4
13 => 4
14 => 4

listNodes :  - 0 -  - 1 -  - 4 -  - 6 -  - 7 -  - 13 -  - 14 -

LeftSide de chaque noeud :
0 : ( - 0 -  - 1 -  - 2 -  - 3 -  - 6 -  - 7 -  - 8 -  - 9 -  - 10 -
     - 11 - )
1 : ( - 0 -  - 6 -  - 7 -  - 8 -  - 9 -  - 10 -  - 11 - )
4 : ( - 0 -  - 1 -  - 6 -  - 7 -  - 8 -  - 9 -  - 10 - )
6 : ( - 0 -  - 1 -  - 6 -  - 7 -  - 8 -  - 9 - )
7 : ( - 0 -  - 1 -  - 2 -  - 3 -  - 6 -  - 7 -  - 8 - )
13 : ( - 0 -  - 1 -  - 2 -  - 6 -  - 7 -  - 8 -  - 9 - )
14 : ( - 0 -  - 1 -  - 6 -  - 7 -  - 8 -  - 9 - )

parent :
[1] = {0}
[4] = {0}
[6] = {4}
[7] = {0}
[13] = {0}
[14] = {13}
```

```
children :
[0] = { - 1 -  - 4 -  - 7 -  - 13 - }
[1] = {}
[4] = { - 6 - }
[6] = {}
[7] = {}
[13] = { - 14 - }
[14] = {}

table : tableJump.xml

lastNumber : 16
```

6 Utilisation des TDA en C++

Chacun des TDA définis a été implémenté, c'est-à-dire que chacun représente une classe en C++. Les attributs de ces classes sont explicités dans la partie **5.3**. Pour ce qui est des méthodes, chaque classe dispose : d'un constructeur par défaut, d'un constructeur de recopie, d'un destructeur, des surcharges d'opérateurs nécessaires ainsi que des opérations de la STL redéfinies (intersection, union...), des fonctions définies précédemment, et éventuellement d'itérateurs sur la classe.

Aux fonctions définies lors de l'étude de cas, ont été ajoutées d'autres fonctions telles que : une méthode `clear`, la surcharge de différentes opérations de la STL et une méthode `affiche`.

Voici la liste des API :

6.1 Itemset

Constructeur par défaut
```
Itemset()
```

Constructeur par recopie
```
Itemset(const Itemset &)
```

Destructeur
```
~Itemset()
```

Remise à zéro d'un objet de type *Itemset*
```
void clear()
```

Surcharge de l'opérateur =
```
Itemset & operator=(const Itemset &)
```

Surcharge de l'opérateur !=
```
bool operator !=(Itemset) const
```

Surcharge de l'opérateur ==
```
bool operator==(Itemset) const
```

Surcharge de l'opérateur <
```
bool operator<(Itemset)
```

Initialise l'attribut *intSet*
```
void setIntSet(set<int>)
```

Récupère l'attribut *intSet*
```
set<int> getIntSet() const
```

Ajoute un entier à l'ensemble

`void` **add**`(int)`

Retire un entier à l'ensemble
`void` **remove**`(int)`

Teste si un entier appartient à cet ensemble
`bool` **isIn**`(int) const`

Teste si un ensemble est vide
`bool` **isEmpty**`() const`

Effectue l'intersection entre deux objets *Itemset*
`Itemset` **I_intersection**`(Itemset)`

Effectue l'union entre deux objets *Itemset*
`Itemset` **I_union**`(Itemset)`

Effectue la différence entre deux objets *Itemset*
`Itemset` **I_minus**`(Itemset)`

Teste si un ensemble est contenu dans un autre
`bool` **I_includes**`(Itemset)`

Itérateur sur un objet de type *Itemset*
`typedef set<int>::iterator` **I_iterator**

Affiche un *Itemset*
`void` **affiche**`() const`

6.2 Table

Constructeur par défaut
`Table()`

Constructeur par recopie
`Table(const Table &)`

Destructeur
`~Table()`

Remise à zéro d'un objet de type *Table*
`void` **clear**`()`

Initialise l'attribut *name*
`void` **setName**`(char *)`

Retourne l'attribut *name*
`char *` **getName**`() const`

Initialise l'attribut *nbTuples*
`void` **setNbTuples**`(int)`

Retourne l'attribut *nbTuples*
`int` **getNbTuples**`() const`

Initialise l'attribut *nbItems*
`void` **setNbItems**`(int)`

Retourne l'attribut *nbItems*
`int` **getNbItems**`() const`

Initialise l'attribut *mapping*
`void` **setMapping**`(map<char *,int>)`

Retourne l'attribut *mapping*
`map<char *,int>` **getMapping**`() const`

Initialise l'attribut *xmlReader*
`void` **setXmlReader**`(xmlTextReaderPtr)`

Retourne l'attribut *xmlReader*
`xmlTextReaderPtr` **getXmlReader**`() const`

Initialise l'attribut *currentTuple*
`void` **setCurrentTuple**`(int)`

