Laboratoire de Recherche en Informatique
Université Paris-Sud XI : UMR 8623
Numéro d'ordre : 9119

Thèse en vue de l'obtention du diplôme de docteur en Informatique

Transformations de Mots, d'Arbres et de Statistiques.

Adrien Vieilleribière

Cette thèse a été soutenue le 22 Septembre 2008.
Le jury était constitué de

Nicole Bidoit	Université Paris-Sud	Présidente
Dominique Laurent	Université Cergy Pontoise	Rapporteur
Joachim Niehren	INRIA, Lille Nord Europe, Projet Mostrare	Rapporteur
Yuzuru Tanaka	Hokkaido University, Faculty of Engineering	Examinateur
Michel de Rougemont	Université Paris II	Directeur de thèse

Remerciements

Je tiens à remercier en premier lieu mon directeur de thèse, Michel de Rougemont qui a su encadrer ma recherche avec la rigueur nécessaire, tout en me laissant une grande liberté de travail. L'excellence du niveau scientifique et l'abondance des questions passionnantes dont il a pu me faire profiter ont largement contribué aux plaisirs de ce travail.

Je remercie sincèrement Nicole Biboit, Joachim Niehren et Dominique Laurent. Tous trois se sont acquittés de la lourde tâche de lecture du présent manuscrit. Ils l'ont effectué avec une diligence que j'apprécie profondément. Yuzuru Tanaka m'a fait l'honneur de participer à mon jury, qu'il en soit ici remercié.

Je remercie mes collègues du LRI, particulièrement ceux de l'équipe Algorithmique et Complexité, aux côtés desquels j'ai passé ces années de thèse. Je tiens à remercier les initiateurs de mon goût de l'approximation et membres des ACI Vera et Verap ; particulièrement Fréderic Magniez, Marie-Claude Gaudel, Richard Lassaigne et Sylvain Peyronnet. Je remercie également les collègues associés au projet Visir ; spécialement Tsuyoshi Sugibuchi et Nicolas Spyratos. Merci à David Forge pour son initiation aux matroïdes et la sympathique collaboration qui en a découlée.

Je remercie chaleureusement mes amis, Kim, Pierre-Alain, Samy, Philipe, Marc & Sophie, Denis, Claudia, Abdelfattah, Bertrand, Valentin, Ekaterina, mon compère homonyme, Thomas, Nicolas, Robin, Sandra, Aurélie ...

Finalement cette thèse ne serait pas sans le soutien des très proches, ma fantastique Aude qui me supporte en toutes circonstances, mon infatigable relecteur de père, ma mère attentionnée et ma sœur. À tous, un grand grand merci, vous m'êtes chers.

Introduction

L'échange de données est devenu, depuis le développement de l'internet, un phénomène incontournable. Des données de tous types, textes dans différentes langues, images, vidéos, s'échangent à chaque instant. Afin d'être analysées, ces données sont souvent structurées à l'aide de balises, ce qui explique le rôle central du modèle XML en base de données. Normaliser un langage commun pour décrire des transformations de fichiers XML est donc nécessaire, et le *W*orld Wide Web Consortium[1] promeut le langage XSLT pour cet usage. L'aspect physique de l'échange sous-entend également des erreurs potentielles, qui naissent de la diversité des matériels, logiciels, et communications. Il est donc essentiel de proposer des traitements robustes aux erreurs.

Permettre des échanges de données opérationnels et efficaces est l'objectif de cette thèse, qui va ainsi développer des techniques d'approximation afin de traiter des données pouvant contenir des erreurs. L'intérêt à long terme est d'étendre les résultats théoriques de la thèse à des fichiers XML et des programmes XSLT. Naturellement, on s'intéresse alors aux arbres étiquetés, comme une simplification de la structure d'un fichier XML, et aux mots comme simplification des arbres. D'un point de vue théorique, un programme XSLT correspond à un transducteur, et une DTD à un langage d'arbres. En pratique, ces objets sont essentiels en échange de données puisqu'ils permettent une interopérabilité indispensable. Trois propriétés sont étudiées en particulier : l'équivalence de transducteurs, ainsi que la consistance et la sûreté de triplets schéma-source, transducteur, schéma-cible. Deux transducteurs sont équivalents si tout couple entrée-sortie accepté par l'un est aussi accepté par l'autre. La consistance garantit q'il est possible de transformer une instance-source dans le schéma-cible avec le transducteur, et permet donc de s'assurer que cela a un sens d'utiliser ce transducteur. La sûreté garantit la pertinence de la transformation en s'assurant que toutes les instances-sources sont transformées dans le schéma-cible. Vérifier ces propriétés est un enjeu à la fois fondamental et industriel. Une des applications est notamment de vérifier la sûreté d'un programme, ou sa consistance avec des spécifications.

Pourquoi vouloir traiter ces questions de manière approchée ? Le cas exact de l'équivalence est étudié depuis le milieu des années 90, et n'a pas trouvé de solutions satisfaisantes. En général, l'équivalence de transducteurs est indécidable même sur des mots. Toutefois, la décidabilité est retrouvée avec des hypothèses restrictives sur les transducteurs, comme le déterminisme ou la linéarité. Malgré tout, les preuves restent non

[1] W3C : organisme de promotion de la compatibilité des technologies de l'internet http://www.w3.org/

constructives ou de complexités élevées. La sûreté est actuellement l'objet d'études intensives [Martens et Neven, 2004, Martens et al., 2008], et est typiquement DEXPTIME dur pour de nombreux cas. Dans cette thèse, l'efficacité de l'approximation résulte de deux aspects. Dans un premier temps, on calcule en temps constant la statistique d'une instance, l'idée étant de baser l'algorithmique sur un échantillonnage des données, avec l'avantage de s'affranchir des erreurs. Dans un second temps, une représentation statistique des transducteurs est construite inductivement, et on démontre que cette représentation est robuste.

Pour les aspects théoriques, le point de départ de cette thèse est la publication de Elgar Fischer, Frédéric Magniez et Michel de Rougemont (LICS 06). Les statistiques ustat ont tout d'abord été introduites dans cette publication pour des mots et des arbres à arité bornée. Ce concept est étendu ici suivant deux axes principaux : la généralisation aux arbres à arité non bornée et aux transducteurs. Son intérêt majeur est de pouvoir relier la distance d'édition avec déplacements entre deux objets à la distance géométrique (la norme $\mathbf{L}_1$) entre leurs représentations.

Cette thèse montre que l'équivalence approchée des machines à états finis (FSM) non déterministes est décidable. Une borne exponentielle est obtenue, et améliorée en temps polynomial pour des transducteurs particuliers. Le cas exact non déterministe est peu abordé dans la littérature, et ne l'est jamais de façon effective : on trouve seulement une borne de dureté (PTIME dur).

Pour une machine à états finis non déterministe, on prouve que la consistance est décidée en temps constant dans sa version approchée (linéaire dans le cas exact). On montre également que la sûreté est décidable en temps polynomial, contrairement au cas exact où la décision est un problème PSPACE-complet.

Un outil d'enseignement a été implémenté pour modéliser graphiquement les notions abordées dans la thèse : FSM, XSLT, automates, consistance, sûreté... Cet outil a été exploité afin de fournir des exemples d'applications d'échange de données. Ainsi, des transducteurs particuliers permettent de visualiser des logs (trafic sur un site Internet) et des réponses à des requêtes OLAP.

Structure de cette thèse

Dans un premier temps, les définitions des concepts abordés dans la thèse sont explicités. Les statistiques ustat de mots et d'arbres, et les résultats associés sur les distances sont développés dans un second chapitre. Ensuite, nous étudions plus précisément l'équivalence des transducteurs, pour arriver dans un quatrième chapitre à l'échange de données à proprement parler, c'est-à-dire aux notions de consistance et de sûreté. Enfin, l'outil de visualisation implémenté pour cette thèse et ses applications sont présentés.

Chapitre 1 : Définitions Préliminaires

Les classes de langages considérées dans cette thèse sont celles des langages réguliers de mots et d'arbres dont une caractérisation est rappelée au travers de la définition d'automate. Les complexités des opérations classiques sur de tels automates (Vide, Appartenance, Intersection et Inclusion) sont ensuite précisées.

La notion de distance d'édition avec déplacements est ensuite présentée, d'abord pour les mots, puis dans le cadre des arbres étiquetés. Les testeurs présentés dans cette thèse sont construits pour cette distance particulière.

Nous rappelons finalement ce qu'est un testeur dans un cadre général : pour une classe de structures munie d'une distance, un testeur est un algorithme qui sait séparer les instances équivalentes des instances « loin d'être équivalentes » pour cette distance.

Chapitre 2 : Statistiques — Mots, Arbres, Automates

Ce chapitre définit la statistique d'une instance et d'un langage en décomposant cette définition par étapes. Ces objets statistiques, précédemment introduits par [Fischer et al., 2006] pour les langages de mots (et d'arbres à arité bornée) sont étendus dans cette thèse aux arbres à arité non bornée et aux transducteurs linéaires.

Un Mot est représenté par un vecteur statistique qui décrit les fréquences d'apparition de ses sous-mots d'une longueur donnée. Le calcul de ce vecteur est alors linéaire en la taille du mot et peut être approché en temps constant par échantillonnage. Réciproquement, étant donné un vecteur, il est possible de décider de l'existence d'un mot dont la statistique est ce vecteur, ou d'une suite de mots dont la statistique asymptotique converge vers ce vecteur. Cette question, déjà traitée pour une longueur fixée, est ici nouvellement résolue dans le cas asymptotique, en montrant qu'une statistique est asymptotiquement inversible si et seulement si un certain système d'inéquations est satisfaisable. Un double lien existe entre distance et norme entre statistiques puisque [Fischer et al., 2006] montre que (1) si deux mots sont proches, leurs statistiques sont proches ; (2) si deux mots de même longueur ont des statistiques proches, alors ces mots sont proches.

Un arbre est représenté par sa matrice statistique. Elle décrit les fréquences des chemins (et des étiquettes) dans l'encodage de Rabin de l'arbre. Cette matrice est calculable en temps linéaire, et il est également possible de l'approximer par échantillonnage. On montre alors que la distance entre arbres est à nouveau reflétée par la distance géométrique entre matrices. Quelques résultats d'inversion sont ensuite étendus au cas des arbres.

Un langage vu dans l'espace statistique peut être approximé par une union de polytopes. Cette approximation peut être calculée de manière inductive, en décrivant des statistiques de boucles, puis en les combinant linéairement lorsqu'elles sont compatibles. Nous montrons que ce calcul peut être réalisé en temps polynomial. Précédemment obtenu pour les statistiques par bloc dans [Fischer et al., 2006], ce calcul est ici adapté pour les statistiques uniformes.

Chapitre 3 : Transducteurs — Statistiques, Distances, Équivalence

Dans ce chapitre, nous montrons comment vérifier l'équivalence approchée de transducteurs linéaires, d'abord pour des transducteurs de mots, puis dans le cas des arbres. L'idée principale est de généraliser l'approximation proposée dans le chapitre précédent, en montrant que cette approximation reflète encore (géométriquement) la distance d'édition avec déplacements entre transducteurs :

- Si deux transducteurs sont proches, alors leurs représentations sont proches.
- Si deux transducteurs sont loin, alors leurs représentations sont loin.

Nous étudions les XSL-Transducteurs, une notion de transduction d'arbres proposée par [Martens et Neven, 2004]. Nous généralisons ainsi les machines à états finis, en se basant sur les fondements d'un langage de type XSLT, *i.e.* adapté à la transformation d'arbres (de la racine aux feuilles).

Nous étendons la représentation géométrique proposée pour les automates au cas des XSL-Transducteurs linéaires. Les statistiques d'un XSL-Transducteur sont alors approximées en décrivant les statistiques de ses boucles, puis en les combinant lorsqu'elles sont compatibles. Nous modélisons ainsi la représentation statistique d'un transducteur par un système d'inéquations, puis nous confirmons que cette nouvelle représentation permet de décider l'équivalence approchée.

L'équivalence approchée peut en effet être décidée par une inclusion approchée des représentations et nous démontrons que cette approche est à la foi sûre (répond oui pour des transducteurs équivalents) et robuste (répond non pour des transducteurs loin). L'idée centrale est de comprendre comment une erreur ε sur les transducteurs se traduit par une erreur ε' sur leurs représentations.

Chapitre 4 : Échange Approché de Données

Dans ce chapitre, nous considérons un transducteur et des schémas, et nous nous intéressons à la consistance (est-ce qu'il existe une instance-cible valide ?), et à la sûreté (est-ce que toutes les instances-cibles sont valides ?). Pour les langages réguliers de mots et les machines à états finis, on montre que la consistance est linéaire, alors que sa version approchée peut être testée en temps constant. La complexité de ce test ne dépend que de ε, des schémas, et du transducteur, mais elle est indépendante de la taille du mot d'entrée considéré. Le traitement de la sûreté achève ce chapitre ; nous montrons que sur les mots, la décision exacte est PSPACE-dure alors que sa version approchée est polynomiale.

Chapitre 5 : Applications

Une application web a été développée et est présentée au travers de quelques exemples.

Sur les mots, elle permet l'application de transducteurs, potentiellement non déterministes et/ou pondérés, et des opérations classiques sur le langage obtenu[2]. En entrant les schémas et le transducteur, le langage des mots consistants (ou sûrs, non-sûrs ...) peut être calculé, et visualisé dans divers formats. Certains calculs de distances[3] entre langages sont essentiellement de même nature que le calcul de consistance et sont donc calculables exactement. Les testeurs de proximité fournissent quant à eux, une alternative bien plus efficace puisqu'ils permettent l'approximation de la distance correspondante en temps constant.

Sur les arbres, l'application autorise la transformation par des programmes XSLT[4] ainsi que la confrontation à une DTD, permettant ainsi de généraliser la décision de la consistance au cas des arbres. Son interface sera utilisée pour présenter des changements de schémas particuliers :
$XML \rightarrow SVG$ **:** des représentations graphiques (histogrammes, « camemberts »...) sont créées à partir de données XML bruitées. Nous appliquons ensuite ces transformations pour visualiser automatiquement les réponses numériques d'une requête OLAP.
$XML \rightarrow KML$ **:** un fichier de logs sera transformé en carte du monde, fournissant ainsi la base d'un outil d'analyse géographique du trafic d'une application web.

[2] confrontation à un langage-cible, recherche de la transformation la moins coûteuse...
[3] par exemple la distance de Hamming, la distance d'édition, ainsi que leurs versions généralisées
[4] A l'aide de processeurs XSLT 1.0 et 2.0 : Xsltproc et Xalan

Table des matières

Notations

- $\mathbb{N}$: l'ensemble des entiers naturels.
- $\mathbb{N}^+$: l'ensemble des entiers naturels non nuls.
- $\mathbb{Z}$: l'ensemble des entiers relatifs.
- $\mathbb{Q}$: l'ensemble des rationnels.
- $\mathbb{R}$: l'ensemble des réels.
- $\{n, m\}$: l'ensemble des entiers compris entre n et m.
- $\lfloor r \rfloor$ désigne la partie entière inférieure de r et $\lceil r \rceil$ sa partie entière supérieure.
- Pour des éléments e, e_1, e_2 et des ensembles E, E_1, E_2 :

  $e \in E$ signifie que e appartient à E.

  $e_1, e_2 \in E$ est le raccourci de $e_1 \in E$ et $e_2 \in E$.

  $(e_1, e_2) \in (E_1 \times E_2)$ est le raccourci de $e_1 \in E_1$ et $e_2 \in E_2$.

  $e \notin E$ signifie que e n'appartient pas à E.

  $\{e \mid P\}$ désigne l'ensemble des éléments e qui satisfont la propriété P.

  $\#E$ désigne le cardinal de l'ensemble E.
- $\mathcal{O}$ (respectivement Ω) est la notation standard de borne supérieure (resp. inférieure) modulo une constante. Θ désigne la conjonction de $\mathcal{O}$ et Ω.
- $\bmod p$ (respectivement $\mathsf{Mod}\ p$) désigne la congruence modulo p pour les classes d'équivalence $\{0, p-1\}$ (resp. $\{1, p\}$).
- Pour Σ un alphabet et $k \in \mathbb{N}$, Σ^k désigne l'ensemble des mots à k lettres et Σ^* l'ensemble des mots sur l'alphabet Σ.
- Pour un mot w, un élément $a \in \Sigma$, et $k \in \mathbb{N}$,

  Λ désigne le mot vide.

  $|w|$ désigne la longueur de w, $|w|_a$ désigne le nombre d'occurrences de a dans w.

  w^2 désigne le mot ww et w^k désigne $\underbrace{ww...w}_{k\ fois}$.

  $w[k]$ désigne la $k^{\text{ième}}$ lettre du mot w en considérant qu'elles sont numérotées à partir de 1.

- Pour un vecteur X décrit sur une base orthogonale $x_1, \ldots, x_n$,

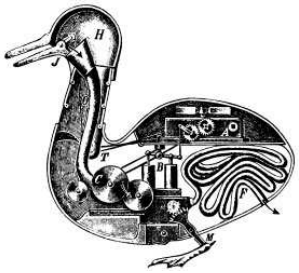
  $X[x_i]$ désigne la coordonnée correspondant à la base x_i.

  $[[X]]$ désigne le vecteur X dans lequel on a renormalisé les coordonnées pour qu'elles se somment à 1 : $[[X]]\,[x_i] = \frac{X[x_i]}{\sum_j X[x_j]}$.

- Signalétique de fin :
  - ✓ : Définition
  - ♪ : Exemple
  - ★ : Théorème
  - ⋆ : Proposition, Lemme ou Corollaire
  - □ : Démonstration

Chapitre 1

Définitions Préliminaires



Nous commençons par introduire la notion de structure étiquetée avec attributs permettant de représenter un fichier XML. Nous donnons ensuite deux façons classiques de présenter des classes de ce type de structures en rappelant ce qu'est un automate et une DTD. Intuitivement, ces objets représentent un langage (régulier) de mots ou d'arbres en donnant des règles définissant ce qu'est un mot ou un arbre valide. Nous donnerons ensuite des propriétés classiques de clôture de ces langages d'automates ainsi que les complexités de leurs opérations usuelles (appartenance, inclusion, vide). Ces résultats permettront ultérieurement de déduire des restrictions « tractables » ainsi que des bornes supérieures générales pour certains problèmes comme la consistance ou la sûreté. Enfin, nous présentons les testeurs et ainsi le type d'approximation que nous considérons dans ce mémoire : des algorithmes qui séparent les structures satisfaisant une propriété de celles « loin » de la satisfaire. La notion de distance entre structures sur laquelle les testeurs seront basés est rappelée : c'est la distance d'édition avec déplacements.

Sommaire

1.1 Structures et Langages

Structures Étiquetées

Classiquement, une *Structure* est la donnée d'un ensemble D, de fonctions sur D, de relations sur D et d'éléments distingués. On écrit $U = (D, f_1, ..., f_k, R_1, ...R_l, a_1, ..., a_m)$ où les fonctions $f_i : D^{a_i} \to D$ ont une arité a_i, les relations $R_j \subseteq D^{a_j}$ ont une arité a_j, et $a_k \in D$ sont des éléments distingués de D.

Exemple 1.1 (Structures)

Voici deux exemples de structures :

(a) Le mot 101001 peut être vu comme un ensemble $D = \{1, 6\}$ d'éléments avec une relation binaire *Succ* donnant un ordre linéaire à cet ensemble, et une relation unaire désignant les positions dans le mot occupées par un 1 (ici $\{1, 3, 6\}$).

(b) $\mathbb{N} = (\{0, 1, 2, \ldots\}, +, *, =, 0)$ est l'arithmétique avec des fonctions binaires $+$ et $*$, une relation binaire $=$, et un élément distingué 0.

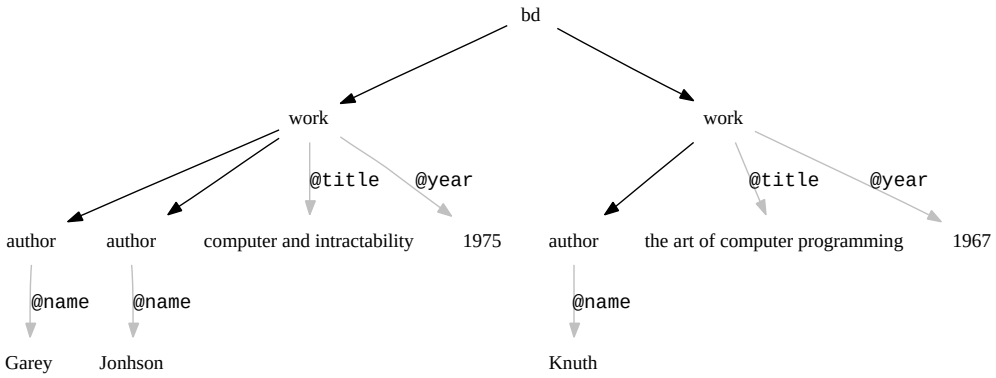


FIG. 1.1 – *Un arbre t_S associé à un fichier F.xml*

Dans ce document, nous considérons des *structures étiquetées avec attributs* : une structure possède alors deux domaines : D l'ensemble de ses nœuds ; et $\mathcal{S}$ l'ensemble des valeurs de leurs attributs. Chaque noeud est étiqueté (par la fonction $L : D \to \Sigma$) par une lettre d'un alphabet fini Σ et possède un ensemble d'attributs $A' \subseteq A$ ayant une valeur dans $\mathcal{S}$ (donné par une fonction partielle $\lambda : D \times A \to \mathcal{S}$).

Exemple 1.2 (Structures étiquetées avec attributs)

(a) Considérons la classe des mots binaires, avec des attributs dans $\{A_1, A_2\}$, et des valeurs dans $\{a, c\}$. Le mot $w = 1.0_{[A_1=a]}.1_{[A_1=c,A_2=c]}.0_{[A_1=a]}.1.0_{[A_1=a]}$, pour $D = \{1, 6\}$, est tel que $L(2) = 0$ et $\lambda(2, A_1) = a$.

(b) L'arbre XML donné par la figure 1.1 est également une structure étiquetée avec attributs.

- $\Sigma = \{bd, work, author\}$
- $A = \{name, title, year\}$
- $D = \{\Lambda, 1, 2, 11, 12, 21\}$
- $Child = \{(\Lambda, 1), (\Lambda, 2), (1, 11), (1, 12), (2, 21)\}$
- NextSibling $= \{(1, 2), (11, 12)\}$
- $\mathcal{S} =$ { Garey, Jonhson, Knuth, computer and intractability, the art of computer programming, 1975, 1967 }
- $L(\Lambda) = bd$, $L(1) = L(2) = work$, $L(11) = L(12) = L(21) = author$
- $\lambda(1, title) =$ computer and intractability, $\lambda(1, year) = 1975$, $\lambda(11, name) =$ Garey ...
- $r = \Lambda$

♪

Nous concentrons notre étude sur la classe des arbres ordonnés à arité non bornée[5]. Le domaine D d'un arbre fini ordonné constitue un ensemble de positions que l'on notera comme dans l'exemple XML : Λ pour la racine, $\{1, 2, \ldots\}$ pour les fils de la racine, $\{11, 12, \ldots\}$ pour les fils du premier fils de la racine, etc ... Donnons tout d'abord une représentation binaire de ces structures.

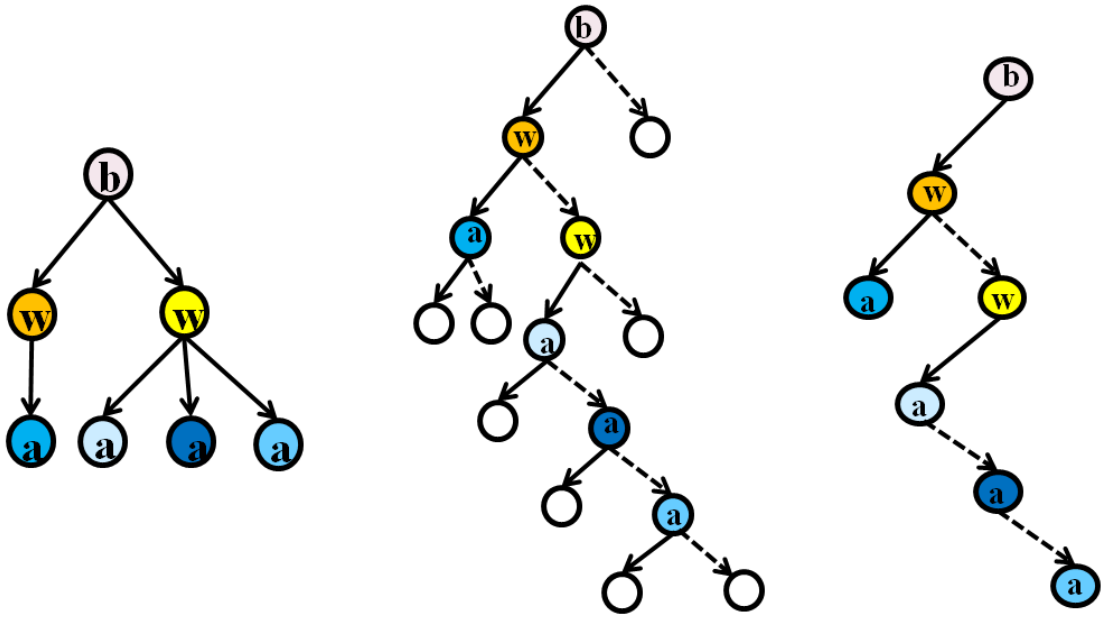


Fig. 1.2 – *Un encodage* fcns *et sa version étendue*

Définition 1.3 (Encodage fcns)

L'encodage fcns (first child next sibbling) consiste à représenter un arbre (unranked) par un arbre binaire en utilisant deux pointeurs pour chaque noeud, un pour son premier fils et un pour son successeur droit, avec pour convention que le nouveau symbole $\bigcirc$ désigne l'absence d'un tel successeur.

- Pour une feuille f étiquetée $a \in \Sigma$, $\mathsf{fcns}(f) = a(\bigcirc, \bigcirc)$
- Pour un arbre $t = a(t_1 \ldots t_n)$, $\mathsf{fcns}(t) = a(\mathsf{fcns}(t_1 \ldots t_n), \bigcirc)$
- Pour une haie $h = t_1 \ldots t_n$ avec $n \geq 2$, $\mathsf{fcns}(h) = \mathsf{fcns}(t_1)[\mathsf{fcns}(t_2 \ldots t_n)]_2$

Intuitivement, le troisième cas revient à encoder le sous-arbre le plus à gauche en laissant le pointeur de frère pour l'encodage de la forêt restante. Formellement $t[t']_i$ désigne la substitution dans l'arbre t de la position i par l'arbre t'. ✓

[5]en anglais : unranked ordered trees

On parlera d'*arbre binaire étendu* pour désigner les encodages d'arbres unranked dans lesquels les éléments $\bigcirc$ sont oubliés. Un noeud peut donc avoir un successeur droit même en l'absence d'un successeur gauche comme dans la figure 1.2.
Ces structures sont alors dans la classe

$$\mathbf{K} = \{I \mid I = (\Sigma, A, D, \mathcal{S}, \text{LeftChild}, \text{RightChild}, r, L, \lambda)\}$$

où LeftChild représente la relation de premier fils d'un arbre unranked et RightChild la relation de frère.

Langages

L'étude des langages est cruciale en informatique. Par exemple, une DTD spécifie un ensemble d'arbres, c'est-à-dire un langage d'arbres. Étant donné une représentation A_L d'un langage, on veut pouvoir résoudre des problèmes de décision classiques tels que :

- **Vide** : $\exists x \in A_L$?
- **Appartenance** : $x \in A_L$?
- **Intersection** : $\exists x \in A_L \cap A_{L'}$?
- **Inclusion** : $x \in A_L \Rightarrow x \in A_{L'}$?

En informatique, une représentation classique des langages repose sur les automates. Un *Langage Régulier* est alors un ensemble de structures reconnues par un automate : celles qui possèdent une[6] exécution acceptante. Nous rappellerons ici ce qu'est un automate de mots et un automate d'arbres et détaillerons les complexités respectives des problèmes de décisions cités.

[6] au moins une dans le cas non déterministe

1.1.1 Langages de Mots

Définition 1.4 (Automate de mots (NFA))

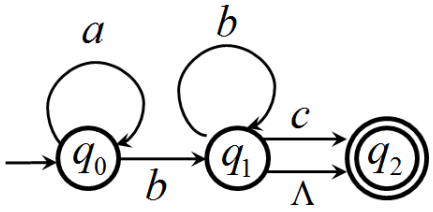
Un *automate* est donné par un 5-uplet $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ où Σ est un alphabet fini, Q un ensemble fini d'états, $I \subseteq Q$ un ensemble d'états initiaux, $F \subseteq Q$ un ensemble d'états finaux et $\delta \subseteq Q \times (\Sigma \cup \Lambda) \times Q$ une relation de transition ; on rappelle que Λ désigne le mot vide. ✓

Un automate est dit *déterministe* quand I est un unique état et δ une fonction (partielle) de $Q \times \Sigma$ dans Q. Si l'automate ne change jamais spontanément d'état (*i.e.* $\delta \subseteq Q \times \Sigma \times Q$) l'automate est dit Λ-*libre*.

Une *exécution* d'un mot $w = w_1 ... w_n$ sur un automate $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ peut être vue comme une séquence $(q_i, a_i, q_{i+1})_{i=\{1,m\}}$ où $m \geq n$, $q_0 \in I$, et il existe une injection f strictement croissante de $\{1, n\}$ dans $\{1, m\}$ telle que si $i \in f(\{1, n\})$, alors $a_i = w[f^{-1}(i)]$ et $(q_i, a_i, q_{i+1}) \in \delta$ et sinon $a_i = \Lambda$ et $(q_i, \Lambda, q_{i+1}) \in \delta$. Une exécution est dite *acceptante* quand elle s'achève dans un état final ($q_{m+1} \in F$).

Exemple 1.5

Le langage L, défini par l'expression régulière $a^* b^+ c^?$, est l'ensemble des mots commençant par un nombre arbitraire (éventuellement 0) de a, suivi d'un nombre quelconque strictement positif de b, et éventuellement d'un c. Ce langage est accepté par l'automate $\mathcal{A} = (\Sigma, Q, I, F, \delta)$: $\Sigma = \{a, b, c\}$, $Q = \{q_0, q_1, q_2\}$, $I = \{q_0\}$, $F = \{q_2\}$, $\delta = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, c, q_2), (q_1, c, \Lambda)\}$



Par exemple *abc*, *bbb*, *aaaab* sont des mots de L. Cela indique que l'on peut lire un de ces mots w de l'état initial (flèche entrante), en terminant dans un état final (doublement cerclé), en suivant une séquence de transitions. Ces transitions sont spontanées (étiquetées Λ) ou lisent un nouveau caractère de w. ♪

Décisions Classiques

La complexité des problèmes proposés se mesure avec les paramètres principaux n, la taille du mot, et m la taille de l'automate : $n_{\mathcal{A}} + a_{\mathcal{A}}$ où $n_{\mathcal{A}}$ est le nombre de sommets de $\mathcal{A}$, et $a_{\mathcal{A}}$ son nombre d'arêtes.

Vide : Le problème de décider **Vide** est d'une complexité temporelle linéaire en la taille de l'automate. On rappellera également l'existence d'algorithmes non déterministes et logarithmiques en espace [Khuller et Raghavachari, 1997], ou d'approches en temps logarithmique (dans un modèle de calcul parallèle [Vardi, 1998]) permettant de résoudre cette question.

Appartenance : Décider si un mot w appartient au langage défini par un automate déterministe est un problème dit « linéaire combiné » *i.e.* linéaire en la taille du mot et linéaire en la taille de l'automate ($\mathcal{O}(|w| \times \|\mathcal{A}|)$) ; la complexité structurelle est LogSpace (Théorème 3 de [Lohrey, 2001]). Pour un automate non déterministe $\mathcal{A}$, la décision peut se faire en temps $\mathcal{O}(|w|.|\mathcal{A}|^2)$ et le problème caractérise la classe LOGDCFL (Lemme 11 de [Segoufin, 2003]).

Intersection : L'intersection de deux langages réguliers est bien sûr régulière et peut se voir comme le « produit cartésien » de leurs automates. On peut donc construire l'automate de l'intersection [7] en temps $\mathcal{O}(|\mathcal{A}|.|\mathcal{A}'|)$.

Inclusion : Le problème de l'inclusion pour les langages réguliers de mots donnés par des automates est un problème PSPACE. Dans le cas général (automates non-déterministes, expressions régulières de mots) ce problème est PSPACE-complet. Une meilleure complexité est obtenue en limitant l'expressivité des langages : par exemple l'inclusion est un problème PTIME pour les automates m-ambigus [Stearns et Hunt, 1981, Stearns et Hunt, 1985], NP-complet pour les automates reconnaissant des langages finis [Hunt et al., 1976], ou coNP-complet pour des expressions régulières formées de concaténation de $a, b, c...$ et de $a^*, b^*, c^*...$ [Martens et al., 2004].

[7] plus précisément, pour $\mathcal{A}$ un NFA à n sommets et a arêtes et $\mathcal{A}'$ un NFA n' sommets et a' arêtes, on sait construire un NFA $\mathcal{A}''$ reconnaissant l'intersection des deux langages avec $n.n'$ états et $\mathcal{O}(a.a'+a.n'+a'.n)$ arêtes en temps $\mathcal{O}(a.a' + a.n' + a'.n)$.

Autres propriétés de clôture : L'ensemble des langages réguliers de mots est clos par union, concaténation et étoile. La construction des automates résultant de ces opérations est bien connue [Rabin et Scott, 1959, Hopcroft et Ullman, 1969]. Son coût est linéaire pour des automates non déterministes. On peut par ailleurs conserver le déterminisme à un coût identique à celui de la multiplication de matrices [Hopcroft et Ullman, 1990]. Enfin, tout automate non déterministe est équivalent à un automate déterministe [Rabin et Scott, 1959]. La procédure de déterminisation repose sur une construction de sous-ensembles de l'espace d'états. Cela conduit à une construction en temps exponentiel mais réalisable en espace polynomial [Stockmeyer et Meyer, 1973], dont le résultat donne accès en temps linéaire au complémentaire du langage.

1.1.2 Langages d'Arbres

D'un point de vue théorique, on peut définir une classe d'arbres comme l'ensemble des arbres acceptés par un automate et on parlera alors de langage régulier d'arbres. Il existe différentes façons de parcourir un arbre à l'aide d'un automate (des feuilles vers la racine, et inversement ...) qui conduisent à plusieurs modèles [Doner, 1965, Doner, 1970, Thatcher et Wright, 1968, Comon et al., 1997].

En pratique, dans le contexte XML, la notion de schéma définit le type de contenu, la syntaxe, et éventuellement la sémantique d'un arbre. Deux types de schémas sont principalement utilisés :

- **XSD** (XML Schema Description [xmlSchema, 2001]) permet de définir un schéma, de façon structurée, en respectant le format XML, approuvé par le W3C[8].
- **DTD** (Document Type Definition [dtd, 1995]) qui décrit la hiérarchie des étiquettes, sans considération sur les valeurs d'attribut utilisées.

Le choix des DTD est particulièrement bien adapté ici puisque l'on va s'intéresser à des décisions concernant essentiellement la structure des étiquettes. Nous allons définir ce qu'est un automate d'arbres, puis voir qu'une DTD est un cas particulier d'automate d'arbres.

[8] http://www.w3.org/XML/Schema

Automate d'Arbres

Une exécution d'un automate d'arbres sur un arbre annote l'arbre par des états jusqu'à avoir annoté chacun des noeuds de l'arbre. Une exécution ascendante marque les noeuds, des feuilles à la racine (de proche en proche jusqu'à marquer la racine), et est dite acceptante si le nœud-racine est finalement marqué par un état final. Intuitivement, on souhaite que la garde d'une transition puisse décrire la situation pour un nombre quelconque de fils. Pour conserver une représentation finie des transitions dans le cadre à arité non bornée, on utilise les expressions régulières. On pourra marquer un noeud x, en déclenchant une transition à la lecture d'une étiquette, lorsque la séquence des marques de ses fils suit une certaine expression régulière.

