

# Modèles de données semi-structurées et contraintes d'inclusion

A.C. Caron, D. Debarbieux et Y. Roos  
Université des Sciences et Technologies de Lille  
Laboratoire d'Informatique Fondamentale de Lille, Bâtiment M3  
Campus Scientifique  
F-59655 Villeneuve d'Ascq Cedex  
{caronc,debarbie,yroos}@lifl.fr

*Résumé* On appelle donnée semi-structurée une donnée irrégulière, hétérogène, dont la structure, inconnue a priori, doit être définie en l'inférant a posteriori à partir de la donnée elle-même. Ici, une donnée semi-structurée sera modélisée par un graphe orienté étiqueté, avec une racine unique. L'exploration d'une telle donnée se fait à partir de la racine, en suivant les chemins, suites d'étiquettes des arcs du graphe. Plus généralement, une requête  $q$  sera une expression régulière de chemins dont le résultat sera l'ensemble des noeuds accessibles en suivant les chemins mots du langage  $q$ . A cette notion de requête est liée celle de contrainte d'inclusion. Informellement, une contrainte d'inclusion est une relation liant deux expressions régulières de chemins  $q$  et  $r$ , sur une donnée fixée, et elle peut s'exprimer de la manière suivante : "Pour cette donnée, le résultat de la requête  $q$  est inclus dans celui de la requête  $r$ ". Nous introduisons un nouveau modèle issu de l'apprentissage automatique et le comparons à des modèles connus dans le domaine des données semi-structurées, en insistant plus particulièrement sur les contraintes d'inclusion.

*Abstract* Semistructured data are irregular and heterogeneous. Their schema is unknown or missing and has to be deduced from data themselves. In this paper, semistructured data will be modeled by a rooted edge-labeled directed graph. For this model, query languages exploit the notion of path expression. A path expression is a sequence of edge labels whose result is the set of nodes reachable with this path. More generally, a regular path expression is a regular language of path expressions, whose result is the set of nodes reachable with these paths. Inclusion constraints is a binary relation over regular path expressions  $q$  and  $r$ , over a given data, and can be expressed as "for this data, answer of query  $q$  is included in answer of query  $r$ ". In this paper, we introduce a new model derived from machine learning framework, and compare it with known semistructured data models, and focus on inclusion constraints.

*Mots clefs* données semi-structurées, XML, typage, contraintes d'inclusion,

expressions régulières de chemins

*Keywords* semistructured data, regular path expressions, inclusion constraints, XML, typing

## 1 Introduction

Les données semi-structurées sont des données dont la structure, inconnue *a priori*, doit être définie en l'inférant *a posteriori* à partir de la donnée elle-même. Dans la mesure où la structure inférée peut tenir compte du type de traitement qu'on souhaite appliquer à la donnée, ceci peut être vu comme un avantage. Un exemple de données semi-structurées sont les documents XML(eXtensible Markup Language [XMLa]), ou plus largement un ensemble de documents et de liens permettant de passer de l'un à l'autre. Pour résumer, les données semi-structurées sont bien souvent complexes, hétérogènes et de formats variés. Elle ne s'expriment que très difficilement dans les modèles de bases de données relationnelles ou objets ; il est donc nécessaire de disposer de modèles adaptés à ces données.

Il y a différentes façons de modéliser ce type de donnée : une représentation arborescente convient particulièrement à la modélisation d'un document XML sans référence, mais ne sera pas l'objet de cet article. Ici, une donnée semi-structurée sera vue comme un graphe orienté étiqueté où les noeuds représentent les objets, simples ou complexes, composant la donnée et où les arcs représentent la composition de ces objets, par agrégation ou par association. Cette représentation est celle décrite dans [ABI 00].

Des langages de requêtes, basés sur des expressions régulières de chemins, permettent l'exploration de ces données, comme dans le projet Lore [ABI 97a], Strudel [FER 97] ou les langages de requêtes pour XML ([XMLb, DEU 99] ...). Dans ces langages, on utilise les étiquettes des arcs du graphe pour former des chemins et naviguer dans la donnée. Par exemple, le chemin `assuré.ayant-droit.voiture` dans la donnée décrite figure 1, permet d'atteindre les noeuds 2 et 8.