Retourne l'attribut *currentTuple*
`int` **getCurrentTuple**`() const`

Ouvre le fichier XML et place sur le premier tuple
`bool` **rewind**`()`

Lit les en-têtes du fichier XML
`void` **readHeader**`()`

Initialise le tableau des correspondances
`void` **initCorrespond**`()`

Lit un tuple dans le fichier XML
`Itemset` **readTuple**`()`

Ferme le fichier XML en cours de traitement
`void` **close**`()`

Ecrit les en-têtes dans le fichier XML ainsi que la liste des items

`void` **writeHeader**`()`

Ecrit un tuple dans le fichier XML
`void` **writeTuple**`(Itemset)`

Ecrit la fin du fichier XML et ferme le fichier
`void` **writeEnd**`()`

Retourne la liste de tous les items de la table
`Itemset` **getListItems**`()`

Récupère l'item par son nom
`int` **getItemByName**`(char *)`

Initialise le nom de l'item
`void` **setItemName**`(int, char *)`

Récupère le nom de l'item
`char *` **getNameOfItem**`(int)`

Affiche une *Table*
`void` **affiche**`()`

6.3 Element

Constructeur par défaut
`Element()`

Constructeur par recopie
`Element(const Element &)`

Destructeur
`~Element()`

Remise à zéro d'un objet de type *Element*
`void` **clear**`()`

Surcharge de l'opérateur =
`Element &` **operator=**`(const Element &)`

Surcharge de l'opérateur !=
`bool` **operator !=**`(Element) const`

Surcharge de l'opérateur ==
`bool` **operator==**`(Element) const`

Surcharge de l'opérateur <
`bool` **operator<**`(Element) const`

Initialise l'attribut *number*
`void` **setNumber**`(int)`

Retourne l'attribut *number*
`int` **getNumber**`() const`

Initialise l'attribut *elementSet*
`void` **setElementSet**`(Itemset &) const`

Retourne l'attribut *elementSet*
`Itemset` **getElementSet**`() const`

Affiche un *Element*
`void` **affiche**`() const`

6.4 SetOfElements

Constructeur par défaut
`SetOfElements()`

Constructeur par recopie
`SetOfElements(const SetOfElements &)`

Destructeur
`~SetOfElements()`

Remise à zéro d'un objet de type *Element*
`void` **clear**()

Surcharge de l'opérateur =
`SetOfElements &` **operator=**`(const SetOfElements &)`

Surcharge de l'opérateur !=
`bool` **operator !=**`(SetOfElements)`

Surcharge de l'opérateur ==
`bool` **operator==**`(SetOfElements)`

Initialise l'attribut *collection*
`void` **setCollection**`(set<Element>)`

Retourne l'attribut *collection*
`set<Element>` **getCollection**`() const`

Initialise l'attribut *nbElements*
`void` **setNbElements**`(int)`

Retourne l'attribut *nbElements*
`int` **getNbElements**`() const`

Initialise l'attribut *name*
`void` **setName**`(char *)`

Retourne l'attribut *name*
`char *` **getName**`() const`

Initialise l'attribut *table*
`void` **setTable**`(char *)`

Retourne l'attribut *table*
`char *` **getTable**`() const`

Effectue l'union de deux *SetOfElements* uniquement sur les *elementSet* des éléments formant cet ensemble
`SetOfElements` **unionSets**`(SetOfElements)`

Effectue l'union de deux *SetOfElements*
`SetOfElements` **SoE_union**`(SetOfElements)`

Effectue la différence de deux *SetOfElements*
`SetOfElements` **SoE_minus**`(SetOfElements)`

Charge la collection à partir d'un fichier XML
`void` **load**`()`

Sauvegarde la collection dans un fichier XML
`void` **save**`()`

Ajoute un élément à la collection
`void` **add**`(Element)`

Retire un élément de la collection
`void` **remove**`(Element)`

Récupère le numéro d'un élément par l'ensemble associé
`int` **getNumberBySet**`(Itemset)`

Récupère l'ensemble associé à un élément par son numéro
`Itemset` **getSetByNumber**`(int)`

Retourne un ensemble contenant les *number* des éléments de l'ensemble
`Itemset` **getNumbers**`()`

Retourne le nombre d'éléments dans l'ensemble

`int` **getSize**`()`

Calcule la fermeture des *elementSet* des éléments de l'ensemble
`void` **closureAll**`(char *)`