Définition 1.6 (Automate d'Arbres (NFHA))
Un *automate d'arbres* est donné par un 4-uplet $A = (\Sigma, Q, F, \delta)$ où Σ est un alphabet, Q un ensemble fini d'états, $F \subseteq Q$ un ensemble d'états finaux, et δ un ensemble fini de règles du type $a(R) \to q$ où $R \subseteq Q^*$ est un langage régulier sur Q. ✓

Exemple 1.7
Soit l'automate $\mathcal{A} = (\Sigma, Q, F, \delta)$ où $\Sigma = \{a, b, w\}$, $Q = \{q_a, q_b, q_w\}$, $F = \{q_b\}$, et δ donnée par $a(\Lambda) \to q_a$, $w(q_a^+) \to q_w$ et $b(q_w^*) \to q_b$.
Le langage accepté par cet automate est l'ensemble des arbres de racine b, dont les fils sont des w et les petits-fils des a. Une exécution acceptant un arbre particulier est donnée par la Figure 1.3. ♪

On parlera d'automate déterministe (DFHA) lorsque l'état d'un noeud p ne dépend que de son étiquette et de ses successeurs, *i.e.* si pour toutes règles $a(R_1) \to q_1$ et $a(R_2) \to q_2$, soit $R_1 \cap R_2 = \emptyset$, soit $q_1 = q_2$.

DTD

La notion de DTD est complètement connectée à celle d'automate d'arbres puisqu'une DTD définit un automate d'arbres particulier dans lequel l'ensemble des états est aussi l'alphabet ($Q = \Sigma$). C'est typiquement le cas de l'exemple précédent que l'on pourrait ré-écrire $\{b : w^*, w : a^+\}$.

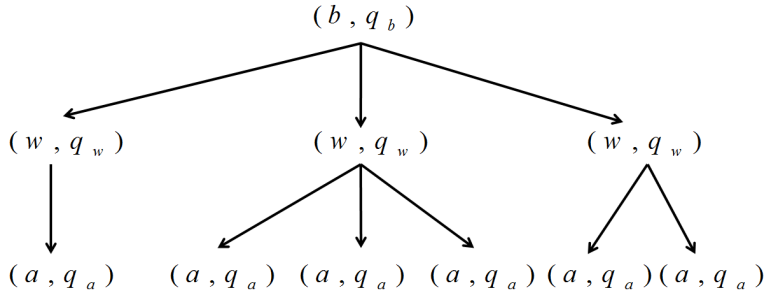


FIG. 1.3 – *Exemple d'une exécution acceptante de l'automate d'arbres dans l'exemple 1.7*

Dans la pratique, on gère aussi les attributs et on spécifie le contenu d'un élément en indiquant le nom, l'ordre et le nombre d'occurrences autorisées des sous-éléments. L'ensemble constitue alors la définition des hiérarchies valides d'éléments (avec attributs) et de textes.

Exemple 1.8

La DTD de la figure 1.4 reflète l'automate donné précédemment en exemple. Un arbre comme celui de la figure 1.1 (*page* 15) est valide pour cette DTD.

```
<!ELEMENT db (work*)>
<!ELEMENT work (author+)>
<!ATTLIST work title CDATA #REQUIRED
               year CDATA>
<!ELEMENT author (EMPTY)>
<!ATTLIST author name CDATA #REQUIRED>
```

FIG. 1.4 – *Une DTD-Source :* $\mathbf{K}_S$

```
<!ELEMENT bib (livre*)>
<!ELEMENT livre (titre, auteur*)>
<!ELEMENT auteur #PCDATA>
<!ELEMENT titre #PCDATA>
```

FIG. 1.5 – *Une DTD-cible :* $\mathbf{K}_T$

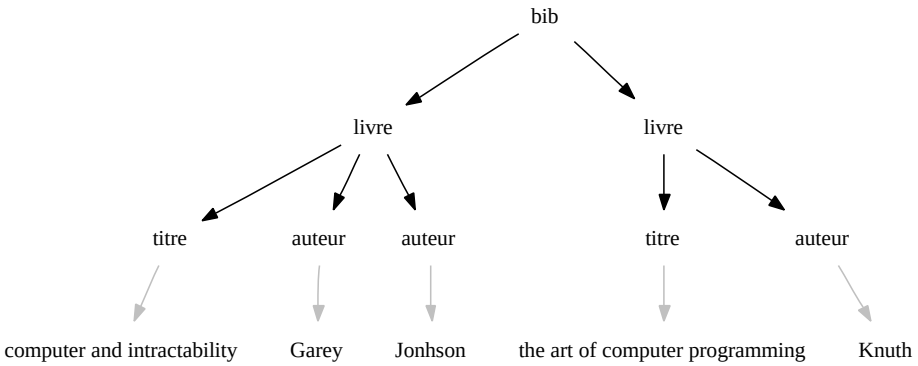


Fig. 1.6 – *Une instance valide pour la DTD-cible* $\mathbf{K}_T$

♪

Décisions Classiques

La complexité des décisions classiques dépend évidemment de la représentation utilisée pour les langages horizontaux (les expressions R). Une possibilité est de décrire ces langages horizontaux par les automates de mots non déterministes (NFA). On peut également vouloir manipuler des combinaisons booléennes d'expressions régulières et une façon de les représenter est de le faire par des automates alternants (AFA).

Vide : La décision du vide d'un automate est dans PTIME pour les NFHA(NFA) et PSPACE-Complet pour les NFHA(AFA) (Théorème 8.5.8 de [Comon et al., 2007]).

Appartenance : Décider si un arbre appartient au langage reconnu par un automate est un problème linéaire en la taille de l'arbre. Si on souhaite également considérer l'automate comme une entrée, on parle alors du problème d'appartenance uniforme dont la complexité est rappelée par la table 1.1 issue du théorème 8.5.6 de [Comon et al., 2007].

Entrée	Complexité de l'appartenance uniforme
NFHA(NFA)	temps linéaire combiné
DFHA(DFA)	temps linéaire
NFHA(AFA)	NP-complet
DFHA(AFA)	PTIME

Tab. 1.1 – *Décision de l'appartenance d'un arbre au langage d'un automate d'arbres*

Inclusion : Comme pour les automates de mots, l'inclusion est efficace (PTIME) pour les automates d'arbres déterministes dont les langages horizontaux sont déterministes, et reste décidable mais plus difficile (typiquement EXPTIME) pour les automates non déterministes. La table 1.2, basée sur le théorème 8.5.9 de [Comon et al., 2007] explicite les complexités pour différents types d'automates d'arbres.

Entrée	Complexité en Temps
DFHA(DFA)	PTIME
NFHA(NFA)	EXPTIME-Complet
DFHA(AFA)	PSPACE-Complet
DTD(exp. reg. 1-non-ambiguës)	PTIME
DTD	PSPACE-Complet

Tab. 1.2 – *Complexité de l'inclusion pour les langages d'automates d'arbres*

Notons finalement que l'inclusion dans PTIME a récemment été généralisée pour les DTD déterministes — celles pour lesquelles les langages horizontaux sont des expressions régulières dont les automates de Glushkov sont déterministes — [Champavère et al., 2008].

Propriétés de clôture

Il existe différents encodages d'un arbre en arbre binaire, et certains permettent la conservation de la reconnaissabilité, *i.e.* L est reconnaissable par un automate d'arbres si et seulement si $encodage(L)$ est simplement reconnaissable (par un automate d'arbres binaire).

Les langages réguliers d'arbres à arité bornée étant clos par union, complémentation et intersection (on pourra voir par exemple [Hopcroft et Ullman, 1990] ou le théorème 5 de [Comon et al., 1997]) ; on en déduit que les langages réguliers d'arbres (à arité non bornée) sont également clos par union, complémentation et intersection.

De la même manière que pour les mots, une construction basée sur les parties des états d'un automate (NFHA) permet d'obtenir un automate déterministe (DFHA) équivalent et cette déterminisation peut également provoquer une augmentation exponentielle du nombre d'états. Minimiser le nombre d'états d'un DFHA reconnaissant un langage mène à un unique automate normalisé (au renommage des états près). La minimisation devient

par contre délicate lorsqu'on veut tenir compte de la taille des représentations des langages horizontaux [Cristau et al., 2005, Martens et Niehren, 2005].

Autres Langages d'Arbres

Définir ce qui est valide est aussi le rôle des « schémas » comme Relax NG [relaxNG, 2001], XML Schema [xmlSchema, 2001] et Schematron [schematron, 2004]. Ceux-ci sont préférentiellement exprimés en syntaxe XML alors que les DTD ont une syntaxe spécifique. La différence la plus significative du point de vue de la puissance d'expression est que les DTD spécifient les arbres valides et ne permettent pas de restriction (négative). On ne peut donc pas exprimer une restriction sur l'imbrication des éléments (un élément "A" ne peut contenir un autre élément "A" à n'importe quel niveau) ni définir des domaines de validité pour la valeur d'un champ.

D'un point de vue des langages, les DTD ne s'intéressent qu'aux étiquettes et sont des langages dits « locaux » [Makoto, 1995]. Pour une DTD non-ambiguë, il est possible de construire un automate déterministe descendant permettant de reconnaître l'ensemble des encodages (fcsn) des arbres valides. Il est par ailleurs connu que ces langages sont strictement moins expressifs que les langages définis par les automates (déterministes) ascendants. Les XML Schemas, fournissent un cadre intermédiaire permettant d'accéder aux informations des ancêtres du noeud courant. Ils correspondent aux DTD « restreignant la compétition » [Bex et al., 2005] et sont alors adaptés à une perspective de validation descendante en une passe [Martens et al., 2005].

RelaxNG fournit, quand à lui, un cadre plus général que les précédents puisqu'il permet de représenter tous les langages réguliers. Il existe par ailleurs plusieurs outils permettant des transformations entre schémas : [Trang, 2004] permet par exemple de convertir :

- une DTD en XML Schema ou en schéma RelaxNG (en notation compacte ou XML),
- un schéma RelaxNG en DTD ou en XML Schema (lorsque c'est possible).

1.2 Distance et Approximation

1.2.1 Distances

Aussi appelée distance de Levenshtein ou déformation dynamique temporelle, la *distance d'édition* entre deux mots w et w', est le nombre minimal d'opérations pour transformer w en w' en effectuant les opérations élémentaires suivantes : ajout, suppression ou substitution d'un caractère. La distance d'édition relative est alors la distance absolue divisée par le maximum des longueurs de w et w'.

Notons que la distance d'édition entre deux mots w et w' peut simplement être obtenue par programmation dynamique en temps $(|w|+1) \times (|w'|+1)$ et en espace $\min(|w|, |w'|)$.

Exemple 1.9
Soit $w = 01011111$ et $w' = 0000111$. En partant de w, on peut remplacer les deux premiers 1 en 0 et supprimer le dernier 1. On obtient ainsi w' en trois opérations élémentaires. Il est par ailleurs facile de montrer qu'au moins trois opérations sont nécessaires donc $dist(w, w') = 3$ et $\mathsf{d}(w, w') = 3/8$. ♪

Ajoutons maintenant un nouveau type d'opération élémentaire, *le déplacement*, qui consiste à déplacer un sous-mot à une autre position (pour un coût de 1). La distance est à nouveau le nombre minimal d'opérations élémentaires pour passer d'un mot à l'autre, et la distance relative cette même distance divisée par $\max(|w|, |w'|)$.

Exemple 1.10
Soit $w = 01010111$ et $w' = 000011111$. On peut transformer w à w' avec deux déplacements (de 01) et une insertion (de 0) :

$$w = 01010111 \xrightarrow{m} 010011111 \xrightarrow{m} 000111111 \xrightarrow{i} 000011111 = w'$$

$$\mathsf{dist}(w, w') = 3 \text{ et } \mathsf{d}(w, w') = 3/9 = 1/3.$$

♪

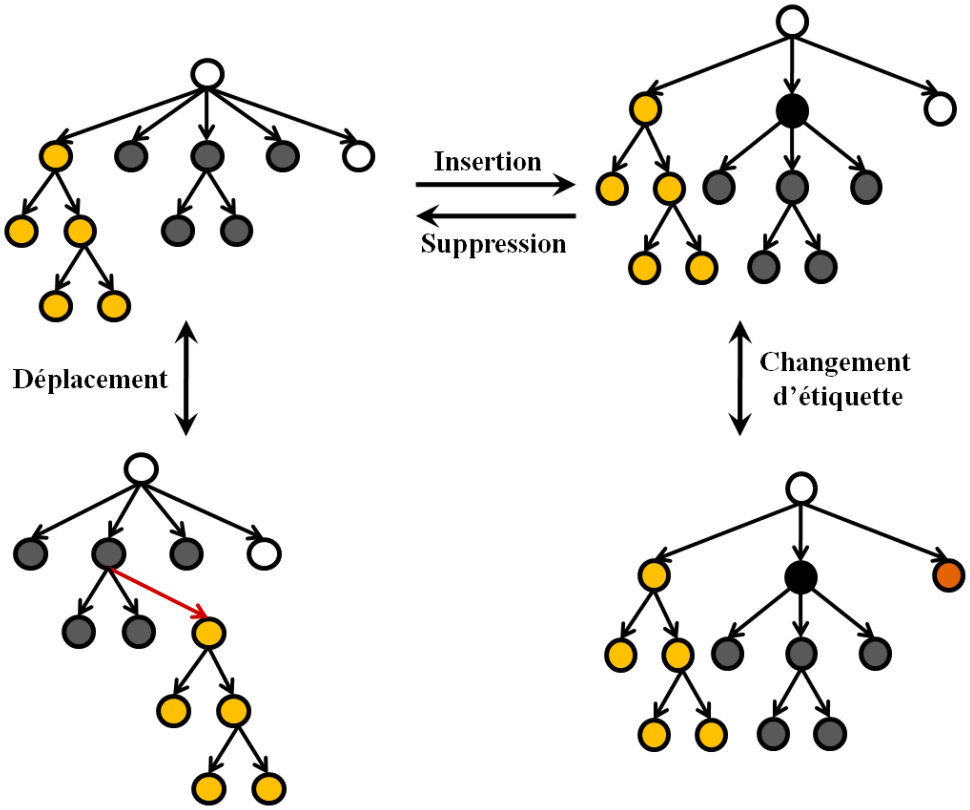


Fig. 1.7 – *Distance d'édition avec déplacements : les quatre opérations élémentaires*

La *distance d'édition avec déplacements* se généralise naturellement aux arbres. La distance entre deux arbres est alors le nombre minimum d'opérations élémentaires nécessaire pour transformer un arbre en un autre. Les opérations élémentaires sont similaires au cas des mots : on peut, à un coût unitaire :

- Modifier l'étiquette d'un nœud.
- Insérer ou supprimer un nœud qui n'est pas la racine (les éventuels fils de ce nœud deviennent alors des fils du père de ce nœud-source, insérés à la place déterminée par l'arête père-fils).
- déplacer tout un sous-arbre à une autre position.

Comme précédemment, la distance relative est la distance absolue divisée par le maximum de la taille des deux arbres considérés, la taille étant ici le nombre de nœuds de l'arbre.

L'**insertion** d'un noeud est donnée par une position et une étiquette de Σ. La position peut être de 4 types : sous une feuille ; en tant que dernier fils d'un noeud ; à la place d'un noeud existant (le noeud inséré décale alors la sous-haie issue de cette position[9] sur la droite) ; ou en tant que nouveau père d'une sous-haie (comme l'exemple donné par la Figure 1.7).

La **suppression** d'un noeud v fait hériter au père v de ce noeud l'éventuelle descendance de v.

Le **déplacement** permet de déplacer le sous-arbre t enraciné en la position initiale à une position destination, en suivant les mêmes règles que pour l'insertion sur l'arbre privé de t.

Qu'il s'agisse de mots ou d'arbres, nous avons maintenant des distances entre instances que nous pouvons généraliser aux distances entre langages. La distance d'une instance à un langage est la distance de cette instance à l'instance du langage la plus proche ; la distance entre deux langages est la plus petite distance entre deux instances des langages. Formellement :

$$\mathsf{dist}(w, L_1) = \min_{w_1 \in L_1} \mathsf{dist}(w, w_1)$$

$$\mathsf{dist}(L_1, L_2) = \min_{w_1 \in L_1, w_2 \in L_2} \mathsf{dist}(w_1, w_2)$$

Remarquons que cette généralisation n'est pas spécifique à une distance, même si dans cette thèse, nous l'utilisons pour la distance d'édition avec déplacements. Notons finalement que les minimums (min) doivent être remplacés par des inf dans le cas de distances continues (par exemple les distances relatives).

Nous allons finalement détailler le type d'approximation utilisé dans ce document en rappelant la notion de testeur. Cette notion sera appliquée dans un cadre particulier : celui de distances basées sur la distance d'édition avec déplacements sur les mots et les arbres étiquetés. Elle est rappelée ici dans le cadre général d'une classe de structures munie d'une distance.

[9] le cadran dessous / à droite de cette position *i.e.* le noeud et sa clôture par les relations de fils et de frère droit

1.2.2 Proximité et Testeurs

La principale motivation pour modifier le modèle de calcul usuel est que la plupart des problèmes sont trop difficiles (par exemple NP-difficile). L'approche traditionnelle consiste à approximer un problème de calcul et l'approximation porte sur la sortie, comme dans le cas des schémas d'approximation en temps polynomial (PTAS). Cette approche trouve ses limites dans la caractérisation [Arora et Safra, 1998] de NP par les preuves vérifiables de manière probabiliste, qui implique des résultats de non-approximabilité [Feige et al., 1996].
Le property testing suggère d'approcher un problème de décision (par exemple, la 3-colorabilité) en accord avec une distance donnée sur les objets étudiés (par exemple, les graphes). Deux simplifications sont introduites : une probabiliste et une autre combinatoire. Étant donnée une distance sur les objets considérés (par exemple, le nombre d'arêtes dont diffèrent deux graphes), un ε-testeur pour une propriété :

- accepte tout objet satisfaisant cette propriété
- refuse, avec grande probabilité, tout objet ε-loin de ceux la satisfaisant.

Ce modèle rend possible la vérification d'une multitude de propriétés.

Définition 1.11 (Proximité)
Deux structures de taille respective n et m sont dites ε-proches si leur distance est inférieure à $\epsilon \times \max(n, m)$. Dans le cas contraire, on dit qu'elles sont ε-loin.
Une classe $\mathbf{K}$ est ε-inclue dans une classe $\mathbf{K}'$ si toute structure de $\mathbf{K}$ (sauf éventuellement un nombre fini) est ε-proche de $\mathbf{K}'$. On note alors $\mathbf{K} \subseteq_\epsilon \mathbf{K}'$ et les classes sont dites ε-loin dans le cas contraire. Deux classes sont ε-équivalentes, noté $\mathbf{K} \equiv_\varepsilon \mathbf{K}'$, si elles vérifient l'$\varepsilon$-inclusion dans les deux sens. ✓

Soit $\mathbf{K}$ une classe de structures finies U, telle que les mots ou les arbres. Une propriété P est un sous-ensemble de $\mathbf{K}$. Une formule F sur $\mathbf{K}$ est définie dans une logique telle que la logique du premier ordre ou la logique monadique du second ordre.

Définition 1.12
Soit P une propriété sur $\mathbf{K}$. Une structure $U \in \mathbf{K}$ ε-satisfait P, ou encore $U \models_\varepsilon P$, si U est ε-proche d'une structure $U_0 \in \mathbf{K}$ qui satisfait P (que l'on notera telle que $U_0 \models P$).
✓

La notation $U \not\models_\varepsilon P$ sera utilisée pour signifier que U est ε-éloignée de toute structure U_0 satisfaisant P, *i.e.* telle que $U_0 \models P$.

Définition 1.13 (Testeur [Goldreich et al., 1998])
Soit $\varepsilon > 0$. Un *ε-testeur* pour une propriété $P \subseteq \mathbf{K}$ est un algorithme probabiliste A tel que, pour toute structure $U \in \mathbf{K}$ fournie en entrée :

1. Si $U \models P$, alors A accepte avec probabilité 1 ;
2. Si $U \not\models_\varepsilon P$, alors A rejette avec probabilité au moins 2/3.

✓

Le seuil 2/3 sur la probabilité d'accepter est bien sûr arbitraire. Il peut être remplacé par n'importe quelle constante $\gamma > 1/2$, entraînant un facteur multiplicatif en $log(1/\gamma)$ dans la complexité du testeur. Il suffit en effet d'exécuter $log(1/\gamma)$ fois un testeur fonctionnant pour le seuil 2/3, puis de prendre la majorité de ses réponses pour obtenir le seuil γ.

Définition 1.14
Une propriété $P \subseteq \mathbf{K}$ est *testable*, s'il existe un algorithme probabiliste A tel que, pour tout réel $\varepsilon > 0$ donné en entrée, $A(\varepsilon)$ est un ε-testeur de P, et la complexité en temps de A dépend uniquement de ε. ✓

Dans le cas classique, une propriété sépare une classe d'instances en deux zones : celles satisfaisant la propriété et celles ne la satisfaisant pas. L'approximation introduite dans ce document se fonde sur la notion de testeur et une propriété distingue alors trois types d'instances : celles qui satisfont la propriété, celles loin de satisfaire la propriété[10] et une zone intermédiaire : les instances proches de satisfaire la propriété. On souhaite alors garantir le résultat de la décision pour les deux premières zones, quitte à tolérer des décisions erronées dans la zone intermédiaire (grisée).

[10] *i.e.* loin de toute instance satisfaisant la propriété

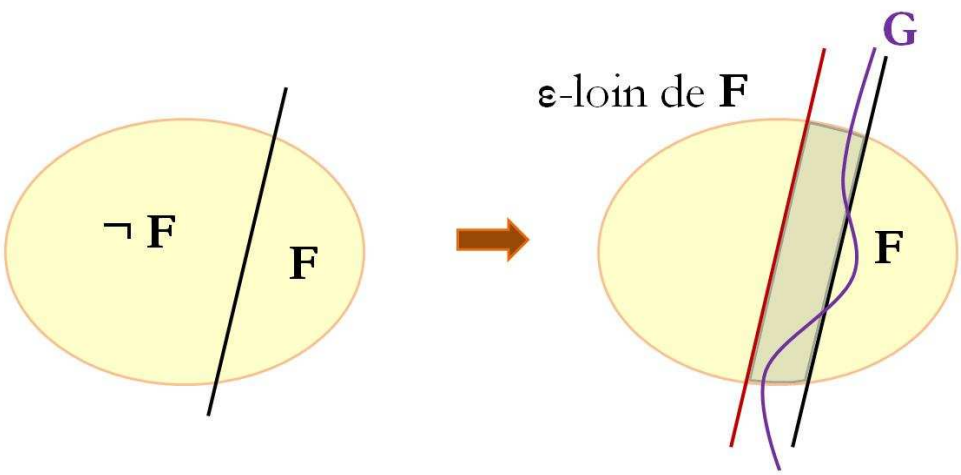
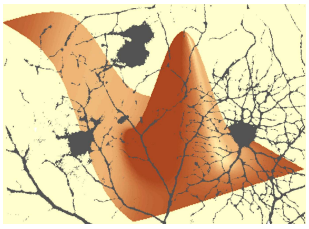


FIG. 1.8 – *Intuition de l'Approximation d'une Propriété*

Chapitre 2

Statistiques : Mots, Arbres, Automates



La notion de testeur a été définie pour une distance quelconque. Dans cette thèse, nous considérons le cadre de la distance d'édition avec déplacements. Cette distance particulière a déjà montré son intérêt dans des travaux précédents, puisqu'il a été démontré que l'équivalence approchée de langages réguliers de mots et d'arbres est testable pour cette distance [Fischer et al., 2006]. Cette approche est ici présentée en introduisant une représentation statistique guidée par cette distance. Un intérêt essentiel de cette représentation est que la statistique d'une instance est calculable en temps linéaire, mais surtout qu'elle peut être approchée efficacement via un échantillonnage de l'instance. Ceci constitue une différence majeure avec l'étude des dénombrements de sous-arbres [Asai et al., 2002, Abe et al., 2002], ou des sous-arbres induits [Zaki, 2002, Wang et al., 2004, Yang et al., 2004] : dans notre cas, un noeud est impliqué dans un nombre borné de sous-structures d'une taille donnée. Cette représentation statistique des instances est ensuite étendue pour les automates de manière inductive. Elle permet par exemple de décider efficacement l'inclusion approchée de deux langages par la comparaison de leurs représentations statistiques.

La statistique d'un mot est la fréquence de ses sous-mots (de ses sous-blocs) d'une longueur donnée. On détaillera, pour un mot donné, des méthodes linéaires de calcul des statistiques de ce mot : c'est le théorème 2.6 (*page* 39).
On montre par le fait 2.8 (*page* 42) comment échantillonner un mot et en déduire une bonne approximation de ses statistiques. La nouveauté de cette section réside principalement dans le traitement du problème réciproque : l'implémentation approchée *i.e.* étant donnée une statistique, est-il possible de trouver une suite de mots dont les statistiques convergent vers cette statistique ? Le théorème 2.18 (*page* 50) permet effectivement de décider ce problème approché efficacement et cette décision polynomiale a par ailleurs été automatisée en Maple. Ces nouvelles propriétés visant à donner des intuitions sur les statistiques ne sont pas indispensables à la compréhension de la suite. Nous finirons par montrer en quoi notre représentation statistique est liée à la distance d'édition avec déplacements : si deux mots de même longueur ont des statistiques proches, alors ces mots sont proches.

La statistique d'un arbre est la fréquence des chemins (et des étiquettes) d'une longueur donnée dans l'encodage fcnf de l'arbre (d'arité non bornée). On montre qu'à nouveau on peut, étant donné un arbre, calculer ses statistiques en temps linéaire. La nouveauté est l'extension de l'échantillonnage pour des arbres à arité non bornée : l'algorithme proposé calcule une matrice statistique approchée, grâce à un nombre constant de requêtes, une requête sur l'arbre étant l'accès à l'étiquette d'un noeud ou la lecture d'un pointeur (de l'encodage fcns). Cet l'échantillonnage conduit à une bonne approximation des statistiques d'un arbre avec grande probabilité, bonne dans le sens inférieure à ε pour la norme $\mathbf{L}_1$. On donnera finalement les grandes lignes de l'adaptation de l'implémentation approchée pour les arbres. Seules les définitions des statistiques sont ici essentielles pour la compréhension de la suite.

La statistique d'un automate vise à représenter les statistiques de ses mots (ou arbres). On montre alors comment construire de façon inductive une bonne approximation de ces statistiques par une union de polytopes (proposition 2.31), qui sont des enveloppes convexes de points correspondants aux boucles de cet automate. En comparaison de l'approche similaire proposée dans [Fischer et al., 2006] pour les automates de mots et arbres à arité bornée, l'apport est de considérer le cas des arbres à arité non bornée.

Sommaire

2.1 Statistiques de Mots

Soit k un entier strictement positif fixé, Σ un alphabet (fini) et w un mot sur Σ. Pour tout mot u de longueur k, on note $|w|_u$ le nombre d'occurrences de u dans w.

Définition 2.1 (ustat)
Étant donné k un entier non nul, ustat_k est un vecteur décrivant les fréquences d'apparition de chaque sous-mot de longueur k.

$$\forall u \in \Sigma^k,\ \mathsf{ustat}_k(w)[u] \stackrel{\text{def}}{=} \frac{|w|_u}{|w|-k+1}$$

$\mathsf{ustat}_k(w)$ est alors le vecteur à $|\Sigma|^k$ coordonnées contenant une coordonnée par sous-mot de longueur k. ✓

La valeur de ce vecteur, pour la coordonnée u, correspond à la distribution de probabilité qu'un sous-mot de longueur k, tiré uniformément (parmi l'ensemble des sous-mots de longueur k), soit ce u particulier, *i.e.*

$$\mathsf{ustat}_k(w)[u] = \Pr_{j=0,\ldots,n-k}\Big[w[j+1, j+2, \ldots, j+k] = u\Big] \tag{2.1}$$

Exemple 2.2
Pour $\Sigma = \{0, 1\}$, $w = 00011111$ de longueur $n = 8$

- Pour $k = 2$, $n - k + 1 = 7$ et le nombre d'occurrences de 00, 01, 10, 11 est respectivement 2, 1, 0 et 4. Le vecteur statistique associé (dans une description lexicographique des coordonnées) est donc :

$$\mathsf{ustat}_2(w) = (\ 2/7\ ,\ 1/7\ ,\ 0\ ,\ 4/7\ )$$

- Pour $k = 3$, $n - k + 1 = 6$, et l'ordre d'énumération est maintenant 000, 001, 010, 011, 100, 101, 110 et 111. Le vecteur statistique est alors :

$$\mathsf{ustat}_3(w) = \Big(\ 1/6\ ,\ 1/6\ ,\ 0\ ,\ 1/6\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 1/2\ \Big)$$

♪

2.1.1 Liens entre Statistiques et Distances

Il existe un lien profond entre la distance d'édition avec déplacement entre deux mots et la norme $\mathbf{L}_1$ entre leurs statistiques :

1. Si deux mots sont proches, leurs statistiques sont proches (Théorème 2.3).
2. Si deux mots (assez grands, de même longueur) ont des statistiques proches, alors ils sont proches (Théorème 2.4).

Théorème 2.3 (lemme 3.3 de [Fischer et al., 2006])
Soit $\varepsilon \in]0,1[$ *et soient* w *et* w' *deux mots de longueur* $n = \Omega(\frac{1}{\varepsilon})$. *Si* $\mathsf{dist}(w, w') \leq 5\varepsilon^2 n$ *alors* $\left|\mathsf{ustat}_{\lceil 1/\varepsilon \rceil}(w) - \mathsf{ustat}_{\lceil 1/\varepsilon \rceil}(w')\right| \leq 6.1\varepsilon$ ★

Démonstration : Pour chaque opération élémentaire, explicitons l'impact maximal fait sur le vecteur statistique. Effectuer une opération élémentaire de distance d'édition avec déplacements sur le mot w conduit à changer le voisinage autour d'au plus trois positions de w. Le cas le plus défavorable est celui du déplacement : il impacte les voisinages autour de trois positions de w : les positions de début et de fin du sous-mot déplacé et la position où ce sous-mot est réinséré. Pour chacun de ces trois voisinages, au plus $2(k-1)$ sous-mots sont affectés par cette opération. Une opération élémentaire de la distance engendre donc une différence en norme $\mathbf{L}_1$ sur les statistiques inférieures à $3 \cdot 2 \cdot (k-1) \leq \frac{6}{n \cdot \varepsilon}$ où n est la longueur de w. Par induction sur la distance absolue, on conclue que pour $\varepsilon^2 \cdot n$ opérations, la différence en norme $\mathbf{L}_1$ est inférieure à 6ε. □

Énoncer une réciproque de ce résultat du type « si deux mots ont des statistiques proches alors ils sont proches » nécessite des précautions. En effet deux mots de tailles respectives très différentes peuvent avoir les mêmes statistiques sans être proches. Pour des mots de même longueur, la réciproque est par contre toujours vraie et peut s'obtenir par un argument non constructif. C'est en effet un corollaire des travaux de [Fischer et al., 2006], en particulier la contraposée du lemme 3.6. Notons que nous n'exposons que très partiellement ce résultat profond et techniquement difficile, sans détailler les mesures statistiques intermédiaires (bstat, bustat) introduites pour son obtention.

Théorème 2.4 (Robustesse de ustat)

Pour tout alphabet fini Σ, $\varepsilon \in]0,1[$, *et* w *et* w' *deux mots sur* Σ *de longueur* $n = \Omega\left(\frac{\ln|\Sigma|.|\Sigma|^{2/\varepsilon}}{\varepsilon^5}\right)$ *: Si* $|\mathsf{ustat}_{\lceil 1/\varepsilon \rceil}(w) - \mathsf{ustat}_{\lceil 1/\varepsilon \rceil}(w')| \leq 6.5\varepsilon$, *alors* $\mathsf{dist}(w,w') \leq 5\varepsilon n$. ★

Démonstration : La robustesse de bustat issue de celle de bstat permet d'obtenir celle de ustat :

- Pour des mots de même longueur, la robustesse peut être montrée pour bstat :

$$dist(w,w') \leq \left(\frac{1}{2}|\mathsf{bstat}(w) - \mathsf{bstat}(w')| + \varepsilon\right) \times n$$

  où bstat désigne la distribution des blocs de k lettres consécutives.
- La distribution bstat est toujours proche de la distribution bustat :
  pour tout mot w assez grand, il existe un mot v obtenu à partir de w par l'effacement de $\mathcal{O}(\frac{(ln|\Sigma|)|\Sigma|^{2/\varepsilon}}{\varepsilon^4})$ lettres tel que $|\mathsf{bustat}(w) - \mathsf{bstat}(v)| \leq \frac{\varepsilon}{2}$.
- La robustesse de bustat est donc une conséquence de celle de bstat.
- Par un argument probabiliste, on montre que —pour tout mot w assez grand— $|\mathsf{bustat}(w) - \mathsf{ustat}(w)| = \mathcal{O}(\frac{(ln|\Sigma|)|\Sigma|^{2/\varepsilon}}{\varepsilon^4})$.
- La robustesse de ustat est alors une conséquence de la robustesse de bustat.

□

Remarque 2.5

L'introduction de la statistique intermédiaire bstat vient des idées suivantes : si deux mots de même longueur ont des bstat proches, alors on peut transformer l'un en l'autre avec peu d'opérations : un bloc sur-représenté est remplacé par un bloc sous-représenté jusqu'à l'obtention de l'égalité des bstat et, si les bstat sont égaux, les mots sont proches via des déplacements.

Par contre, et contrairement à ustat, deux mots peuvent être proches et avoir des bstat loin. En effet, décaler la position initiale (depuis laquelle on découpe en blocs de taille fixée) peut changer radicalement la distribution de blocs. Prendre en compte ces possibles décalages est alors l'idée de la dernière mesure introduite par bustat.

2.1.2 Calcul et Approximation

Pour calculer les valeurs du vecteur statistique ustat d'un mot w, il suffit de déplacer sur le mot w une fenêtre de longueur k en mettant à jour un vecteur de dimension constante ($|\Sigma|^k$). Ceci suggère un mode de calcul en continu (streaming) du vecteur $\mathsf{ustat}_k(w)$.

Proposition 2.6

Pour tout k, et tout mot w de Σ^*, toutes les valeurs non nulles de $\mathsf{ustat}_k(w)$ peuvent être calculées en temps $\mathcal{O}(|w|)$ $\star$

Ce calcul peut être vu comme une exécution sur un automate $\mathcal{A}^{\Sigma,k}_{Bruijn}$

- d'états $Q = \{u \in \Sigma^* \mid |u| \leq k-1\}$
- d'état initial $I = \Lambda$
- et de relation de transition définie par δ :
  - pour chaque $u \in \Sigma^*$ tel que $|u| < k-1$, et tout $a \in \Sigma$, $(u, a, ua) \in \delta$.
  - pour chaque $u \in \Sigma^*$ tel que $|u| = k-1$, et tout $a \in \Sigma$, en posant u' la restriction de u à ses $k-1$ dernières lettres alors : $(u, a, u'a) \in \delta$, et cette transition incrémente le compteur de ua.

Il est également possible de calculer dans le même temps les statistiques uniformes de tous les ordres inférieurs ou égaux à k, en ajoutant les incrémentations des compteurs additionnels correspondants (voir la figure 2.1). Ces procédures nécessitent à priori une mémoire exponentielle en k puisqu'il y a un nombre exponentiel de compteurs (un par coordonnée du vecteur calculé). Nous pouvons réduire cette complexité spatiale vis-à-vis de k, par exemple en effectuant une passe par compteur. La dépendance en k est alors réduite à une complexité spatiale linéaire en k ; l'aspect linéaire en la taille du mot est conservé, mais nous perdons l'aspect en ligne de l'algorithme.