A cette représentation est liée la notion de *contrainte d'inclusion* introduite par Abiteboul et Vianu dans [ABI 97b]. Informellement, une contrainte d'inclusion est une relation liant deux expressions régulières de chemin  $q$  et  $r$ , sur une donnée fixée, et elle peut s'exprimer de la manière suivante : *pour cette donnée, le résultat de la requête  $q$  est inclus dans celui de la requête  $r$* . Outre l'optimisation de requêtes, ces contraintes d'inclusions permettent d'extraire des relations sémantiques présentes dans la donnée. Par exemple `assuré.voiture  $\subseteq$  compagnie` signifie que tout contrat d'assurance de voiture pris par un assuré est souscrit auprès d'une compagnie. C'est ce deuxième point qui nous intéresse particulièrement.

Nous voulons ici comparer plusieurs méthodes de typage des noeuds d'une donnée semi-structurée, un typage étant une application qui associe un type à un ensemble de noeuds, et permet de construire un graphe *schéma* de la

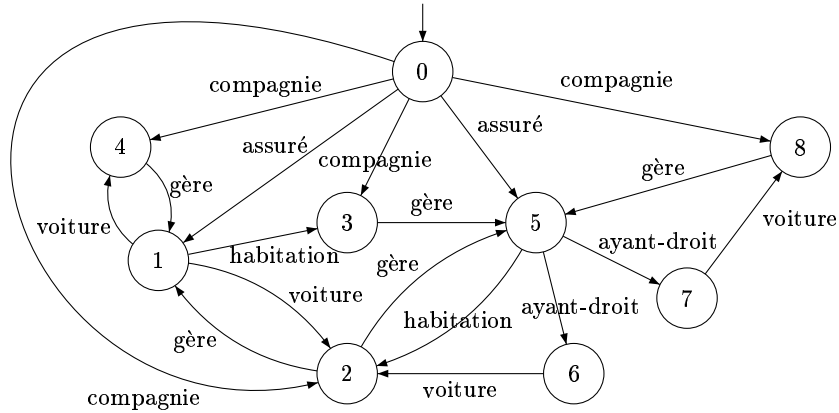


Figure 1: Une représentation de donnée semi-structurée.

donnée initiale, que nous appellerons *guide*. Connaître un schéma de la donnée est utile pour définir des requêtes. A ce titre, la connaissance des contraintes d'inclusion est une aide à la formulation de ces requêtes.

Pour la comparaison des guides, nous examinerons les points principaux suivants :

- préservation des chemins présents dans la donnée initiale,
- préservation des contraintes d'inclusions présentes dans la donnée initiale,
- taille (nombre de noeuds) du guide par rapport à la donnée initiale.

La première méthode étudiée est celle utilisée dans le système Lore pour construire un *dataguide* [GOL 97] à partir d'une donnée semi-structurée. La seconde, définie dans [NES 98], type chaque noeud de la donnée en fonction de sa signature (quels chemins peut-on prendre à partir du noeud) et de son contexte (de quelle manière arrive-t-on sur ce noeud). Pour terminer, nous utiliserons une méthode issue du domaine de l'apprentissage de langages rationnels [DEN 01], adaptée aux données semi-structurées, afin de définir des guides appelés MAM. Notons que ces derniers n'ont pas été définis dans le but de calculer efficacement des requêtes (comme c'était le cas pour les dataguides par exemple). En effet, dans ces guides peuvent apparaître des contraintes d'inclusion qui n'étaient pas présentes dans la donnée initiale. Ces dernières permettent en revanche d'inférer des relations de sous-typage sur les noeuds.

## 2 Les dataguides

Les dataguides ont été définis dans le cadre du projet Lore [Lor], dont l'objectif était de définir un système de gestion de données semi-structurées. Avant d'introduire ces guides, nous allons poser quelques définitions élémentaires.

**Définition 1** Soit  $S$  une donnée semi-structurée de racine  $r$ . Soit  $\Sigma$  l'alphabet des étiquettes des arcs de  $S$ .

