Sémantique et extension du langage C2QL pour la composition de techniques protégeant la confidentialité dans le nuage

Santiago Bautista

Juillet 2017

Résumé

Des applications de tout genre manipulent des personnelles de ses utilisateurs et utilisent le cloud pour s'exécuter ou s'héberger. Différentes techniques existent pour protéger la confidentialité de ces données. Pendant ce stage on a étudié la sémantique et prouvé les propriétés algébriques d'un langage permettant de décrire efficacement la composition de plusieurs de ces techniques, comme la fragmentation et le chiffrement.

Mots clés : privacy, cloud-computing, semantics, proof of correctness, algebraic laws, fragmentation, optimisation

Table des matières

1 Introduction

De plus en plus de logiciels sont développés pour être exécutés dans le cloud, et ses logiciels, de quelque sorte qu'ils soient (messagerie, gestion d'agenda personnel, commande de pizza ou reconnaissance vocale) traitent des données personnelles, qu'ils doivent donc protéger.

Une des propriétés qui doit être garantie dans la protection des données personnelles est la confidentialité. Différentes techniques existent pour protéger la confidentialité des données, comme par exemple le chiffrement et la fragmentation.

Dans sa thèse de 2016, Ronan Cherrueau montre que chacune de ces techniques a des avantages et des inconvénients, mais qu'*en composant les différentes techniques ensemble* on peut profiter de tous les avantages de ces techniques en éliminant la plupart des inconvénients. Il développe donc un langage, nommé C2QL (pour *Cryptographic Compositions for Query Language*), qui permet de décrire une telle composition de techniques de sécurisation des données pour en vérifier la correction et raisonner plus facilement.

Le langage se présente comme un ensemble de fonctions que l'on peut composer entre elles. Parmi ces fonctions, il y a les fonctions classiques pour faire des requêtes dans des bases de données, telles que la projection et la sélection, tout comme des fonctions décrivant la protection des données, comme le chiffrement ou la fragmentation.

Un des intérêts de ce langage est que, pour décider comment protéger les données des utilisateurs, le développeur peut suivre un processus simple en trois étapes. D'abord, le développeur écrit les requêtes *en ne tenant compte* ni du fait que le programme s'exécute dans le nuage, ni des mécanismes pour le protéger. Puis, il compose sa requête avec les fonctions de protection nécessaires (qui dépendent du problème en particulier, des contraintes de confidentialité spécifiques) pour avoir une requête sécurisée. Finalement, le développeur utilise des lois de commutation entre les différentes fonctions pour optimiser sa requête sécurisée.

Par conséquent, disposer de lois qui indiquent à quelles conditions les différentes fonctions du langage commutent est très important.

Or, si dans sa thèse R. Cherrueau donne la plupart de ces lois, il n'a pas eu le temps de les démontrer, ni de contempler tous les cas de figure.

C'est pourquoi, pendant ce stage, j'ai complété l'ensemble de lois fournies (section 3.2) dans la thèse de Ronan et j'ai formalisé la sémantique des différentes fonctions (section 3.1) pour ensuite démontrer la correction de ces lois (section 3.3).

Une description du contexte dans lequel s'inscrit ce stage est donnée à la section 2, et une discussion sur les aspects qui n'ont pas été traités est donnée à la section 4.

2 Contexte

2.1 De l'importance de composer les techniques de protection

De plus en plus d'applications, en particulier les applications web et les applications pour téléphone portable, cherchent à utiliser le nuage, soit pour stocker du code ou des données, soit pour faire des calculs, voir les deux à la fois.

En effet, le nuage peut offrir des services (que ce soit sous forme d'infrastructure, de plateforme ou de logiciel) disposant d'une forte disponibilité et faciles à redimensionner.

Autrement dit, pouvoir utiliser le nuage est devenu un enjeu de la conception logicielle, à cause des avantages de disponibilité et redimensionnement que cela offre.

Mais ce n'est pas le seul enjeux de la conception logicielle.

Vu que la plupart de ces applications manipulent des données personnelles, garantir la *confidentialité* des données est également un enjeux de ces applications là ; tout comme le sont les *performances* pour garantir une meilleure expérience à l'utilisateur de l'application.

Dans sa thèse, R. Cherrueau s'intéresse à trois techniques particulières utilisées dans le développement logiciel et regarde comment elles interagissent avec les trois enjeux cités plus haut. Ces trois techniques, qu'on va décrire brièvement, sont le *chiffrement*, la *fragmentation verticale* et l'*exécution* de l'application *chez l'utilisateur*.