Itérateur sur un objet de type *SetOfElements*
`typedef set<Element> ::iterator` **SoE_iterator**

Affiche un *SetOfElements*
`void` **affiche**`()`

6.5 Order

Constructeur par défaut
`Order()`

Constructeur par recopie
`Order(const Order &)`

Destructeur
`~Order()`

Remise à zéro d'un objet de type *Order*
`void` **clear**`()`

Initialise l'attribut *J*
`void` **setJ**`(SetOfElements)`

Retourne l'attribut *J*
`SetOfElements` **getJ**`() const`

Initialise l'attribut *imPred* pour l'*Element*
`void` **setImPred**`(Element, SetOfElements)`

Retourne l'attribut *imPred* pour l'*Element*
`SetOfElements` **getImPred**`(Element)`

Initialise l'attribut *imSucc* pour l'*Element*
`void` **setImSucc**`(Element, SetOfElements)`

Retourne l'attribut *imSucc* pour l'*Element*
`SetOfElements` **getImSucc**`(Element)`

Initialise l'attribut *name*
`void` **setName**`(char *)`

Retourne l'attribut *name*
`char *` **getName**`()`

Initialise l'attribut *table*
`void` **setTable**`(char *)`

Retourne l'attribut *table*
`char *` **getTable**`()`

Charge la collection de l'ordre à partir d'un fichier XML
`void` **load**`()`

Sauvegarde la collection de l'ordre dans un fichier XML
`void` **save**`()`

Retourne l'ensemble des numéros de éléments de J
`Itemset` **getJnumbers**`()`

Insère un élément dans l'ordre
`void` **insert**`(Element)`

Retire un élément de l'ordre
`void` **remove**`(Element)`

Retourne les éléments maximaux de l'ordre
`SetOfElements` **max**`()`

Retourne les éléments minimaux de l'ordre
`SetOfElements` **min**`()`

Recopie l'ordre dans un autre
`Order` **duplicate**`()`

Renvoie l'ordre qui correspond à l'idéal de l'ensemble donné
`Order` **getIdeal**`(SetOfElements)`

Retire le filtre de l'*Element* de l'ordre
`Order` **removeFilter**`(Element)`

Retourne les successeurs immédiats de l'ensemble donné
`SetOfElements` **imSuccIdeal**`(SetOfElements)`

Retourne les prédecesseurs immédiats de l'ensemble donné
`SetOfElements` **imPredIdeal**`(SetOfElements)`

Affiche un *Order*
`void` **affiche**`()`

6.6 Node

Constructeur par défaut
`Node()`

Constructeur par recopie
`Node(const Node &)`

Destructeur
`~Node()`

Remise à zéro d'un objet de type *Node*
`void` **clear**`()`

Surcharge de l'opérateur =
`Node &` **operator=**`(const Node &)`

Surcharge de l'opérateur !=
`bool` **operator !=**`(Node) const`

Surcharge de l'opérateur ==
`bool` **operator==**`(Node) const`

Surcharge de l'opérateur <
`bool` **operator<**`(Node) const`

Teste si un nœud est nul
`bool` **isNull**`() const`

Initialise l'attribut *number*
`void` **setNumber**`(int &) const`

Retourne l'attribut *number*
`int` **getNumber**`() const`

Initialise l'attribut *itemset*
`void` **setItemset**`(Itemset &) const`

Retourne l'attribut *itemset*
`Itemset` **getItemset**`() const`

Initialise l'attribut *closure*
`void` **setClosure**`(Itemset &) const`

Retourne l'attribut *closure*
`Itemset` **getClosure**`() const`

Initialise l'attribut *support*

`void` **setSupport**`(int &) const`

Retourne l'attribut *support*
`int` **getSupport**`() const`

Initialise l'attribut *confidence*
`void` **setConfidence**`(float &) const`

Retourne l'attribut *confidence*
`float` **getConfidence**`() const`

Initialise l'attribut *specializable*
`void` **setSpecializable**`(bool &) const`

Retourne l'attribut *specializable*
`bool` **getSpecializable**`() const`

Initialise l'attribut *processed*
`void` **setProcessed**`(bool &) const`

Retourne l'attribut *processed*
`bool` **getProcessed**`() const`

Affiche un *Node*
`void` **affiche**`() const`

6.7 SetOfNodes

Constructeur par défaut
`SetOfNodes()`

Constructeur par recopie
`SetOfNodes(const SetOfNodes &)`

Destructeur
`~SetOfNodes()`

Remise à zéro d'un objet de type *SetOfNodes*