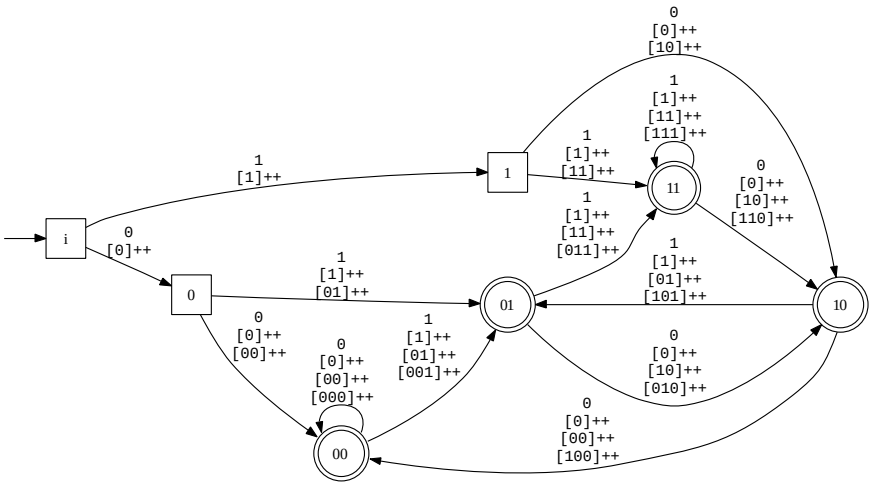


FIG. 2.1 – ustat_1, ustat_2 *et* ustat_3 *en temps linéaire sur l'alphabet* $\{0,1\}$.

Exemple 2.7

Le calcul de $\mathsf{ustat}_1(w)$, $\mathsf{ustat}_2(w)$ et $\mathsf{ustat}_3(w)$ en temps linéaire est donné pour l'alphabet $\{0,1\}$ par la figure 2.1.

L'étiquetage des arêtes sur cette figure a le sens suivant : la première ligne de l'étiquette est la lettre lue, les lignes suivantes sont de la forme $[u]++$ pour signifier que lors de cette transition, le compteur associé à u (un mot de longueur inférieure à k) est incrémenté.

Le résultat final X du calcul est donc un vecteur d'entiers dont la normalisation est le vecteur stat recherché.

Rappelons que la normalisation d'un vecteur (non nul) est simplement la multiplication scalaire de ce vecteur par l'inverse de sa norme : $\left[\left[\overrightarrow{X}\right]\right][u] = \dfrac{\overrightarrow{X}[u]}{\sum_{v\in\Sigma^k}\overrightarrow{X}[v]}$. ♪

Calcul Approché

Comme déjà vu dans les travaux de [Fischer et al., 2006], ces statistiques sur les mots peuvent être approchées en temps constant.

Algorithme 1 : Échantillonnage d'une Statistique de Mot.

Données : Un mot w, un nombre d'échantillons N, et une taille d'échantillon k
Sorties : $\widehat{\mathsf{ustat}}_k^N(w)$
$max := |w| - k$;
X le tableau initialisé à 0 pour tout $u \in \Sigma^k$;
pour *i de* 1 *à* N **faire**
  Choisir uniformément $j \in \{1, max\}$;
  $u = w\,[\{j, j+k\}]$;
  $X[u] := X[u] + 1/N$;
fin
retourner X

En effet, pour chaque itération de l'algorithme 1, l'incrémentation de $X[u]$ est une variable de Bernoulli $X_u \in \{0, 1\}$ avec probabilité de succès $p_u = \mathsf{ustat}(w)[u]$. Les N tirages aléatoires étant indépendants, la borne de Chernoff peut donc être appliquée sur chaque coordonnée : $\Pr[X_u < p_u - \varepsilon] + \Pr[X_u > p_u + \varepsilon] < 2e^{-2N\varepsilon^2}$. La taille de l'échantillon N est alors choisie pour que la probabilité d'erreur de l'algorithme soit inférieure à $1/3$ (naturellement généralisable à toute constante strictement positive).

Théorème 2.8 (Corollaire 3.1 de [Fischer et al., 2006])

Pour tout k fixé, et tout mot w de Σ^*, il existe un échantillonneur qui approxime $\mathsf{ustat}_k(w)$ *en temps de calcul* $\mathcal{O}(k^3(\ln|\Sigma|)|\Sigma|^{k/2})$ ★

Démonstration (sketch) : L'échantillonneur consiste à approximer ustat par le vecteur moyen de tirages élémentaires où un tirage élémentaire choisit un sous-mot aléatoire de longueur k, depuis une position uniformément distribuée. Par définition des vecteurs ustat_k, l'espérance de ce tirage est ustat_k. La convergence est donc garantie pour la norme $\mathbf{L}_1$ *i.e.*

$$\lim_{N\to\infty} \sum_{u\in\Sigma^k} \left| \mathsf{ustat}(w)[u] - \widehat{\mathsf{ustat}_k^N}(w)[u] \right| = 0$$

Dans [Fischer et al., 2006], ce résultat à été affiné en montrant que pour $N = \mathcal{O}(k^3|\Sigma|^{2k}\ln|\Sigma|)$ itérations de ce tirage élémentaire, une sortie proche du vecteur complet ustat_k est obtenue avec grande probabilité. La proximité se mesure ici par la norme $\mathbf{L}_1$, la somme des valeurs absolues des différences sur l'ensemble des coordonnées ; proche signifie alors à moins de $\varepsilon = 1/k$; et grande probabilité signifie alors avec probabilité plus que $2/3$.[11]

$$\Pr_{N=\mathcal{O}(k^3|\Sigma|^{2k}\ln|\Sigma|)} \left[\sum_{u\in\Sigma^k} \left| \mathsf{ustat}(w)[u] - \widehat{\mathsf{ustat}_k^N}(w)[u] \right| > 1/k \right] \leq 2/3$$

□

La sortie de l'algorithme 1 ne dépend que de k et de Σ. L'approximation de ces statistiques se fait donc en temps dépendant seulement de Σ et k, mais pas de la taille du mot donné en entrée.

[11] facilement généralisable pour un $p < 1$ arbitraire

2.1.3 Inversion

Après avoir vu comment passer (de façon exacte ou approximative) d'un mot à son vecteur statistique, nous allons nous intéresser à l'opération inverse consistant à retrouver des mots possédant une statistique donnée.

Définition 2.9 (Inversion de statistique)
Soit $k \in \mathbb{N}^+$ et $\lambda = \left(\lambda^1, ..., \lambda^{|\Sigma|^k}\right)$ une statistique d'ordre k. On dit que w est un *inverse* de λ, quand $\mathsf{ustat}_k(w) = \lambda$. S'il existe un tel mot w, on dira que λ est inversible. ✓

Exemple 2.10 (Une statistique non inversible)
Soit $\Sigma = \{0, 1\}$ et posons $\lambda = (1/3, 2/3, 0, 0)$. λ ne possède aucun inverse. En effet, pour inverser λ, il faudrait la présence d'au moins deux occurrences de 01. Ceci impose la présence d'au moins une occurrence de 10, et contredit ainsi la valeur $\lambda[10] = 0$. ♪

Remarque 2.11
Il est toujours possible d'inverser une statistique uniforme : Pour tout $k > 0$, il existe un mot $w_k \in \Sigma^*$ de longueur $|w_k| = |\Sigma|^k + k - 1$ qui contient comme sous-mots tout mot sur Σ de longueur k. En effet, l'automate $\mathcal{A}^{\Sigma,k}_{Bruijn}$ sur lequel se déroule une exécution du calcul dans la proposition 2.6 (*page* 39) donne implicitement un générateur de mots de statistique uniforme : tout chemin eulérien recouvrant $Q' = \left\{w \in \Sigma^{k-1}\right\}$ produit un mot dont le ustat_k est uniforme. En faisant abstraction des étiquettes de la restriction de cet automate à Q', on reconnaît un graphe de Bruijn [de Bruijn, 1946, Good, 1946](de dimension k, sur $|\Sigma|$ symboles), ce qui garantit l'existence d'un tel chemin. En décrivant un chemin minimal, de l'état initial à Q', puis un chemin eulérien dans Q', on obtient des mots (minimaux) ayant un ustat uniforme. Par exemple pour $\Sigma = \{0, 1, 2\}$, $w = 001021122$, $k = 2$, et tout a,$b \in \Sigma$, $\mathsf{ustat}_2(w)[ab] = 1/9$.

Si l'inverse est cherché d'une taille particulière, les idées de [Kontorovich, 2004] peuvent être suivies pour réduire à nouveau le problème à l'existence d'un chemin eulérien dans un nouveau graphe. La notion d'inversion asymptotique est nouvellement introduite ici pour capturer l'aspect asymptotique de cette question. En réduisant ce problème à la résolution d'un système linéaire, on aura une décision en temps polynomial. Requérir des mots de taille croissante impose de suivre un grand nombre de boucles dans le graphe de Bruijn d'ordre k, ce qui fait que la proportion de ces boucles devient la seule chose significative d'un point de vue statistique.

Inversion asymptotique approchée

Définition 2.12 (inversion asymptotique)
Pour tout vecteur unitaire a de $\mathbb{R}^{\Sigma^k}$, on dira que a est asymptotiquement inversible, noté $Inv^{\infty}(a)$, s'il existe une suite d'instances (de mots) de taille croissante dont la suite des statistiques d'ordre k converge vers a pour la norme $\mathbf{L}_1$. ✓

L'existence d'une suite croissante suggère que les parties finies ne comptent que marginalement. La proposition suivante illustre qu'il est effectivement suffisant de se concentrer sur les boucles pour décider de l'inversion asymptotique.

Proposition 2.13
Sur l'alphabet binaire, $k = 2$, et $a = (a_1, a_2, a_3, a_4)$ un vecteur unitaire de réels, $Inv^{\infty}(a)$ si et seulement si $a_2 = a_3$. ⋆

Démonstration : Entre deux apparitions d'un motif 01, il existe nécessairement un sous-mot 10. Le nombre de 01 est toujours le même que le nombre de 10 plus ou moins un et donc asymptotiquement on impose $a_2 = a_3$. Réciproquement, la suite de mots $\left((00)^{\lfloor a_1.n \rfloor}(01)^{2.a_2.n}(11)^{a_4.n}\right)_{n\in\mathbb{N}}$ convient. □

Nous allons donner l'intuition de la réduction à un système linéaire par un exemple.

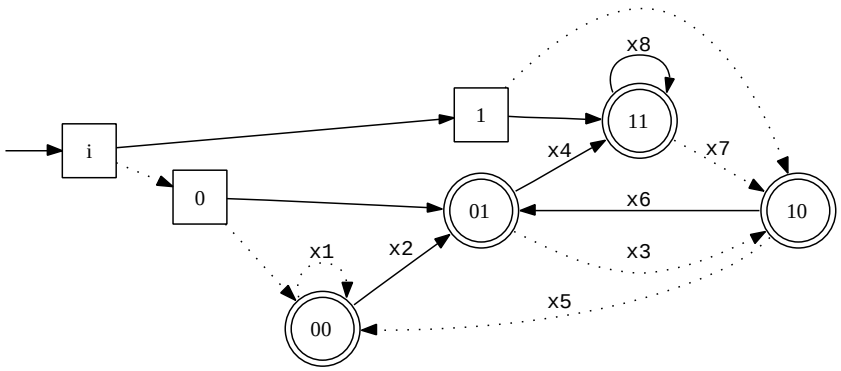


Fig. 2.2 – $\mathcal{A}^{\{0,1\},2}_{Bruijn}$

.

Exemple 2.14

Reprenons l'automate $\mathcal{A}^{\{0,1\},2}_{Bruijn}$ décrit pour le calcul du vecteur statistique d'un mot en représentant les arêtes 0 en pointillés, les arêtes 1 par des flèches pleines, et x_i représentant intuitivement la proportion du nombre de passages par cette arête dans l'exécution d'un long mot sur cet automate.

Essayons par exemple d'inverser (asymptotiquement) $a = (2/7, 1/7, 0, 4/7)$.
Si on peut effectivement le faire, alors il existe $x_1, \ldots, x_8$ dans $[0, 1]$ tels que

$$\sum_{i=1}^{8} x_i = 1$$

les statistiques sont respectées :

$$\begin{cases} 00 : x_1 + x_5 = 2/7 \\ 01 : x_2 + x_6 = 1/7 \\ 10 : x_3 + x_7 = \ \ 0 \\ 11 : x_4 + x_8 = 4/7 \end{cases}$$

et les flots sont conservés :

$$\begin{cases} x_2 = x_5 & \text{en l'état } 00 \\ x_3 + x_4 = x_2 + x_6 & \text{en l'état } 01 \\ x_4 = x_7 & \text{en l'état } 10 \\ x_3 + x_7 = x_5 + x_6 & \text{en l'état } 11 \end{cases}$$

Ce système ne possède pas de solution. Cette statistique peut pourtant être effectuée en suivant la suite d'états $i, 0, 00, 00, 01; 11; 11; 11; 11$. On retrouve alors le mot 00011111 dont on a déjà montré qu'il était un inverse de cette statistique avec l'exemple 2.2 (*page* 36) pour $k = 2$.

Par contre, asymptotiquement, on ne peut pas obtenir cette statistique.
En effet, $x_3 = x_7 = 0$, ce qui implique $x_4 = 0$ puis $x_3 + x_4 = 0$ et donc $x_2 = x_6 = 0$; or on voulait $x_2 + x_6 = 1/7$. On n'a donc pas de mot arbitrairement long possédant a comme statistique, et on peut aussi expliquer cela par le fait de pouvoir générer des 01 ; il est nécessaire de générer (à deux près) le même nombre de 10 ; on va donc s'écarter asymptotiquement de ce vecteur d'au moins 1/7. ♪

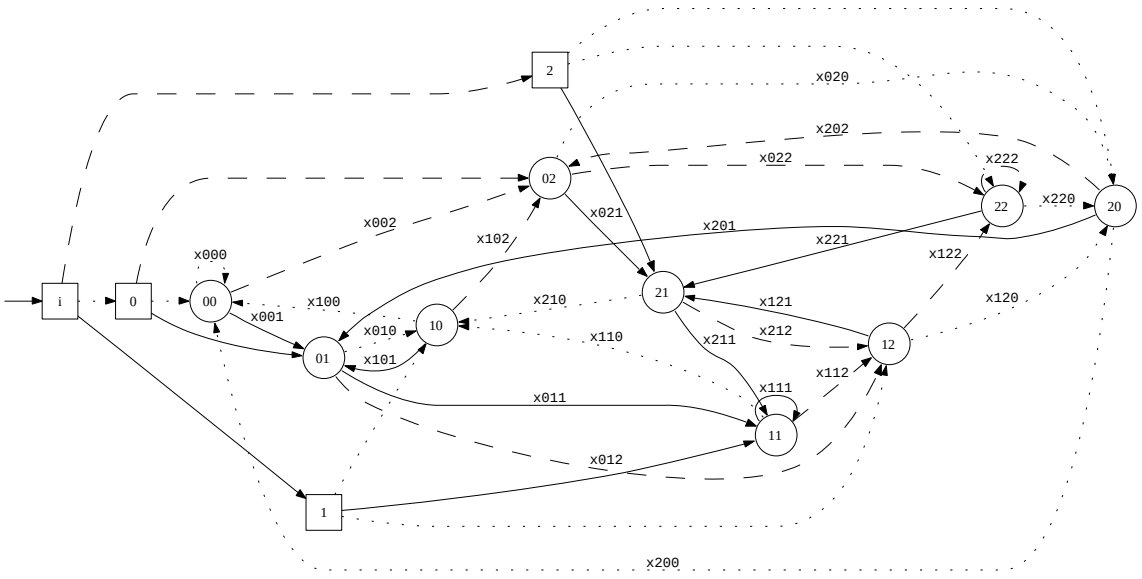


Fig. 2.3 – $\mathcal{A}^{\{0,1,2\},2}_{Bruijn}$

.

Proposition 2.15

Sur l'alphabet ternaire, $k = 2$, et $a = (a_1, a_2, a_3, \ldots, a_9)$ un vecteur unitaire de réels, $Inv^{\infty}(a)$ si et seulement si le système suivant possède une solution :

1. $\sum_i x_i = 1$ *où x_i désigne la fréquence d'utilisation de l'arête i.*
2. *les statistiques sont respectées,* i.e. *la somme des poids entrant dans un état correspond à la statistique recherchée pour ce sous-mot (l'étiquette de cet état).*

$$\begin{cases} x_{000} + x_{100} + x_{200} &= a_1 \quad (\textit{état } 00) \\ x_{001} + x_{101} + x_{201} &= a_2 \quad (\textit{état } 01) \\ x_{002} + x_{102} + x_{202} &= a_3 \quad (\textit{état } 02) \\ x_{010} + x_{110} + x_{210} &= a_4 \quad (\textit{état } 10) \\ \qquad\qquad \ldots & \\ x_{022} + x_{122} + x_{222} &= a_9 \quad (\textit{état } 22) \end{cases}$$

3. *Les flots sont conservés,* i.e. *la somme des poids entrant dans un état est égal à la somme des poids qui en sortent.*

$$\begin{cases} x_{100} + x_{200} &= x_{001} + x_{002} \quad (\textit{état } 00) \\ x_{001} + x_{101} + x_{201} &= x_{010} + x_{011} + x_{012} \quad (\textit{état } 01) \\ \qquad\qquad \ldots & \\ x_{022} + x_{122} &= x_{220} + x_{221} \quad (\textit{état } 22) \end{cases}$$

$\star$

Chaque valeur de statistique génère une contrainte ; chaque état spécifie aussi une contrainte pour la conservation de son flot. [12] Pour décider si une statistique peut être approchée de la sorte, il nous suffit donc de connaître l'ensemble des solutions du système ainsi généré pour le vecteur a associé. Si l'on tient compte des arêtes permettant d'arriver dans la partie des états cerclés, on retrouve un problème d'inversion exact. La partie non cerclée influence au plus $k-1$ sous-mots de longueur k. Pour des mots de plus en plus longs, l'impact de cette phase « initialisation » sur la statistique tend vers zéro[13].

[12] Par exemple pour l'état 02, on génère la contrainte $x_{002} + x_{102} + x_{202} = x_{020} + x_{021} + x_{022}$.

[13] $\leq (k-1)/(n-k+1)$ pour un mot de longueur n

Exemple 2.16 (Inversion à l'ordre 3 sur l'alphabet binaire)
A partir de l'automate $\mathcal{A}_{Bruijn}^{\{0,1,3\},2}$ précédemment considéré pour le calcul de ustat pour l'alphabet binaire et $k = 3$, on définit la matrice d'adjacence suivante possédant une variable par arête de l'automate.

$$M := \begin{bmatrix} x_1 & 0 & 0 & 0 & x_2 & 0 & 0 & 0 \\ x_3 & 0 & 0 & 0 & x_4 & 0 & 0 & 0 \\ 0 & x_5 & 0 & 0 & 0 & x_6 & 0 & 0 \\ 0 & x_7 & 0 & 0 & 0 & x_8 & 0 & 0 \\ 0 & 0 & x_9 & 0 & 0 & 0 & x_{10} & 0 \\ 0 & 0 & x_{11} & 0 & 0 & 0 & x_{12} & 0 \\ 0 & 0 & 0 & x_{13} & 0 & 0 & 0 & x_{14} \\ 0 & 0 & 0 & x_{15} & 0 & 0 & 0 & x_{16} \end{bmatrix}$$

On engendre les équations du système par les contraintes suivantes :

- Les variables sont dans [0,1] et se somment à 1 : $0 \leq x_i \leq 1$ pour $i \in \left\{1, |\Sigma|^{k+1}\right\}$, $\sum_{i=1}^{|\Sigma|^{k+1}} x_i = 1$.

- Les flots sont préservés : la somme des valeurs entrantes dans un état est aussi la somme de ses valeurs sortantes.

$$x_1 + x_3 = x_1 + x_2,\ x_5 + x_7 = x_3 + x_4,\ x_9 + x_{11} = x_5 + x_6,\ x_{13} + x_{15} = x_7 + x_8,$$
$$x_2 + x_4 = x_9 + x_{10},\ x_6 + x_8 = x_{11} + x_{12},\ x_{10} + x_{12} = x_{13} + x_{14},\ x_{14} + x_{16} = x_{15} + x_{16}$$

- Les masses sont respectées : la distribution stationnaire est celle qui est recherchée. La somme des valeurs entrantes d'un noeud doit correspondre à la statistique recherchée (pour ce sous-mot) *i.e.* la somme sur une ligne i doit redonner la valeur de $a[i]$. Pour le vecteur $\lambda := \left(\frac{1}{2}, \frac{1}{10}, \frac{1}{10}, 0, \frac{1}{10}, 0, 0, \frac{1}{5}\right)$ on obtient les équations :

$$x_1 + x_2 = \frac{1}{2},\ x_3 + x_4 = \frac{1}{10},\ x_5 + x_6 = \frac{1}{10},\ x_7 + x_8 = 0,$$
$$x_9 + x_{10} = \frac{1}{10},\ x_{11} + x_{12} = 0,\ x_{13} + x_{14} = 0,\ x_{15} + x_{16} = \frac{1}{5}.$$

En résolvant le système, on obtient les solutions suivantes :

$$\begin{aligned} x_5 = \frac{1}{10},\ x_9 = \frac{1}{10}, x_{16} = \frac{1}{5}, \\ x_1 = \frac{1}{2} - x_3, \\ x_4 = \frac{1}{10} - x_3, \\ x_6 = x_7 = x_8 = x_{10} = x_{11} = x_{12} = x_{13} = x_{14} = x_{15} = 0 \\ 0 \leq x_2 = x_3 \leq \frac{1}{10}. \end{aligned}$$

dont une solution est :

$$S = \begin{bmatrix} 2/5 & 0 & 0 & 0 & 1/10 & 0 & 0 & 0 \\ 1/10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/5 \end{bmatrix}$$

On vérifie ainsi que l'on obtient la propriété de distribution stationnaire : $S \cdot a = a$. ♪

A l'aide d'une solution, on peut également retrouver une manière d'inverser asymptotiquement cette statistique. En effet, il suffit de suivre les arêtes avec une proportion d'utilisation égale à la valeur calculée par la variable correspondante. On procède comme suit. Tant que l'on a des valeurs positives dans S, on cherche une boucle simple[14], qui ne contient que des arêtes de poids strictement positifs. Le poids minimal de ses arêtes est retiré de chacune d'elles et retenons que le nombre d'itérations de cette boucle sera proportionnel à ce poids multiplié par sa longueur. On achève donc la procédure avec un certain nombre de boucles simples pondérées, que l'on va effectuer dans un ordre arbitraire, avec les proportions voulues.

[14]de longueur maximale

Exemple 2.17

Considérons l'exemple précédent. Trois boucles sont à considérer :

- la boucle $000 \to 001 \to 010 \to 100 \to 000$ avec un poids $4 \cdot 1/10$
- la boucle $111 \to 111$ avec un poids $1/5$
- la boucle $000 \to 000$ avec un poids $2/5$

Par une utilisation de ces boucles, proportionnellement à ces poids, on explicite la suite de mots $w_n = (000)^{\lfloor 2n/5 \rfloor}.(0001)^{4\cdot\lfloor n/10 \rfloor}.(111)^{\lfloor n/5 \rfloor}$, qui vérifie bien

$$\mathsf{ustat}_2(w_n) \xrightarrow{n\to\infty} a = \left(\frac{1}{2}, \frac{1}{10}, \frac{1}{10}, 0, \frac{1}{10}, 0, 0, \frac{1}{5}\right)$$

♪

Théorème 2.18

Pour tout vecteur a dans $\mathbb{R}^{|\Sigma|^k}$, on sait décider $Inv^\infty(a)$ en temps polynomial en $|\Sigma^k|$. ★

En effet, la construction du système $\mathcal{S}$ de contraintes linéaires est généralisée dans le cas général comme suit :

- Les $|\Sigma|^{k+1}$ variables sont dans [0,1] et se somment à 1.
- Les flots sont préservés : la somme des valeurs sortantes dans un état est aussi la somme de ses valeurs entrantes :

$$\forall q, \sum_{(q,e,q')\in\delta} x_{(q,e,q')} = \sum_{(q'',e',q)\in\delta} x_{(q'',e',q)}$$

- Les masses sont respectées : la somme des poids entrants[15] correspondent à la fréquence d'occurrence de l'état q de $\mathcal{A}^{\Sigma,k}_{Bruijn}$. Pour $int()$: $\Sigma^k \to \{1, |\Sigma|^k\}$ la fonction qui associe un mot à sa position dans l'ordre lexicographique,

$$\sum_{(q',e,q)\in\delta} x_{(q',e,q)} = a_{int(q)}$$

[15] et donc aussi des poids sortants par le point précédent

Une fois le système $\mathcal{S}$ associé au vecteur a, on a :

1. Si $\mathcal{S}$ possède une solution, la méthode fournit un ensemble de boucles simples munies d'une pondération. En construisant une suite de mots (en itérant ces boucles simples proportionnellement à leurs poids), on explicite une suite de mots dont les statistiques convergent vers a pour la norme $\mathbf{L}_1$. Ceci montre alors que a est asymptotiquement inversible.
2. Si a est asymptotiquement inversible, soit w_i une suite croissante de mots tels que $\mathsf{ustat}_k(w_i) \xrightarrow{n\to\infty} a$. En rajoutant au plus k lettres à chaque w_i, on peut supposer que chaque exécution de w_i sur l'automate $\mathcal{A}^{\Sigma,k}_{Bruijn}$ décrit une boucle $\hat{w}_i$. Cette boucle se décompose en un ensemble de boucles simples (éventuellement imbriquées). Les statistiques produites par ces boucles simples sont indépendantes de l'ordre dans lequel on les a parcourues : l'idée est qu'un état q de $\mathcal{A}^{\Sigma,k}_{Bruijn}$ encode les dernières k lettres lues et ainsi toute l'information du point de vue de l'impact statistique. Puisque la distribution des fréquences d'utilisation des arêtes vit dans un compact, on peut donc extraire une sous-suite convergente $w'_i = \sigma(\hat{w}_i)$ dont la fréquence d'utilisation des arêtes converge vers $X = (x_{q,a})_{q\in\Sigma^k, a\in\Sigma}$. Par construction, X satisfait toutes les contraintes de $\mathcal{S}$:
   - X est unitaire.
   - les flots sont préservés puisqu'on ne considère que des boucles.
   - les masses sont respectées puisqu'on extrait d'une suite produisant des statistiques qui convergent en norme $\mathbf{L}_1$ vers a.

   On confirme donc que $\mathcal{S}$ possède une solution.

Finalement, la résolution de $\mathcal{S}$ est celle d'un système d'équations linéaires, polynomial en $\mathcal{S}$, et donc polynomial en $|\Sigma^k|$. J'ai par ailleurs implémenté la création automatique de ce système puis sa résolution en Maple.

2.2 Statistiques d'Arbres

Pour étendre la définition des vecteurs statistiques aux arbres, on doit d'abord clarifier quelles sous-structures sont dénombrées. Comme pour les mots, la statistique d'ordre 1 d'un arbre est le vecteur de fréquence de ses étiquettes. Pour les statistiques d'ordres supérieurs, commençons par considérer l'encodage de fcns étendu d'un arbre XML, *i.e.* l'arbre étendu [16] dans lequel on représente le premier fils d'un noeud comme son successeur gauche, et le frère droit d'un noeud comme son successeur droit. Cet encodage possède l'avantage de conserver l'ensemble des noeuds de l'arbre n-aire, et d'en avoir une représentation binaire avec le même nombre d'arêtes.

Un segment de longueur k dans l'encodage fcns étendu, suit soit un successeur gauche (0), soit un successeur droit (1). Son type u est un mot binaire de longueur $k-1$ décrivant la séquence des successeurs.

Le vecteur de statistique squelettique gstat d'un arbre est un vecteur unitaire de dimension 2^{k-1} : chaque coordonnée représente un type et prend pour valeur la fréquence[17] de ce type. La statistique squelettique d'un arbre ne considère pas les étiquettes, et ne concerne donc que la structure hiérarchique de l'arbre.

Chaque segment de longueur k possède k étiquettes et il peut bien sûr exister plusieurs segments de même type mais d'étiquettes différentes. On peut donc affiner une statistique squelettique en précisant, pour chaque type (présent) u, la distribution $\mathsf{stat}_k[u]$ des différentes séquences d'étiquettes pour ce type. Notons que cette distribution est exactement de même nature qu'une statistique de mots d'ordre k. La matrice stat_k contient tous les $\mathsf{stat}_k[u]$. C'est donc une matrice à 2^{k-1} lignes et Σ_S^k colonnes dans laquelle chaque ligne est de norme unitaire.

[16] l'arbre est de degré au plus deux

[17] Le nombre d'occurrences de ce type divisé par le nombre total de segments de longueur k.

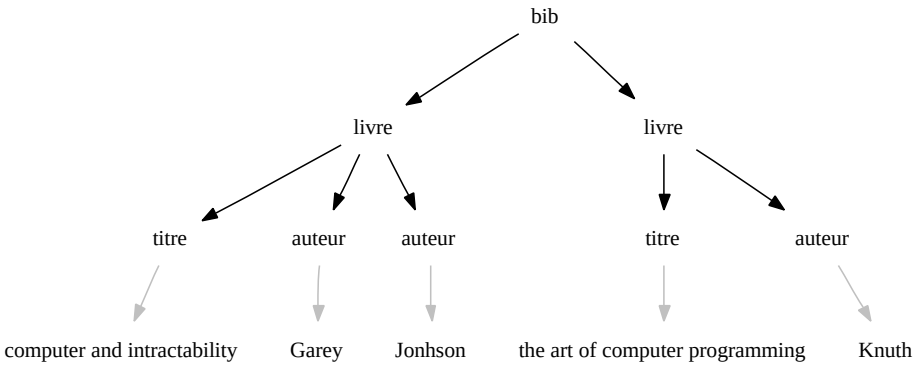


FIG. 2.4 – *Une instance-cible : $t' = \mathscr{T}(t)$*

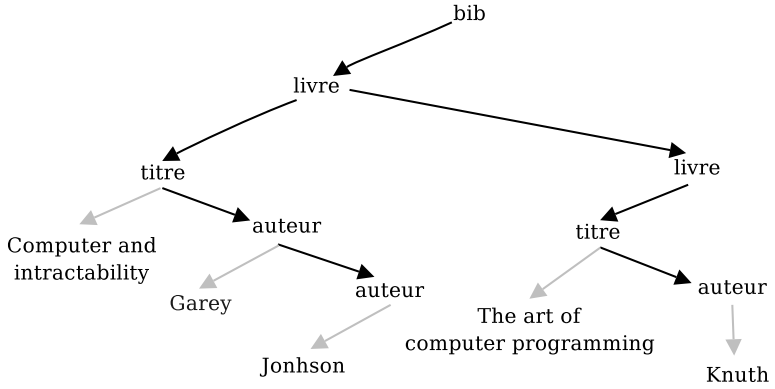


FIG. 2.5 – *t'' : l'encodage de Rabin de t'*

Définition 2.19

Pour t un arbre, $\mathsf{ustat}_k(t)$ est une matrice unitaire à 2^{k-1} lignes et Σ^k colonnes. Pour $u \in \{0,1\}^{k-1}$ un type, et $a = a_1 \dots a_k$ une séquence d'étiquettes,

$$\mathsf{ustat}_k(t)[u][a] = \mathsf{gstat}_k(t)[u] \times \mathsf{stat}_k(t)[u][a]$$

✓

Exemple 2.20

Par souci de concision, nous ne donnerons que les valeurs non nulles du vecteur stat en désignant la coordonnée par la séquence des premières lettres des k étiquettes considérées. Les ordonnées (les types) sont données dans l'ordre lexicographique et rappelées à gauche des vecteurs.

Dans l'encodage fcns étendu de l'arbre de la figure 2.4 (*page* 53), on observe trois arêtes gauches (premier fils) et quatre arêtes droites (frère droit). La statistique squelettique

d'ordre 2 de l'arbre t de la figure 2.4, noté comme un vecteur colonne dont on rappelle en colonne de gauche le type de segment considéré : $\mathsf{gstat}_2(t) = \begin{array}{c|l} 0 & 3/7 \\ 1 & 4/7 \end{array}$

Parmi les arêtes gauches, deux sont étiquetées par 'titre'-'auteur' et une par 'bib'-'livre', correspondant à la première ligne de $\mathsf{stat}_2(t)$ et $\mathsf{ustat}_2(t)$. Les arêtes droites sont de trois sortes : celle 'auteur'-'auteur', 'livre'-'livre' et les deux étiquetées 'titre'-'auteur'. Avec ces notations, types en lignes et étiquettes en colonnes, on a :

$\mathsf{stat}_2(t)$	aa	bl	ll	lt	ta
0	0	1/3	0	2/3	0
1	1/4	0	1/4	0	1/2

$\mathsf{ustat}_2(t)$	aa	bl	ll	lt	ta
0	0	1/7	0	2/7	0
1	1/7	0	1/7	0	2/7

La statistique uniforme de l'arbre en l'arête droite 'titre'-'auteur' est :

$$\mathsf{ustat}(t)[1][ta] = \mathsf{gstat}(t'')[1] \times \mathsf{stat}(t)[1][ta] = 4/7.1/2 = 2/7.$$

On retrouve au numérateur le nombre d'arêtes de ce genre présentes dans t, et au dénominateur le nombre d'arêtes de l'arbre. ♪

2.2.1 Lien entre Distances et Statistiques

Nous devons montrer que la représentation matricielle est correcte et robuste, *i.e.*

– si $\mathsf{dist}(t,t') \leq f(\varepsilon)$ alors $|\mathsf{ustat}(t) - \mathsf{ustat}(t')| \leq c \cdot \varepsilon$,

– si $|\mathsf{ustat}(t) - \mathsf{ustat}(t')| \leq g(\varepsilon)$ alors $\mathsf{dist}(t,t') \leq c' \cdot \varepsilon$.

Le premier point (correction) est facile mais le second (robustesse) est plus difficile.

Correction

Pour chaque opération élémentaire, on explicite l'erreur faite sur les matrices, puis on conclut par induction sur la distance absolue.

Fait 2.21

Soit $n = \Omega(\frac{1}{\varepsilon}) = |t| = |t'|$. *Si* $\mathsf{dist}(t,t') \leq 1$ *alors* $|\mathsf{ustat}_k(t) - \mathsf{ustat}_k(t')| \leq 3(2^k-1)/(n-1)$.