Le chiffrement Lorsqu'il est bien utilisé, le chiffrement permet de garantir la confidentialité des données de l'utilisateur. De plus, dans certains cas, des calculs peuvent être faits sur les données chiffrées. On appelle chiffrement homomorphe un chiffrement avec lequel on peut effectuer des calculs avec les données chiffrées. Les chiffrement homomorphes totaux, comme celui de Gentry (**référence à ajouter**) sont pour l'instant trop contraignants pour pouvoir être utilisés dans la plupart des applications, mais les chiffrements homomorphes partiels, c'est à dire les chiffrement avec lesquels on peut effectuer *certaines* opérations sur les données chiffrées, peuvent se révéler très utiles. C'est le cas des chiffrements déterministes (dont les chiffrements symétriques) qui sont des chiffrement homomorphes partiels, permettant le test d'égalité.

Dans tous les cas, le chiffrement implique un surcoût en terme de calculs, donc diminue les performances. Dans certains cas, il améliore la confidentialité et permet l'utilisation du nuage.

L'exécution côté client Si le programme était exécuté entièrement par la machine de l'utilisateur, cela serait à la fois bon pour la confidentialité (car les données ne seraient pas du tout exposées aux risques liés à l'utilisation du nuage et du réseau) et pour les performances, puisque, à moins de traiter une trop grande quantité de données dans un calcul hautement parallélisable, les performances du cloud sont moins bonnes que celles des machines des utilisateurs. Par contre, l'exécution côté client ne permet pas de profiter des avantages du cloud.

La fragmentation verticale consiste à séparer les différentes données que manipule le programme entre deux clouds n'ayant aucun rapport entre eux (à deux endroits géographiques différents, gérés par des entités différentes, etc. . .). Ceci permet de protéger celles des données personnelles qui sont constituées d'une *association* de deux données. Par exemple, dans une application stockant un ensemble de rendez-vous, l'association $(\text{date}, \text{lieu})$ doit être protégée, car une personne malveillante ayant accès à ces deux informations là à la fois pourrait suivre l'utilisateur de l'application.

La fragmentation verticale contribue à protéger la confidentialité, mais d'une façon souvent moins forte que celles du chiffrement ou de l'exécution côté client. Par contre, chacun des fragments de données qu'elles génère peuvent être opérés séparément, en introduisant ainsi, lorsque le programme le permet, une dose de parallélisation supplémentaire qui peut

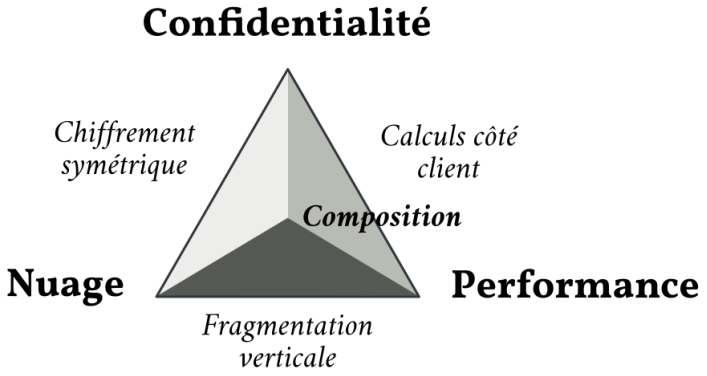

FIGURE 1 – Enjeux et techniques dans le cloud-computing

(image provenant de la thèse de Ronan Cherrueau)

améliorer les performances et permettre de tirer encore plus d'avantages du nuage.

La figure 1 résume comment ces trois techniques là interagissent avec les trois enjeux cités plus haut.

Dans sa thèse, Ronan Cherrueau montre qu'en composant ces différentes techniques on peut à la fois profiter des avantages du nuage, protéger la confidentialité et améliorer les performances d'un programme.

Pour pouvoir décrire comment s'effectue une telle composition, pour pouvoir vérifier la correction d'une telle composition et pour pouvoir raisonner dessus, Cherrueau a introduit un langage : C2QL.

2.2 Un langage pour décrire la composition : C2QL

Le langage C2QL donne une façon d'exprimer comment les techniques mentionnées ci-dessus se composent avec les fonctions classiques de l'algèbre relationnelle.

Les données manipulées sont donc représentées sous forme de tables, ou relations, c'est à dire un ensemble de lignes contenant des valeurs pour chacun(e) des différent(e)s attributs ou colonnes considéré(e)s.

Dans l'exemple de l'application stockant des rendez-vous, cette table contiendrait autant de lignes que de rendez-vous stockés dans l'application et (par exemple) trois colonnes ou attributs : nom de l'utilisateur ayant stocké le rendez-vous, date du rendez-vous et lieu du rendez-vous.