`void` **clear**`()`

Surcharge de l'opérateur =
`SetOfNodes &` **operator=**`(const SetOfNodes &)`

Surcharge de l'opérateur !=
`bool` **operator !=**`(SetOfNodes)`

Surcharge de l'opérateur ==
`bool` **operator==**`(SetOfNodes)`

Initialise l'attribut *collection*
`void` **setCollection**`(set<Node>)`

Retourne l'attribut *collection*
`set<Node>` **getCollection**`() const`

Ajoute un nœud à l'ensemble
`void` **add**`(Node)`

Retire un nœud de l'ensemble
`void` **remove**`(Node)`

Récupère un nœud par son *itemset*
`Node` **getNodeByItemset**`(Itemset)`

Récupère un nœud par son *number*
`Node` **getNodeByNumber**`(int)`

Récupère l'*itemset* par son *number*
`Itemset` **getItemsetByNumber**`(int)`

Met à jour le fermé de tous les nœuds de l'ensemble
`void` **updateClosureAll**`(Itemset)`

Itérateur sur un objet de type *SetOfNodes*
`typedef set<Node>::iterator` **SoN_iterator**

Affiche un *SetOfNodes*
`void` **affiche**`()`

6.8 RuleTree

Constructeur par défaut
`RuleTree()`

Constructeur par recopie
`RuleTree(const RuleTree &)`

Destructeur
`~RuleTree()`

Remise à zéro d'un objet de type *RuleTree*
`void` **clear**`()`

Initialise l'attribut *listNodesItem*
`void` **setListNodesItem**`(map<Node, int>)`

Retourne l'attribut *listNodesItem*
`map<Node, int>` **getListNodesItem**`() const`

Initialise l'attribut *listNodes*
`void` **setListNodes**`(SetOfNodes)`

Retourne l'attribut *listNodes*
`SetOfNodes` **getListNodes**`() const`

Initialise l'attribut *parent* pour le premier *Node*
`void` **setParent**`(Node, Node)`

Retourne l'attribut *parent* pour le *Node*
`Node` **getParent**`(Node)`

Initialise l'attribut *children* pour le *Node*
`void` **setChildren**`(Node, Itemset)`

Retourne l'attribut *children* pour le *Node*
`Itemset` **getChildren**`(Node)`

Initialise l'attribut *name*
`void` **setName**`(char *)`

Retourne l'attribut *name*
`char *` **getName**`() const`

Initialise l'attribut *table*
`void` **setTable**`(char *)`

Retourne l'attribut *table*
`char *` **getTable**`() const`

Initialise l'attribut *orderSup*
`void` **setOrderSup**`(Order)`

Retourne l'attribut *orderSup*
`Order` **getOrderSup**`() const`

Initialise l'attribut *orderInf*
`void` **setOrderInf**`(Order)`

Retourne l'attribut *orderInf*
`Order` **getOrderInf**`() const`

Initialise l'attribut *lastNumber*
`void` **setLastNumber**`(int)`

Retourne l'attribut *lastNumber*
`int` **getLastNumber**`() const`

Initialise l'attribut *listItems*
`void` **setListItems**`(Itemset)`

Retourne l'attribut *listItems*
`Itemset` **getListItems**`() const`

Retourne l'item impliqué par le *Node*
`int` **getItemByNode**`(Node)`

Retourne le nœud par son *number*
`Node` **getNodeByNumber**`(int)`

Retourne tous les numéros des nœuds de l'arborescence
`Itemset` **getNodesNumbers**`()`

Convertit un *Itemset* en *SetOfElements*
`SetOfElements` **itemsetToSetOfElements**`(Itemset)`

Convertit un *Itemset* en *SetOfNodes*
`SetOfNodes` **itemsetToSetOfNodes**`(Itemset)`

Charge l'arborescence à partir d'un fichier XML
`bool` **load**`()`

Sauvegarde une règle dans le fichier XML
`void` **saveRule**`(ofstream &, int, Table)`

Sauvegarde l'arborescence dans un fichier XML
`void` **save**`(char *)`

Ajoute un fils à un nœud
`void` **addChild**`(Node, Node)`

Met à jour un nœud
`void` **updateNode**`(Node)`

Insere un nœud dans l'arborescence par son *itemset* et son père
`void` **insert**`(Itemset, Node)`

Retire un nœud de l'arborescence
`void` **remove**`(Node)`

Spécialise un nœud de l'arborescence

`void` **specialize**`(Node)`

Effectue un saut dans l'arborescence des règles

`bool` **jump**`(Node)`

Affiche un *RuleTree*

`void` **affiche**`()`

6.9 Fichiers sources

Pour chaque TDA, deux fichiers sont utiles : un `.hpp` et un `.cpp`.