- Un chemin est un mot obtenu par concaténation des lettres de  $\Sigma$ .
- Soit  $p = l_1 \dots l_n$  un chemin. Le noeud  $o$  est accessible par  $p$  s'il existe des noeuds  $o_1, \dots, o_{n-1}$  tels que  $(r, l_1, o_1), \dots, (o_{n-1}, l_n, o)$  sont des arcs de  $S$ .
- L'ensemble cible d'un chemin  $p$  est l'ensemble des noeuds accessibles par  $p$ .
- Une requête est un langage régulier de chemins.
- Le résultat de la requête  $q$  est l'union des ensembles cibles des chemins de  $q$ .
- Une contrainte d'inclusion  $p \subseteq q$  où  $p$  est une expression rationnelle et  $q$  est un mot signifie que le résultat de la requête  $p$  est inclus dans le résultat de la requête  $q$ .

**Exemple 1** Sur la donnée de la figure 1, l'ensemble cible de `assuré.voiture` est  $\{2,4\}$ , l'ensemble cible de `assuré.ayant-droit.voiture` est  $\{2,8\}$ . Ainsi le résultat de la requête `assuré.(ayant-droit.voiture + voiture)` est  $\{2,4,8\}$ .

R. Goldman et J. Widom ont défini dans [GOL 97] un guide appelé dataguide dont la définition est la suivante :

**Définition 2** Soit  $S$  une donnée semi-structurée. On appelle dataguide de  $S$  tout graphe  $D$  qui vérifie :

- Chaque chemin de  $S$  a exactement une instance dans  $D$ .
- Chaque chemin de  $D$  est un chemin de  $S$ .
- Deux chemins différents  $l$  et  $l'$  ont le même ensemble cible dans la source si et seulement si ils ont la même cible dans le dataguide.

Cette définition nous garantit que le dataguide a exactement le même ensemble de chemins que la donnée initiale.

A partir de cette définition, nous avons montré la proposition suivante :

**Proposition 1** Le dataguide d'une donnée  $S$  est unique (à un isomorphisme près) et il est le résultat de l'application sur la source  $S$  de l'algorithme classique de déterminisation des automates finis de mots [HOP 79]

**Corollaire 1** Il en découle que ce dataguide peut avoir une taille exponentielle par rapport à la taille de la source.

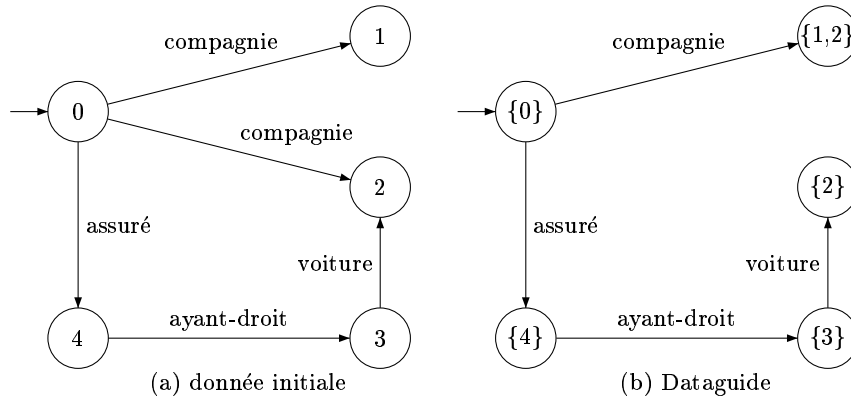


Figure 2: Dataguide.

La figure 2 montre un exemple de donnée semi-structurée et de dataguide associé. On peut remarquer que la contrainte `assuré.ayant-droit.voiture`  $\subseteq$  `compagnie` n'est plus vérifiée dans le dataguide et ne peut être retrouvée qu'à travers une inclusion d'ensembles  $\{2\} \subseteq \{2,3\}$ .

Ainsi les dataguides ne satisfont pas notre critère principal : la préservation des contraintes d'inclusion. Par ailleurs, la taille d'un dataguide peut être exponentielle (dans le pire des cas) par rapport à la taille de la donnée initiale. Pour réduire la taille, R. Goldman et J. Widom proposent dans [GOL 99] plusieurs méthodes permettant de regrouper des états du dataguide. Malheureusement, ces nouveaux guides ne conservent pas forcément l'ensemble des chemins.