Les opérateurs empruntés à l'algèbre relationnelle présents dans ce langage sont

- La projection, notée π qui consiste à ne considérer que certains des attributs de la table.
- La sélection, notée σ qui consiste, pour une table donnée, à ne considérer que les lignes satisfiant un certain prédicat.
- La jonction naturelle, notée $\bowtie$, qui combine les informations de deux tables en fonction des attributs qu'elles ont en commun.

— L'aggrégation et la réduction (notées respectivement group et fold) permettant, respectivement, de regrouper les lignes qui partagent les mêmes valeurs pour un certain ensemble d'attributs et d'effectuer des opérations sur les groupes de lignes ainsi obtenus.

Les opérateurs relatifs à la protection des données présents dans le langage sont

— La fragmentation verticale, notée frag, qui sépare une table en deux tables : la première contenant certains des attributs (colonnes) de la table d'origine, et la deuxième contenant le reste des attributs de la table d'origine.
— La défragmentation verticale, notée defrag, qui effectue l'opération inverse : à partir de deux tables n'ayant pas d'attributs en commun, reconstruit une seule table. Un attribut spécial, *id*, servant à identifier chaque ligne et créer lors de la fragmentation d'une table, rend la défragmentation possible.
— Le chiffrement, noté crypt qui chiffre un attribut dans une table.
— Le déchiffrement, noté decrypt qui déchiffre un attribut dans une table.

« Où est passée le calcul côté client ? » est peut-être une question que vous vous posez peut-être en ce moment. La raison pour laquelle il n'y a aucun opérateur dans C2QL qui permette d'exprimer le rapatriement des données sur la machine de l'utilisateur est que cette gestion est faite implicitement, pour garantir que les requêtes C2QL protègent toujours la confidentialité, par conception. En effet, dans le langage C2QL on suppose que les données à protéger sont toujours soit un attribut, dans lequel cas il peut être protégé par chiffrement, soit une association de deux attributs, dans lequel cas elle peut être protégée par fragmentation. Par conséquent, dans une requête, le premier déchiffrement ou la première défragmentation (ou l'absence de chiffrement ou de fragmentation dans des cas où ils auraient été nécessaires) brisent la protection des données et requièrent que les données soient rapatriées chez l'utilisateur pour cette opération et toutes les opérations postérieures.

Ainsi, on peut, en regardant une expression C2QL et en connaissant les contraintes de confidentialité à respecter, déduire à quel moment les données doivent être rapatriées chez l'utilisateur ; sans que cette opération aie besoin d'apparaître explicitement dans l'expression.

Rajouter un exemple serait probablement utile

TODO Expliquer les contraintes de sécurité avant l'aspect implicite du c.c.

Ce langage a été défini comme un Langage de Domaine Spécifique Embarqué dans Idris, qui est un langage à types dépendants. Le système de typage d'Idris et les types donnés aux opérateurs permettent de vérifier, lorsqu'on écrit une requête en C2QL que celle-ci aura un sens au moment de l'exécution.

2.3 Utiliser C2QL : commuter les opérateurs

Dans sa thèse, Cherrueau expose un méthode pour se servir facilement et efficacement de C2QL, en trois étapes.

Première étape : écrire la version locale de l'application. Le développeur peut commencer par écrire les requêtes de son application sans tenir compte de l'utilisation du cloud ni de la protection des données. C'est ce qu'on appelle ici une version « locale » du programme ; c'est

le programme tel qu'il pourrait s'exécuter dans la machine de l'utilisateur, localement. Cette version de l'application est beaucoup plus facile à écrire qu'une version utilisant efficacement les techniques de protection.

Deuxième étape : ajouter les techniques de protection nécessaires. Le langage C2QL suppose que les contraintes de confidentialité sont de deux types : soit il s'agit d'une association entre deux attributs que l'on veut protéger, dans lequel cas on choisit de mettre les deux attributs de l'association dans des fragments distincts grâce à la fragmentation verticale, soit il s'agit de la valeur d'un attribut que l'on veut protéger, dans lequel cas soit on chiffre cette valeur, soit on la décompose sur plusieurs fragments avec la technique exposée par Aggarwal (**TODO rajouter la référence**). Dans les deux cas on peut passer des versions locales des requêtes à des versions protégées en faisant apparaître à droite de la requête les fonctions de protections composées avec leurs fonctions réciproques.

Troisième étape : faire commuter les différentes fonctions. En faisant commuter les différents opérateurs qui apparaissent dans les requêtes, on peut aboutir à une requête optimisée, profitant des avantages du cloud et de bonnes performances. Pour cela, on a besoin de savoir à quelles conditions les différents opérateurs de C2QL peuvent commuter.

C'est sur cet ensemble de lois qui indiquent à quelle condition deux fonctions de C2QL peuvent commuter que je me suis intéressé pendant mon stage.