Effectuer une opération élémentaire de distance d'édition avec déplacements conduit à changer sur l'arbre t le voisinage autour d'au plus trois noeuds. Le cas le plus défavorable est celui du déplacement : il modifie les statistiques à l'endroit où on déconnecte le sous-arbre, à la place où on réinsère ce sous-arbre, et au sommet du sous-arbre déplacé. Pour chacun de ces trois voisinages, on modifie au plus une statistique dans laquelle le noeud considéré est une feuille ; au plus deux statistiques dans lesquelles le noeud considéré est un noeud interne de profondeur maximale ; au plus 2^{k-1} statistiques dans lesquelles le noeud considéré est la racine, etc ... Au total, on a bien modifié moins de $\sum_{i=0}^{k-1} 2^i = 2^k - 1$ statistiques. En itérant ce fait $\varepsilon.n$ fois, on obtient :

Proposition 2.22

Soit $n = \Omega(\frac{1}{\varepsilon}) = |t| = |t'|$. *Si* $\mathsf{d}(t,t') \leq \varepsilon$ *alors*

$$|\mathsf{ustat}_k(t) - \mathsf{ustat}_k(t')| \leq 3\varepsilon \times (2^{1/\varepsilon} - 1).$$

$\star$

Robustesse

Le raisonnement général pour montrer que des arbres (de même taille) ayant des statistiques proches sont proches (pour la distance d'édition avec déplacements) se résume comme suit :

1. Pour un arbre t, la matrice statistique M se décompose en deux sous-matrices : $M = A + L$ où A correspond aux segments terminant sur un noeud interne de t et L à ceux terminant sur une feuille.
2. Nous compressons ensuite l'encodage fcns de l'arbre, en suivant la proposition de [Fischer et al., 2006]. Une étape élémentaire consiste alors à compresser les feuilles, *i.e.* à les supprimer en modifiant l'étiquette de leur père pour y faire apparaître la structure de ces feuilles[18] compressées. Un exemple d'une telle compression est donné par la figure 2.6 (*page* 57).
3. Nous construisons inductivement les matrices M^i_{2k-i} pour $1 \leq 0 \leq k$. M^0_{2k} est la matrice statistique d'ordre $2k$ de l'arbre t ; M^1_{2k-1} la matrice statistique (d'ordre $2k-1$) de t après une étape de compression, M^2_{2k-2} la matrice statistique après deux étapes de compression, et ainsi de suite jusqu'à M^k_k. De la même manière que M, les matrices M^i_{2k-i} se décomposent en $M^i_{2k-i} = A^i_{2k-i} + L^i_{2k-i}$. Nous montrons que lors de cette compression, les $M^i_{2k_i}$ restent proches. Soit v un noeud compressé, deux cas sont possibles :
   - Pour un v loin du squelette, A_{2k-i+1} représente les mêmes segments que A^i_{2k-i} (sauf peut-être un) *i.e.* si les M_i sont initialement proches, alors les L_i sont également proches.
   - Pour v à distance un du squelette, v est alors compressé sur le squelette. Le nombre de feuilles gauches (respectivement droites) de ce type est en proportion d dans l'arbre. Remarquons que A^{i+1}, et donc également L^{i+1} change alors proportionnellement à ce d et nous nous ramenons alors à estimer le nombre de feuilles à distance 1 du squelette. Pour ce faire, nous estimons le nombre $A_k[u]$ par deux techniques différentes. D'abord par la définition directe, puis en introduisant les matrices N (correspondant aux noeuds possédant à la fois une feuille gauche et une feuille droite). La différence $N_k[u] - A_k[u]$ approxime en effet le nombre de

[18] respectivement de ces sous-arbres pour plusieurs étapes de compression

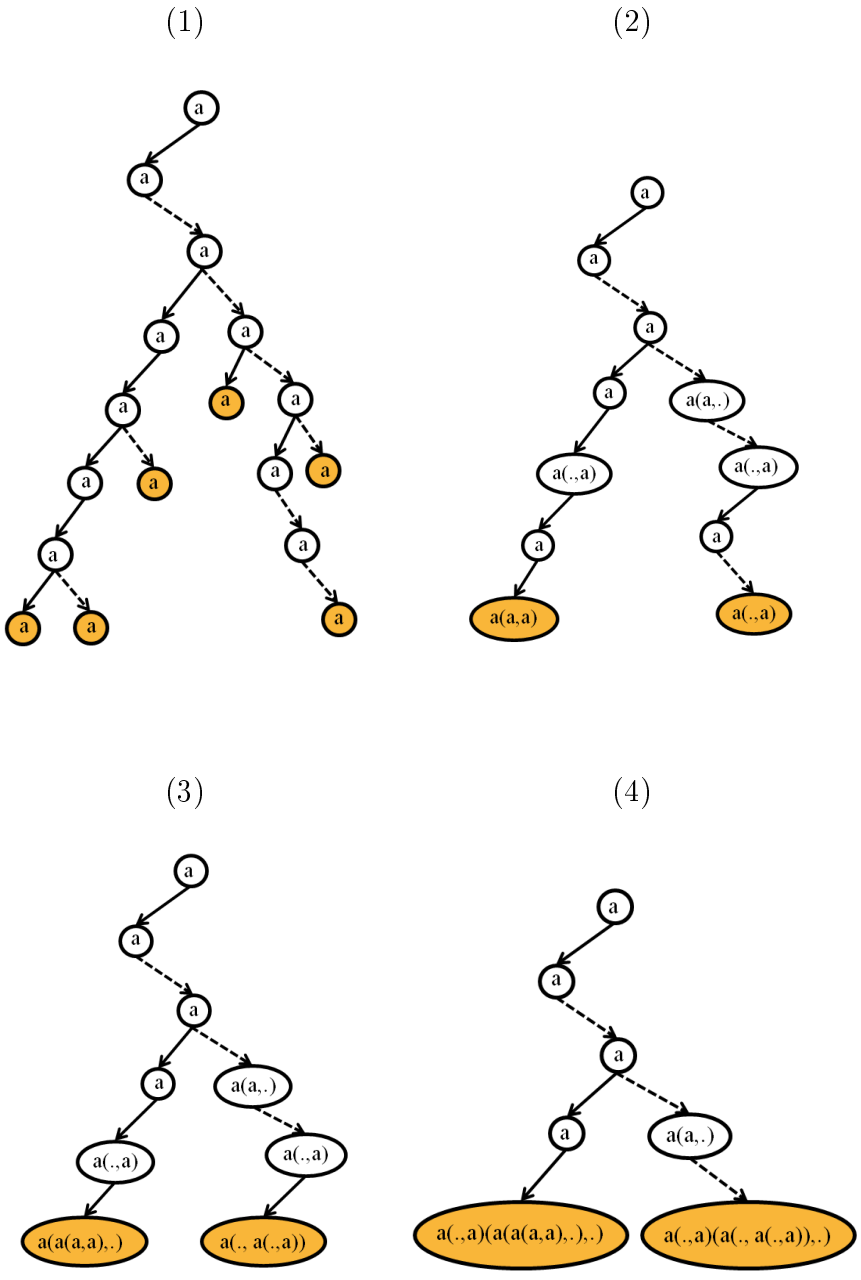


Fig. 2.6 – *Exemple : Trois étapes de compression*

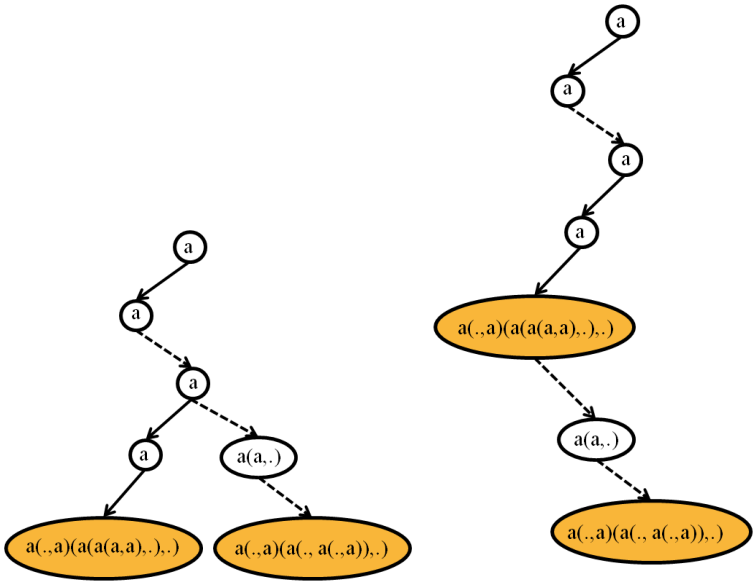


Fig. 2.7 – *Exemple de linéarisation*

feuilles à distance 1 du squelette. Nous estimons alors A_k^{i+1} et L_k^{i+1} depuis A_{k+1}^i et L_{k+1}^i, en suivant les équations inductives suivantes :

$$\begin{aligned}
A_k[u] &= A_{k+1}[0.u] + A_{k+1}[1.u] + \varepsilon \\
L_k[u] &= L_{k+1}[0.u] + L_{k+1}[1.u] \\
N0_k[u] &= A_{k+1}[u.1] + L_{k+1}[u.0] \\
N1_k[u] &= A_{k+1}[u.0] + L_{k+1}[u.1] \\
N_k[u] &= N0_k[u] + N1_k[u]
\end{aligned}$$

Par une induction sur le nombre d'étapes de compression, nous montrons qu'un M_{2k-i}^i (pour t) reste proche du $M_{2k-i}^{\prime i}$ (pour t') pour tout i de 1 à k.

4. Après k étapes d'induction, nous obtenons M_{2k}^k, et peu ($\leq \varepsilon.n$) de noeuds ont toujours à la fois un fils gauche et un fils droit. Linéariser le squelette, (voir la figure 2.7 (*page* 58)) coûte alors au plus $\varepsilon.n$ déplacements : la dégradation de la précision est une fraction constante de la taille des arbres.

5. Les versions linéarisées des deux arbres induisent des statistiques qui sont des statistiques de mots sur un alphabet plus grand (de taille 2^k). Nous utilisons alors la robustesse sur les mots — le théorème 2.4 (*page* 38) — pour en déduire la robustesse sur les arbres.

2.2.2 Calcul Approché

Algorithme 2 : Echantillonage des Statistiques d'un Arbre

Données : Un arbre t, un nombre d'échantillons N, et une taille d'échantillon k
Résultat : Une Approximation de $\mathsf{ustat}_k(t)$
$j := 0$;
tant que $j < N$ **faire**
  Choisir un noeud i uniformément ;
  Π_i^k : l'ensemble des chemins de racine i de longueur $k-1$;
  /* des chemins dans l'encodage fcns étendu de t */
  si $\Pi_i^k \neq \emptyset$ **alors**
    pour *chaque* π_j *de* Π_i^k **faire**
      Renvoyer π_j ;
      /* i.e. renvoyer son type $\in \{0,1\}^{k-1}$ et son étiquette $\in \Sigma^k$*/
    fin
    j :=j+1;
    /* des nouveaux chemins sont obtenus */
  fin
fin
retourner *la matrice moyenne des chemins (étiquetés) renvoyés* ;

En générant suffisamment de segments aléatoires de l'arbre, l'algorithme 2 (*page* 59) permet d'approcher la matrice statistique pour la norme $\mathbf{L}_1$. Une étape consiste à générer un segment, c'est-à-dire un type (mot de $\{0,1\}^{k-1}$) et une séquence d'étiquettes (mot de Σ^k) ; l'algorithme va alors approcher la matrice statistique en faisant la moyenne des segments renvoyés. La robustesse de l'approximation vient du fait que l'espérance de renvoyer un couple (type,étiquette) donné est proportionnelle au nombre de chemins de l'arbre possédant ce type et ces étiquettes. Pour un couple donné, la convergence pour ce couple (type,étiquette) particulier est alors une application directe de la borne de Chernoff. L'application d'une « union bound » étant l'approximation précédente à la matrice complète.

Chaque passage dans la boucle génère un chemin (cas Π_i^k non vide) ou n'en produit pas (Π_i^k vide). Le second cas n'apparaît que si on tire un noeud qui n'est racine d'aucun chemin de longueur $k-1$; La probabilité de cet évènement est majorée par $\frac{k-1}{k}$, *i.e.* la probabilité de ne rien générer en p boucles est exponentiellement petite en p et ne dépend que de k. Pour suffisamment d'itérations, la probabilité que l'algorithme réussisse N tirages peut donc être rendue arbitrairement proche de 1.

2.2.3 Inversion

A nouveau on peut s'intéresser au problème inverse : étant donnée une matrice de statistiques, on voudrait décider s'il existe un arbre possédant de telles statistiques. Nous allons commencer par nous restreindre à l'inversion de gstat, *i.e.* on veut décider s'il existe une suite d'arbres (de taille croissante) dont les statistiques squelettiques suivent un vecteur donné.

Proposition 2.23

Toute statistique squelettique d'ordre 2 est inversible. ⋆

En effet, une statistique squelettique d'ordre 2 est du type $\mathsf{gstat}_2 = \begin{matrix} 0 \\ 1 \end{matrix} \begin{pmatrix} x \\ 1-x \end{pmatrix}$. Pour une statistique rationnelle, posons $m = \mathsf{pgcd}\,(x, 1-x)$. En considérant l'arbre filiforme à mx arêtes gauches suivies de $m(1-x)$ arêtes droites, on obtient une inversion exacte. Puisque cela vaut pour n'importe quel multiple de m, ceci montre aussi l'inversion (exacte) asymptotique de tout vecteur rationnel. Un vecteur réel peut être approché par une suite de vecteurs rationnels et, par diagonalisation, on obtient que tout vecteur est asymptotiquement inversible.

Dès l'ordre 3, des contraintes — du type de celles impliquant 01 à l'ordre 2 sur les mots — apparaissent naturellement pour les statistiques squelettiques d'arbres :

Proposition 2.24

Sur l'alphabet binaire, $k = 2$, et $X = (a_1, a_2, a_3, a_4)$ un vecteur unitaire de réels,
$Inv^{\infty}\,(a_1, a_2, a_3, a_4)$ si et seulement si $((a_2 \leq a_1 + a_3) \wedge (a_3 \leq a_2 + a_4))$. ⋆

Démonstration : Supposons que $a_2 \leq a_1 + a_3$ et $a_3 \leq a_2 + a_4$. Posons $x_3 = min(a_2, a_3)$, $x_2 = a_2 - x_3$, $x_4 = a_3 - x_3$, $x_1 = a_1 - x_2$, et $x_5 = a_4 - x_4$ et considérons des arbres du type de l'arbre de la figure 2.8 (*page* 61) de tailles croissantes. Le nombre de statistiques erronées, par rapport à l'inversion exacte, est inférieur à $4k$ (les raccordements entre la phase x_i et la phase x_{i+1}), et donc la suite de vecteurs statistiques pour de tels arbres converge bien vers (a_1, a_2, a_3, a_4).

Réciproquement, supposons $Inv^{\infty}\,(a_1, a_2, a_3, a_4)$. Deux statistiques de type 01 ne peuvent être connectées que par une arête de type 00 (comme dans la partie x_2 de

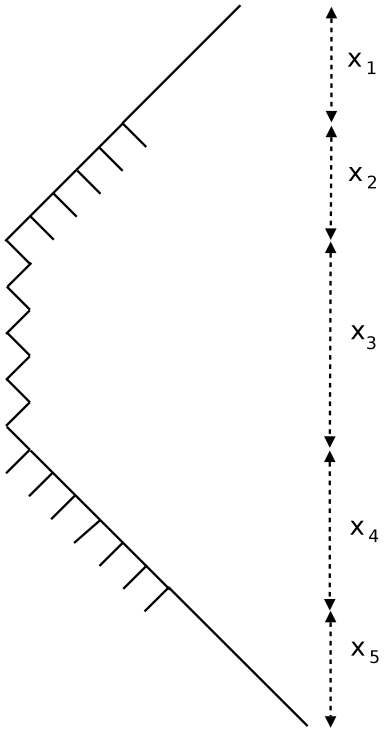


Fig. 2.8 – *Inversion asymptotique de statistique squelettique d'arbres pour $k = 3$.*

$$\begin{array}{ll} 001 : a_2 & \left[\begin{array}{ccc} 2a_1 & (a_3, a_5) & a_1 \\ a_2 & a_6 & (a_2, a_5) \\ a_2 & (a_3, a_6) & (a_7, a_6) \\ a_7 & (a_3, a_6) & (a_2, a_3) \\ a_7 & a_3 & (a_4, a_7) \\ a_8 & 2a_8 & (a_6, a_4) \end{array}\right] \\ 010 : a_3 & \\ 011 : a_4 & \\ 100 : a_5 & \\ 101 : a_6 & \\ 110 : a_7 & \end{array}.$$

Tab. 2.1 – *Compatibilité de statistiques d'ordre 4*

la figure 2.8) ou par une arête de type 10 (partie x_3). On a donc bien à la limite $a_2 \leq a_1 + a_3$. Symétriquement, connecter deux statistiques de type 10 se fait comme dans les parties x_3 ou x_4 de la figure 2.8 *i.e.* $a_3 \leq a_2 + a_4$. □

Pour $k = 4$, combiner deux statistiques de la colonne gauche de la table 2.1 induit d'engendrer au moins un élément de sa partie droite.

Conjecture 2.25

Pour un alphabet fini, $k > 0$ un entier, et X une matrice unitaire de réels de dimension $2^{k-1} \times |\Sigma|^k$, $Inv^{\infty}(X)$ est décidable en temps polynomial en la dimension de l'espace statistique.

Une généralisation de ce résultat aux arbres pourrait être de commencer par montrer qu'une statistique squelettique est inversible si et seulement si un système linéaire est satisfaisable. Cette première étape nécessite l'adaptation des graphes de Bruïjn. La construction formelle de l'hyper-graphe, dont les sommets représentent les matrices statistiques « canoniques » pourrait constituer le point de départ d'une généralisation en ce sens. La seconde étape est d'étendre le raisonnement des statistiques squelettiques en tenant compte des étiquettes. Une matrice statistique inversible devra vérifier les contraintes issues de la cohérence de ses étiquettes, en plus des contraintes de son aspect squelettique. Les contraintes supplémentaires sont typiquement celles trouvées dans le cas des mots, et la difficulté supplémentaire est simplement l'alourdissement du formalisme qui permet de décrire ces contraintes depuis la matrice statistique.

2.3 Statistiques d'un Automate

Nous savons déjà comment associer à un mot ou à un arbre, un point —dans l'espace des statistiques— composé d'une coordonnée par sous-instance de longueur k. Intuitivement, la représentation statistique d'un langage va représenter les statistiques de ses éléments :

$$\mathsf{ustat}(L) = \{\mathsf{ustat}(I) \mid I \in L\}$$

2.3.1 Les Statistiques d'un Automate de Mots

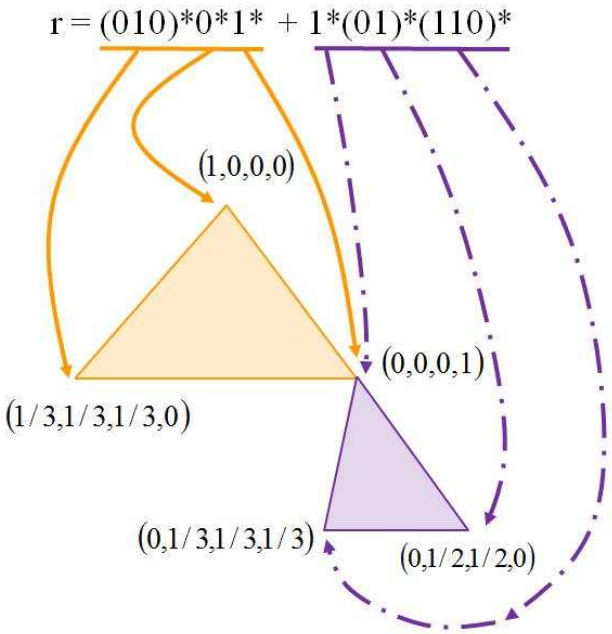


Fig. 2.9 – *Exemple de Statistique d'un Automate de Mots*

Nous allons donner une manière inductive de construire $\mathcal{H}_{A_L}$, une union finie de polytopes, qui permet d'approximer l'ensemble des statistiques des (longs) mots reconnus par l'automate. Pour ce faire on commence par donner la représentation d'une boucle puis on combine ces représentations lorsqu'elles sont compatibles. Une *boucle* désigne un chemin revenant à son origine *i.e.* une séquence d'arêtes consécutives $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots \xrightarrow{\ldots} q_p \xrightarrow{a_p} q_1$. Une boucle est dite *simple* quand elle ne contient pas de boucle de taille strictement inférieure.

La statistique d'ordre k d'une boucle est de même nature que pour les mots, *i.e.* un vecteur unitaire de dimension Σ^k avec une coordonnée par sous-mot de longueur k. Ce vecteur représente la statistique asymptotique de cette boucle, c'est-à-dire la statistique obtenue en itérant infiniment cette boucle. En la coordonnée u (un sous-mot de longueur k), ce vecteur peut également être vu comme la probabilité d'obtenir u en partant d'une position uniforme aléatoire de la boucle.

Définition 2.26 (Statistique uniforme d'une boucle)
Soit $\mathcal{A}$ un automate reconnaissant exactement w^*, et k un entier. $\mathcal{H}^k_{\mathcal{A}}[u]$ est donnée par :

$$\Pr_{j=1,\dots,|w|} \left[w[j, j+1 \bmod k, \dots, j+k-1 \bmod k] = u \right]$$

✓

Exemple 2.27
Pour $\mathcal{A}$ un automate sur l'alphabet binaire $\{0,1\}$ reconnaissant $(0001)^*$ et $k=2$, $\mathcal{H}^2_{\mathcal{A}}$ — donné dans l'ordre lexicographique[19]— est le vecteur $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0)$. ♪

Pour un graphe G orienté, on notera $\widehat{G}$ le graphe de ses composantes fortement connexes, *i.e.* le graphe ou chaque sommet correspond à une composante fortement connexe, avec la convention d'une arête orientée entre deux composantes s_1 et s_2 lorsqu'il existe une arête d'un des sommets de s_1 vers un des sommets de s_2. Dans la suite, si π est un chemin de $\widehat{\mathcal{A}}$ et C une boucle de $\mathcal{A}$, on dit que C est dans π si C est dans une composante fortement connexe correspondant à un sommet de π.

Définition 2.28 (boucles compatibles)
Soit $\mathcal{A}$ un automate et π un chemin acceptant de $\widehat{\mathcal{A}}$. Un ensemble fini C de boucles de $\mathcal{A}$ est dit π-*compatible* lorsque chacune de ces boucles est dans π. Cet ensemble C est dit $\mathcal{A}$-compatible (ou simplement compatible lorsque le contexte ne prête pas à confusion) lorsqu'il existe un tel chemin π. ✓

[19] *i.e.* dans l'ordre : 00,01,10,11

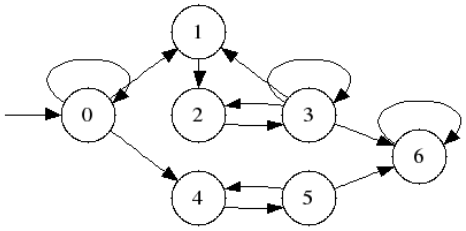


FIG. 2.10 – *Le graphe sous-jaçant à $\mathcal{A}$.*

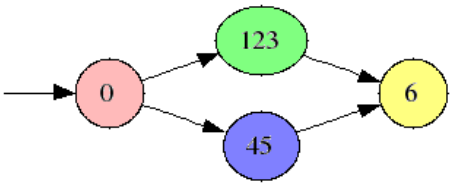


FIG. 2.11 – *Le graphe de ses composantes fortement connexes : $\widehat{\mathcal{A}}$*

Exemple 2.29

Supposons que la structure de l'automate $\mathcal{A}$ soit donnée par le graphe de la Figure 2.10. Dans le graphe de ses composantes connexes $\widehat{A}$ donné par la Figure 2.11, notons π_1 le chemin supérieur $(q_0 \to q_{213} \to q_6)$ et π_2 le chemin inférieur $(q_0 \to q_{45} \to q_6)$.
La boucle $q_1 \to q_2 \to q_3 \to q_1$ est dans π_1.

Dans cet exemple, on peut observer des ensembles de boucles de $\mathcal{A}$:

- π_1-compatibles :
  - $\{q_0 \to q_0\ ,\ q_2 \to q_3 \to q_2\ ,\ q_6 \to q_6\}$
  - $\{q_1 \to q_2 \to q_3 \to q_1\ ,\ q_6 \to q_6\}$
  - $\{q_1 \to q_2 \to q_3 \to q_1\ ,\ q_2 \to q_3 \to q_2\ ,\ q_3 \to q_3\}$.
- π_2-compatibles : $\{q_0 \to q_0\ ,\ q_4 \to q_5 \to q_4\ ,\ q_6 \to q_6\}$.
- non $\mathcal{A}$-compatibles : $\{q_1 \to q_2 \to q_3 \to q_1\ ,\ q_4 \to q_5 \to q_4\}$.

♪

La représentation statistique d'un automate est construite par induction. Soit S un ensemble de boucles $\mathcal{A}$-compatibles $b_1, \ldots, b_p$, de statistiques $u_1, \ldots, u_p$. Une décomposition pour S est un ensemble de réels $(\alpha_i)_{i\in\{1,p\}}$ avec chaque α_i dans $[0,1]$ et $\sum_{i=1}^{p} \alpha_i = 1$.

- Pour une décomposition fixée, la représentation de cette décomposition dans l'espace statistique est un point dont les coordonnées sont : $\mathcal{H}_S^{\alpha_1,\ldots,\alpha_p} = \sum_i \alpha_i.u_i$.
- L'ensemble de boucles compatibles S est alors représenté dans l'espace statistique par l'ensemble de ses décompositions possibles, c'est-à-dire l'ensemble des combinaisons linéaires (se sommant à 1) des statistiques de ces boucles :

$$\mathcal{H}_S = \bigcup_{\substack{0\leq\alpha_i\leq 1 \\ \sum_i \alpha_i = 1}} H_S^{\alpha_1,\ldots,\alpha_p}$$

- Un chemin π de $\widehat{\mathcal{A}}$ est représenté dans l'espace statistique par la représentation statistique de l'ensemble des boucles π-compatibles, de taille inférieure à $m.k^2$.
- La représentation d'un automate $\mathcal{A}$ est finalement donnée par l'ensemble (fini) des représentations de ses chemins $\pi \in \widehat{\mathcal{A}}$.

D'un point de vue théorique, c'est l'erreur tolérée ε qui paramètre la construction de l'approximation. Dans la pratique, l'approximation se fait pour les statistiques d'ordre $k = \lceil 1/\varepsilon \rceil$ et les constructions sont plus agréables à paramétrer par k.

Définition 2.30 (Statistiques d'un automate de mots)
Soit $\mathcal{A}$ un automate de mots à m états, et k un entier.

$$\mathcal{H}_{\mathcal{A}}^{k} = \bigcup_{\pi\in\widehat{\mathcal{A}}} \bigcup_{\substack{\{s_1,\ldots,s_l\} \ \pi\text{-compatibles} \\ |s_i|\leq k^2.m}} \mathsf{Enveloppe-Convexe}(\mathsf{stat}(s_1), \ldots, \mathsf{stat}(s_l))$$

✓

Par construction, cette approximation des statistiques des mots reconnus par un automate est une union finie d'enveloppes convexes. On rappelle que l'enveloppe convexe d'un ensemble de points est le plus petit convexe contenant ces points [20] et qu'un polytope est un polyèdre (sous-espace défini par des hyperplans) borné. Chaque enveloppe convexe étant aussi une union finie de polytopes, la construction garantit donc la proposition suivante.

Proposition 2.31
Pour tout automate de mots $\mathcal{A}$ et tout entier $k > 0$, $\mathcal{H}^k_{\mathcal{A}}$ est une union finie de polytopes.
$\star$

Construction de $\mathcal{H}^k_{\mathcal{A}}$

Étant donné un automate $\mathcal{A}$ à m états et un entier k, la construction de $\mathcal{H}_{\mathcal{A}}$ peut se faire de façon inductive. Remarquons que l'on pourrait énumérer toutes les boucles de taille mk^2 mais cette méthode n'est pas efficace puisqu'il peut y avoir un grand nombre de boucles, $\mathcal{O}(|\Sigma|^{m.k^2})$. Pour améliorer cette approche, il nous suffit en fait de considérer les statistiques générées par ces boucles, et non ces boucles en elles-mêmes.

Nous procédons alors par induction sur la taille des chemins entre deux états possibles de $\mathcal{A}$. Soient u et v deux mots de Σ^{k-1}, et t dans $\{0, 1, \ldots, mk\}$, posons ${}^u\mathrm{P}^v_t$ une matrice $m \times m$ dans laquelle une entrée (i, j) est l'ensemble des statistiques correspondant à un chemin de longueur $k + t$ entre les états i et j, commençant par u et finissant par v.

Initialisation : pour chaque u, v de longueur $k-1$, nous construisons ${}^u\mathrm{P}^v_0$ directement depuis $\mathcal{A}^k$: chaque entrée non vide (i, j) est un ensemble de vecteurs unitaires correspondant aux statistiques uniformes des séquences de k arêtes dans $\mathcal{A}$: $i = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \ldots q_{k-1} \xrightarrow{a_k} q_k = j$, commençant par u et finissant par v.

[20] $\mathsf{Enveloppe-Convexe}(p_1, \ldots, p_s) = \bigcup_{0 \leq \lambda_i \leq 1 \ , \ \sum_i \lambda_i = 1} \lambda_i p_i$

Induction : Intuitivement, une étape d'induction consiste à ajouter l'effet d'une arête (de l à j) aux statistiques des chemins en cours (de longueur $k+t$, de l'état i à l'état l), en ajoutant la nouvelle statistique engendrée par cette nouvelle arête. L'équation inductive correspondante est alors :

$$ {}^{u}\mathrm{P}_{t+1}^{v_1 v_2 \ldots v_{k-1}}[i,j] = \bigcup_{l,a} M^{v_k}[l,j] \otimes \left(\frac{t+1}{t+2} . {}^{u}\mathrm{P}_{t}^{a v_1 v_2 \ldots v_{k-2}}[i,l] \oplus \frac{1}{t+2} . \chi_{a v_1 v_2 \ldots v_{k-1}} \right), \quad \text{où} $$

- M^e est la matrice d'adjacence de l'automate restreinte aux étiquettes e, *i.e.* $M^e[i,j] = 1$ si $(i,e,j) \in \delta$ et 0 sinon. La convention pour la multiplication est alors standard : $0 \otimes \{x_1, \ldots, x_n\} = \emptyset$ et $1 \otimes \{x_1, \ldots, x_n\} = \{x_1, \ldots, x_n\}$
- χ_w désigne l'indicatrice de w : le vecteur unitaire de poids 1 en la coordonnée w. Pour un ensemble $\{x_1, \ldots, x_n\}$, un vecteur statistique χ, et $\alpha, \lambda \in [0,1]$: $\alpha.\{x_1, \ldots, x_n\} \oplus \lambda.\chi = \{x'_1, \ldots, x'_n\}$ avec $x'_i[u] = \alpha x_i[u] + \lambda.\chi[u]$

Lemme 2.32

Etant donné $\mathcal{A}$ un automate à m états et un entier $k = 1/\varepsilon$, un ensemble H de $(|\Sigma|^{k^2}+1)$-tuples de vecteurs peut être construit en temps $m^{|\Sigma|^{\mathcal{O}(k^2)}}$ tel que $|H| \leq m^{|\Sigma|^{\mathcal{O}(k^2)}}$ et $\mathcal{H}_{\mathcal{A}} = \bigcup_{S \in H} \mathsf{Enveloppe-Convexe}(S)$. ⋆

Démonstration : Soient les matrices $({}^{u}\mathrm{P}_{t}^{v})_{u,v \in \Sigma^{k-1}, t=0 \ldots mk^2}$ construites précédemment par induction. A la fin de ce calcul, les diagonales de ces matrices contiennent les statistiques des boucles de longueur au plus $k + mk^2$, sauf la jonction issue de l'itération. Les statistiques uniformes sont alors simplement obtenues en ajoutant ces statistiques de liaison :

$$ {}^{u}\hat{\mathrm{P}}_{t}^{v}[i,i] = \frac{t+1}{t+k} . {}^{u}\mathrm{P}_{t}^{v}[i,i] \oplus \frac{k-1}{t+k} . \sum_{i=1}^{k-1} \chi_{v_i \ldots v_{k-1} u_1 \ldots u_i} $$

Un tuple de $(|\Sigma|^{k^2}+1)$ boucles est compatible si et seulement si il existe un chemin de l'automate passant par tous les états des origines de ces boucles, et cette condition peut être testée en temps polynomial par une multiplication de matrices sur une algèbre appropriée. La borne supérieure sur la taille et le temps de calcul se déduit alors de l'observation que seules $m^{|\Sigma|^{\mathcal{O}(k^2)}}$ statistiques sont considérées. □

2.3.2 Les Statistiques d'une DTD

Comme pour les automates de mots, la représentation statistique d'une DTD est définie de manière inductive, en donnant d'abord la statistique d'une boucle, puis en combinant ces boucles par une notion de compatibilité généralisée pour les arbres. La statistique d'une boucle est alors la statistique d'arbre de cette boucle infiniment répétée.

Généralisons tout d'abord la notion de boucle aux arbres. Nous associons à une DTD S un ensemble de boucles simples. Une *boucle* t_i est un arbre étendu[21] avec une feuille distinguée compatible avec la racine de t_i *i.e.* de même étiquette, et les successeurs libres permettant l'itération. Si la racine de t_i a un successeur gauche (respectivement droit), l'élément distingué ne possède pas de successeur gauche (respectivement droit). Une boucle est dite *simple* lorsqu'aucun de ses sous-arbres étendus (stricts) n'est une boucle. La feuille distinguée d'une boucle est représentée soulignée, et '.' désigne l'absence de successeurs. Par exemple $t_i = a(b, \underline{a})$ ou $a(., \underline{a})$, sont des boucles simples et $a(a(\underline{a}, .), .)$ est une boucle qui n'est pas simple. La séquence des successeurs (dans l'encodage fcns) de la racine à l'élément distingué est appelée le type de la boucle et noté comme précédemment par un mot de $\{0, 1\}^*$ avec la convention 0 pour la relation de premier fils, et 1 pour la relation de frère droit. La $m^{\text{ème}}$ itération de l'arbre t_i via l'élément distingué est notée $(t_i)^m$. Soit t_a un *arbre terminal* de racine étiquetée par a et associé à une DTD, *i.e.* un sous-arbre valide pour l'étiquette a dont aucun chemin (dans l'encodage fcns) ne fait apparaître deux fois la même étiquette. Pour une boucle simple t_i, une *boucle dérivée* de t_i est constituée de la boucle simple t_i à laquelle des arbres terminaux t_a sont connectés aux noeuds étiquetés a de t_i.

A chaque boucle simple ou dérivée, on associe une matrice s_i notée comme précédemment, en ligne les types et en colonne les étiquettes :

$$s_i = \mathsf{ustat}(t_i^*) = \lim_{n \to \infty} \mathsf{ustat}\left((t_i)^n\right)$$

Deux boucles sont *compatibles* si il existe un arbre valide où ces deux boucles apparaissent.

[21] C.f. Définition et Exemple 1.2 (*page* 16)

Exemple 2.33

Considérons la DTD donnée par les quatre règles :

- $root : a^* + b$
- $a : c.d$
- $c : a.f + g$
- $b : e^*$

A l'ordre 2, les boucles simples de cette DTD sont $t_1 = a(.,\underline{a})$, $t_2 = e(.,\underline{e})$, et $t_3 = a(c(\underline{a}(.,f),d),.)$. L'arbre $t_a = a(c(g,d),.)$ est un arbre terminal pour a et $t_c = c(g,.)$ est un arbre terminal pour c. $t_4 = a(c(g,d),\underline{a})$ une boucle dérivée de t_1 .