### 3 Les guides perfects

Dans [NES 98] les auteurs proposent de typer chaque noeud de la donnée initiale en ayant comme contrainte importante de contrôler le nombre de types. Ce typage prend en compte non seulement les chemins entrants (comme le dataguide) mais aussi les chemins sortants. Le guide obtenu à partir de ce typage sera appelé guide perfect.

Pour construire le guide perfect il faut regrouper dans la donnée initiale des noeuds qui ont les mêmes chemins entrants et sortants. L'algorithme, dont nous donnons figure 3 le pseudo-code, construit une relation d'équivalence  $\mathcal{R}_S$  sur l'ensemble des noeuds d'une donnée  $S$ . De manière classique, cet algorithme de construction utilise une méthode de point fixe pour calculer les classes d'équivalence:

A partir de la relation  $\mathcal{R}_S$ , on peut définir un graphe  $P$  appelé guide perfect de  $S$  dont les noeuds sont des classes d'équivalence de  $\mathcal{R}_S$  et dont les arcs sont tels que:  $(T, \alpha, T')$  arc de  $P$  si et seulement si  $\exists n \in T, \exists n' \in T', (n, \alpha, n')$  arc de  $S$ .

**Remarque 1** *Deux noeuds ayant le même type ont les mêmes chemins entrants et sortants. Le réciproque n'est pas toujours vraie.*

```

construire un type par noeud
typer tous les noeuds avec tous les types
tant que le point fixe n'est pas atteint faire
  pour tous les noeuds N de la donnée faire
    pour tous les types T possibles de N faire
      pour toutes les flèches sortantes  $(T, \alpha, T')$  faire
        si  $\exists N'$  tq  $(N, \alpha, N')$  soit un arc de la donnée et que T' soit un type de N'
          alors N n'est pas typable par T
        fin si
      fin pour
    pour toutes les flèches entrantes  $(T', \alpha, T)$  faire
      si  $\exists N'$  tq  $(N', \alpha, N)$  soit un arc de la donnée et que T' soit un type de N'
        alors N n'est pas typable par T
      fin si
    fin pour
  fin pour
fin pour
Regrouper les types qui ont le même ensemble de noeuds dans une même classe d'équivalence.

```

Figure 3: Algorithme de construction du guide perfect

**Exemple 2** *La figure 4 montre une donnée initiale et le guide perfect associé.*

Nous pouvons désormais évaluer le guide perfect par rapport à nos objectifs.

**Proposition 2**

- *Le guide perfect a le même ensemble de chemins que la donnée initiale.*
- *On peut borner la taille du guide perfect par la taille de la donnée initiale.*
- *Les contraintes d'inclusion sont préservées.*

Ces résultats correspondent aux objectifs fixés mais il faut les nuancer: le guide perfect ne va pas, dans la majorité des cas, être beaucoup plus petit que la donnée initiale. En effet toute irrégularité dans la donnée va être source de nouveaux chemins et donc de nouveaux noeuds dans le guide. Le guide perfect va donc avoir un bon comportement sur les données régulières qui sont plus du domaine des bases de données classiques que des données semi-structurées.

Dans [NES 98], le guide perfect est en réalité utilisé pour initialiser un algorithme de K-clustering qui va typer le document semi-structuré avec exactement K types. Le guide ainsi obtenu est une approximation du guide perfect et ne préserve plus les chemins.

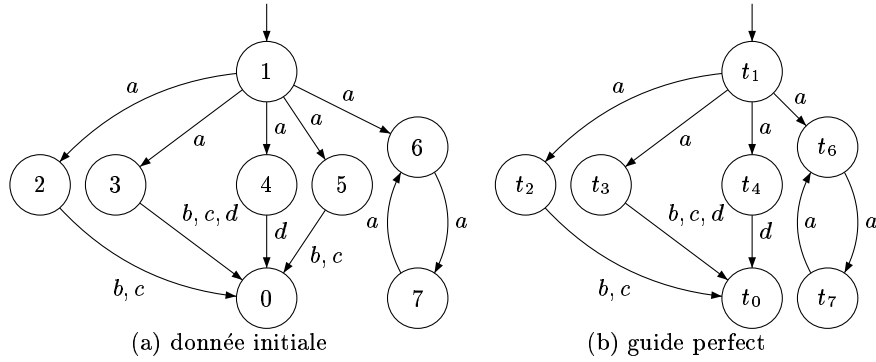


Figure 4: Guide perfect.