3 Contribution

Dans mon travail avec l'ensemble de lois de C2QL, j'ai commencé par poser une sémantique formelle pour les différentes fonctions du langage C2QL (section 3.1),puis j'ai complété cet ensemble de lois, en m'intéressé à toutes les combinaisons possibles de lois que l'on pouvait vouloir commuter dans le langage (section 3.2), ensuite j'ai démontré la correction sémantique de ces lois-là (section 3.3) et en ce moment je travaille sur l'automatisation de l'optimisation des requêtes, en réfléchissant à comment déterminer, parmi les commutations possibles, lesquelles il faut choisir pour optimiser une requête(section 3.4).

3.1 Établir des définitions

Les définitions des différents opérateurs présentes dans la thèse de 2016 sont données en français, ce qui facilite leur compréhension, mais rend impossible une preuve mathématique de la correction des lois les concernant.

J'ai donc commencé par poser des définitions formelles des fonctions du langage C2QL.

Pour former ces définitions, j'ai du faire des choix à plusieurs reprises. À chaque fois, j'ai utilisé les deux mêmes critères pour guider mes choix : d'une part il faut que les définitions que je choisi permettent au langage d'être le plus expressif possible, d'autre part il faut que les définitions choisies facilitent une démonstration rigoureuse de la correction des lois de commutation.

Pour un document contenant l'ensemble des définition, se référer à l'annexe A.

Dans ce rapport je vais me centrer principalement sur celles des définitions où il y a eu des choix à faire (en exposant les motivations qui ont permit d'effectuer un choix plutôt qu'un autre) et celles des définitions où il y a des différences par rapport aux définitions qui étaient suggérées par les notations de la thèse, en expliquant les motivations de ces différences.

Des tuples ou des fonctions ?

Dans son livre de 1982, Ullman parle de deux façons de définir les relations (tables) de l'algèbre relationnelle : soit comme un sous-ensemble d'un produit de domaines (donc comme un ensemble de tuples, où chaque tuple représente une ligne de la table et chaque coordonnée de chaque tuple correspond à la valeur pour un attribut donné), soit comme un ensemble de fonctions définies sur l'ensemble des attributs (chaque fonction correspond donc à une ligne de la table, et la valeur d'un attribut pour une ligne donnée est son image par cette fonction).

Ullman décide d'utiliser la première définition pour le reste de son livre, et ne détaille pas plus la deuxième définition. C'est pourtant cette deuxième définition que j'ai décidé de prendre dans ce cas ci, puisque que ce soit pour le chiffrement comme pour la fragmentation, on fait ici référence aux différentes colonnes d'une table par leur nom (toutes les colonnes ont un nom, et tous les noms des colonnes sont différents) et il est donc plus facile de raisonner, par exemple, sur la fragmentation, avec cette définition-là : en effet, la fragmentation se définit alors en thermes de simples restrictions sur certains ensembles.

Des sélections portant sur plus d'un attribut à la fois.

Dans certaines des lois présentes dans la thèse, la notation suggère que les prédicats utilisés lors des filtrages ne portent que sur un seul attribut à la fois.

Ainsi, par exemple, dans la loi (7) à la page 30 de la thèse concernant la commutation entre la jointure naturelle et la sélection, on lit

$$\sigma_{p\alpha \wedge q\beta} \circ \bowtie \equiv \bowtie \circ (\sigma_{p\alpha}, \sigma_{q\beta}) \qquad \text{si } \alpha \in \Delta \text{ et } \beta \in \Delta'$$

où Δ et Δ' représentent respectivement le schéma relationnel du premier argument et le schéma relationnel du deuxième argument.

Cette notation suggère que les prédicats p et q portent chacun sur un seul attribut (respectivement α et β).

Le problème est qu'il y a certains prédicats qui ne peuvent pas ce décomposer en des prédicats portant chacun sur un seul attribut.

Par exemple, si on dispose d'une base données concernant des ornithorynques, qu'un des attributs de la base de données indique la date où l'ornithorynque a vu un kangourou pour la première fois, et qu'un autre des attributs indique la date où l'ornithorynque a pondu des oeufs pour la première fois, et que l'on ne veut s'intéresser qu'aux ornithorynques ayant vu des kangourous avant de pondre des œufs pour la première fois, alors le prédicat ("date de première vision d'un kangourou" < "date de première ponte des oeufs") ne peux pas être décomposé en deux prédicats qui porteraient l'un sur la date de la première vision d'un kangourou et l'autre sur la date de la première ponte d'œufs.

Ainsi, dans ma définition de ce qu'est une sélection et ce qu'est un prédicat, j'ai supposé

que la valeur de vérité d'un prédicat dépendait de plusieurs attributs à la fois, et pour refléter cela j'utilise la notion de *domaine d'un prédicat*.