Le `.hpp` contient toutes les déclarations de fonctions, autrement dit les classes, et le `.cpp` contient l'implémentation des méthodes définies.

Pour pouvoir utiliser, par exemple, le TDA Table, il faudra inclure dans le fichier courant le `table.hpp`, et dans la ligne de commande il faudra penser à compiler `table.cpp`.

itemset.hpp	itemset.cpp
table.hpp	table.cpp
element.hpp	element.cpp
setOfElements.hpp	setOfElements.cpp
order.hpp	order.cpp
node.hpp	node.cpp
setOfNodes.hpp	setOfNodes.cpp
ruleTree.hpp	ruleTree.cpp

Les fichiers `.cpp` sont compilés et donc disponibles dans une librairie pour utiliser le TDA Table, il suffit d'inclure dans le fichier le `table.hpp` ainsi que la librairie `TDANavig.so`.

7 Conclusion

L'objectif était d'implanter les TDA définis dans l'étude de cas à l'aide de la STL en C++. Cet objectif est atteint. La bibliothèque de fonctions est disponible, toutes les fonctions ont été testées et validées sur différents jeux d'essai. L'équipe Algorithmique dispose donc d'une nouvelle brique pour sa plate-forme ARF. Il reste à intégrer dans la plate-forme les appels vers ces TDA.

Le présent rapport constitue un document de référence pour l'utilisation de cette bibliothèque de fonctions et devrait permettre aux futurs programmeurs d'implanter les différents algorithmes de l'équipe.

Les fonctions proposées dans cette bibliothèque ne sont pas optimisées pour les performances. Ceci était la volonté de l'équipe qui préfèrait avoir, dans un premier temps, un produit stable plutôt qu'efficace. Cette bibliothèque gagnerait à être optimisée. Par exemple, pour savoir si une règle est spécialisable la méthode actuelle consiste à calculer des fermetures et vérifier si l'item x est toujours impliqué. Ce calcul de fermetures nécessite un parcours complet de la table, ce qui permet au passage de calculer le support. Si le support n'est pas une priorité, une première amélioration consisterait à calculer les fermetures à l'aide des inf-irréductibles uniquement. Une autre amélioration consisterait à tester l'inclusion par rapport aux inf-irréductibles maximaux plutôt que de calculer des fermetures.

De la même façon, l'espace mémoire utilisé n'a pas été optimisé. Par exemple, dans le TDA RuleTree, une même règle peut apparaître à différents endroits. On pourrait envisager une autre implémentation qui éliminerait ce genre de redondances.

D'un point de vue plus personnel, ce TER m'a permis d'approfondir mes connaissances sur la STL et de mieux en maîtriser les subtilités. Il m'a permis aussi de découvrir la manipulation de documents XML à partir du C++. Cela m'a permis de redécouvrir les automates (déterministes et à pile) afin de réaliser le parsing des documents XML. Je pense que cette expérience de programmation me sera utile pour la suite.

Ce TER m'a également permis de découvrir le monde passionnant de la recherche. Il m'a confortée dans mon idée de poursuivre mes études par un mastère recherche. J'y ai appris et compris la nécessité de la rigueur dans le travail et de communiquer des documents propres et clairs si l'on souhaite que son travail soit exploité par d'autres.

Références

[1] Swami A. Agrawal R., Imielinski. *Mining Association Rules between Sets of Items in Large Databases*, 1993.

[2] Silicon Graphics. *Standard Template Library Programmer.* http ://www.sgi.com/tech/stl/.

[3] Nicolai M. Josuttis. *The C++ Standard Library - A Tutorial and Reference.* ADDISON-WESLEY, 1999.

[4] Raynaud O. Medina R. *Jump navigation in ideals lattice for knowledge discovery.*

[5] Raynaud O. Medina R., Nourine L. *Interactive Association Rules Discovery using Ideals Refinement*, 2003.

[6] Daniel Veillard. *The XML C parser and toolkit of Gnome.* http ://www.xmlsoft.org.