## 4 AFER - MAM

Dans cette partie, nous présentons un guide basé sur les travaux de Denis, Lemay et Terlutte qui ont introduit, dans [DEN 01], la notion d'*automate fini à résiduels premiers*, dans le cadre de l'apprentissage automatique. Ces automates peuvent être vus comme des automates non déterministes canoniques. Ils ont de plus la propriété d'être plus petits que l'automate minimal déterministe reconnaissant le même langage.

Nous avons utilisé ces automates pour construire un guide qui conserve les contraintes d'inclusion, et qui est souvent plus petit que le guide *perfect*.

Nous allons, dans cette section, utiliser la terminologie classique de la théorie des langages ([HOP 79]), et considérer notre donnée semi-structurée comme un automate de mots. Celui-ci a un unique état initial, tous ses états sont terminaux. Ainsi, un chemin de la donnée semi-structurée est un mot reconnu par l'automate associé.

Nous allons donner une série de définitions qui vont permettre d'introduire l'AFER (Automate Fini à Etats Résiduels) canonique associé à une donnée semi-structurée.

**Définition 3** *Étant donné un langage  $L$  et un mot  $u$  construit sur le même alphabet  $\Sigma$ , le résiduel de  $L$  par rapport à  $u$ , noté  $u^{-1}L$ , est le langage*

$$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$$

**Définition 4** *Soit  $A = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$ , un automate. Pour tout état  $q$  de  $Q$ , on définit  $\text{post}(q)$  par :*

$$\text{post}(q) = \{v \in \Sigma^* \mid \delta(q, v) \cap Q_f \neq \emptyset\}$$

**Définition 5** Un automate  $A = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$ , reconnaissant un langage  $L$  est un AFER s'il est un automate (non déterministe) qui vérifie

$$\forall q \in Q \exists u \in \Sigma (\text{post}(q) = u^{-1}L \wedge u^{-1}L \neq \emptyset)$$

**Définition 6** Un langage résiduel  $u^{-1}L$  est premier s'il n'est pas égal à l'union des langages résiduels qu'il contient strictement.

On peut, maintenant, définir l'AFER canonique associé à un langage, automate dont chaque état est associé à un résiduel premier du langage.

**Définition 7** Étant donné un langage régulier  $L$ , l'AFER canonique associé à  $L$  est l'automate  $A = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$  défini par :

- $\Sigma$  est l'alphabet associé à  $L$ ;
- $Q$  ensemble des états.  $Q = \{u^{-1}L \mid u^{-1}L \text{ est premier}\}$ ;
- $Q_0$  ensemble des états initiaux.  $Q_0 = \{u^{-1}L \in Q \mid u^{-1}L \subseteq L\}$ ;
- $Q_f$  ensemble des états finaux.  $Q_f = \{u^{-1}L \in Q \mid \varepsilon \in u^{-1}L\}$ ;
- $\delta$ , fonction de transition, est définie par :  $\delta(u^{-1}L, x) = \{v^{-1}L \in Q \mid v^{-1}L \subseteq (ux)^{-1}L\}$ .

L'AFER canonique associé à une donnée semi-structurée est en fait l'AFER canonique associé au langage des chemins de cette donnée; il a donc exactement le même ensemble de chemins que la donnée initiale.

De plus, l'AFER canonique a de bonnes propriétés de taille comme le montre la proposition suivante :

**Proposition 3** ([DEN 01]) Soit  $L$  un langage et  $R$  un AFER qui reconnaît  $L$ . Si  $R$  n'est pas isomorphe à l'AFER canonique  $C$  qui reconnaît  $L$ , alors  $R$  a strictement plus d'états que  $C$  ou  $R$  a strictement moins de transitions que  $C$ .