Avec cette définition là, la loi de commutation entre la jonction et la sélection devient

$$\begin{aligned} \sigma_p \circ \bowtie &= \bowtie \circ (\sigma_p, \mathrm{id}) && \text{si } \mathrm{dom}(p) \subset \delta_1 \\ \sigma_p \circ \bowtie &= \bowtie \circ (\mathrm{id}, \sigma_p) && \text{si } \mathrm{dom}(p) \subset \delta_2 \end{aligned}$$

(où δ_1 et δ_2 sont les schémas relationnels de, respectivement, le premier et le deuxième argument).

Compter, ou agréger et réduire ?

Le développeur peut vouloir agir sur plusieurs lignes à la fois.

Le choix qui a été fait dans la thèse, c'est de traiter l'exemple de la fonction count, qui permet de regrouper, *en les comptant*, les lignes ayant les mêmes valeurs pour un certain ensemble d'attributs.

Je me suis posé la questions de savoir si les opportunités d'optimisation exposées dans la thèse par rapport à la fonction count et à sa façon de commuter avec les autres fonctions lui étaient propres, ou si elles étaient généralisables à toutes les fonctions d'agrégation. Plutôt que de m'intéresser à la fonction count, je me suis donc intéressé aux fonctions group et fold, qui permettent de regrouper, *en appliquant une fonction quelconque aux autres attributs*, les lignes ayant les mêmes valeurs pour un certain ensemble d'attributs.

Défragmentation et jointure, un air de famille trompeur

3.2 Compléter les propriétés

3.3 Prouver les propriétés

3.4 Optimiser les requêtes

4 Travail futur

5 Conclusion

Annexe A : Définitions sémantiques

Le but de ce document est de donner une définition formelle des fonctions dont est composé le langage C2QL.

Préambule

Définition 1 *On appelle* nom d'attribut *toute chaîne de caractères.*

Ici, pour simplifier, on appelle chaîne de caractères tout mot sur l'alphabet

$$\Sigma = \{a, \ldots, z\} \cup \{A, \ldots, Z\} \cup \{0, \ldots, 9\}$$

Vu que le nom d'attribut « id » joue un rôle particulier, on appelle, par opposition, nom d'attribut régulier *tout nom d'attribut autre que « id ».*

Définition 2 *On appelle* schéma relationnel *tout ensemble de noms d'attributs réguliers.*

Définitions générales

Définition 3 *On appellera* valeur *tout élément d'un certain ensemble $\mathcal{V}$, que l'on suppose non-vide, infini, dénombrable, et stable par formation de n-uplets (i.e. $\forall k \in \mathbf{N}, \mathcal{V}^k \subset \mathcal{V}$).*

Définition 4 *On appelle* relation *de schéma relationnel Δ tout ensemble de fonctions de $\Delta \cup \{id\}$ dans $\mathcal{V}$.*

Chacun des éléments de la relation (chacune de ces fonctions) est appelé(e) ligne.

Pour chaque ligne l de la relation et chaque α de Δ, $l(\alpha)$ est appelé attribut *de* nom *α pour la ligne l.*

L'image de id est appelée identifiant de la ligne, *et elle est, au sein de chaque relation, unique pour chaque ligne.*

Définition 5 *On appelle* S *l'ensemble des schémas relationnels possibles. Autrement dit, on pose* $\mathrm{S} = \mathcal{P}(\Sigma^* \setminus \{id\})$.

On appelle R *l'ensemble des relations possibles,*

et on introduit la fonction sch *de* R *dans* S *qui à une relation associe son schéma relationnel.*

Projections et sélections

Définition 6 *Pour tout ensemble δ de noms d'attributs réguliers, on appelle* projection sur les attributs *δ la fonction suivante :*

$$\begin{array}{rccc} \pi_\delta : & \mathrm{R} & \to & \mathrm{R} \\ & r & \mapsto & \{l|_{(\delta \cap \mathrm{sch}(r)) \cup \{id\}} / l \in r\} \end{array}$$

Définition 7 *On appelle* L *l'ensemble de toutes les lignes possibles.*

On appelle prédicat *toute fonction de* L *dans* $\{true, false\}$.

On appelle domaine *d'un prédicat* p *le plus petit ensemble* D *tel que :*

$$\forall (l, l') \in \mathrm{L}^2, (l|_D = l'|_D \Rightarrow p(l) = p(l'))$$

et on le note $\mathrm{dom}(p)$.