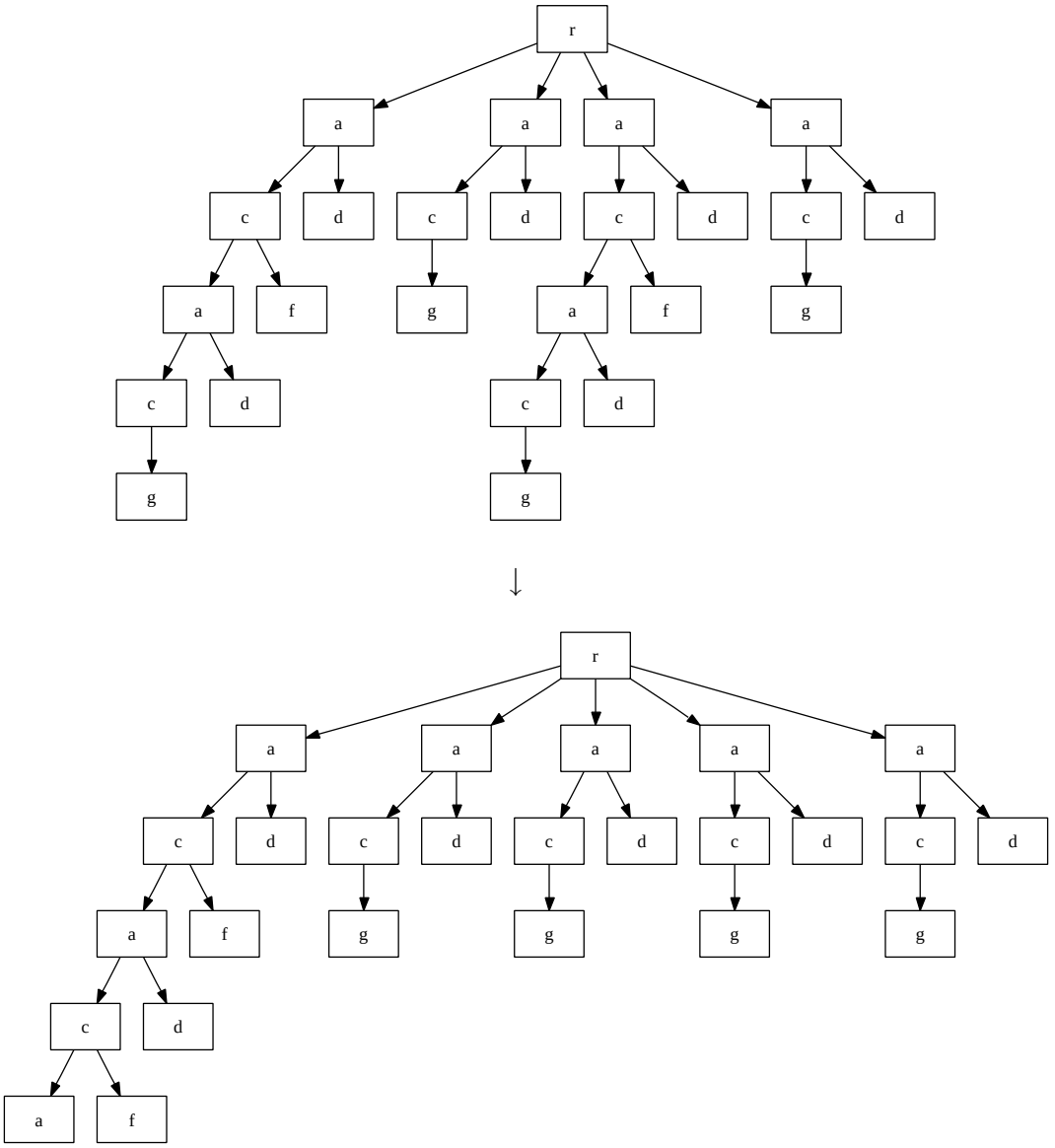
Pour $k = 2$, on obtient les matrices élémentaires suivantes :

$$s_1 = \begin{array}{c|c|} & aa \\ \hline 0 & 0 \\ \hline 1 & 1 \\ \hline \end{array} \quad , \quad s_2 = \begin{array}{c|c|} & ee \\ \hline 0 & 0 \\ \hline 1 & 1 \\ \hline \end{array} \quad , \; et \; s_3 = \begin{array}{c|cccc|} & ac & af & ca & cd \\ \hline 0 & 1/4 & 0 & 1/4 & 0 \\ 1 & 0 & 1/4 & 0 & 1/4 \\ \hline \end{array}$$

et la matrice de la boucle dérivée t_4 donnée par $s_4 = \begin{array}{c|cccc|} & aa & ac & cd & cg \\ \hline 0 & 0 & 1/4 & 0 & 1/4 \\ 1 & 1/4 & 0 & 1/4 & 0 \\ \hline \end{array}$

Les boucles t_1, t_3 et t_4 sont compatibles alors que la boucle horizontale t_2 n'est compatible avec aucune d'entre elles. Si un grand arbre t est proche de la DTD, alors $\mathsf{ustat}(t)$ est nécessairement proche de s_2 ou de $\{\alpha_1 \cdot s_1 + \alpha_3 \cdot s_3 + \alpha_4 \cdot s_4, \alpha_1 + \alpha_3 + \alpha_4 = 1\}$. En effet, les boucles simples de même nature peuvent être regroupées par des déplacements (voir figure 2.12 (*page* 71)).

Notons que la réciproque nécessiterait la prise en compte d'une contrainte linéaire reliant le nombre d'occurrences de la boucle horizontale aa au nombre d'occurrences des boucles verticales (enracinées en a) : il conviendrait alors d'imposer la contrainte supplémentaire $\alpha_1 \leq \alpha_3 + \alpha_4$. ♪



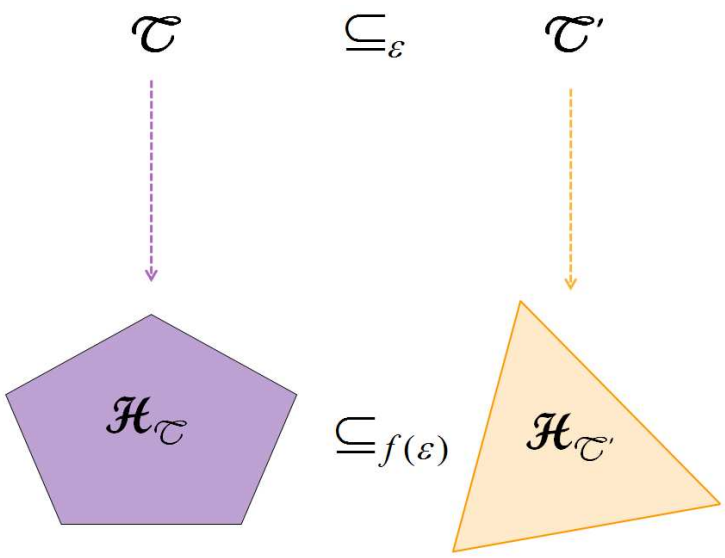
En effectuant trois déplacements, les boucles de même type sont regroupées.

- le sous-arbre gauche — celui qui a pour racine le fils le plus à gauche de r — possédera des statistiques qui convergent vers s_3 (pour un nombre d'itérations de cette boucle qui tend vers l'infini).
- la haie restante (celle depuis les autres fils a de r) aura quand à elle une statistique tendant vers s_4.

Fig. 2.12 – *Regroupement de Boucles Simples d'Arbres.*

Chapitre 3

Transducteurs : Statistiques, Distances, Équivalence



Décider si deux descriptions formelles sont équivalentes est une question centrale en informatique, particulièrement dans les domaines comme la vérification ou le test [Lee et Yannakakis, 1996].

Définition 3.1 (Équivalence)
Lorsqu'un transducteur $\mathscr{T}$ accepte un couple d'instances (I, I'), on notera $(I, I') \in \mathscr{T}$.
On dira que $\mathscr{T}_1$ est inclus dans $\mathscr{T}_2$, noté $\mathscr{T}_1 \subseteq \mathscr{T}_2$, lorsque $(I, I') \in \mathscr{T}_1 \Rightarrow (I, I') \in \mathscr{T}_2$.
On dira qu'ils sont équivalents, noté $\mathscr{T}_1 \equiv \mathscr{T}_2$, lorsque $\mathscr{T}_1 \subseteq \mathscr{T}_2$ et $\mathscr{T}_2 \subseteq \mathscr{T}_1$. ✓

Dans le cas général, l'équivalence de deux transducteurs est un problème indécidable. Griffiths a déjà démontré ce résultat pour les transducteurs (non déterministes) Λ-libres, c'est-à-dire qui ne lisent ni n'écrivent le mot vide (Λ). Ces transducteurs Λ-libres ont la propriété suivante : la taille de la sortie est linéaire dans la taille de l'entrée. Ce résultat d'indécidabilité s'étend naturellement aux transducteurs d'arbres ainsi qu'à des classes plus restreintes de transductions [Karhumäki et Lisovik, 1999] et peu de résultats effectifs et positifs existent dans le cas non déterministe. L'équivalence est malgré tout montrée décidable pour les transducteurs linéaires lettre à lettre dans le cas descendant [Andre et Bossut, 1993] ou ascendant [Andre et Bossut, 1995]. Ces travaux ne fournissent cependant pas de borne de complexité pour la décision de ces équivalences, contrairement à l'approche proposée dans ce chapitre.

Néanmoins, il existe plusieurs restrictions intéressantes permettant d'obtenir la décidabilité de l'équivalence. Sur les mots, l'équivalence est décidable pour les transducteurs déterministes ([Bird, 1973],[Valiant, 1974]), et ce résultat a été généralisé dans le cas déterministe pour deux [Gurari, 1982] puis plusieurs bandes [Harju et Karhumäki, 1991], ainsi que pour certains transducteurs d'arbres : les transducteurs ascendants déterministes [Zachar, 1978], les transducteurs ascendants finiment valués [Seidl, 1990] ou les transducteurs MSO déterministes [Engelfriet et Maneth, 2005]. Notons finalement que pour des transducteurs déterministes complets, il vient d'être montré que l'équivalence peut être décidée en temps polynomial [Maneth et Seidl, 2007].

Ce chapitre donne une méthode effective pour décider de l'équivalence approchée dans le cas non déterministe Λ-libre. Nous relâchons alors un problème indécidable (l'équivalence pour des transducteurs ε-libres) en une version approchée décidable.

Dans la suite, les transducteurs sont donc supposés Λ-libres, et nous nous donnons une borne β sur la taille de sortie d'une transition. Comme pour les automates, la *taille* d'un XSL-Transducteur $\mathscr{T}$, notée $|\mathscr{T}|$, est son nombre d'états ajouté à la taille de sa relation de transition. La taille d'une arête est la taille de son instance de sortie (longueur du mot de sortie ou nombre de noeuds de l'arbre de sortie), et la relation de transition a comme taille la somme des tailles de ses arêtes.

Définition 3.2 (Équivalence Approchée)
On dira que $\mathscr{T}_1$ est ε-inclus dans $\mathscr{T}_2$, noté $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$ lorsque pour tout $(I, I') \in \mathscr{T}_1$ avec $|I| \geq \frac{\beta}{\varepsilon}.\max(|\mathscr{T}_1|, |\mathscr{T}_2|)$, il existe $(I_1, I'_1) \in \mathscr{T}_2$ tel que I_1 est ε-proche de I et I'_1 est ε-proche de I'. On dira qu'ils sont ε-équivalents, noté $\mathscr{T}_1 \equiv_\varepsilon \mathscr{T}_2$, lorsque $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$ et $\mathscr{T}_2 \subseteq_\varepsilon \mathscr{T}_1$. ✓

Nous allons définir une représentation géométrique d'un XSL-Transducteur qui va nous permettre de tester l'ε-inclusion en testant l'ε' inclusion des représentations. Cette construction de $\mathcal{H}_\mathscr{T}$ est une adaptation de celle qui est appliquée aux automates et suit le même principe : on définit la représentation d'une boucle, puis on étend la notion de compatibilité pour obtenir une nouvelle construction inductive.

L'idée générale est ensuite de montrer que $\mathcal{H}_\mathscr{T}$ est une bonne approximation de $\mathsf{ustat}(\mathscr{T})$ pour les grands couples d'instances, et ainsi une représentation fidèle des couples (de grande taille) proches d'être en relation pour $\mathscr{T}$. On va en effet montrer que la représentation statistique associée aux XSL-Transducteurs permet de décider du problème de l'inclusion approchée. Il suffira alors d'appliquer cette décision de l'ε-inclusion dans les deux sens pour décider l'ε-équivalence.

Sommaire

3.1 Mots

3.1.1 String-Transducteur

Un String-Transducteur transforme des mots en mots. Il fonctionne de la même manière qu'un automate de mots, à la différence que ses transitions lisent et écrivent des caractères.

Définition 3.3 (String-Transducteur (FSM))
Un *String-Transducteur* est donné par $\mathscr{T} = (\Sigma, \Sigma', Q, I, F, \delta)$ où Σ est l'alphabet d'entrée, Σ' l'alphabet de sortie, Q un ensemble d'états, $I \in Q$ un état initial, $F \subseteq Q$ les états finaux et $\delta \subseteq Q \times (\Sigma \cup \{\Lambda\}) \times (\Sigma')^* \times Q$ la relation de transition. ✓

On parlera d'arête d'un String-Transducteur pour désigner un élément de sa relation de transition. Pour tout String-Transducteur on sait construire un transducteur normalisé équivalent. La *normalisation* $\mathscr{T}'$ d'un String-Transducteur $\mathscr{T} = (\Sigma, \Sigma', Q, I, F, \delta)$ est un String-Transducteur équivalent où on a majoré la taille de sortie des arêtes par 1, *i.e.* $\mathscr{T}' = (\Sigma, \Sigma', Q', I, F, \delta')$ avec $\delta' \subseteq Q \times (\Sigma \cup \{\Lambda\}) \times (\Sigma' \cup \{\Lambda\}) \times Q$.

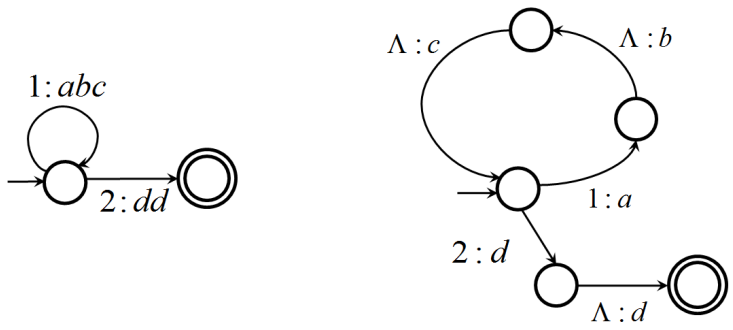


Fig. 3.1 – *Un String-Transducteur et une de ses normalisations possibles*

Dans le cas général (non déterministe), un mot w peut potentiellement être transformé en plusieurs mots ; on notera $\mathscr{T}(w)$ l'ensemble de ces mots. Un String-Transducteur est dit déterministe lorsque la relation de transition est une fonction $\delta : Q \times \Sigma \times (\Sigma')^* \to Q$. Dans le cas déterministe, $\mathscr{T}(w)$ contient donc au plus un élément. Dans le cas général, $\mathscr{T}(w)$ est un langage régulier de mots.

Proposition 3.4

Soit un mot w de longueur n, et $\mathscr{T}$ un String-Transducteur normalisé à $n_{\mathscr{T}}$ états et $a_{\mathscr{T}}$ arêtes. On sait construire $\mathscr{T}(w)$, la transduction de w par $\mathscr{T}$, en temps $\mathcal{O}(|w|.n_{\mathscr{T}}.a_{\mathscr{T}})$. L'automate engendré (FSA) possède moins de $|w|.n_{\mathscr{T}}$ états et moins de $2.|w|.n_{\mathscr{T}}.a_{\mathscr{T}}$ arêtes. ⋆

Ce résultat, démontré dans l'annexe A.6 (*page* 176), montre entre autre que l'image d'un mot par un transducteur est un langage régulier. Cette propriété reste vraie pour l'image inverse : L'image réciproque d'un mot par un String-Transducteur est encore un langage régulier que l'on obtient en complexité linéaire en la taille du mot comme l'image directe de ce mot par le String-Transducteur inverse. L'*inverse* $\mathscr{T}'$ d'un String-Transducteur normalisé $\mathscr{T} = (\Sigma, \Sigma', Q, I, F, \delta)$ est le transducteur $\mathscr{T}' = (\Sigma', \Sigma, Q, I, F, \delta')$ avec $(q, i, o, q') \in \delta'$ si et seulement si $(q, o, i, q') \in \delta$.

Les transducteurs sont Λ-libres et la taille de sortie de leurs transitions est bornée par β *i.e.* dans le cas des mots, les transitions sont alors restreintes à un sous-ensemble

$$\delta \subseteq Q \times \Sigma \times \left(\bigcup_{i=1}^{\beta} (\Sigma')^i \right) \times Q.$$

3.1.2 Statistiques d'un String-Transducteur

Intuitivement, la statistique d'un String-Transducteur $\mathscr{T}$ approche les statistiques des couples de mots (entrée/sortie) en relation pour $\mathscr{T}$.

Nous allons calculer une approximation des statistiques d'un String-Transducteur en donnant une représentation statistique de la transformation pour chaque chemin π de $\widehat{\mathscr{T}}$. L'approximation pour un chemin particulier de $\widehat{\mathscr{T}}$ est alors la statistique d'un ensemble de boucles compatibles, elle-même approchée comme l'ensemble des décompositions particulières de cet ensemble de boucles compatibles. Reste alors à préciser l'étape d'initialisation de l'induction en spécifiant ce qu'est une statistique de boucle.

Il convient d'être vigilant puisque le simple « produit carthésien » des couples de statistiques n'est pas une façon robuste d'approcher cet objet. En effet, il faut par exemple pouvoir séparer une boucle $1|a$ d'une boucle $1|aaaaaa$, qui produisent les mêmes statistiques d'entrée et de sortie, mais sont pourtant loin d'être équivalentes [22].

Pour pouvoir séparer ces deux String-Transducteurs, une coordonnée supplémentaire capturant ces différences de longueur est donc utilisée. Nous approchons ainsi

$$\left\{ \left(\mathsf{ustat}(I), \mathsf{ustat}(I'), \frac{|I'|}{|I|} \right) \mid (I, I') \in \mathscr{T} \right\}.$$

Pour le String-Transducteur $\mathscr{T} = (\Sigma, \Sigma', Q, I, F, \delta)$, nous construisons inductivement une représentation géométrique de la transformation dans un espace constitué de trois types de coordonnées :

1. $\mathsf{ustat}(I)$: de dimension $|\Sigma|^k$.
2. $\mathsf{ustat}(I')$: de dimension $|\Sigma'|^k$.
3. la dernière coordonnée est utilisée pour la distorsion des longueurs (sous-espace de dimension 1 inclus dans l'intervalle $[1, \beta]$).

[22] les longueurs des couples acceptés diffèrent complètement : un couple $(1^n, a^n)$ accepté par la première boucle, est loin de tout mot $(1^n, a^{6n})$ accepté par la seconde boucle

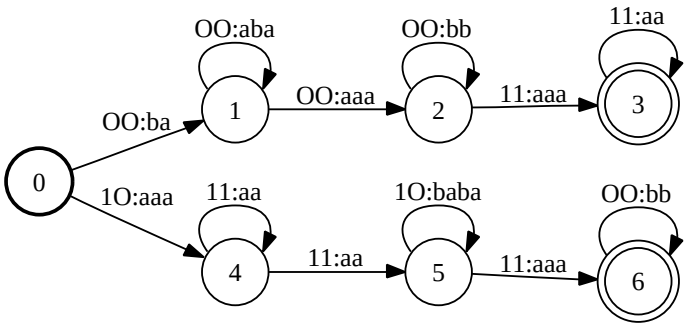


Fig. 3.2 – *Un String-Transducteur*

L'induction calque alors les manières de traduire une instance-source I par le String-Transducteur $\mathscr{T}$. Une exécution particulière acceptante de $\mathscr{T}$ sur I suit un chemin π de $\widehat{\mathscr{T}}$ suivant lequel I peut être traduite en suivant un certain nombre de fois un ensemble de boucles compatibles sur ce chemin. Sur ce chemin π, les statistiques de I vont essentiellement pouvoir se décomposer sur les statistiques d'une base de boucles π-compatibles.

Exemple 3.5 (L'approximation d'un String-Transducteur)
La Figure 3.3 représente l'approximation obtenue pour le String-Transducteur de la Figure 3.2. ♪

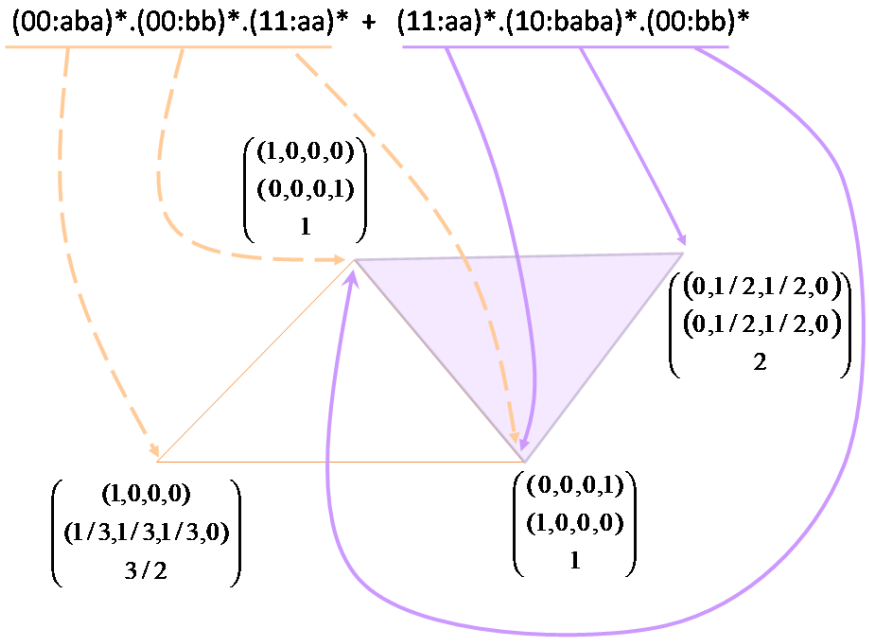


Fig. 3.3 – *La Représentation Statistique du String-Transducteur de la Figure 3.2*

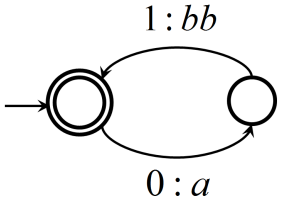


FIG. 3.4 – *Exemple de boucle simple*

Le cas d'une boucle

Dans cet espace, chaque boucle s de $\mathscr{T}$ est associée au point $(\mathcal{I}_s, \mathcal{O}_s, \rho_s)$ où :

1. $\mathcal{I}_s$ représente les statistiques uniformes de l'entrée de la boucle,
2. $\mathcal{O}_s$ représente les statistiques uniformes de sortie de s,
3. ρ_s est le facteur d'expansion de cette boucle *i.e.* $\frac{l_{out}(s)}{l_{in}(s)}$, où l_{in} (respectivement l_{out}) représente la longueur d'entrée (respectivement de sortie) de la boucle.

Exemple 3.6 (Représentation géométrique d'une boucle)
Donnons la représentation géométrique pour $k = 2$ du String-Transducteur réduit à une boucle de la Figure 3.4.

1. Les premières coordonnées définissent les densités des couples-lettres sur l'alphabet-source (les densités en 00, 01, 10 et 11). Il s'agit donc de $(0, 1/2, 1/2, 0)$.
2. Les coordonnées suivantes désignent les densités de sortie pour aa, ab, ba et bb : 0 pour aa, 1/3 pour ab, ba ou bb.
3. La dernière coordonnée ne dépend pas de k et restera donc égale à 3/2 pour tout k.

Le String-Transducteur de la Figure 3.4 est représenté pour $k = 2$ par

$$\begin{pmatrix} (0, 1/2, 1/2, 0) \\ (0, 1/3, 1/3, 1/3) \\ 3/2 \end{pmatrix}$$

♪

Pour une décomposition sur une base donnée

Soit $S = \{s_1, \ldots, s_p\}$ un ensemble de boucles compatibles. Une *décomposition* pour S est un ensemble de réels $(\alpha_i)_{i\in\{1,p\}}$ avec chaque α_i dans $[0,1]$ et $\sum_{i=1}^{p} \alpha_i = 1$. Pour une décomposition donnée, la représentation géométrique de cette décomposition est un point dont les coordonnées sont :

$$\mathcal{H}_S^{\alpha_1,\ldots,\alpha_p} = \begin{pmatrix} \sum_i \alpha_i.\mathcal{I}_{s_i} \\ \left[\left[\left(\sum_{s_i\in S} \alpha_i.\rho_{s_i}.\mathcal{O}_{s_i}\right)\right]\right] \\ \sum_i \alpha_i.\rho_{s_i} \end{pmatrix},$$

On rappelle que $[[x]]$ désigne la renormalisation $\frac{x}{|x|}$ du vecteur x. Cette renormalisation disparaît dans le cas d'un String-Transducteur qui préserve la longueur.

Pour un ensemble de boucles compatibles

La représentation d'un ensemble S de boucles compatibles doit alors être générée comme les points précédents, mais pour une décomposition arbitraire sur cette base de statistiques, et non plus fixée par un seul $(\alpha_i)_{i\in\{1,p\}}$.

$$\mathcal{H}_S = \bigcup_{\substack{0\leq\alpha_i\leq 1 \\ \sum_i \alpha_i = 1}} H_S^{\alpha_1,\ldots,\alpha_p}$$

Pour les String-Transducteurs préservant la longueur, on obtient la notion d'enveloppe convexe traditionnelle ([Cormen et al., 1990], Section 33.3). Dans le cas de distorsions, on perd l'aspect de convexité et on parlera alors *d'enveloppe pondérée*.

Exemple 3.7

Considérons les boucles s_1 d'entrée a et de sortie cc et s_2 d'entrée bb et de sortie ccd et $k = 2$. $\mathsf{ustat}(s_1) = \begin{pmatrix} (1,0,0,0) \\ (1,0,0,0) \\ 2 \end{pmatrix}$, $\mathsf{ustat}(s_2) = \begin{pmatrix} (0,0,0,1) \\ (1/3,1/3,1/3,0) \\ 3/2 \end{pmatrix}$.

$\mathcal{H}_{\{s_1,s_2\}}$ est l'ensemble des interpolations $\alpha s_1 \oplus (1-\alpha)s_2$ pour α dans $[0,1]$.

$$\mathcal{H}^{\alpha}_{\{s_1,s_2\}} = \begin{pmatrix} (\alpha,0,0,1-\alpha) \\ 1/\left((3\alpha/2+1/2)+2\left(\alpha/2-1/2\right)\right) \,.\, (3\alpha/2+1/2, 1/2-\alpha/2, 1/2-\alpha/2, 0) \\ \alpha/2+3/2 \end{pmatrix}$$

$\mathcal{H}_{\{s_1,s_2\}}$ contient par exemple le point pour $\alpha = 1/4$,

$$\frac{1}{4}s_1 \oplus \frac{3}{4}s_2 \quad : \quad \begin{pmatrix} (1/4,0,0,3/4) \\ [[(\frac{7}{8},\frac{3}{8},\frac{3}{8},0)]] \\ \frac{1}{4}*2+\frac{3}{4}*\frac{3}{2} \end{pmatrix} = \begin{pmatrix} (1/4,0,0,3/4) \\ (\frac{7}{13},\frac{3}{13},\frac{3}{13},0) \\ 13/8 \end{pmatrix}.$$

♪

Pour un chemin de $\widehat{\mathscr{T}}$

Une traduction le long d'un chemin π du graphe des composantes fortement connexes $\widehat{\mathscr{T}}$ est essentiellement une décomposition d'un ensemble de boucles π-compatibles. La représentation des traductions le long d'un chemin π est ainsi définie par

$$\mathcal{H}_{\pi} = \bigcup_{\text{ensemble } S \text{ de boucles } \pi\text{-compatibles}} \mathcal{H}_S$$

Pour un String-Transducteur Lambda-Free

Pour représenter $\mathscr{T}$, nous ferons l'union des représentations de chacun des chemins de $\widehat{\mathscr{T}}$ (il n'y en a qu'un nombre fini).

$$\mathcal{H}_{\mathscr{T}} = \bigcup_{\pi \in \widehat{\mathscr{T}}} \mathcal{H}_{\pi}$$

Exemple 3.8

Dans l'exemple 3.5 (*page* 79), posons π_1 le chemin supérieur de $\widehat{\mathscr{T}}$ et π_2 le chemin inférieur. Les boucles π_1-compatibles sont les trois boucles de la partie supérieure (Figure 3.2). La représentation statistique pour π_1 est alors $\mathcal{H}_{\pi_1} = \alpha_1 \cdot u_1 \oplus \alpha_2 \cdot u_2 \oplus \alpha_3 \cdot u_3$, avec $\alpha_1, \alpha_2, \alpha_3 \in [0,1]$, $\alpha_1+\alpha_2+\alpha_3 = 1$, et u_i correspondant aux statistiques de la boucle simple i. De la même manière, $\mathcal{H}_{\pi_2} = \alpha'_1 \cdot u'_1 \oplus \alpha'_2 \cdot u'_2 \oplus \alpha'_3 \cdot u'_3$, pour u'_i les statistiques des boucles simples inférieures, $\alpha'_1, \alpha'_2, \alpha'_3 \in [0,1]$ et $\alpha'_1 + \alpha'_2 + \alpha'_3 = 1$. Finalement $\mathcal{H}_{\mathscr{T}} = \mathcal{H}_{\pi_1} \cup \mathcal{H}_{\pi_2}$. ♪

Cette représentation $\mathcal{H}_{\mathscr{T}}$ peut être vue comme une union finie d'ensembles de solutions de systèmes d'inéquations linéaires. On note une différence clé avec le cas des mots, puisqu'une union finie de polytopes est suffisante pour les automates, alors qu'on manipule ici une union finie d'intersections d'hyperplans. Toutes les unions sont finies, sauf celle du passage de $\mathcal{H}_S^{\alpha_1,\dots,\alpha_p}$ à $\mathcal{H}_S$ qui, elle, décrit un système à p variables.

Proposition 3.9

Pour tout String-Transducteur $\mathscr{T}$, $\mathcal{H}_{\mathscr{T}}$ est une union finie d'arrangements d'hyperplans. ⋆

On remarque aussi qu'un automate de mots est la restriction d'un String-Transducteur à sa partie entrante. La représentation statistique d'un automate de mots présentée dans la section 2.3 (*page* 63) correspond ainsi à la projection sur les $|\Sigma|^k$ premières coordonnées de la représentation donnée pour les String-Transducteurs .

3.1.3 Équivalence Approchée de String-Transducteurs

Dans cette section, nous montrons que la représentation statistique proposée permet de décider de l'équivalence approchée de deux String-Transducteurs . Cette procédure de décision, qui sera décrite par un algorithme, repose sur les idées suivantes :

1. Les String-Transducteurs proches ont des représentations proches :
   - On montre qu'un couple de grande taille $(I, I') \in \mathscr{T}$ a des statistiques[23] proches de $\mathcal{H}_{\mathscr{T}}$.
   - Si des instances sont proches, leurs statistiques sont proches.
   - On en déduit que lorsque $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$ alors $\mathcal{H}_{\mathscr{T}_1} \subseteq_{\varepsilon'} \mathcal{H}_{\mathscr{T}_2}$.
2. Réciproquement, deux String-Transducteurs loin ont des représentations loin :
   - Tout point de $\mathcal{H}_{\mathscr{T}}$ peut être approché par les statistiques de couples de $\mathscr{T}$.
   - Si $\mathscr{T}_1 \not\subseteq_\varepsilon \mathscr{T}_2$, on peut construire une suite de couples d'instances (de tailles croissantes) de $\mathscr{T}_1$ qui sont tous loin de $\mathscr{T}_2$.
   - On peut extraire de cette suite une sous-suite dont les statistiques convergent vers un point de $\mathcal{H}_{\mathscr{T}_1}$ qui est, par construction, le centre d'une boule de rayon ε' n'intersectant pas $\mathcal{H}_{\mathscr{T}_2}$.
3. La procédure générale consiste à chercher s'il existe une telle boule. La correction est alors garantie puisqu'on montre que :
   - Si les String-Transducteurs sont équivalents, alors il n'existe pas de boule de ce type.
   - Si les String-Transducteurs sont ε-loin, alors on a choisi le pas de la grille suffisamment petit pour détecter cette boule.

La présentation est maintenant organisée comme suit : nous présentons l'algorithme, puis nous montrons sa correction, sa robustesse, pour conclure que l'algorithme est un testeur. Enfin nous montrons que la complexité en temps de cet algorithme est polynomiale pour les String-Transducteurs préservant la longueur, et majorée par une exponentielle dans le cas général.

[23] $\left(\mathsf{ustat}(I), \mathsf{ustat}(I'), \frac{|I'|}{|I|}\right)$

Algorithmes

L'ε-équivalence entre deux String-Transducteurs $\mathscr{T}_1$ et $\mathscr{T}_2$ est ici décidée en testant la cohérence de deux systèmes d'inéquations linéaires $\mathcal{S}_1$ et $\mathcal{S}_2$. Ces systèmes sont obtenus en relaxant les approximations statistiques ($\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$) proposées pour les transducteurs. Leur cohérence est alors testée en relâchant certaines contraintes d'un paramètre guidé par la preuve de robustesse qui suivra, *i.e.* si les transducteurs sont ε-loin, on trouvera un témoin de cette propriété dans l'espace des statistiques, sur une grille de pas $f(\varepsilon)$.

Deux algorithmes suivant ce principe sont présentés. Tout d'abord l'algorithme 3 (*page* 86) pour le cas particulier des String-Transducteurs à un état, puis sa généralisation au cas de plusieurs états par l'algorithme 4. Dans le premier cas, on montre qu'il est suffisant de traiter le cas des boucles simples, en obtenant ainsi une meilleure complexité que dans le cas général. La correction s'obtient par les deux propriétés suivantes :

1. si $\mathscr{T}_1 \equiv \mathscr{T}_2$ les algorithmes 3 et 4 renvoient vrai.

2. si $\mathscr{T}_1 \not\subseteq_\varepsilon \mathscr{T}_2$, ces algorithmes renvoient faux.

On garde en tête l'intuition de $\mathcal{H}_{\mathscr{T}}$: Un point $x = (\mathcal{I}_x, \mathcal{O}_x, \rho_x)$ appartient à $\mathcal{H}_{\mathscr{T}_2}$ si et seulement si il se décompose sur une base $(s'_1, \dots s'_p)$ *i.e.* si et seulement si le système suivant possède une solution.

$$\mathcal{S}^x_{\mathscr{T}_2} = \begin{cases} 0 \leq \alpha_i \leq 1, \forall i \in \{1, p\} \\ \displaystyle\sum_{i=1}^{t'} \alpha_i = 1 \\ \mathcal{I}_x = \displaystyle\sum_{i=1}^{t'} \alpha_i . \mathcal{I}_{s'_i} \\ \left|\left| \displaystyle\sum_{i=1}^{t'} \alpha_i . \rho_{s'_i} . \mathcal{O}_{s_i} \right|\right| . \mathcal{O}_x = \displaystyle\sum_{i=1}^{t'} \alpha_i . \rho_{s'_i} . \mathcal{O}_{s'_i} \\ \rho_x = \displaystyle\sum_{i=1}^{t'} \alpha_i . \rho_{s'_i} \end{cases}$$

Algorithme 3 : L'ε-inclusion pour les String-Transducteurs fortement connexes

Données : $\mathscr{T}_1$ et $\mathscr{T}_2$ Deux String-Transducteurs à un état ; $\varepsilon \in]0,1[$.

Sorties : La décision de $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$

Construire $\mathcal{H}_{\mathscr{T}_2}$ comme un système d'inéquations linéaires;

$\varepsilon^* := 12.2\varepsilon\sqrt{\varepsilon}$;

Construire $\mathcal{S}_2$ l'ensemble des systèmes d'équations de la relaxation de $\mathcal{H}_{\mathscr{T}_2}$ par ε^* ;

$Res := 1$;

pour chaque *boucle simple b de* $\mathscr{T}_1$ **faire**

  Calculer $x = (I_b, O_b, l_b)$;

  si *x n'est la solution d'aucun des systèmes de* $\mathcal{S}_2$ **alors** $Res := 0$

fin

retourner *Res*

Algorithme 4 : Testeur pour l'ε-inclusion

Données : Deux String-Transducteurs ; $\varepsilon \in]0,1[$.

Sorties : La décision de $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$

Construire $\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$ comme ensembles de systèmes d'inéquations linéaires;

$\tilde{\varepsilon} := 0.1 \times \sqrt{\varepsilon} \times \frac{\rho_{min}}{\rho_{max}}$;

Construire $\mathcal{S}_1$ et $\mathcal{S}_2$ les systèmes d'inéquations des relaxations respectives de $\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$ par $\tilde{\varepsilon}/3$;

$Res := 1$;

pour chaque *x point de la grille sur* $[0,1]^{|\Sigma|^k} \times [0,1]^{|\Sigma'|^k} \times [1,\beta]$ *de pas* $\tilde{\varepsilon}/3$ **faire**

  si *x vérifie* $\mathcal{S}_1$ **alors**

    si *x n'est la solution d'aucun des systèmes de* $\mathcal{S}_2$ **alors** $Res := 0$

  fin

fin

retourner *Res*

Relaxation de système d'inéquations : Nous illustrons par un exemple l'étape de relaxation d'un système utilisée dans les deux algorithmes. La relaxation d'un système $\mathcal{S}$ est un nouveau système $\mathcal{S}'$ avec des inégalités conservées ($\varepsilon_1 + \varepsilon_2 + \varepsilon_3 \leq \varepsilon'$, $\alpha_i \in [0,1]$, $\sum_i \alpha_i = 1$), et où on autorise une erreur ε' à se répartir (additivement) sur les autres inégalités de $\mathcal{S}$.