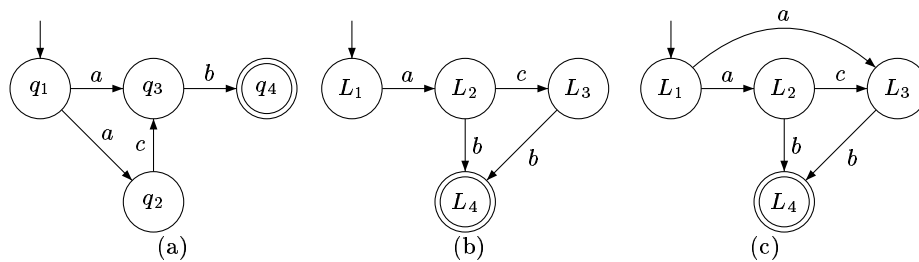


Figure 5: Où est l'AFER canonique?

**Exemple 3** La figure 5 présente trois automates qui reconnaissent le même langage  $L = ab + acb$ . Le premier (a) n'est pas un AFER car l'état  $q_2$  n'est pas un résiduel de  $L$ . En effet  $\text{post}(q_2) = cb$  qui n'est pas un résiduel de  $L$ . Le deuxième (b) est un AFER reconnaissant  $L$ , mais il n'est pas canonique car il n'est pas maximum en regard du nombre de transitions (on peut ajouter  $\delta(L_1, a) = L_3$  sans modifier le langage reconnu par l'automate). Enfin l'automate (c) est l'AFER canonique associé à  $L$ .

Nous ne présenterons pas ici de mode de construction de l'AFER canonique (on pourra se référer à [DEN 01]). On signalera juste que l'une des méthodes est de construire l'automate déterministe minimal et de la réduire. Le coût de la construction d'un AFER canonique est donc potentiellement exponentiel. Comme le montre la figure 6, l'AFER canonique associé à une donnée ne conserve pas les contraintes d'inclusion : la donnée initiale (a) vérifie la contrainte d'inclusion  $g \subseteq ea$ , ce qui n'est pas le cas de son AFER canonique (b).

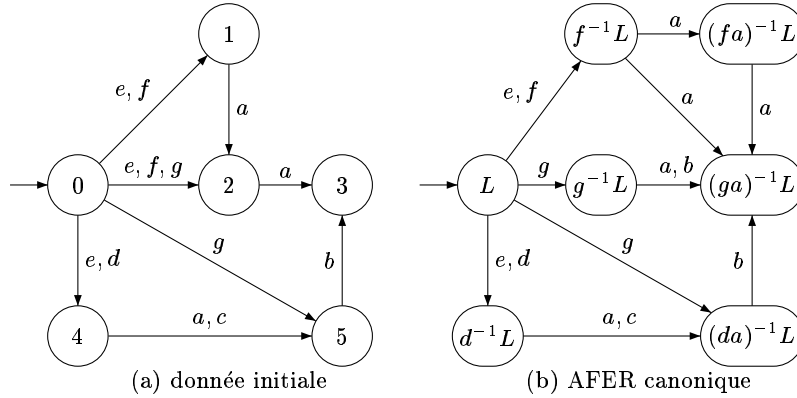


Figure 6: Une donnée et son AFER canonique, ayant pour ensemble de chemins  $L = eaa + ea + eab + ecb + dab + dcb + faa + fa + ga + gb$ .

L'AFER canonique de correspond pas à nos critères. En revanche, c'est le cas pour le miroir de l'AFER canonique associé au miroir du langage de la donnée initiale. C'est ce dernier que nous prendrons comme guide de la donnée.

**Définition 8** Soit un alphabet  $\Sigma$ .

- Soit  $u = x_1x_2 \dots x_n$  un mot de  $\Sigma^*$ . Le miroir de  $u$  est le mot  $u^R = x_n \dots x_2x_1$ .
- Soit  $L$  un langage sur  $\Sigma$ . Le miroir de  $L$  est le langage  $L^R = \{u^R \mid u \in L\}$ .
- Soit un automate  $A = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$ . Le miroir de  $A$  est l'automate  $A^R = \langle \Sigma, Q, Q_f, Q_0, \delta^R \rangle$ , où  $\delta^R = \{(q, x, q') \mid (q', x, q) \in \delta\}$ .