Définition 8 *On appelle* sélection de prédicat p*, pour tout prédicat* p*, la fonction :*

$$\begin{array}{rccc} \sigma_p : & \mathrm{R} & \to & \mathrm{R} \\ & r & \mapsto & r \cap p^{-1}(\{true\}) \end{array}$$

Jointure naturelle

Définition 9 *Si* r *et* r' *sont deux relations et que* l *est un élément de* r*. On appelle* correspondants de l dans r' *l'ensemble des lignes* l' *telles que*

$$\forall \alpha \in \mathrm{sch}(r) \cap \mathrm{sch}(r'), l'(\alpha) = l(\alpha)$$

On note $\mathrm{cor}_{r,r'}(l)$ *l'ensemble de ces lignes-là.*

Définition 10 *Si* l *et* l' *sont deux lignes correspondantes, on appelle* concaténation de l et de l'*, notée* $l.l'$ *la fonction de* $\mathrm{sch}(l) \cup \mathrm{sch}(l') \cup \{id\}$ *définie par :*

$$\begin{cases} l.l'(\alpha) = l(\alpha) & si\ \alpha \in \mathrm{sch}(r) \setminus \mathrm{sch}(r') \\ l.l'(\alpha) = l'(\alpha) & si\ \alpha \in \mathrm{sch}(r') \setminus \mathrm{sch}(r) \\ l.l'(\alpha) = l(\alpha) = l'(\alpha) & si\ \alpha \in \mathrm{sch}(r) \cap \mathrm{sch}(r') \\ l.l'(id) = \gamma & \gamma\ \textit{étant un identifiant frais} \end{cases}$$

où on appelle identifiant frais *une valeur qui ne soit l'identifiant d'aucune autre ligne dans le système.*

Définition 11 *Pour* r *et* r' *deux relations, on appelle* jointure naturelle *de* r *et* r' *la relation*

$$r \bowtie r' = \{l.l' / l \in r, l' \in \mathrm{cor}_{r,r'}(l)\}$$

On utilisera aussi la notation préfixe. En effet, on vient de définir la fonction

$$\begin{array}{rccc} \bowtie : & \mathrm{R}^2 & \to & \mathrm{R} \\ & (r, r') & \mapsto & r \bowtie r' \end{array}$$

Fragmentation et défragmentation

La défragmentation est presque un cas particulier de jointure naturelle, où l'identifiant serait considéré comme un attribut en commun pour les deux tables et il serait le seul.

Définition 12 *Deux relations r et r' sont dites* unifiables *si :*

$$\mathrm{sch}(r) \cap \mathrm{sch}(r') = \emptyset$$

On remarquera que deux relations unifiables non vides sont également joignables.

On note Ru *l'ensemble des paires de relations unifiables, qui est un sous-ensemble de* R^2.

Définition 13 *Pour tout ensemble de noms d'attributs réguliers δ on appelle* fragmentation de fragment gauche *δ l'application suivante :*

$$\begin{array}{rccl} \mathrm{frag}_\delta & \mathrm{R} & \to & \mathrm{Ru} \\ & r & \mapsto & (\{l|_{(\mathrm{sch}(r)\cap\delta)\cup\{id\}} / l \in r\}, \{l_{(\mathrm{sch}(r)\setminus\delta)\cup\{id\}} / l \in r\}) \end{array}$$

Définition 14 *On dit que deux lignes l et l' sont* unifiables *si elles partagent le même identifiant et que les relations correspondantes sont unifiables.*

On définit alors leur unification $\mathrm{Unif}(l, l')$ *comme la fonction définie sur* $\mathrm{sch}(l) \cup \mathrm{sch}(l') \cup \{id\}$ *par*

$$\begin{cases} \mathrm{Unif}(l,l')(\alpha) = l(\alpha) & si\ \alpha \in \mathrm{sch}(l) \\ \mathrm{Unif}(l,l')(\alpha) = l'(\alpha) & si\ \alpha \in \mathrm{sch}(l') \\ \mathrm{Unif}(l,l')(id) = l(id) = l'(id) & \end{cases}$$

Définition 15 *On appelle* défragmentation *la fonction de* Ru *à valeur dans* R *définie par :*

$$\begin{array}{rccl} \mathrm{defrag}: & \mathrm{Ru} & \to & \mathrm{R} \\ & (r, r') & \mapsto & \{\mathrm{Unif}(l,l') / l(id) = l'(id), l \in r, l' \in r'\} \end{array}$$

Chiffrement et déchiffrement

Vu que pour l'instant on s'intéresse uniquement aux contenus des tables pour démontrer la correction sémantique des lois de composition, on ne parlera pas pour l'instant des éventuelles clefs de chiffrement et déchiffrement.

Définition 16 *On appelle* chiffrement *tout couple* $\mathtt{c}$ *de fonctions* $(\mathrm{Enc}, \mathrm{Dec})$ *de $\mathcal{V}$ dans $\mathcal{V}$ vérifiant* $\mathrm{Dec} \circ \mathrm{Enc} = \mathrm{id}$.