Exemple 3.10

Le système donné pour $\mathcal{S}^x_{\mathscr{T}_2}$ (page 85), relâché avec le paramètre ε' devient :

$$\mathcal{S}^x_{\mathscr{T}_2}(\varepsilon') = \begin{cases} 0 \leq \alpha_i \leq 1, \forall i \in \{1,p\} \\ \sum_{i=1}^{t'} \alpha_i = 1 \\ \varepsilon_1 + \varepsilon_2 + \varepsilon_3 \leq \varepsilon' \\ I_x \leq \sum_{i=1}^{t'} \alpha_i . \mathcal{I}_{s'_i} + \varepsilon_1 \\ I_x \geq \sum_{i=1}^{t'} \alpha_i . \mathcal{I}_{s'_i} - \varepsilon_1 \\ O_x \leq \dfrac{\sum_{i=1}^{t'} \alpha_i . \frac{l_{out}(s'_i)}{l_{in}(s'_i)} . \mathcal{O}_{s'_i}}{\left|\left| \sum_{i=1}^{t'} \alpha_i . \dfrac{l_{out}(s'_i)}{l_{in}(s'_i)} . \mathcal{O}_{s_i} \right|\right|} + \varepsilon_2 \\ O_x \geq \dfrac{\sum_{i=1}^{t'} \alpha_i . \frac{l_{out}(s'_i)}{l_{in}(s'_i)} . \mathcal{O}_{s'_i}}{\left|\left| \sum_{i=1}^{t'} \alpha_i . \dfrac{l_{out}(s'_i)}{l_{in}(s'_i)} . \mathcal{O}_{s_i} \right|\right|} - \varepsilon_2 \\ l_x \leq \sum_{i=1}^{t'} \alpha_i . \rho_{s'_i} + \varepsilon_3 \\ l_x \geq \sum_{i=1}^{t'} \alpha_i . \rho_{s'_i} - \varepsilon_3 \end{cases}$$

♪

Assurons-nous que les statistiques d'un couple de mots en relation pour un String-Transducteur $\mathscr{T}$ sont bien "approximativement représentées" par $\mathcal{H}_{\mathscr{T}}$, et donnons l'intuition de cette affirmation par un exemple.

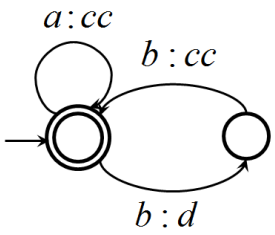


FIG. 3.5 – *Le String-Transducteur de l'Exemple 3.11*

Exemple 3.11

Considérons le String-Transducteur à deux boucles compatibles donné par la Figure 3.5. Le mot d'entrée $w = a^{500}b^{500}$ se décompose sous la forme $w = a^{500}(bb)^{250}$; l'image de w par $\mathscr{T}$ est $w' = \mathscr{T}(w) = c^{1000}(ccd)^{250}$.

On pose $\beta_1 = \frac{500}{500+250*2} = 1/2$ et $\beta_2 = \frac{2*250}{500+250*2} = 1/2$ ($k_1 = 500$ et $k_2 = 250$) et nous allons voir que $\frac{1}{2}s_1 \oplus \frac{1}{2}s_2$ approche bien $(\mathsf{ustat}(w), \mathsf{ustat}(w'), |w|/|w'|)$, où $\oplus$ représente la même opération que celle présentée pour combiner des boucles compatibles[24].

$$
\begin{aligned}
\mathsf{ustat}_1(w) &= (a : 500/1000, b : 500/1000) = (a : 1/2, b : 1/2) \\
&= 1/2 * \mathsf{ustat}(a) + 1/2 * \mathsf{ustat}(b) \\
&= \beta_1 \mathsf{ustat}(\mathcal{I}_{s_1}) + \beta_2 \mathsf{ustat}(\mathcal{I}_{s_2}) \\
|w'|/|w| &= \tfrac{1750}{1000} = 1.75 \\
\rho &= \beta_1\rho_{s_1} + \beta_2\rho_{s_2} = \tfrac{1}{2} * 2 + \tfrac{1}{2} * 3/2 = 1.75. \\
x' &= \left[\left[\sum_i \beta_i.\rho_{s_i}.\mathcal{O}_{s_i}\right]\right] \\
&= \left[\left[\tfrac{1}{2} * 2 * (cc : 1) + \tfrac{1}{2} * \tfrac{3}{2} * (cc : \tfrac{1}{3}, cd : \tfrac{1}{3}, dc : \tfrac{1}{3})\right]\right] \\
&= [[cc : 1.25, cd : 0.25, dc : 0.25]] \\
&= (cc : \tfrac{1.25}{1.75}, cd : \tfrac{0.25}{1.75}, dc : \tfrac{0.25}{1.75}) \\
\mathsf{ustat}_2(w') &= \tfrac{1}{1749}(cc : 1250, cd : 250, dc : 249)
\end{aligned}
$$

[24] *i.e.* une combinaison linéaire des coordonnées d'entrée, une combinaison linéaire des distorsions, et la renormalisation des coordonnées de sortie pondérées par leur distorsions

Trois différences distinguent x' et $\mathsf{ustat}_2(w')$:

- le dernier cc issu de a est compté dans x' sans exister dans w' : c'est la différence entre statistique asymptotique et statistique tronquée.
- symétriquement, le dernier dc issu de b est compté dans x' sans exister dans w'. A nouveau au plus k occurrences de trop sont introduites pour chaque s_i.
- le dernier c issu des a et le premier c issu des b n'est pas compté dans x'. A chaque changement de boucle on néglige donc dans x' au plus k occurrences de sous-mots de longueur k.

♪

Correction

Étendons les remarques faites sur l'exemple pour un String-Transducteur général : les String-Transducteurs proches ont des représentations proches. Tout d'abord nous montrons que les statistiques de tout couple de longs mots (accepté par un String-Transducteur) sont « approximativement conformes » à la représentation statistique de ce transducteur.

Proposition 3.12

Soit $k = 1/\varepsilon \in \mathbb{N}^+$ et (w, w') un couple de mots accepté par un String-Transducteur Λ-libre $\mathscr{T}$ à m états et b boucles simples, et tel que $|w| \geq N_{\min} = 3.(b+1).(k+1).(1+2m)$. Alors, il existe $(\widehat{w}, \widehat{w'}) \in \mathscr{T}$ tel que :

1. $\mathsf{d}(w, \widehat{w}) \leq 2\varepsilon$
2. $\mathsf{d}(w', \widehat{w'}) \leq 2\varepsilon$
3. $\left(\mathsf{ustat}(\widehat{w}), \mathsf{ustat}(\widehat{w'}), \frac{|\widehat{w'}|}{|\widehat{w}|}\right)$ *est 3ε-proche de $\mathcal{H}_{\mathscr{T}}$.*

⋆

Démonstration : Soit $\mathscr{T}$ possédant m états, a arêtes, et b boucles simples, $k \in \mathbb{N}^+$, $(w, w') \in \mathscr{T}$. Pour simplifier l'explication, nous supposons $(w, w') \in \mathscr{T}^k$, sans perdre de généralité puisqu'il suffit de rajouter quelques lettres pour supposer w de longueur un multiple de k. Soit π un chemin de $\widehat{\mathscr{T}}$ tel que w puisse être transformé en w' en restant dans la restriction de $\mathscr{T}$ à π ; l'existence d'un tel chemin est garantie par le fait que $(w, w') \in \mathscr{T}$.

En suivant π, w peut être décomposé, pour un certain j, en $w = v_0u_1v_1u_2 \dots u_jv_j$ où :

- $|v_0|, \dots, |v_j| \leq m.k$
- $C_1 \dots C_c$ sont des composantes fortement connexes de $\widehat{\mathscr{T}}$ $(c \leq m)$
- chaque u_i est transformé dans la composante C_i.

A l'intérieur d'une composante fortement connexe, u_i se décompose sur un ensemble de boucles, et pour chaque u_i, les différentes occurrences d'une même boucle sont regroupées. On définit alors $\widehat{w} = v'_0(u'_1)^{k_1}v'_1(u'_2)^{k_2} \dots v'_{b'-1}(u'_{b'})^{k_b}v'_b$ où chaque v'_i est une séquence de moins de m arêtes sur Σ^k, $S = \{u'_1 \dots u'_{b'}\}$ est un ensemble de boucles π-compatibles ; et $b' \leq 2^{a^k}$; Posons

$$\widehat{w'} = out(v_0)(out(u'_1))^{k_1}out(v_1)(out(u'_2))^{k_2} \dots out(v_{b-1})out((u'_{b'}))^{k_b}out(v_b) \qquad (3.1)$$

Par construction $(\widehat{w}, \widehat{w'}) \in \mathscr{T}^k \subseteq \mathscr{T}$ et les opérations élémentaires effectuées pour passer de w à $\widehat{w}$ sont : les déplacements pour regrouper les boucles (εn), et d'éventuelles modifications sur les v'_i, chaque v'_i étant de longueur inférieure à $m.k$. En sommant ce nombre d'opérations élémentaires, on obtient :

$$\mathsf{dist}(w, \widehat{w}) \leq 3(b'+1)m.k + \varepsilon n \qquad (3.2)$$

Pour β la plus longue sortie d'une arête de $\mathscr{T}$:

$$\mathsf{dist}(w', \widehat{w'}) \leq 3\beta(b'+1)m.k + \varepsilon n \qquad (3.3)$$

La distance entre $\left(\mathsf{ustat}(\widehat{w}), \mathsf{ustat}(\widehat{w'}), |\widehat{w}| / |\widehat{w'}|\right)$ et $\mathcal{H}_{\mathscr{T}}$ peut maintenant être majorée en explicitant un point (x, x', ρ) de $\mathcal{H}_{\mathscr{T}}$ adapté à la décomposition précédente de $(\widehat{w}, \widehat{w'})$.

Posons $n_0 = \sum_i k_i$, $n_1 = \sum_i k_i.l_{in}(s_i)$ où $l_{in}(s_i)$ désigne la longueur de la boucle s_i, $\beta_i = \frac{k_i.l_{in}(s_i)}{n_1}$ et finalement $x = \sum_i \beta_i.\mathcal{I}_{s_i}$, $x' = \left[\left[\sum_i \beta_i.\rho_{s_i}.\mathcal{O}_{s_i}\right]\right]$ et $\rho = \sum_i \beta_i.\rho_{u_i}$.

Clairement $\sum_i \beta_i = 1$ et $(x, x', \rho) \in \mathcal{H}_{u'_1 \ldots u'_b} \subseteq \mathcal{H}_{\mathscr{T}}$. En reprenant les sources d'erreurs dévoilées dans l'exemple :

$$|\mathsf{ustat}(\widehat{w}) - x|_1 \leq \frac{6.(b'+1).k.(1+m)}{n_0 - k + 1} \tag{3.4}$$

$$|\mathsf{ustat}(\widehat{w'}) - x'|_1 \leq \beta \frac{6.(b'+1).k.(1+m)}{n_0 - k + 1} \tag{3.5}$$

$$|\frac{|\widehat{w}'|}{|\widehat{w}|} - \rho| \leq (\beta+1) \frac{6.(b'+1).k.(1+m)}{n_0} \tag{3.6}$$

En choisissant n_0 assez grand pour garantir que les parties droites des équations (3.4) (3.5) et (3.6) soient inférieures à ε, nous garantissons la 3ε-proximité à $\mathcal{H}_{\mathscr{T}}$.

- Pour l'équation (3.4), $n_0 \geq 3.(b'+1).k.(1+m) + k - 1$ est suffisant.
- Pour l'équation (3.5), il suffit de multiplier la borne précédente par β.
- Cette nouvelle borne suffit aussi pour l'équation 3.6.

Il suffit d'ajouter $b'.m.k$ pour obtenir la borne voulue pour n_0 *i.e.*

$$n = |w| \geq 6.(b'+1).(k+1).(1+2m).$$

Cette borne est alors réintroduite dans la version relative de l'équation (3.2) ce qui assure $\frac{(b'+1).m.k}{n} \leq \varepsilon$ et garantit ainsi

$$\mathsf{d}(w, \widehat{w}) \leq \varepsilon + \frac{(b'+1).m.k}{n} \leq 2\varepsilon.$$

Le cas de l'équation (3.3) est similaire :

$$\frac{\beta(b'+1).m.k}{n} \leq \varepsilon \Rightarrow \mathsf{d}(w', \widehat{w'}) \leq \varepsilon + \beta \frac{(b'+1).m.k}{n} \leq 2\varepsilon$$

□

Dans la suite, nous utilisons le théorème 2.3 (*page* 37) : deux mots proches ont des statistiques proches.[25] Ce théorème assure que les statistiques de (w, w') sont proches de $(\widehat{w}, \widehat{w'})$ et par conséquent de $\mathcal{H}_{\mathscr{T}}$. Ceci valant quel que soit le couple de mots assez long, on conclut que deux String-Transducteurs ε-proches ont des représentations ε'-proches. Pour des transducteurs proches, les erreurs introduites par chaque flèche du diagramme de la figure 3.6 sont contrôlées :

- horizontalement par définition de l'ε-équivalence
- verticalement par les résultats précédents.

$$\begin{array}{ccc} (w_1, w_1') & \rightarrow & (w_2, w_2') \\ \downarrow & & \downarrow \\ (\widehat{w_1}, \widehat{w_1'}) & & (\widehat{w_2}, \widehat{w_2'}) \\ \downarrow & & \downarrow \\ (x_1, x_1', \rho_1) & & (x_2, x_2', \rho_2) \end{array}$$

Fig. 3.6 – *Intuition de la Correction*

Théorème 3.13

Soit $\varepsilon > 0$. Si deux String-Transducteurs Λ-libres $\mathscr{T}_1$ et $\mathscr{T}_2$ sont ε-proches alors ils ont des représentations statistiques $(3\varepsilon + 12.2\sqrt{\varepsilon})$-proches pour la norme $\mathbf{L}_1$. ★

Montrons la réciproque de la proposition 3.12, à savoir que chaque point de $\mathcal{H}_{\mathscr{T}}$ peut être approché par les statistiques de couples de mots en relation dans $\mathscr{T}$.

Proposition 3.14

Pour tout point x de $\mathcal{H}_{\mathscr{T}}$, on peut construire un couple de mots $(w, w') \in \mathscr{T}$ tel que $(\mathsf{ustat}(w), \mathsf{ustat}(w'), |w'|/|w|)$ soit ε-proche de x ⋆

[25] Pour $n = \Omega(\frac{1}{\varepsilon}) = |w| = |w'|$, si $\mathsf{dist}(w, w') \leq \varepsilon^2 n$ alors $|\mathsf{ustat}_k(w) - \mathsf{ustat}_k(w')| \leq 6.1\varepsilon$.

Démonstration : Soit $x \in \mathcal{H}_{\mathscr{T}}$. Par définition de $\mathcal{H}_{\mathscr{T}}$, cela signifie qu'il existe un chemin π de $\widehat{\mathscr{T}}$, un ensemble $S = s_1, \ldots, s_p$ de boucles π-compatibles, ainsi que $\alpha_1, \ldots, \alpha_p$ (dans $[0,1]$ et sommant à 1) tels que

$$x = \mathcal{H}_S^{\alpha_1,\ldots,\alpha_p} = \begin{pmatrix} \sum_{i=1}^{p} \alpha_i.\mathcal{I}_{s_i} \\ \left[\left[\left(\sum_{s_i \in S} \alpha_i.\rho_{s_i}.\mathcal{O}_{s_i}\right)\right]\right] \\ \sum_i \alpha_i.\rho_{s_i} \end{pmatrix}.$$

Nous choisirons l'ordre des s_i de telle sorte qu'on puisse effectuer les boucles $s_1, \ldots, s_p$ dans cet ordre précis[26], quitte à parcourir des transitions entre ces boucles.

Soit $n \in \mathbb{N}$, et $(u_0, u'_0), \ldots, (u_p, u'_p)$ un témoin de la compatibilité. Posons :

$w = u_0(\mathcal{I}_{s_i})^{\lfloor n.\alpha_1 \rfloor} u_1 (\mathcal{I}_{s_i})^{\lfloor n.\alpha_2 \rfloor} \ldots u_{p-1} (\mathcal{I}_{s_p})^{\lfloor n.\alpha_p \rfloor} u_p$ et

$w' = u'_0(\mathcal{O}_{s_1})^{\lfloor n.\alpha_1 \rfloor} u'_1 (\mathcal{O}_{s_2})^{\lfloor n.\alpha_2 \rfloor} \ldots u'_{p-1} (\mathcal{O}_{s_p})^{\lfloor n.\alpha_p \rfloor} u_p.$

Le couple (w, w') est accepté par $\mathscr{T}$, et nous allons voir que la norme $\mathbf{L}_1$ entre $(\mathsf{ustat}(w), \mathsf{ustat}(w'), |w'|/|w|)$ et x, tend vers 0 quand la taille de w tend vers l'infini. L'erreur absolue[27] sur l'entrée est engendrée par différents facteurs :

1. les occurrences à cheval sur tout ou partie de chacun des u_i (ces occurrences de w sont absentes du point de vue de x)
2. les éventuelles occurrences à cheval sur deux boucles différentes (pour le cas d'un u_i réduit au mot vide par exemple)
3. les erreurs dues aux arrondis $\lfloor n.\alpha_i \rfloor$.

Chaque u_i est de longueur inférieure à $2.m$. Le premier cas engendre au plus $m(p+1)$ occurrences erronées, le second au plus $2(k-1)p$ et le troisième moins de $\sum_{i=1}^{p} m^{n.\alpha_i - \lfloor n.\alpha_i \rfloor} \leq pm$.

[26] on doit avoir $s_i < s_j$ lorsque s_i et s_j sont dans deux composantes connexes différentes connectées par π et que l'état représentant (entre autre) s_i précède strictement l'état représentant s_j. L'ordre choisi pour les boucles d'une même composante fortement connexe, n'a en revanche aucune importance.

[27] *i.e.* le nombre d'occurrences qui diffèrent

L'erreur totale absolue sur l'entrée est donc majorée par $(p+1)(m+2k)+2(k-1)m+pm \leq (p+1)(m+4k-2)$.

L'erreur faite sur la longueur de w' provient exactement des mêmes raisons :

$$\begin{aligned}\left|\sum_i \alpha_i.\rho_{s_i} - \frac{|w'|}{|w|}\right| &\leq \sum_{i=1}^{p} |u_i| + (k-1)\times p + \sum_{i=1}^{p} \rho_i \times |\widehat{\alpha}_i - \alpha_i| \\ &\leq pm + p(k-1) + \sum_{i=1}^{p} \rho_i \\ &\leq p(m+k+\rho_{max}).\end{aligned}$$

L'erreur sur la statistique de sortie est issue des raisons précédentes auxquelles s'ajoute l'influence de l'approximation des α_i vis-à-vis de la renormalisation. L'erreur précédente est donc multipliée par au plus β. L'erreur absolue totale est donc majorée par $3.p.\beta(m+k+\rho_{max})$ ce qui assure que l'erreur relative, inférieure à $(3.p.\beta(m+k+\rho_{max}))/(1+n)$, tend vers 0 quand n tend vers l'infini.

Pour n assez grand, on peut donc bien rendre $(\mathsf{ustat}(w), \mathsf{ustat}(w'), |w'|/|w|)$ aussi proche de x que l'on souhaite. □

Les résultats précédents permettent d'établir la correction.

Proposition 3.15

Si $\mathscr{T}_1$ et $\mathscr{T}_2$ sont deux transducteurs Λ-libres équivalents, alors l'algorithme 3 renvoie vrai.

⋆

Démonstration : Supposons $\mathscr{T}_1 \equiv \mathscr{T}_2$ et que le test renvoie faux. Prenons une boucle b qui n'est solution d'aucun système. Le couple (w_i, w_o) — entrée de la boucle, sortie de la boucle — est alors proche d'être accepté par $\mathscr{T}_1$ et loin de l'être par $\mathscr{T}_2$. Plus précisément, on a $(\hat{w}_i, \hat{w}_0) \in \mathscr{T}_1$ tel que tout $(w, w') \in \mathscr{T}_2$ possède des statistiques $9\varepsilon/10$ loin de $\mathcal{H}_{\mathscr{T}_1}$. Par la proposition 2.3 (*page* 37), pour tout $(w, w') \in \mathscr{T}_2$ $\mathsf{dist}(w_i, w)$ ou $\mathsf{dist}(w_0, w')$ est nécessairement plus grand que $\frac{5}{6.1\times 2}\frac{\sqrt{9\varepsilon/10}}{p}$. On contredit ainsi $\mathscr{T}_1 \subseteq \mathscr{T}_2$ et donc l'équivalence de $\mathscr{T}_1$ et $\mathscr{T}_2$. □

Robustesse

Cette section est dédiée à montrer que si deux String-Transducteurs sont loin, l'algorithme donné permet de les rejeter. La partie centrale de la preuve utilise le fait que si deux String-Transducteurs sont ε-loin alors leurs représentations sont ε'-loin.

Proposition 3.16

Soient $\mathscr{T}_1$ et $\mathscr{T}_2$ deux String-Transducteurs Λ-libres. Si $\mathscr{T}_1$ est ε-loin de $\mathscr{T}_2$ alors on peut trouver une boule, centrée dans $\mathcal{H}_{\mathscr{T}_2}$, et de rayon $\varepsilon' = \theta(\varepsilon_2)$, qui n'intersecte pas $\mathcal{H}_{\mathscr{T}_1}$. $\star$

Ce résultat est une conséquence du fait que lorsque des String-Transducteurs sont loin, on peut extraire une suite de couples de mots de l'un, tous éloignés des couples en relation pour l'autre. Cette suite peut toujours être choisie de taille croissante.

Proposition 3.17

Soient $\mathscr{T}_1$ et $\mathscr{T}_2$ deux String-Transducteurs Λ-libres. Si $\mathscr{T}_1$ est ε-loin de $\mathscr{T}_2$ alors il existe une suite $(w_i, w'_i)_{i\in\mathbb{N}} \in \mathscr{T}_1$ telle que la longueur des mots w_i et w'_i croît avec i et pour tout $(w, w') \in \mathscr{T}_2$, $\mathsf{dist}(w_i, w) + \mathsf{dist}(w'_i, w') \geq \varepsilon_2 = \Omega(\sqrt{\varepsilon})$. $\star$

Démonstration de la Proposition 3.17 : Par l'absurde, supposons que $\mathscr{T}_1$ soit ε-loin d'être inclus dans $\mathscr{T}_2$ et l'existence d'un nombre fini de (w_i, w'_i) vérifiant la conclusion[28] de la proposition 3.17. Nécessairement chaque boucle de $\mathscr{T}_1$ peut être arbitrairement approchée par $\mathscr{T}_2$. On contredit alors le fait qu'il existe un couple de mots (suffisamment longs) témoignant de l'« ε-éloignement ». $\square$

L'existence de cette suite permet alors d'extraire le centre d'une boule qui « sépare » les deux représentations statistiques.

[28] pour tout $(w, w') \in \mathscr{T}_2$, $\mathsf{dist}(w_i, w) + \mathsf{dist}(w'_i, w') \geq \varepsilon_2 = \Omega(\sqrt{\varepsilon})$

Démonstration de la Proposition 3.16 : Supposons que $\mathscr{T}_1$ est ε-loin d'être inclus dans $\mathscr{T}_2$, et prenons les $(w_i, w'_i)_{i\in\mathbb{N}} \in \mathscr{T}_1$ de la proposition précédente. Chaque (w_i, w'_i) se décompose sur une base S_i de sommets de $\mathscr{T}_1$. Il existe donc une base infiniment répétée, ce qui garantit de pouvoir extraire une sous-suite $(\widehat{w}_i, \widehat{w}_i)_{i\in\mathbb{N}}$ dont tous les éléments se décomposent sur cette base commune. La suite $(\mathsf{ustat}_k(\widehat{w}_i), \mathsf{ustat}_k(\widehat{w}_i))_{i\in\mathbb{N}}$ prend ces valeurs dans le compact $[0,1]^{|\Sigma|^k} \times [0,1]^{|\Sigma'|^k} \times [1,\beta]$, ce qui permet de garantir l'existence d'une sous-suite $(w_i^*, w_i'^*, \rho_i)$ qui converge vers $(\mathcal{I}_a, \mathcal{O}_a, \rho_a)$. Ce point est le centre de la boule qui va nous permettre de séparer $\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$.
Pour n'importe quel couple de mots $(w, w') \in \mathscr{T}_2$ et tout i, $\mathsf{dist}(w_i, w^*)+\mathsf{dist}(w', w'^*)+\tilde{\varepsilon} \geq \varepsilon$ et donc par la proposition 2.3 (*page* 37), $|\mathsf{ustat}_k(w_i)-\mathsf{ustat}_k(w^*)|+|\mathsf{ustat}_k(w'_i)-\mathsf{ustat}_k(w'^*)|+\tilde{\varepsilon} \geq \frac{6.5\times\varepsilon}{5\times 2}$. En passant à la limite, on obtient que pour tout $(w, w') \in \mathscr{T}_2$ tels que $\left|\frac{w'}{w} - \rho_a\right| \leq \tilde{\varepsilon}$, $|\mathsf{ustat}_k(w_i) - \mathcal{I}_a| + |\mathsf{ustat}_k(w'_i) - \mathcal{O}_a| \geq 0.65\varepsilon - \tilde{\varepsilon}$.
En posant $\varepsilon' = 0.65\varepsilon - \tilde{\varepsilon}$ pour un $\tilde{\varepsilon}$ bien choisi[29] , on obtiendra que la boule centrée en $(\mathcal{I}_a, \mathcal{O}_a, \rho_a)$ et de rayon ε' n'intersecte jamais $\mathcal{H}_{\mathscr{T}_1}$. □

Nous avons exhibé un point a de $\mathcal{H}_{\mathscr{T}_2}$ loin de $\mathcal{H}_{\mathscr{T}_1}$. Si la difficulté n'est pas la possibilité de combiner les boucles, c'est le témoin qu'une de ses boucles est déjà « loin de $\mathcal{H}_{\mathscr{T}_1}$ ». Dans ce cas, le critère de test est simplifié : la décision se fait uniquement par un critère sur les sommets extrémaux des représentations statistiques.

Corollaire 3.18

Soient $\mathscr{T}_1$ et $\mathscr{T}_2$ deux String-Transducteurs Λ-libres. Si $\mathscr{T}_1$ fortement connexe est ε-loin de $\mathscr{T}_2$ alors il existe un sommet de $\mathcal{H}_{\mathscr{T}_2}$, ε'-loin pour la norme $\mathbf{L}_1$ de $\mathcal{H}_{\mathscr{T}_1}$. ⋆

[29] le $\tilde{\varepsilon}$ du corollaire suivant par exemple

Démonstration : Soit $a = (\mathcal{I}_a, \mathcal{O}_a, \rho_a)$ le point de la preuve précédente. On peut décomposer sur la base infiniment répétée choisie pour l'extraction. L'un des sommets (=boucle) de cette décomposition est nécessairement loin de $\mathcal{H}_{\mathscr{T}_1}$ puisque sinon on pourrait montrer que a est aussi proche de $\mathcal{H}_{\mathscr{T}_1}$ et ainsi contredire la proposition précédente. On a donc un sommet $s = (\mathcal{I}_s, \mathcal{O}_s, \rho_s)$ tel que pour tout $(w, w') \in \mathscr{T}_1$, au moins l'un des trois cas suivants est vérifié :

1. $|\mathsf{ustat}(w) - \mathcal{I}_a| \geq \varepsilon'/3$
2. $|\mathsf{ustat}(w') - \mathcal{O}_a| \geq \frac{\varepsilon'}{3} \times \frac{\rho_{min}}{\rho_{max}}$
3. $\left|\frac{|w'|}{|w|} - \rho_a\right| \geq \tilde{\varepsilon}$

L'erreur en norme $\mathbf{L}_1$ pour ce s est donc au minimum $\frac{\varepsilon'}{3} \times \frac{\rho_{min}}{\rho_{max}} - \tilde{\varepsilon}$. Si cette erreur est plus grande que $\tilde{\varepsilon}$, s est $\tilde{\varepsilon}$ loin de $\mathcal{H}_{\mathscr{T}_2}$ (pour la norme $\mathbf{L}_1$). Il suffit donc de poser $\tilde{\varepsilon} = \frac{0.65\varepsilon}{6} \times \frac{\rho_{min}}{\rho_{max}}$. □

Corollaire 3.19

L'algorithme 3 est un testeur pour l'inclusion des String-Transducteurs Λ-libres fortement connexes. ⋆

En effet, la proposition 3.15 (*page* 94) assure que l'algorithme renvoie vrai pour des transducteurs équivalents. Le corollaire précédent nous garantit quant à lui que l'algorithme renvoie faux pour des transducteurs ε-loin puisqu'il il s'exécute avec une erreur $\tilde{\varepsilon}$ suffisamment petite pour que les hypothèses du corollaire 3.18 (*page* 96) soient vérifiées.

Pour plusieurs composantes fortement connexes, le fait d'être loin peut aussi indiquer que deux boucles présentes dans chacun des transducteurs n'ont pas le même type de compatibilité. Le point discriminant les deux représentations doit donc aussi être cherché plus généralement « à l'intérieur » de $\mathcal{H}_{\mathscr{T}_1}$.

Théorème 3.20

L'algorithme 4 est un ε-testeur pour l'inclusion des String-Transducteurs Λ-libres. ★

Démonstration : Le point a de la preuve de la proposition 3.16 (*page* 95) assure que l'on satisfait la condition du si (externe) pour un point x $\tilde{\varepsilon}/3$-proche de a. Chacun des systèmes de $\mathcal{S}_2$ renvoie faux puisque ce a est encore au moins $\frac{2\times\tilde{\varepsilon}}{3}$-loin de $\mathcal{H}_{\mathscr{T}_2}$. La variable Res est alors affectée à $faux$ et gardera cette valeur jusqu'à être renvoyée. □

Complexité

La procédure de décision exposée dans ce chapitre prend en entrée des ensembles de systèmes linéaires. En testant uniquement la satisfaction de ces systèmes, on garantit que la décision se fait en temps polynomial en la taille des systèmes.

La construction la plus directe pour obtenir une approximation robuste de la définition inductive de $\mathcal{H}_T$ est de considérer toutes les boucles de taille inférieure à $m.k$. Pour un String-Transducteur à a arêtes, ce nombre de boucles est majoré par $a^{m.k}$. Sans perte de généralité, a peut être choisi plus petit que $m^2 \times |\Sigma_S| \times |\Sigma_T|^\beta$, et donc l'ensemble des systèmes générés est de taille $\mathcal{O}\left(\left(m^2 \times |\Sigma_S| \times |\Sigma_T|^\beta\right)^{m.k}\right)$.

En fait, il n'est pas nécessaire de considérer toutes ces boucles. En effet, seules les statistiques d'entrée, les statistiques de sortie et les distorsions de ces boucles comptent pour la construction de $\mathcal{H}_{\mathscr{T}}$. Pour un mot d'entrée de longueur n, il n'existe que $(n-k+1)^{|\Sigma_S|^k}$ statistiques d'entrée différentes ; le nombre de statistiques de sortie à considérer est inférieur à $(\beta.n-k+1)^{|\Sigma_T|^k}$; et moins de $\beta.n$ distorsions sont possibles. Dans la construction de $\mathcal{H}_{\mathscr{T}}$ le nombre de (x, x', ρ) à considérer, est donc un $\mathcal{O}\left((p\,(\beta+1))^{(\Sigma_S+\Sigma_T)^k}\right)$ où p est une longueur de boucle suffisante pour garantir la robustesse. Pour les String-Transducteurs, la preuve de la robustesse nous assure que $p = m.k^2$ est suffisant pour une construction robuste de $\mathcal{H}_{\mathscr{T}}$, *i.e.* il suffit de considérer un nombre de statistiques polynomial en m.

Construction inductive de $\mathcal{H}_{\mathscr{T}}$

La construction de $\mathcal{H}_{\mathscr{T}}$ peut alors se faire par induction sur la taille t des chemins entre deux états possibles du String-Transducteur $\mathscr{T}$. Soit P_t la matrice $m \times m$ dans laquelle l'entrée (i, j) est en ensemble de vecteurs correspondants à un chemin de longueur t entre les états i et j. $P_t[i, j]$ contient des ensembles de points. Un point est donné par sa statistique d'entrée $x \in [0, 1]^{|\Sigma_S|^k}$; sa statistique de sortie $y \in [0, 1]^{|\Sigma_T|^k}$; et sa distorsion $\rho \in [1, \beta]$.

On construit également $P_t^g[i, j]$ représentant les possibles blocs gauches. Un point (de cette matrice d'ensembles de points) est alors donné par son bloc gauche d'entrée $x_t^g \in (\Sigma_S)^t$; son bloc gauche de sortie $y_t^g \in (\Sigma_T)^{t.\beta}$; et sa distorsion gauche : $\rho_t^g \in [1, \beta]^t$. Symétriquement, on pose $P_t^d[i, j]$ pour ses différents blocs droits.

La propriété centrale de cette construction est que la matrice P_{t+1} s'obtient depuis P_1,...,P_t, P_1^g, P_1^d, ... , P_{k-1}^g et P_{k-1}^d par une équation inductive. De la même façon que l'induction a été obtenue pour les automates (*page* 68), les matrices P_t sont calculées de façon incrémentale pour t de 1 à $m.k$. Finalement les représentations des boucles (éléments diagonaux de P_t) sont conservées. La question de compatibilité d'un ensemble de boucles revient à décider de l'existence d'un chemin passant par les origines de ces boucles et cette décision est également polynomiale. Puisque seules $\mathcal{O}\left((m.k\,(\beta+1))^{(\Sigma_S+\Sigma_T)^k}\right)$ statistiques sont à considérer, la représentation de $\mathcal{H}_{\mathscr{T}}$ est construite en temps polynomial en m.

3.2 Arbres

3.2.1 XSL-Transducteur

XSLT est un langage central dans le monde XML et beaucoup de qualités reconnues de XML reposent en fait sur l'utilisation de XSLT : productions de versions diffusables (XHTML, SVG ...), pérennité des documents, ouverture des formats, interopérabilité, etc.

La première motivation est d'associer un style à un document XML, tout comme on associe une feuille de style CSS à une page HTML. Pour remédier aux limitations du CSS que sont par exemple l'impossibilité d'extraire les valeurs des attributs ou de créer de nouvelles données, un nouveau langage XML a été proposé : XSL[30]. Cette thèse s'intéresse aux transformations permettant de convertir un document XML en un autre document XML et s'intéresse alors à XSLT[31], aux dépens des transformations de XML en formats non structurés (txt, pdf ...) considérées par XSL-FO[32].

Comme pour les automates d'arbres, il existe une littérature fournie de transformateurs d'arbres dont deux familles importantes se distinguent selon qu'elles s'appliquent de la racine aux feuilles (transducteur descendant) [Andre et Bossut, 1993, Andre et Bossut, 1998, Martens et Neven, 2004] ou des feuilles à la racine (transducteur ascendant) [Zachar, 1978]. Précédemment introduites pour les arbres à arité bornée, les transformations déterministes descendantes sont connues pour être moins expressives que les transformations ascendantes [Engelfriet, 1975]. Notons que d'autres modèles sont étudiés sur les arbres d'arité non bornée comme les transducteurs à jetons introduits par [Milo et al., 2003] dont la version à 0 jeton correspond aux Macro Tree Transducers de [Maneth et al., 2005].

Motivés par le cadre de transformation de XML [World Wide Web Consortium, 2006] par XSLT [World Wide Web Consortium, 1999], nous allons principalement nous intéresser au cas descendant introduit par [Martens et al., 2008].