Un résultat classique lie ces trois définitions :

**Proposition 4** *Soit un automate  $A = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$ . Si  $A$  reconnaît le langage  $L$ , alors  $A^R$  reconnaît le langage  $L^R$ .*

**Définition 9** *Soit  $L$ , un langage rationnel. Le MAM associé à  $L$  est le miroir de l'AFER canonique associé au langage miroir de  $L$ .*

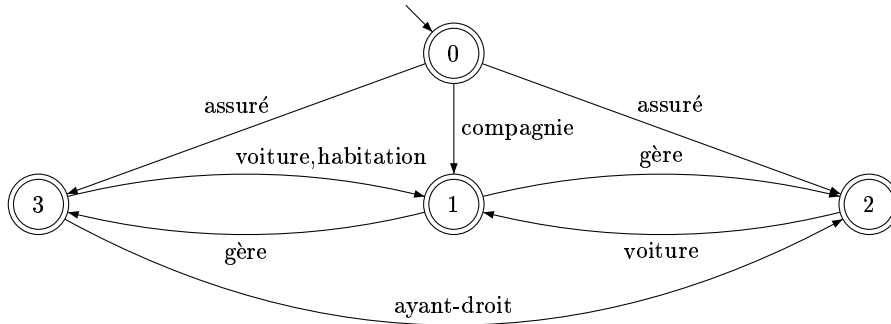


Figure 7: Le MAM associé à la donnée de la figure 1.

On va maintenant évaluer le guide MAM par rapport à nos objectifs. Deux résultats sont directement issus de l'étude des AFER : le guide MAM peut avoir une taille exponentielle par rapport à la donnée initiale et il a exactement le même ensemble de chemins que celle-ci.

Concernant la préservation des contraintes d'inclusion, nous avons montré la proposition suivante :

**Proposition 5** *Soit  $S$ , une donnée semi-structurée. Toute contrainte d'inclusion  $u \subseteq v$  présente dans  $S$  est aussi présente dans le guide MAM associé à  $S$ .*

Ce résultat est une conséquence de la proposition suivante :

**Proposition 6** *Pour tout langage  $L$ , pour tous mots  $u$  et  $v$ , la contrainte d'inclusion  $u \subseteq v$  est présente dans le MAM associé à  $L$  si et seulement si  $u^{-1}L \subseteq v^{-1}L$ .*

La proposition 6 permet de caractériser les contraintes d'inclusion présentes dans le guide MAM. Remarquons que, contrairement aux autres guides présentés ici, le guide MAM contient des contraintes qui n'étaient pas dans le document initial. On peut se poser la question du sens et de l'utilité de ces contraintes. Certaines de ces contraintes sont issues de la clôture transitive des contraintes existantes et d'autres sont nouvelles. Contrairement aux autres guides, le guide MAM ne peut pas être utilisé pour calculer des requêtes. En effet, le résultat d'une requête sur le guide MAM peut donner un sur-ensemble du résultat de cette

même requête sur le document initial. En revanche, ces nouvelles contraintes signifient que des noeuds de la donnée ont des points communs : toutes les requêtes que l'on peut faire à partir de l'un peuvent être faites à partir de l'autre. Par exemple la contrainte `assuré.ayant-droit`  $\subseteq$  `assuré` présente dans la figure 7 est nouvelle par rapport à la donnée initiale figure 1 et peut être interprétée comme *un ayant droit est une spécialisation d'un assuré*. Ces informations peuvent être exploitées par un langage typé de manipulation de données semi-structurées (comme CDuce [BEN 02]).

Du point de vue de la complexité des algorithmes de construction des trois guides étudiés, on peut établir que, dans le pire des cas, la construction du dataguide et du MAM sont en  $\mathcal{O}(2^n)$  et celle du guide perfect est en  $\mathcal{O}(n^3)$ ,  $n$  étant le nombre de noeuds de la donnée initiale.