Pour toute valeur v de $\mathcal{V}$ on note $\mathtt{c}(v) = \mathrm{Enc}(v)$ *et* $\mathtt{c}^{-1}(v) = \mathrm{Dec}(v)$

Définition 17 *Pour une ligne l définie sur Δ, pour α un attribut, et pour* $\mathtt{c}$ *un chiffrement, on appelle* version de l chiffrée pour α avec le chiffrement $\mathtt{c}$ *la ligne notée* $\mathtt{c}(l)_\alpha$ *définie par :*

$$\begin{cases} \forall \beta \in \Delta \setminus \{\alpha\} & \mathtt{c}(l)_\alpha(\beta) = l(\beta) \\ & \mathtt{c}(l)_\alpha(\alpha) = \mathtt{c}(l(\alpha)) \quad si\ \alpha \in \Delta \end{cases}$$

De même, on définit la version de l déchiffrée pour α avec le chiffrement $\mathtt{c}$, *notée* $\mathtt{c}^{-1}(l)_\alpha$, *par :*

$$\begin{cases} \forall \beta \in \Delta \setminus \{\alpha\} & \mathtt{c}^{-1}(l)_\alpha(\beta) = l(\beta) \\ & \mathtt{c}^{-1}(l)_\alpha(\alpha) = \mathtt{c}^{-1}(l(\alpha)) \quad si\ \alpha \in \Delta \end{cases}$$

Définition 18 *Pour α un nom d'attribut et c un chiffrement, on appelle* fonction de chiffrement de α par c *la fonction*

$$\begin{array}{rccc} \text{crypt}_{\alpha,c}: & \text{R} & \to & \text{R} \\ & r & \mapsto & \{c(l)_\alpha / l \in r\} \end{array}$$

De même, on appelle fonction de déchiffrement de α par c *la fonction*

$$\begin{array}{rccc} \text{decrypt}_{\alpha,c}: & \text{R} & \to & \text{R} \\ & r & \mapsto & \{c^{-1}(l)_\alpha / l \in r\} \end{array}$$

Définition 19 *On dit d'un prédicat p et un chiffrement c sont compatibles pour l'attribut α s'il existe un autre prédicat $c_\alpha \Rightarrow p$ ne dépendant que de p, c et du nom d'attribut α tel que*

$$\forall l \in L, p(l) = (c_\alpha \Rightarrow p)(c(l)_\alpha)$$

Agrégation

Définition 20 *Pour δ un ensemble de noms d'attributs réguliers, on appelle* nom de groupe pour δ *toute application n définie de δ à valeurs dans $\mathcal{V}$.*

On remarque que tout nom de groupe est une ligne.

δ est appelé domaine *du nom de groupe n, et noté* $\text{dom}(n)$.

De plus, pour r une relation, on définit l'ensemble des noms de groupe de r pour δ *:*

$$r_\delta = \{l|_\delta / l \in r\}$$

Définition 21 *Pour r une relation et n un groupe, on appelle* groupe de r pour le nom n *l'ensemble des éléments de r coïncidant avec n sur* $\text{sch}(r) \cap \text{dom}(n)$. *On le note r_n.*

Autrement dit :

$$r_n = \{l \in r / l|_{\text{sch}(r)\cap\text{dom}(n)} = n|_{\text{sch}(r)\cap\text{dom}(n)}\}$$

De plus, on appelle identifiants du groupe r_n *l'ensemble des identifiants des lignes du groupe. On note* $\text{IDs}(r_n)$ *cet ensemble.*

Autrement dit :

$$\text{IDs}(r_n) = \{l(id) / l \in r_n\}$$

Définition 22 *On dira qu'une application f est* plus petite *qu'une application g si f est une restriction de g.*

On dira qu'un nom de groupe n_0 est minimal *pour une relation r donnée si c'est une plus petite application n pour laquelle le groupe de r pour n vaut r_{n_0}.*

Définition 23 *Pour r une relation, n un nom de groupe, et α un attribut de* $(\text{sch}(r) \setminus \text{dom}(r)) \cup \{id\}$, *on appelle* valeurs du groupe r_n pour l'attribut α *la fonction*

$$\begin{array}{rccc} r_n(\alpha): & \text{IDs}(r_n) & \to & \mathcal{V} \\ & l(id) & \mapsto & l(\alpha) \end{array}$$

Remarque : Souvent, on supposera que l'ensemble des identifiants possible est totalement ordonné et on s'en servira pour considérer des fonctions définies sur un ensemble d'identifiants (par exemple les $r_n(\alpha)$ définis ci-dessus) comme des listes.