[30] eXtensible Stylesheet Language : http://www.w3.org/Style/XSL/
[31] XSL Transformations : http://www.w3.org/TR/xslt
[32] XSL Formating Objects : http://www.w3.org/TR/xsl/

Soit $\mathbf{K}_S$ un ensemble de Σ_S arbres et $\mathbf{K}_T$ un ensemble de Σ_T arbres. Un XSL-Transducteur associe, de façon descendante, à un nœud v d'étiquette dans Σ_S, un nouveau sous-arbre sur Σ_T. Cette modélisation est motivée par le langage XSLT dans lequel cette fonctionnalité est standard.

Soit H_{Σ_T} l'ensemble des séquences finies d'arbres finis, (haies, forêts). $H_{\Sigma_T}[Q]$ est l'ensemble des haies (séquences finies d'arbres finis) de H_{Σ_T} dans lesquelles les feuilles sont étiquetées par $\Sigma_T \cup Q$ au lieu de Σ_T.

Définition 3.21

Un *XSL-Transducteur* d'arbres entre Σ_S arbres et Σ_T arbres est donné par (Q, q_0, δ) où : δ est un ensemble de règles de la forme $(\Sigma_S, Q) \rightarrow H_{\Sigma_T}[Q]$ tel que si $(q_0, a) \rightarrow h \ \in \delta$, alors h est soit vide, soit h est un unique arbre dont la racine est étiquetée par Σ. ✓

Exemple 3.22

En pratique, le XSL-Transducteur est souvent déterministe et défini par un programme XSLT $\mathscr{T}$.

```
<xsl:output method="xml" indent="yes"/>
   <xsl:template match="/bd">
       <bib> <xsl:apply-templates/> </bib>
   </xsl:template>
   <xsl:template match="work">
      <livre>
         <titre/>
         <xsl:apply-templates/>
      </livre>
   </xsl:template>
   <xsl:template match="author">
      <auteur/>
   </xsl:template>
```

FIG. 3.7 – *Un Programme XSLT* $\mathscr{T}$

Le XSL-Transducteur donné par le code XSLT peut être décrit de façon équivalente dans notre formalisme par :

- $\Sigma_S = \{$ db, work, author $\}$,
- $\Sigma_T = \{$ bib, livre, auteur, titre $\}$,
- $Q = \{q\}$,
- $\delta = \{(q, db) \rightarrow bib(q); (q, work) \rightarrow livre(titre, q); (q, author) \rightarrow auteur\}$.

♪

La transformation par la δ d'un arbre t dans l'état q, notée $T^q(t)$, est définie inductivement comme suit : si $t = \Lambda$ alors $T^q(t) := \Lambda$; si $t = a(t_1 \dots t_n)$ et qu'il existe un élément $(q, a) \rightarrow h$ dans δ alors $T^q(t)$ contient la haie h dans laquelle on a remplacé chaque noeud u de h étiqueté par l'état p par une haie de $T^p(t_1) \dots T^p(t_n)$. S'il n'y a pas de règle $(q, a) \rightarrow h$ dans δ, alors $T^q(t) := \Lambda$. Finalement, la transformation de t par T, notée $\mathscr{T}(t)$ est définie par $T^{q_0}(t)$.

Un XSL-Transducteur est dit déterministe si pour chaque paire (q, a), il existe au plus une règle dans δ et dans ce cas $\mathscr{T}(t)$ s'interprète comme un arbre. Dans le cas non déterministe, on notera également $\mathscr{T}(t)$ pour désigner l'ensemble des transformés possibles, c'est-à-dire un ensemble d'arbres ou chacun résulte d'un ensemble de choix résolvant le non-déterminisme.

La partie droite d'une règle est dite *effaçante* si elle possède un état apparaissant à son plus haut niveau, et *k-copiante* lorsque qu'elle contient k feuilles étiquetées par des états. Une règle est *linéaire* si exactement une feuille de sa partie droite est étiquetée par un état, et un XSL-Transducteur est *linéaire* lorsque toutes ses règles sont linéaires.

Exemple 3.23 (Règles linéaire, non linéaire, effaçante)
Considérons l'alphabet d'entrée $\{a, b\}$, l'alphabet de sortie $\{c, d, e\}$, $Q = p, q$ et $T = (Q, p, \delta)$ les correspondances entre δ et les templates XSLT sont données ci dessous :

1. $(p, a) \rightarrow c(d)$:
```
<xsl:template match="a" mod="p">
  <c>
    <d/>
  </c>
</xsl:template>
```
2. $(p, b) \rightarrow e(q)$:
```
<xsl:template match="b" mod="p">
  <e>
    <xsl:apply-templates mod="q"/>
  </e>
</xsl:template>
```
3. $(q, a) \rightarrow cp$:
```
<xsl:template match="a" mod="q">
  <c/>
  <xsl:apply-templates mod="p"/>
</xsl:template>
```
4. $(q, b) \rightarrow d(pq)$:
```
<xsl:template match="b" mod="q">
  <d>
    <xsl:apply-templates mod="p"/>
    <xsl:apply-templates mod="q"/>
  </d>
</xsl:template>
```

La règle (4) copie les fils du noeud courant deux fois : une copie est transformée dans l'état p et l'autre dans l'état q : elle est 2-copiante. La règle (3) copie les fils du noeud courant une seule fois mais aucun père n'est donné à cette copie : elle est effaçante. ♪

Précisons le coût algorithmique de la transformation d'un arbre :

Proposition 3.24

Pour un XSL-Transducteur $\mathscr{T}$ déterministe et un arbre t, on sait construire une représentation de $\mathscr{T}(t)$ en temps :

1. $\mathcal{O}(|t|)$ *pour $\mathscr{T}$ linéaire.*
2. $\mathcal{O}(\exp^{|t|})$ *pour $\mathscr{T}$ non linéaire.*

$\star$

Démonstration :

1. Le résultat naturel est un temps $\mathcal{O}(|t| + |\mathscr{T}_t|)$ qui est clairement linéaire en $|t|$.
2. Même avec de la non-linéarité, la taille de t reste bornée par β^j où $\beta = \max_{q \in Q, a \in \Sigma_S} (|\delta(q,a))|$ est le nombre maximal de nœuds dans la partie droite d'une règle de $\mathscr{T}$, et j est la hauteur de t. Comme $j \leq |t|$, on déduit que pour tout XSL-Transducteur déterministe $\mathscr{T}$, $|\mathscr{T}(t)| \leq \beta^{|t|}$, *i.e.* la sortie est au pire exponentielle en la taille de t. Sur l'alphabet unaire, une règle 2-copiante peut transformer un arbre filiforme à n noeuds en arbre binaire complet de profondeur n, à 2^n feuilles, et donc de taille $\Omega(\exp(|t|))$. Une représentation de $\mathscr{T}(t)$ peut donc potentiellement avoir une taille $\Theta(\exp^{|t|})$.

$\square$

Nous allons voir ce qui peut être généralisé pour des transducteurs d'arbres. Nous étendons d'abord la notion de $\mathcal{H}_{\mathscr{T}}$ pour un XSL-Transducteur puis on montre que l'ε'-inclusion entre les objets géométriques générés peut à nouveau servir de témoin à l'ε-inclusion des transducteurs.

3.2.2 Statistiques d'un XSL-Transducteur linéaire

Pour étendre la représentation aux XSL-Transducteurs linéaires, clarifions la notion de boucle. Soit $\widehat{\mathscr{T}}$ le graphe dont les sommets sont les états du XSL-Transducteur et dont chaque arête correspond à une une règle $(a, q) \to h \in \delta$. Le type d'une arête (ou d'une règle) désigne le type du segment reliant la (première) racine à la feuille étiquetée par un état, son entrée désignera a, sa sortie h. Un chemin est une succession d'arêtes et une boucle est un chemin d'un état à lui-même ; ils induisent donc :

- un segment d'entrée : son type est la concaténation des types des arêtes, et son étiquetage est la concaténation des entrées des arêtes
- une haie de sortie : pour $\pi_i.\pi_2$, dans π_1, la feuille étiquetée de la haie de sortie de π_1 est remplacée par la haie de π_2.

Comme pour le cas des mots, une boucle est dite *simple* lorsqu'elle ne contient aucune boucle strictement plus petite.

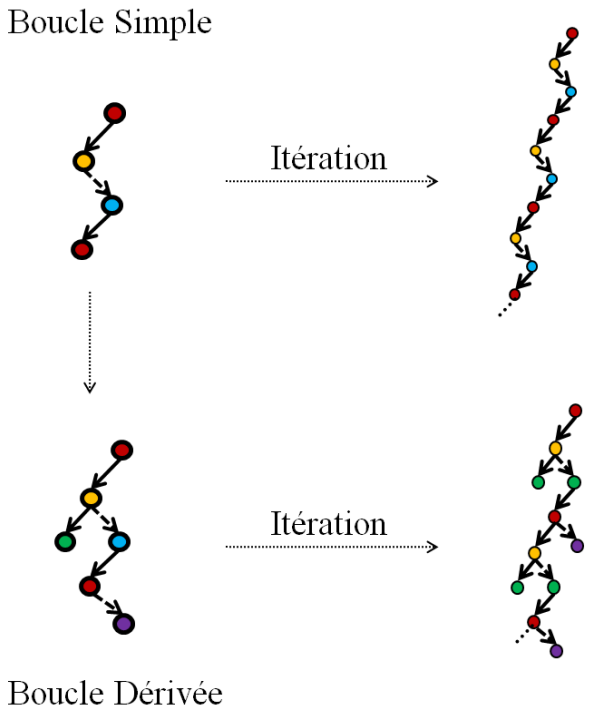


Fig. 3.8 – *Exemples d'itérations de boucles d'un XSL-Transducteur.*

Le cas d'une boucle

La statistique d'une boucle est de même nature que la statistique d'une boucle pour les mots, à la différence que la statistique d'entrée (et de sortie) est maintenant une statistique d'arbre étendu (de haie).

Comme dans le cas des mots, la statistique d'une boucle simple b d'un XSL-Transducteur se compose de trois parties : la statistique d'entrée de la boucle, sa statistique de sortie (deux statistiques d'arbres d'ordre k), et la distorsion $\frac{|out(b)|}{|in(b)|}$, où $|t|$ désigne le nombre de noeuds de l'arbre t.

Pour une décomposition sur une base donnée

Comme pour les String-Transducteurs, deux boucles sont compatibles si on peut les itérer (voir 3.8), dans n'importe quel ordre. La différence avec le cas des String-Transducteurs est que la statistique d'entrée (respectivement de sortie) d'une boucle est maintenant dotée d'un type : celui du chemin permettant d'affirmer qu'il s'agit d'une boucle. Comme précédemment, on pose

$$\mathcal{H}_S^{\alpha_1,\dots,\alpha_p} = \begin{pmatrix} \sum_i \alpha_i.\mathcal{I}_{s_i} \\ \left[\left[\left(\sum_{s_i \in S} \alpha_i.\frac{l_{out}(s_i)}{l_{in}(s_i)}.\mathcal{O}_{s_i}\right)\right]\right] \\ \sum_i \alpha_i.\rho_{s_i} \end{pmatrix},$$

où on interprète $\mathcal{I}_{s_i}$ (resp. $\mathcal{O}_{s_i}$) comme la statistique d'arbre de l'entrée (resp. de sortie) de la boucle, et l_{in} (resp. l_{out}) comme le nombre de noeuds en entrée (resp. en sortie) de la boucle s_i.

Donnons un exemple de ces constructions.

Exemple 3.25

Reprenons l'exemple du XSLT de la figure 3.7 (*page* 101) pour expliquer comment sont transformées les statistiques uniformes par ce transducteur.

Les transitions lisant *bd* ou *author* conservent (localement) la taille. Par contre, la transition lisant *work* ne préserve pas la taille et possède un facteur de distorsion de 2. [33] Pour la simplicité de l'explication, ce transducteur possède un unique état, et donc ses boucles simples sur l'unique état du transducteur sont, de fait, compatibles. On remarque que cet état q est relié à lui-même à la fois par une arête verticale et par une arête horizontale. Notons aussi que la première « template » ne s'applique qu'à la racine. Elle n'apparaît dans aucune boucle et sa contribution aux statistiques de sorties converge donc asymptotiquement vers 0.

Donnons finalement les statistiques de ces boucles dans une forme compacte où $t[e]$: val exprime que val est la valeur de ustat en la ligne t (un type de chemin) et en la colonne e (une séquence d'étiquettes).

- Pour $k = 1 : s = \begin{pmatrix} a:1 \\ a:1 \\ 1 \end{pmatrix}$, $s' = \begin{pmatrix} w:1 \\ (l:1/2, t:1/2) \\ 2 \end{pmatrix}$.
  $\mathcal{H}_{\mathscr{T}}$ est l'ensemble des points « interpolés » entre s et s' : pour $\alpha \in [0,1]$,
  $\alpha.s \oplus (1-\alpha).s'$ est associé à $\begin{pmatrix} (a:\alpha, w:1-\alpha) \\ 1/(2-\alpha)\,(a:\alpha, l:1-\alpha, t:1-\alpha) \\ 2-\alpha \end{pmatrix}$

- Pour $k = 2$, trois types de boucles sont à considérer dont voici les représentations :

$$s = \begin{pmatrix} 0[wa]:1 \\ (0[lt]:1/2, 1[ta]:1/2) \\ 2 \end{pmatrix} , \quad s' = \begin{pmatrix} 1[aa]:1 \\ 1[aa]:1 \\ 1 \end{pmatrix} , \quad s'' = \begin{pmatrix} 1[ww]:1 \\ 1[ll]:1 \\ 1 \end{pmatrix}$$

♪

[33] On passe localement d'un noeud *word* à l'arête gauche $0[lt]$ *i.e.* deux noeuds *livre* et *titre*.

Pour un XSL-Transducteur général

La généralisation des autres étapes est parfaitement identique au cas des String-Transducteurs :

$$\mathcal{H}_S = \bigcup_{\substack{0\leq\alpha_i\leq 1\\ \sum_i \alpha_i = 1}} H_S^{\alpha_1,\ldots,\alpha_p}$$

$$\mathcal{H}_\pi = \bigcup_{\text{ensemble } S \text{ de boucles } \pi\text{-compatibles}} \mathcal{H}_S$$

$$\mathcal{H}_{\mathscr{T}} = \bigcup_{\pi\in\widehat{\mathscr{T}}} \mathcal{H}_\pi$$

Exemple 3.26

La représentation statistique construite pour $k = 2$ dans l'exemple 3.25 (*page* 107) est :

$$\mathcal{H}_{\mathscr{T}} = \{\alpha.s \oplus \alpha'.s' \oplus \alpha''.s'' \mid \alpha + \alpha' + \alpha'' = 1\}\,.$$

Le point associé à $\alpha = 1/2$, $\alpha' = \alpha'' = \frac{1}{4}$ est alors

$$\frac{1}{2}.s \oplus \frac{1}{4}.s' \oplus \frac{1}{4}.s'' = \begin{pmatrix} 0[wa] : \frac{1}{2},\ 1[aa] : \frac{1}{4},\ 1[ww] : \frac{1}{4} \\ \left[\left[0[lt] : \frac{1}{2},\ 1[ta] : \frac{1}{2},\ 1[aa] : \frac{1}{4},\ 1[ll] : \frac{1}{4}\right]\right] \\ 2 \times \frac{1}{2} + \frac{1}{4} + \frac{1}{4} \end{pmatrix}$$

$$\textit{i.e.}\quad \mathcal{H}_{\mathscr{T}}^{\frac{1}{2},\frac{1}{4},\frac{1}{4}} = \begin{pmatrix} 0[wa] : \frac{1}{2},\ 1[aa] : \frac{1}{4},\ 1[ww] : \frac{1}{4} \\ 0[lt] : \frac{1}{3},\ 1[ta] : \frac{1}{3},\ 1[aa] : \frac{1}{6},\ 1[ll] : \frac{1}{6} \\ 3/2 \end{pmatrix}$$

♪

Proposition 3.27

Pour tout XSL-Transducteur $\mathscr{T}$, $\mathcal{H}_{\mathscr{T}}$ est une union finie d'arrangements d'hyperplans. ⋆

3.2.3 Équivalence Approchée de XSL-Transducteurs Linéaires

Nous avons vu également comment étendre la robustesse de ustat (proposition 2.3 (*page* 37)) au cas des arbres. Les preuves suivantes sont alors identiques au cas des String-Transducteurs si on remplace chaque utilisation de la robustesse sur les mots par sa version généralisée aux arbres : la proposition 2.22 (*page* 55).

Proposition 3.28

Si $\mathscr{T}_1$ et $\mathscr{T}_2$ sont deux XSL-Transducteurs équivalents Λ-libres, alors l'algorithme 6 renvoie vrai. $\star$

La propagation de l'erreur étant plus importante que pour le cas des String-Transducteurs, le pas de grille dans le cas des arbres doit donc être choisi plus finement que pour le cas des mots. Pour garantir la robustesse du test, l'erreur ε' utilisée pour le test devra alors être exponentiellement plus petite que l'erreur initiale ε. En adaptant la proposition 3.16 (*page* 95) avec le corollaire 2.22 (*page* 55), on peut à nouveau extraire une suite croissante sur la base infiniment répétée dont une sous-suite converge vers un point permettant de séparer les deux XSL-Transducteurs.

Proposition 3.29

Si un XSL-Transducteur linéaire Λ-libre $\mathscr{T}_1$ est ε-loin du transducteur Λ-libre $\mathscr{T}_2$, alors on peut trouver une boule, centrée dans $\mathcal{H}_{\mathscr{T}_2}$, et de rayon $\varepsilon' = \theta(\varepsilon \times (2^{1/\varepsilon} - 1))$, qui n'intersecte pas $\mathcal{H}_{\mathscr{T}_1}$. $\star$

Le théorème 3.20 (*page* 98) se généralise finalement en tenant compte de ce nouveau pas de grille.

Théorème 3.30

L'algorithme 6 est un ε-testeur pour l'inclusion des XSL-Transducteurs Λ-libres. $\bigstar$

Algorithmes :

Algorithme 5 : L'ε-inclusion pour les XSL-Transducteurs linéaires fortement connexes

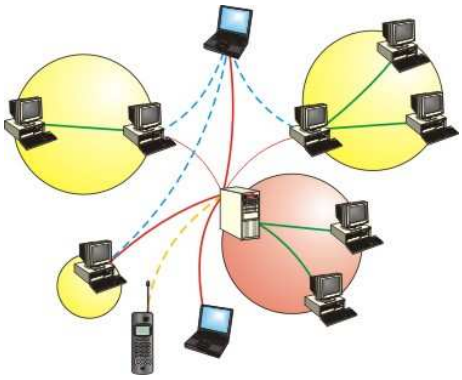
Données : Deux XSL-Transducteurs linéaires à un état ; $\varepsilon \in]0,1[$.
Sorties : La décision de $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$
Construire $\mathcal{H}_{\mathscr{T}_2}$ comme un système d'inéquations linéaires;
$\varepsilon^* := \varepsilon / \left(3(2^{1/\varepsilon})\right)$;
Construire $\mathcal{S}_2$ l'ensemble des systèmes d'équations de la relaxation de $\mathcal{H}_{\mathscr{T}_2}$ par ε^* ;
$Res := 1$;
pour chaque *boucle simple b de* $\mathscr{T}_1$ **faire**
  Calculer $x = (I_x, O_x, l_x)$;
  si *x n'est la solution d'aucun des systèmes de* $\mathcal{S}_2$ **alors** $Res := 0$
fin
retourner *Res*

Algorithme 6 : Testeur pour l'ε-inclusion

Données : Deux XSL-Transducteurs ; $\varepsilon \in]0,1[$
Sorties : La décision de $\mathscr{T}_1 \subseteq_\varepsilon \mathscr{T}_2$
Construire $\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$ comme ensembles de systèmes d'inéquations linéaires;
$\tilde{\varepsilon} := 0.1 \times \varepsilon / \left(3(2^{1/\varepsilon})\right) \times \frac{\rho_{min}}{\rho_{max}}$;
Construire $\mathcal{S}_1$ et $\mathcal{S}_2$ les systèmes d'inéquations des relaxations respectives de $\mathcal{H}_{\mathscr{T}_1}$ et $\mathcal{H}_{\mathscr{T}_2}$ par $\tilde{\varepsilon}/3$;
$Res := 1$;
pour chaque *x point de la grille sur* $[0,1]^{|\Sigma|^k} \times [0,1]^{|\Sigma'|^k} \times [1,\beta]$ *de pas* $\tilde{\varepsilon}/3$ **faire**
  si *x vérifie* $\mathcal{S}_1$ **alors**
    si *x n'est la solution d'aucun des systèmes de* $\mathcal{S}_2$ **alors** $Res := 0$
  fin
fin
retourner *Res*

Chapitre 4

Échange Approché de Données



Nous allons maintenant étudier ce que l'on appellera l'échange de données dont la philosophie peut être caricaturée de la façon suivante : j'ai mon langage (le français) dont je comprends une partie (mon schéma-source), tu as ton langage (l'espéranto) dont tu comprends une partie (le schéma-cible), et il y a un dictionnaire donnant des possibilités d'associations entre ces deux langues. Les deux questions centrales étudiées dans ce chapitre sont alors :

1. La consistance : si je pense à une phrase particulière en français, est-ce-qu'il est possible de trouver une phrase correspondante en espéranto que tu comprends ?
2. La sûreté : est-ce-que, pour toutes les phrases françaises que je comprends et quelque soit la manière dont j'utilise le dictionnaire, je tomberais toujours sur une phrase que tu comprends ?

L'idée est de considérer un schéma-source, un schéma-cible et des contraintes décrivant comment sont liées source et cible. Des cadres généraux ont précédemment été proposés pour les bases de données relationnelles [Fagin et al., 2002, Arenas et al., 2004, Fagin et al., 2005, Calvanese et al., 2006] ou XML [Arenas et Libkin, 2005]. Les travaux cités s'intéressent à des contraintes du type « si je satisfais telle propriété (formule)

sur la source, alors je dois aussi satisfaire telle autre propriété sur la cible ». Pour une instance-source donnée, une solution est une instance du schéma-cible telle que toutes les contraintes sont satisfaites. On parlera de donnée consistante si une telle solution existe, et il peut exister un nombre quelconque de solutions.

Cette manière d'associer une instance-source à un ensemble de solutions décrit indirectement des transformations d'instances. Dans les travaux existants cités, la construction d'une solution (si elle existe) s'effectue par des variantes de l'algorithme de « poursuite » de [Fagin et al., 2002]. Une manière directe de décrire des transformations d'instances est de le faire par un transducteur. Les questions de consistance et de sûreté sont alors reformulées pour des transducteurs de mots ou d'arbres, et les schémas choisis parmi les langages réguliers (de mots et d'arbres).

Savoir construire une solution lorsqu'elle existe peut s'appliquer dans de nombreuses situations pratiques. Chaque exemple de visualisation donné au chapitre 5 est en effet un cas particulier de la construction d'une représentation de l'ensemble des solutions, étant donnés une instance source et un « cadre d'échange » Δ (le schéma-source, le schéma-cible et le transducteur).
Savoir si un cadre d'échange est sûr est « le nerf de la guerre » du type-checking. Décider si un cadre d'échange est sûr est alors un problème qui apparaît naturellement pour les transducteurs d'arbres [Milo et al., 2000, Martens et Neven, 2002, Milo et al., 2003, Martens et al., 2008]. Ce problème de décision a également été abordé dans divers contextes : pour des transformations de XML basés sur les requêtes de chemins [Calvanese et al., 1999] ou pour les bases de données relationnelles transformées par des contraintes logiques (source-cible) [Alon et al., 2003] ...

Ces approches existantes ne considèrent que les versions exactes des problèmes de décision ; pourtant, dans un contexte de données imparfaites, l'aspect de robustesse est essentiel. On verra aussi que ces questions peuvent être difficiles dans le cas exact alors qu'il sera montré comment décider efficacement ses versions approchées.

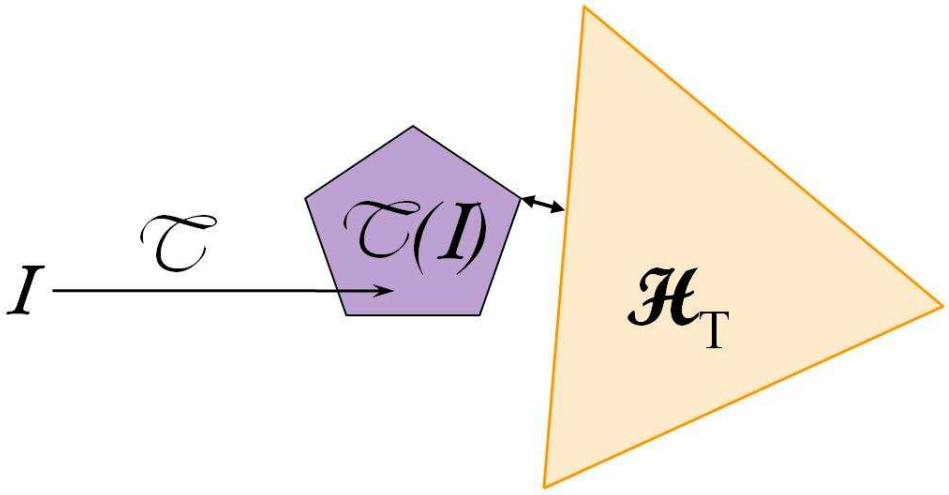
Organisation.

Consistance : On montre que la consistance exacte est linéaire (dans la taille du mot) pour les langages de mots réguliers et les machines à états finis. Puis l'approximation est introduite de trois manières : d'abord sur la source, puis sur la cible, puis à la fois sur la source et la cible. Le résultat principal est que l'on peut décider de l'ε-consistance d'un mot-source en temps constant (qui ne dépend que de ε, des schémas et du transducteur, et pas de la taille du mot-source). Cette propriété se déduit d'une analyse des statistiques de ce mot. La méthode consiste à approximer (par échantillonnage) la statistique de ce mot suffisamment précisément pour pouvoir situer ce point par rapport aux polytopes du langage-source $\mathcal{H}_S$ et aux polytopes associés au transducteur $\mathcal{H}_{\mathscr{T}}$.

Sûreté : Décider de la sûreté est un problème PSPACE-dur dans le cas des mots. La décision peut se faire efficacement en montrant que l'ensemble des mots sûrs est régulier. On montre comment, à partir de ce langage des mots sûrs, on peut introduire l'approximation avec des testeurs d'inclusion sur la source ou la cible. Dans le cas général, $\mathcal{H}_{\mathscr{T}}$ est restreinte en entrée aux polytopes du langage de l'entrée, et comparée (en ses coordonnées de sortie) aux polytopes du langage-cible.

Sommaire

4.1 Consistance



Une première question importante est de savoir si, pour une entrée-source I (typiquement dans le schéma-source $\mathbf{K}_S$), on peut produire, par la transduction, une instance-cible du schéma-cible $\mathbf{K}_T$. On va donc chercher à décider si c'est le cas, et si la réponse est positive, chercher à construire une telle instance-cible appelée *solution*.

Ce problème possède un vaste potentiel d'applications. Il est en effet aisé de modéliser des questions de mise en page ou de transformation de formats sous cette forme. Un objet (l'instance-source) de notre monde (le schéma-source), est traduit dans le monde de quelqu'un d'autre qui propose le format (le schéma-cible) dans lequel il veut obtenir l'information.

4.1.1 Consistance Exacte

Un schéma-source $\mathbf{K}_S$, un schéma-cible $\mathbf{K}_T$, et un transducteur $\mathscr{T}$ définissent un cadre d'échange de données $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$. Étant donnés Δ et une instance-source I appartenant au schéma-source $\mathbf{K}_S$, une *solution* du problème de source-consistance exacte (donné par (Δ, I)) est une instance-cible J appartenant au schéma-cible $\mathbf{K}_T$ telle que I peut être transformée en J par $\mathscr{T}$. On notera $Sol(I)$ l'ensemble (potentiellement infini) de ces solutions, et s'il est non vide, I est dite *source-consistante* pour Δ.

La définition de la source-consistance présentée ci-dessus dans un cadre général de classes munies de distance dénote les instances par I. Dans le cas spécifique des mots, cette instance-source est notée w ; dans le cas des arbres, elle est notée t.

Détaillons maintenant le traitement de la source-consistance dans le cas des mots. L'application de la proposition 3.4 (*page* 77) nous permet la construction d'un automate $\mathscr{T}(w)$ à $n_0 = |w|.n_1$ sommets et moins de $a_0 = 2|w|.n_1.a_1$ arêtes. On peut alors appliquer l'intersection (page 19) qui nous assure la construction de $\mathscr{T}(w) \cap \mathcal{A}_T$ par un automate à $n_0.n_2$ sommets et $a_0.a_2 + a_0.n_2 + a_2.n_0$ arêtes. En remplaçant n_0 et a_0 par leurs majorations respectives en $|w|$, n_1 et a_1, on obtient un automate à $|w|.n_1.n_2$ sommets et $2|w|.n_1.a_1.a_2 + 2|w|.n_1.a_1.n_2 + a_2.|w|.n_1 = \mathcal{O}(|w|.n_1.n_2.a_1.a_2)$ arêtes. On conclue alors :

Proposition 4.1
Soit w un mot, $\mathscr{T}$ un transducteur sous forme normale à n_1 sommets et a_1 arêtes et $\mathcal{A}_T$ un automate (NFA) à n_2 sommets et a_2 arêtes. $Sol(w)$ peut être construit en temps $\mathcal{O}(|w|.n_1.n_2.a_1.a_2)$. La représentation résultante est un NFA à $|w|.n_1.n_2$ sommets et $\mathcal{O}(|w|.n_1.n_2.a_1.a_2)$ arêtes. ⋆

Exemple 4.2
Soit $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$, où :

- le schéma-source $\mathbf{K}_S$ est reconnu par l'automate $\mathcal{A}_S$ de la figure 4.1 ;
- le String-Transducteur $\mathscr{T}$ est donné par la figure 4.2 (*page* 116) ;
- le schéma-cible $\mathbf{K}_T$ est reconnu par l'automate $\mathcal{A}_T$ de la figure 4.3 (*page* 116).

Un mot $w = 21222$ peut être traduit par $\mathscr{T}$ de deux manières :
- en 34455 : via le chemin $q_0 \xrightarrow{2:3} q_4 \xrightarrow{1:4} q_4 \xrightarrow{2:4} q_5 \xrightarrow{2:5} q_5 \xrightarrow{2:5} q_5$.
- en 34464 : via le chemin $q_0 \xrightarrow{2:3} q_4 \xrightarrow{1:4} q_4 \xrightarrow{2:4} q_5 \xrightarrow{2:6} q_4 \xrightarrow{2:4} q_5$.

Un chemin de $\mathcal{A}_T$ accepte 34455 : $q_0 \xrightarrow{3} q_2 \xrightarrow{4} q_2 \xrightarrow{4} q_2 \xrightarrow{5} q_2 \xrightarrow{5} q_2$, ce qui signifie que 34455 appartient au langage-cible. Aucun chemin de $\mathcal{A}_T$ n'accepte le mot 34464 : il n'appartient donc pas au langage-cible. $Sol(w)$ est alors réduit à $\{34455\}$.

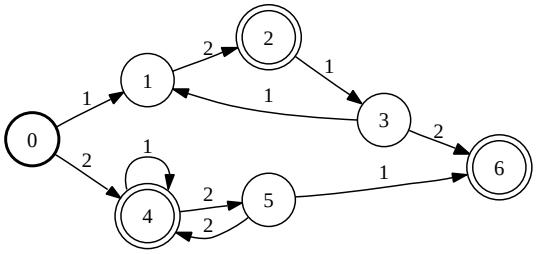


Fig. 4.1 – *L'automate-source* $\mathcal{A}_S$

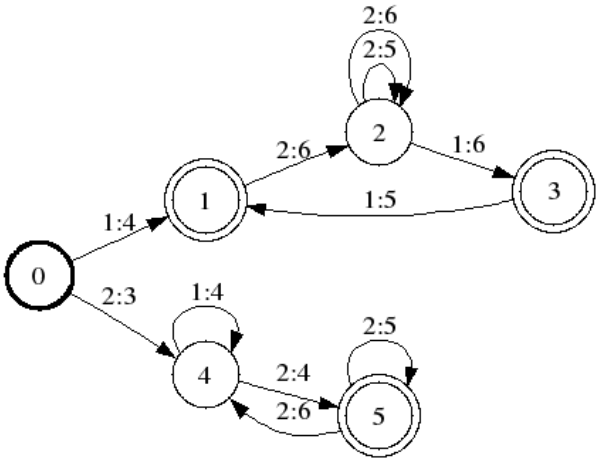


Fig. 4.2 – *le transducteur* $\mathcal{T}$

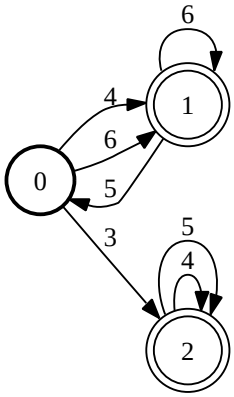


Fig. 4.3 – *L'automate-cible* $\mathcal{A}_T$

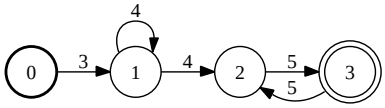


Fig. 4.4 – *L'automate* $Sol(\mathcal{A}^{in}_{4.2})$

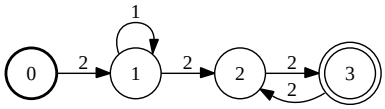


Fig. 4.5 – *L'automate des mots source-consistants*

Plus généralement, l'ensemble des solutions pour n'importe quel mot reconnu de l'automate-source $\mathcal{A}_S$ est représenté par la figure 4.4 et correspond à $Sol(\mathcal{A}_S)$.

On calcule également l'ensemble des mots-sources possédant un ensemble non vide de solutions que l'on représente dans la figure 4.5. ♪

Corollaire 4.3 (source-consistance)
Supposons $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$ *fixés :* $\mathscr{T}$ *donné par un String-Transducteur normalisé,* $\mathbf{K}_S$ *reconnu par l'automate-source* $\mathcal{A}_S$ *sur l'alphabet* Σ*, et* $\mathbf{K}_T$ *par l'automate-cible* $\mathcal{A}_T$*. Pour tout mot* $w \in \Sigma^*$*, décider si* w *est source-consistant pour* Δ *se fait en temps* $\mathcal{O}(|w|)$*. De plus, un automate (NFA) pour* $Sol(w) = \mathscr{T}(w) \cap \mathbf{K}_T$ *peut être construit en temps* $\mathcal{O}(|w|)$*.*
⋆

En effet, déterminer si w est source-consistant revient exactement à déterminer si $Sol(w)$ est vide. Par le théorème 4.1 (*page* 115), la construction de $Sol(w)$ produit un NFA possédant un nombre linéaire d'arêtes et de sommets. L'application directe du test du vide (*page* 19) nous permet de traiter cette question en temps et en espace linéaire en le nombre d'arêtes de $Sol(w)$, et donc en temps et en espace $\mathcal{O}(|w|)$. Les autres cadres nous permettraient des réductions de l'espace utilisé.

4.1.2 Consistance Approchée

On va maintenant chercher à être insensible à un éventuel bruit et donc à tolérer une proportion d'erreurs. Nous avons trois manières de les tolérer :

1. en autorisant de modifier la source,
2. en autorisant de corriger la ou les cibles produites, et
3. en permettant de corriger à la fois le coté source et le coté cible.

Modifier la source

Tolérer une correction uniquement sur l'entrée revient à être proche de $\mathcal{A}_1 = \mathscr{T}^{-1}(\mathcal{A}_T)$. Une manière de faire est de :

- calculer $\mathcal{A}_1$ et $\mathcal{H}_{\mathcal{A}_1}$.
- approcher $\mathsf{ustat}(I)$ et en estimer la distance à $\mathcal{H}_{\mathcal{A}_1}$.
- accepter si cette distance est petite (inférieure à $f(\varepsilon)$) et rejeter dans le cas contraire.