Par ailleurs, nous connaissons des majorations de taille pour ces trois guides. Il est à noter qu'il est possible de construire un exemple pour lequel le dataguide, l'AFER canonique et le MAM ont tous une taille exponentielle par rapport à la donnée initiale. Cependant, il nous semblait que la taille obtenue à partir des données réelles pouvait être très inférieur à cette majoration, en particulier dans le cas du dataguide et du MAM. C'est pourquoi nous avons implémenté ces trois algorithmes et nous avons étudié la taille des guides produits. A titre indicatif, voici trois exemples de données semi-structurées et la taille des guides associés. Ces trois exemples, qui sont à compléter par une étude sur des données plus nombreuses et de plus grande taille, laissent à penser que la taille de notre guide MAM est raisonnable.

Type de données	Taille initiale	Dataguide	Perfect	MAM
Fichier de configuration	180	7	42	8
Site web	1288	315	526	124
Donnée génomique	5284	138	585	119

## 5 Conclusion

Dans cet article, nous avons étudié différents modèles de données semi-structurées, sous la forme de graphes appelés *guides*. Les critères de comparaison que nous avons choisis sont d'une part la préservation des chemins, d'autre part la conservation des contraintes d'inclusion, et enfin la taille du guide par rapport à celle de la donnée initiale. Les deux premiers guides étudiés proviennent du domaine des données semi-structurées, et le troisième est une adaptation d'un modèle utilisé en apprentissage de langages rationnels.

La section précédente permet de dire que le dataguide et le MAM n'ont pas de si mauvaises propriétés de taille. En effet, nous avons constaté sur quelques exemples que la taille de ces guides était bien inférieure à l'exponentielle de la taille de la donnée initiale, et inférieure à celle du guide perfect.

Le dataguide ne préserve pas les contraintes d'inclusion, contrairement au MAM. Ce dernier ne permet pas de calculer le résultat d'une requête. En revanche,

il permet d'inférer des relations sémantiques, sous la forme de nouvelles contraintes d'inclusion, à partir des informations structurelles fournies par la donnée.

## References

- [ABI 97a] ABITEBOUL S., QUASS D., MCHUGH J., WIDOM J. WIENER J., The Lorel Query Language for Semistructured Data, *Journal of Digital Libraries*, vol. 1, n° 1, 1997, p. 68–88.
- [ABI 97b] ABITEBOUL S. VIANU V., Regular Path Queries with Constraints, *Proc. of ACM Symposium on Principles of Database Systems*, 1997.
- [ABI 00] ABITEBOUL S., BUNEMAN P. SUCIU D., *Data on the Web*, Morgan Kaufmann Publishers, 2000.
- [BEN 02] BENZAKEN V., CASTAGNA G. FRISCH A., CDuce: a white paper, presented at the workshop PLAN-X: Programming Languages Technologies for XML, 2002.
- [DEN 01] DENIS F., LEMAY A. TERLUTTE A., Residual Finite State Automata, *Lecture Notes in Computer Science 2010*, 18th Annual Symposium on Theoretical Aspects of Computer Science, 2001, p. 144–157.
- [DEU 99] DEUTSCH A., FERNANDEZ M., FLORESCU D., LEVY A. SUCIU D., A Query Language for XML, *Proc. of World Wide Web Conference*, Toronto, 1999.
- [FER 97] FERNANDEZ M., FLORESCU D., LEVY A. SUCIU D., A Query Language for a Web-site Management System, *SIGMOD Record*, vol. 3, n° 26, 1997, p. 4–11.
- [GOL 97] GOLDMAN R. WIDOM J., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *Proc. of International Conference on Very Large Data Bases*, 1997, p. 436–445.
- [GOL 99] GOLDMAN R. WIDOM J., Approximate DataGuides, *Proc. of the Workshop on Query Processing for Semistructured Data and non-Standard Data Formats*, Jerusalem, Israel, 1999.
- [HOP 79] HOPCROFT J. ULLMAN J., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.
- [Lor] Lore, <http://www-db.stanford.edu/lore/>.
- [NES 98] NESTOROV S., ABITEBOUL S. MOTWANI R., Extracting Schema from Semistructured Data, *Proc. of ACM SIGMOD International Conference on Management of Data*, June 1998.
- [XMLa] The W3C's XML web pages, <http://www.w3.org/XML>.
- [XMLb] XML Query web pages, <http://www.w3.org/XML/Query>.