On définit la longueur de telles listes comme le cardinal de leur ensemble de départ. Par exemple, la longueur de $r_n(\alpha)$ est $|r_n(\alpha)| = |\,\mathrm{IDs}(r_n)|$

Définition 24 *Pour r une relation, et n un nom de groupe, on appelle* ligne de groupe de r pour n *la ligne notée* $\mathrm{lg}_{r,n}$ *définie sur* $\mathrm{sch}(r) \cup \{id\}$ *par :*

$$\begin{cases} \mathrm{lg}_{r,n}(\alpha) = n(\alpha) & \text{si } \alpha \in \mathrm{sch}(r) \cap \mathrm{dom}(n) \\ \mathrm{lg}_{r,n}(\alpha) = r_n(\alpha) & \text{si } \alpha \in (\mathrm{sch}(r) \setminus \mathrm{dom}(n)) \\ \mathrm{lg}_{r,n}(id) = \gamma & \text{où } \gamma \text{ est un identifiant frais} \end{cases}$$

Définition 25 *Pour δ un ensemble de noms d'attributs, on appelle* fonction d'agrégation pour les attributs δ *la fonction suivante :*

$$\begin{array}{rccc} \mathrm{group}_\delta : & \mathrm{R} & \to & \mathrm{R} \\ & r & \mapsto & \{\mathrm{lg}_{r,n} \,/\, n \in r_\delta\} \end{array}$$

Réduction

La plupart du temps, les agrégations sont faites pour pouvoir faire une réduction ensuite.

On suppose que les identifiants des lignes peuvent être totalement ordonnés et donc que les fonctions définies sur des ensembles d'identifiants peuvent être vues comme des listes.

Pour toute liste l on notera $\mathrm{hd}(l)$ le premier élément de la liste, et $\mathrm{tl}(l)$ le reste de la liste.

Dans les définitions qui suivent, f est une fonction de $\mathcal{V}^2$ dans $\mathcal{V}$ et z est un élément de $\mathcal{V}$.

Définition 26 *On appelle* réduction d'une liste t par la fonction f avec l'élément neutre z *la valeur* $\mathrm{red}_{f,z}(t)$ *définie par induction sur la liste par :*

$$\begin{cases} \mathrm{red}_{f,z}(\emptyset) = z \\ \mathrm{red}_{f,z}(t) = \mathrm{red}_{f,f(z,hd(t))}(\mathrm{tl}(t)) \end{cases}$$

Si une valeur v de $\mathcal{V}$ n'est pas une liste, on la considère alors comme une liste à un seul élément et on pose donc $\mathrm{red}_{f,z}(v) = f(z, v)$.

Définition 27 *Pour l une ligne définie sur δ, et α un nom d'attribut régulier, on appelle* réduction de l'attribut α dans la ligne l par la fonction f avec l'élément neutre z *la ligne* $\mathrm{red}_{\alpha,f,z,l}$ *définie sur δ par :*

$$\begin{cases} \mathrm{red}_{\alpha,f,z,l}(\alpha) = \mathrm{red}_{f,z}(l(\alpha)) & \text{si } \alpha \in \delta \\ \mathrm{red}_{\alpha,f,z,l}(\beta) = l(\beta) & \text{si } \beta \neq \alpha \end{cases}$$

Définition 28 *On appelle* fonction de réduction de l'attribut α par la fonction f avec l'élément neutre z *la fonction suivante :*

$$\begin{cases} \mathrm{fold}_{\alpha,f,z} : & \mathrm{R} & \to & \mathrm{R} \\ & r & \mapsto & \{\mathrm{red}_{\alpha,f,z,l} \,/\, l \in r\} \end{cases}$$

Opérations ensemblistes : union, différence, fragmentation horizontale

Définition 29 *On dit que deux tables r et r' sont* défragmentables horizontalement *si elles ont même schéma relationnel e leurs ensembles d'identifiants sont disjoints. Autrement dit,*

$$\begin{cases} \text{sch}(r) = \text{sch}(r') \\ \{l(id)/l \in r\} \cap \{l(id)/l \in r'\} = \emptyset \end{cases}$$

On appelle union *ou* défragmentation horizontale *de deux tables r et r' défragmentables horizontalement la table $r \cup r'$, aussi notée* $\text{hdefrag}(r, r')$.

Définition 30 *On appelle* différence ensembliste *de deux tables r et r' ayant le même schéma relationnel la table $r \setminus r'$.*

Définition 31 *Pour p un prédicat, on appelle* fragmentation horizontale de critère *p la fonction*

$$\begin{array}{rccl} \text{hfrag}: & \text{R} \rightarrow & \text{R}^2 & \\ & r & \mapsto & (\{l \in r / p(l)\}, \{l \in r, \neg p(l)\}) \end{array}$$

On remarquera que les deux tables du résultat sont horizontalement défragmentables.