Pour les mots : L'image inverse d'un automate par un transducteur —détaillée *page* 77 — montre que $\mathcal{A}_1$ est un langage régulier et donne une méthode pour le construire. La proximité d'un mot à ce langage peut alors être décidée par un testeur d'appartenance. On calcule $\mathcal{H}(\mathcal{A}_1)$ par la définition 2.30 (*page* 66). On approxime $\mathsf{ustat}(w)$ par $\widehat{\mathsf{ustat}(w)}$ en tirant des sous-mots aléatoires (théorème 2.8 (*page* 42)). On calcule finalement la distance géométrique entre $\widehat{\mathsf{ustat}(w)}$ et $\mathcal{H}(\mathcal{A}_1)$. Comme dans [Fischer et al., 2006], il s'agit d'un testeur d'appartenance, construit en temps polynomial dans le schéma, et qui, une fois construit, sépare en temps constant les entrées source-consistantes et celles loin d'être source-consistantes.

Exemple 4.4

Dans l'exemple 4.2, on construit la représentation de l'automate de la figure 4.6. Un échantillonnage ne donnant que le sous-mot 1^p sera accepté puisque proche des statistiques de la boucle de l'état 6. Un mot 1^p peut effectivement être corrigé en 21^p2 qui est traduit en 34^p4 et accepté par l'automate-cible $\mathcal{A}_T$ (Figure 4.3 (*page* 116)) par le chemin $q_0 \xrightarrow{3} q_2 \xrightarrow{4} q_2 \xrightarrow{4} q_2 \xrightarrow{4} q_2 \ldots \xrightarrow{4} q_2 \xrightarrow{4} q_2$. ♪

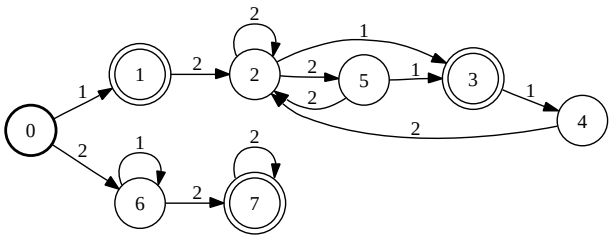


Fig. 4.6 – *Pré-image de l'automate-cible :* $\mathcal{A}_1 = \mathscr{T}^{-1}(\mathcal{A}_T) \cap \mathcal{A}_S$

Pour les arbres : La transduction inverse garantit également une image réciproque régulière. On sait alors construire $\mathcal{A}_1$ par la transduction inverse[34] et en déduire $\mathcal{H}(\mathcal{A}_1)$. Les statistiques de l'arbre-source sont ensuite approximées par échantillonnage de petits sous-arbres (taille inférieure à $k = 1/\varepsilon$). On accepte[35] si la norme $\mathbf{L}_1$ entre le vecteur de statistiques obtenu et $\mathcal{H}_{\mathcal{A}_1}$ est inférieure à $f(\varepsilon)$, et on rejette sinon.

Modifier la cible

De manière symétrique, si on souhaite une robustesse vis-à-vis de la sortie, la distance à considérer dans l'espace statistique (de sortie) est celle entre $\mathsf{ustat}(\mathscr{T}(I))$ et $\mathcal{H}_{\mathcal{A}_T}$. On peut alors construire $\mathcal{H}_{\mathcal{A}_T}$, construire $\mathscr{T}(I)$ explicitement, échantillonner ses statistiques avec le chapitre 2, et finalement accepter lorsque la distance (norme $\mathbf{L}_1$) entre $\mathcal{H}_{\mathcal{A}_T}$ et $\widehat{\mathsf{ustat}(\mathscr{T}(I))}$ est suffisamment petite (inférieure à $f(\varepsilon)$).

ε-Source-Consistance.

Du point de vue pratique, le cas le plus intéressant est de tolérer des erreurs à la fois sur la source et la cible. Une instance est alors dite *ε-source-consistante* si elle est ε-proche d'une instance du schéma-source qui est traduite vers une instance ε-proche du schéma-cible.

Présentons l'algorithme 7 (*page* 120), un ε-Testeur pour la ε-Source-consistance, d'abord dans le cas d'un transducteur déterministe à un état, avant de l'étendre à plusieurs états.

[34] dans le pire des cas en $\mathcal{O}(\exp(\exp(\|\mathcal{A}_T\|)))$.
[35] *i.e.* Vrai est renvoyé

Algorithme 7 : Testeur$_1(w, k, N)$: ε-Source-Consistance à un état

Données : Un mot w, un transducteur $\mathscr{T}$ déterministe à un état,
le schéma-cible $\mathbf{K}_T$ donné sous la forme de sa représentation statistique $\mathcal{H}_{\mathbf{K}_T}$

Résultat : La décision de l'ε-Source-Consistance de w

$k := 1/\varepsilon$;

$\alpha := \min_{a \in \Sigma_s} |\mathscr{T}(a)| > 0$;

$\beta := LCM_{a \in \Sigma_s} |\mathscr{T}(a)|$; /* le plus petit multiple commun des longueurs $\mathscr{T}(a)$ */

tant que N *sorties n'ont pas encore été engendrées* **faire**

  Choisir i dans $\{1, n\}$ uniformément ;

  $a := w[i]$; $\gamma := |\mathscr{T}(a)|$;

  Choisir $b \in_r \{0, 1\}$ avec $Prob(b = 1) = \frac{\gamma}{\beta}$;

  si b=*1* **alors**

    Choisir $j \in_r \{0, ..., \gamma - 1\}$ uniformément et sortir le sous-mot $X_{i,j}$ de longueur k commençant à la position j de $\mathscr{T}(a)$ composé des k lettres suivantes sur la droite.

  fin

fin

$\forall u \in \Sigma^k$, $\widehat{ustat}[u] = \frac{1}{N} \sum \chi_{X_{i,j}=u}$;

si $\mathsf{dist}_{geom}\left(\widehat{ustat}, H_{\mathbf{K}_T}\right) > \varepsilon$ **alors retourner** *Faux* ; **sinon retourner** *Vrai* ;

L'objectif est d'échantillonner le mot-source w (tirer des sous-mots de w) afin d'approcher $\mathsf{ustat}(\mathscr{T}(w))$. Cet objectif nécessite quelques précautions. Si le transducteur $\mathscr{T}$ ne conserve pas la longueur des sous-mots (voir Exemple 4.6 (*page* 122)), le tirage uniforme sur w « produit » un tirage non-uniforme sur $\mathscr{T}(w)$ et donc ne permet pas d'approcher les statistiques $\mathsf{ustat}(\mathscr{T}(w))$.

Pour résoudre ce biais, lors du tirage d'une position i sur w, l'échantillonnage va prendre en compte la longueur de l'image de $w[i]$ par $\mathscr{T}$ (Exemple 4.6).

Lemme 4.5

Fixons $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$, un cadre d'échange, d'alphabet d'entrée Σ, d'alphabet de sortie Σ', et dont le transducteur a une distorsion dans $[\alpha, \beta]$. Pour $0 < \varepsilon < 1$ fixé, supposons $N_0 = \Omega(\beta \ln(1/\varepsilon) \ln(|\Sigma'|)/(\alpha\varepsilon^3))$.

Pour tout mot-source w tel que $|w| = n \geq N \geq N_0$, Testeur$_1(w, \lceil \frac{1}{\varepsilon} \rceil, N)$ est un ε-testeur qui décide si w est ε-source-consistant par rapport à Δ. $\star$

Démonstration : Si w est consistant, le testeur accepte avec grande probabilité. Concentrons-nous donc sur la seconde condition de la définition d'ε-testeur. Soit $|w| = n$ et $X_{i,j}$ une sortie (*i.e.* un sous-mot de longueur k) tirée pour les choix i, j et b. Pour $u \in |\Sigma'|^k$ on définit

$$\widehat{ustat_T}(w, k, N)[u] = \sum_{i=1}^{N} \frac{\#\{X_{i,j} = u\}}{N}$$

Tout couple (i, j) est choisi avec probabilité $\frac{1}{n\beta}$, chaque $X_{i,j}$ est alors obtenu avec la même probabilité ce qui assure que l'espérance de $\widehat{ustat_T}$ est $\mathsf{ustat}(\mathscr{T}(w))$. Nous pouvons alors appliquer la borne de Chernoff pour chaque coordonnée u :

$$Pr(|\widehat{ustat_T}(w, k, N)[u] - \mathsf{ustat}(\mathscr{T}(w))[u]| > \varepsilon) \leq 2\exp(-2\varepsilon^2 N)$$

On applique ensuite une « union bound » à l'ensemble des coordonnées et on remplace N par sa borne inférieure :

$$Pr(|\widehat{ustat_T}(w, k, N) - \mathsf{ustat}(\mathscr{T}(w))| > \varepsilon) \leq |\Sigma'|^k . \exp(-2\varepsilon^2 N_0) \leq 1/6$$

Si on répète $\Theta(N_0.\beta \ln(1/\delta)/\alpha)$ fois la boucle externe, on obtient N échantillons avec une erreur bornée par δ, *i.e.* arbitrairement petite (à la place de 1/6). Il reste à appliquer un testeur d'appartenance de $\widehat{ustat_T}(w, k, N)$ à H_T. Nous acceptons alors lorsque la distance est inférieure à $2.\varepsilon$, et rejetons si ce n'est pas le cas. On conclut que Testeur$_1(w, k, N)$ est un $2.\varepsilon$-testeur. $\square$

Exemple 4.6

Soit $\mathscr{T}$ le transducteur à un état avec $\mathscr{T}(0) = ab$, $\mathscr{T}(1) = a$, $\mathscr{T}(2) = ccc$ et considérons l'entrée $w = 0021111$.

Lorsque l'on tire aléatoirement dans w une position correspondant à une lettre 1, cela correspond à un seul sous-mot (commençant par a) dans l'image. Lorsqu'il s'agit d'un 0 (respectivement 2), on engendre deux (resp. trois) nouvelles positions dans le mot-cible. On va donc tirer aléatoirement un décalage uniformément dans $\{0, 1\}$ (resp. $\{0, 1, 2\}$). Il faut alors considérer les tirages de 2 deux fois moins souvent que les tirages de 1, et trois fois moins souvent que les tirages de 3 pour simuler l'uniformité des tirages dans le mot cible $w' = ababcccaaaa$.

Cette manière d'approcher les statistiques de w' converge vers la statistique

$$\begin{aligned} &aa : 4/7.1/6.3/4 = 1/14 \\ &ab : 2/7.1/3.1/2 = 1/21 \\ &bc : 1/7.1/3.1/2 = 1/42 \quad \approx (0.34, 0.23, 0, 0, 0, 0.1, 0.1, 0, 0.2)\,. \\ &ca : 1/7.1/3.1/2 = 1/42 \\ &cc : 1/7.1/2.2/3 = 1/21 \end{aligned}$$

Les statistiques d'ordre 2 de w' sont en fait $(0.33, 0.22, 0, 0, 0, 0.1, 0.1, 0, 0.22)$. ♪

Une nouvelle fois, les effets de bords empêchant l'exactitude du calcul se réduisent proportionnellement à n, la taille du mot-source w. Remarquons aussi que cette inexactitude rentre tout à fait dans l'esprit de l'approche : on cherche à estimer les statistiques produites en évitant justement la construction et le calcul exact et explicite.

Pour un String-Transducteur déterministe et linéaire.

Soit $\mathscr{T}$ un String-Transducteur déterministe et linéaire à m états. Pour généraliser le testeur précédent, on va échantillonner des sous-mots u de w qui conduisent à des sous-mots de $\mathscr{T}(w)$. Gardons en tête qu'à priori, on ne sait pas dans quel état se trouvera $\mathscr{T}$ lorsqu'il traduira u. On considérera tous les états possibles $S_u \subseteq Q$ en assurant cependant que les états considérés $S_{u_1}, ...S_{u_N}$ sont compatibles, *i.e.* ils peuvent être parcourus en restant dans un ensemble π de composantes de $\widehat{\mathscr{T}}$. Nous énumérons donc tous les π de ce graphe acyclique $\widehat{\mathscr{T}}$. A chaque fois, nous appliquons un testeur, qui teste si w est proche d'un mot $\widehat{w}$, tel qu'une exécution de $\widehat{w}$ commençant dans un état de π ne parcourt que les composantes connexes de π.

Algorithme 8 : Testeur$_2(w, k, N, \pi)$: ε-Source-Consistance pour un Chemin

Données : Un mot w, un transducteur $\mathscr{T}$ linéaire déterministe,
le schéma-source $\mathbf{K}_S$ donné sous la forme de sa représentation statistique $\mathcal{H}_{\mathbf{K}_S}$

Résultat : La décision de l'ε-Source-Consistance de w

Générer $u_1...u_N$ des sous-mots de longueur k de w.

Estimer $\widehat{\mathsf{ustat}(w)}$;

si $\widehat{\mathsf{ustat}(w)}$ *est ε-loin de* $\mathcal{H}_\pi$ **alors** Rejeter;

sinon

associer un ensemble d'états S_{u_i} pour chaque u_i pour lesquels u_i est π-compatible.

Appliquer le Testeur$_1(w, k, N)$ pour chaque état S_{u_i} possible, puisque $\mathscr{T}(u_i)$ est bien défini.

Accepter s'il y a un choix d'états tels que Testeur$_1(w, k, N)$ accepte, rejeter sinon.

fin

Un ensemble π de composantes connexes est *admissible pour* $\mathcal{A}$ s'il existe un mot x et un état q dans une des composantes connexes de π tels que l'exécution de x depuis q rencontre exactement les composantes connexes de π. Un tel mot x est dit π-compatible pour q.

Définition 4.7

Soit $\mathscr{T}$ un String-Transducteur , π un chemin dans le graphe de ses composantes fortement connexes $\widehat{\mathscr{T}}$, et $\mathbf{K}_T$ un schéma-cible. Un mot w est ε-source-consistant *le long de* π, s'il existe un mot w' (ε-proche de w) qui est π-compatible depuis l'état initial q_0 et tel que $\mathscr{T}(w')$ est ε-proche de $\mathbf{K}_T$. ✓

On remarque alors qu'un mot w est ε-source-consistant si et seulement si il existe un chemin π tel que w soit ε-source-consistant le long de π.

Lemme 4.8

Fixons $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$, un cadre d'échange, d'alphabet d'entrée Σ, d'alphabet de sortie Σ', et dont le String-Transducteur $\mathscr{T}$ a une distorsion dans $[\alpha, \beta]$. Pour $0 < \varepsilon < 1$ fixé, supposons $N_0 = \Omega(\beta \ln(1/\varepsilon) \ln(|\Sigma'|)/(\alpha\varepsilon^3))$, et fixons π un chemin de $\widehat{\mathscr{T}}$, le graphe des composantes fortement connexes de $\mathscr{T}$.

Pour tout mot-source w tel que $|w| = n \geq N \geq N_0$, $Testeur_2(w, \lceil \frac{1}{\varepsilon} \rceil, N, \pi)$ est un ε-testeur qui décide si un mot w est ε-source-consistant le long de π. ⋆

Démonstration : La correction de cet algorithme se déduit de la correction de l'algorithme pour un état. Un mot consistant peut être décomposé le long de π et fournit ainsi les u_i et des états pour lesquels l'algorithme renvoie vrai. Si ce n'est pas le cas, c'est le témoin que quelle que soit la manière de traduire l'instance-source, on restera toujours loin (du point de vue statistique) des statistiques du schéma-cible, preuve que $\mathscr{T}(w)$ est loin de $\mathbf{K}_T$ pour la distance d'édition avec déplacements. □

Il reste finalement à décider si un tel π existe :

Algorithme 9 : Testeur$_3(w, k, N)$: ε-Source-Consistance sur les Mots

Générer tous les π et appliquer Testeur$_2(w, k, N, \pi)$.

si *il existe un* π *tel que Testeur*$_2(w, k, N, \pi)$ *accepte* **alors** Accepter **sinon** Rejeter

Théorème 4.9

Fixons $\Delta = (\mathbf{K}_S, \mathscr{T}, \mathbf{K}_T)$*, un cadre d'échange, d'alphabet d'entrée* Σ*, d'alphabet de sortie* Σ'*, et dont le String-Transducteur* $\mathscr{T}$ *a une distorsion dans* $[\alpha, \beta]$*. Pour* $0 < \varepsilon < 1$ *fixé, supposons* $N_0 \geq \beta \ln(1/\varepsilon) \ln(|\Sigma'|)/(\alpha\varepsilon^3)$*. Pour tout mot-source* w *tel que* $|w| = n \geq N \geq N_0$*, Testeur*$_3(w, k, N)$ *est un* ε*-testeur qui décide si un mot* w *est* ε*-source-consistant.* ★

Démonstration : La correction est à nouveau déduite de celle de l'algorithme précédent. Un mot consistant suit un certain π pour produire une instance du schéma-cible et l'algorithme acceptera pour ce π particulier. Réciproquement, si un mot est loin d'être consistant, n'importe quel π donnera une image loin du schéma-cible. Les statistiques de l'image seront donc, pour chaque π, loin de $\mathcal{H}_{\mathbf{K}_T}$ et, par le lemme précédent, l'algorithme renvoie Faux. □

Interprétation : Soit $\mathsf{ustat}(w)$ le vecteur statistique de dimension $|\Sigma|^k$, associé à w. Nous avons deux opérations à effectuer pour montrer que w est ε-consistant :

- trouver w' proche de w et lisible par $\mathscr{T}$.
- trouver une image de $\mathscr{T}(w')$ proche de $\mathbf{K}_T$.

Soit $\mathcal{H}_T$ l'union de polytopes (cible) associée à $\mathbf{K}_T$. Considérons l'automate associé à $\mathscr{T}$ où tous les états sont acceptants et où l'on ignore les sorties. Soit $G_{\mathscr{T}}$ l'union de polytopes associée au langage accepté par cet automate. Chaque polytope correspond à un π. Chaque polytope de $G_{\mathscr{T}}$ qui est ε-proche de $\mathsf{ustat}(w)$ indique un possible w' donné par son vecteur statistique $\mathsf{ustat}(w')[u]$ le long du π fixé.

Définissons $\mathsf{ustat}(\mathscr{T}(w'))$ relativement à π comme l'ensemble de vecteurs statistiques $\mathsf{ustat}[v]$ de dimension $|\Sigma'|^k$ qui sont des images possibles de w' par $\mathscr{T}$. Cet ensemble est défini comme l'enveloppe convexe C des vecteurs limites $\mathsf{ustat}[v]$ tels qu'il existe des états $s_1.....s_p$ dans π tels que pour un i $\mathscr{T}(s_i, u_i) = v_j$, et $\mathsf{ustat}[v_j] = \mathsf{ustat}(w')[u_i]$ si $|v_j| = k$. Si $|v_j| > k$, alors le poids de $\mathsf{ustat}(w')[u_i]$ est uniformément distribué sur tous les sous-mots v de longueur k de v_j. L'ensemble des i pour lesquels il n'y a pas de tels s_i doit être de densité inférieure à ε.

Le testeur est donc le suivant : si C est ε-proche de $\mathcal{H}_T$, alors Accepter, sinon Rejeter.

4.1.3 Généralisations

Transducteurs d'Arbres à Un État

Pour les arbres, c'est la matrice statistique de $\mathscr{T}(I)$ qui doit être approchée pour être confrontée à la représentation géométrique de la DTD-cible. L'algorithme renvoie des chemins de longueurs $k-1$ de l'encodage fcns. Ces chemins doivent être générés de façon uniforme (pour la sortie). La méthode est alors adaptée en permettant une expansion locale au noeud choisi (par au plus k transitions successives) qui détermine la probabilité de devoir tirer un chemin depuis l'image de ce noeud. L'algorithme 10 présente le test de la consistance pour un transducteur à un état. Implicitement, les chemins sont des chemins dans l'encodage fcns étendu de l'arbre.

La probabilité d'échec d'un tirage particulier de cet algorithme est augmentée, par rapport aux mots, par deux facteurs : l'augmentation du facteur multiplicatif (β) à considérer et l'augmentation de la probabilité d'échec (lors d'un appel récursif). Dans le cas des mots, les lettres de la source qui ne produisent pas nécessairement de statistiques d'ordre k dans l'image, sont nécessairement parmi les k dernières lettres. C'est la véritable différence avec le cas des arbres où typiquement un nombre quelconque de sous-arbres[36] peut n'engendrer aucune statistique d'ordre k.

Malgré tout, ces deux facteurs peuvent être majorés indépendamment de l'entrée :

- β ne dépend que de $\mathscr{T}$
- pour tout arbre et tout k, la proportion de noeuds qui ne sont racines d'aucune statistique d'ordre k est inférieure à $\frac{k-1}{k}$.

Comme précédemment, le vecteur $\widehat{ustat}$ — moyenne des chemins renvoyés par l'algorithme — possède une espérance égale à $\mathsf{ustat}(\mathscr{T}(I))$ et peut alors être confronté à la représentation géométrique de la DTD-cible. La distance ($\mathbf{L}_1$) entre est $\widehat{ustat}$ et $\mathcal{H}_{\mathbf{K}_T}$ reflète alors la distance (d'édition avec déplacement) entre $\mathscr{T}(I)$ et $\mathbf{K}_T$.

[36] les arbres verticaux de taille $k-1$ touchant la frontière par exemple

Algorithme 10 : Testeur$'_1(I, k, N)$: ε-Source-Consistance à un État

Données : Un arbre I, un XSL-Transducteur $\mathscr{T}$ déterministe à un état,
le schéma-cible $\mathbf{K}_T$ donné sous la forme d'une représentation statistique robuste $H_{\mathbf{K}_T}$
Résultat : La décision de l'ε-Source-Consistance de I
$k := 1/\varepsilon$;
$\alpha := \min_{a \in \Sigma_s} |\mathscr{T}(a)| > 0$;
$\beta := LCM_{|J| \leq k} 2^{|\mathscr{T}(J)|}$; /* une borne sur le nombre de chemins engendrés depuis l'image d'une transition */
tant que N *sorties n'ont pas encore été produites* **faire**
  Choisir un noeud i uniformément ;
  $a := tag[i]$; $\gamma := |\mathscr{T}(a)|$;
  Π_1 : Chemins de $\mathscr{T}(a)$ de longueur $k-1$;
  Π_2 : Chemins de $\mathscr{T}(a)$ de longueur $< k-1$ se terminant par un noeud f_i étiqueté par un état (sur la frontière de $\mathscr{T}(a)$) ;
  pour *chaque* $\pi \in \Pi_2$ *de feuille* f_i **faire**
    appliquer localement la transduction jusqu'à pouvoir déduire le nombre c_i de chemins enracinés en f_i et de longueur $k - (1 + |\pi|)$;
  fin
  Choisir $b \in_r \{0,1\}$ avec $Prob(b=1) = \dfrac{\#\{\Pi_1\} + \sum_i c_i}{\beta}$;
  si *b=1* **alors**
    début
      Tirer $\pi \in \Pi_1 \cup \Pi_2$ suivant la distribution induite par les poids : 1 pour un $\pi \in \Pi_1$ et c_i pour $\pi \in \Pi_2$;
      si $\pi \in \Pi_1$ **alors** Sortir le chemin π;
      sinon
        I_i : le sous-arbre de I enraciné en f_i;
        Sortir $concat(\pi, Testeur_1(I_i, k - (1 + |\pi|), 1))$ où la concaténation se fait par l'arête (gauche ou droite) spécifiée par f_i
      fin
    fin
  fin
fin
si $\mathsf{dist}_{geom}\left(\widehat{ustat}, H_{\mathbf{K}_T}\right) > \varepsilon$ **alors**
  retourner *Faux* ; **sinon retourner** *Vrai* ;
fin

Intégrer Différentes Erreurs

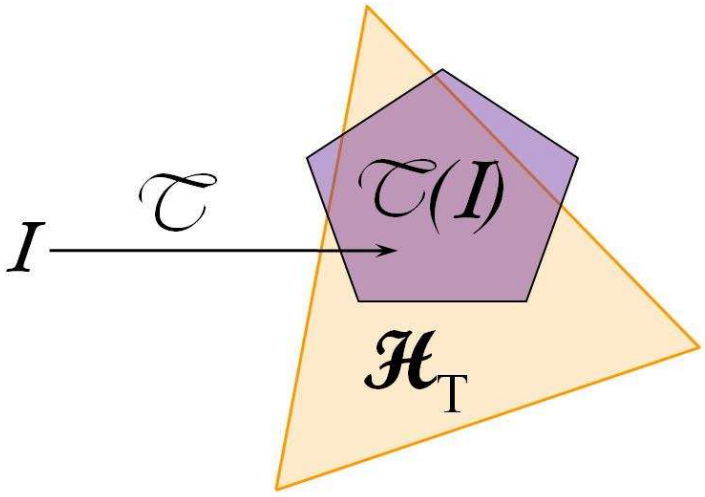
Pour tolérer différentes d'erreurs (sur la source, la cible...), une relaxation générique est proposée. Le principe de la décision suivra alors l'algorithme 11, dans lequel il convient de choisir chaque ε_i en appliquant la fonction de robustesse[37] à l'erreur correspondante que l'on veut tolérer.

Algorithme 11 : Principe du Testeur de la Source-Consistance Approchée

Données : une instance I, les paramètres ε_S, ε_I, ε_J et $\varepsilon_T \in]0,1[$
un transducteur $\mathscr{T}$ donné sous la forme de sa représentation statistique $\mathcal{H}_{\mathscr{T}}$
le schéma-source $\mathbf{K}_S$ donné par $\mathcal{H}_{\mathbf{K}_S}$ et le schéma-cible $\mathbf{K}_T$ donné par $\mathcal{H}_{\mathbf{K}_T}$
Résultat : La décision de la Source-Consistance approchée de I
début
  Estimer $\mathsf{ustat}(I)$ par un vecteur λ;
  $\mathcal{H}'_{\mathbf{K}_S}$: la relaxation de $\mathcal{H}_{\mathbf{K}_S}$ par le paramètre ε_S ;
  $\mathcal{H}'_{\mathscr{T}}$: la restriction de $\mathcal{H}_{\mathscr{T}}$ en l'entrée λ;
  $\mathcal{H}''_{\mathscr{T}}$: la relaxation sur l'entrée de $\mathcal{H}'_{\mathscr{T}}$ par le paramètre ε_I ;
  $\mathcal{H}'''_{\mathscr{T}}$: la relaxation de $\mathcal{H}''_{\mathscr{T}}$ sur la sortie par le paramètre ε_J ;
  $\mathcal{H}'_{\mathbf{K}_T}$: la relaxation de $\mathcal{H}_{\mathbf{K}_T}$ par le paramètre ε_T ;
  $\mathcal{H}$: la conjonction de $\mathcal{H}'_{\mathbf{K}_S}$, $\mathcal{H}'''_{\mathscr{T}}$ et $\mathcal{H}'_{\mathbf{K}_T}$;
  si $\mathcal{H}$ *a une solution* **alors** Accepter **sinon** rejeter
fin

[37] Essentiellement $\varepsilon^3 \to \varepsilon$ pour les mots et $2^\varepsilon \to \varepsilon$ pour les arbres.

4.2 Sûreté



« C'est bien de s'échanger des données mais pourquoi ne pas produire que du pertinent.[38] »

Intuitivement ce sage penseur avait envie d'un échange de données sûr, c'est-à-dire que quelle que soit la manière dont je traduise mon entrée, j'arriverais toujours dans le bon format (l'espéranto que tu comprends). On dira d'une instance-source qu'elle est *sûre* lorsque n'importe quelle transduction arrive dans le schéma-cible. Formellement, I une instance-source de $\mathbf{K}_S$ est sûre si pour tout $(I, J) \in \mathscr{T}$, $J \in \mathbf{K}_T$.

Pour un transducteur déterministe, la sûreté et la consistance sont deux notions équivalentes. Sachant comment décider de la source-consistance en temps linéaire (propriété 4.1 (*page* 115)), on sait aussi décider de la sûreté de celle-ci en temps linéaire. Si le transducteur est non déterministe et/ou que l'on veut vérifier la sûreté d'une classe, la consistance et la sûreté sont deux notions à priori différentes : une instance (ou une classe) peut ainsi être consistante sans être sûre (voir l'Exemple 4.11 (*page* 133)).

[38] philosophe austère contemporain

4.2.1 Décision Exacte

Pour les langages de mots

L'approche directe regroupe le coût de la transduction et le coût de l'inclusion. La décision de la sûreté d'un langage-source est alors un problème PSPACE-complet. En pensant au transducteur identité, on retrouve le problème d'inclusion pour des automates non déterministes déjà PSPACE-dur. La transduction se fait en temps polynomial, et fournit un automate de taille polynomiale (potentiellement non déterministe). Nous avons déjà précisé cette complexité du problème de l'inclusion avec l'automate-cible.

D'une façon générale, on peut importer les autres cas de l'inclusion décrits précédemment (*page* 19) dès que l'on sait exprimer la cible et l'image de la source dans un des cas particuliers proposés.

Cas des arbres

De la même manière, pour un arbre-source t (respectivement un langage $\mathbf{K}_S$), on peut calculer $\mathscr{T}(t)$ (resp. $\mathscr{T}(\mathbf{K}_S)$) pour $\mathscr{T}$ un XSL-Transducteur déterministe (resp. déterministe linéaire) par le fait 3.24 (*page* 104) puis tester l'inclusion avec la table 1.2 (*page* 24).

4.2.1.1 Le langage des mots sûrs

A partir du schéma de sortie et du String-Transducteur , l'algorithme suivant construit un automate capturant le langage sûr maximal.

Algorithme 12 : Calcul d'une représentation du Langage Sûr Maximal

Données : Un automate-cible A_T et un transducteur $\mathscr{T}$

Résultat : Un automate pour le langage sûr maximal

1 Soit $\mathcal{A}_T = (\Sigma', Q, s_0, \delta, F)$ un automate déterministe reconnaissant $\mathbf{K}_T$;

2 $\mathcal{A}' := (\Sigma, Q, s_0, \delta', Q - F)$ avec $\delta'(s_i, e, s_j)$ si et seulement s'il existe un mot u de $\mathscr{T}(e)$ tel que $\delta^*(s_i, u, s_j)$;

3 **retourner** $\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$ *: le complément de* $\mathcal{A}'$

Cette méthode coûte au pire deux exponentielles : une pour complémenter la source (une pour la cible), qui sera ensuite re-déterminisée pour pouvoir être soustraite au schéma-cible. Cette complexité est en fait aussi une borne inférieure puisque décider de l'existence d'un mot sûr est 2-EXPTIME-Complet.

Fait 4.10

$\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$ *reconnaît exactement l'ensemble des mots sûrs.*

Démonstration : Un mot w est reconnu par $\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$ si et seulement si ce mot est sûr *i.e.* $\mathscr{T}(w) \subseteq \mathbf{K}_T$. En effet :

- Si $\mathcal{A}'$ accepte un mot (sur Σ) $e_1...e_n$, alors il existe n mots $w_1, ..., w_n$ sur Σ' tels que $\forall\ \ 1 \leq i \leq n$, $w_i \in \mathscr{T}(e_i)$ et $w_1...w_n$ est rejeté par $\mathcal{A}_T$.
- S'il existe n mots $w_1...w_n$ tels que $\forall\ \ 1 \leq i \leq n$, $w_i \in \mathscr{T}(e_i)$ et $w_1...w_n$ est rejeté par $\mathcal{A}_T$ alors $e_1...e_n$ est accepté par $\mathcal{A}'$.
- Des deux points précédents, on déduit : $\mathcal{A}'$ accepte un mot $e_1...e_n$ si et seulement si il existe un mot dans $exp_{\mathcal{A}'}\{e_1, ..., e_n\}$ qui est rejeté par $\mathcal{A}'$.
- Comme $\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$ est le complément de $\mathcal{A}'$, on a : un mot $e_1...e_n$ est accepté par $\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$ si et seulement si tout mot $w_1...w_n$ tel que $\forall\ \ 1 \leq i \leq n$, $w_i \in \mathscr{T}(e_i)$ est accepté par $\mathcal{A}_T$.

□

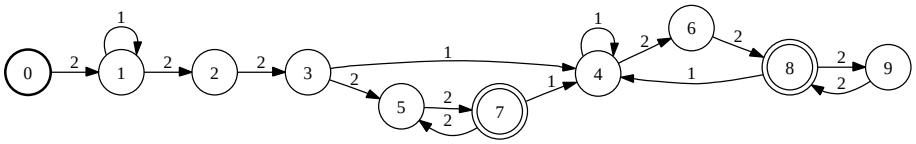


Fig. 4.7 – *L'automate-source $\mathcal{A}'$ des mots non sûrs*

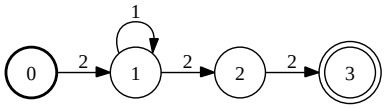


Fig. 4.8 – *L'automate des mots sûrs*

Exemple 4.11 (Exemple 4.2 suite)
Reprenons le transducteur de la figure 4.2 et le schéma-cible $\mathbf{K}_T$ reconnu par l'automate de la figure 4.3 (*page* 116).

Le $\mathcal{A}'$ produit est donné par la figure 4.7 : il peut également se voir comme l'intersection de $\mathscr{T}^{-1}(\mathcal{A}_T{}^c)$ et de l'automate-source $\mathcal{A}_S$, $\mathcal{A}_T{}^c$ dénotant le complémentaire de $\mathcal{A}_T$.

Le mot $w = 22222$ est dans le langage reconnu par $\mathcal{A}'$ puisqu'il est accepté par le chemin $q_0 \xrightarrow{2} q_1 \xrightarrow{2} q_2 \xrightarrow{2} q_3 \xrightarrow{2} q_5 \xrightarrow{2} q_7$. Ce mot n'est donc pas reconnu par l'automate des mots sûrs, donné par la Figure 4.8 et construit comme le complémentaire de $\mathcal{A}'$. Le mot w n'est donc pas sûr.

Nous avons en effet déjà vu dans cet exemple 4.2 (*page* 115), que $\mathscr{T}(w) = \{34555, 34645\}$ et on vérifie bien que l'un de ces mots (34645) n'est effectivement pas dans le langage-cible.

Le mot $w = 22222$ est donc un exemple d'une entrée consistante mais pas sûre. ♪

4.2.1.2 Le langage des arbres sûrs

Intuitivement, le pré-calcul précédent des mots sûrs se fonde sur :

- la possibilité de complémenter les automates.
- la conservation de la régularité par transduction inverse.
- la clôture par intersection.

Ces arguments sont transposables tels quels dans le cadre des langages réguliers d'arbres puisque ces trois propriétés ont déjà été énoncées dans les parties d'introduction aux langages et aux transducteurs d'arbres. Les arbres sûrs forment donc un langage régulier d'arbres, puisque c'est toujours le langage-source privé de l'image réciproque du complément du langage-cible.

D'un point de vue de la complexité, cette approche sans contrainte (sur le transducteur ou sur les langages) peut être non élémentaire en la taille des schémas ; il est donc raisonnable d'imposer des restrictions sur ces objets pour rendre cette méthode « tractable ». La première restriction doit être une classe de langage d'arbres dont la complémentation est tractable et une classe de transducteurs dont le calcul de l'image inverse est tractable.

4.2.2 Sûreté Approchée

La situation peut être déclinée de la même façon que pour la consistance approchée :

Corriger la Source

Si on souhaite que chaque instance-source soit ε-proche d'un mot sûr, on utilise le testeur d'ε inclusion entre $\mathbf{K}_S$ et $\mathcal{A}^{safe}_{\mathcal{A}_T,\mathscr{T}}$.

Corriger la Cible

Si on veut que chaque mot instance soit traduit de façon ε-proche du schéma-cible, le testeur d'ε inclusion est alors $\mathscr{T}(\mathbf{K}_S) \subseteq_\varepsilon \mathbf{K}_T$.

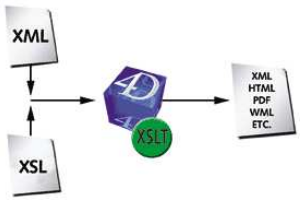
Corriger Source et Cible

Si on souhaite finalement que chaque instance-source soit ε-proche d'une instance, dont chaque transduction est proche du schéma-cible, on procède comme suit :

- On calcule $\mathcal{H}_S$ la représentation statistique du schéma d'entrée à partir de son automate.
- Pour chaque sommet s ayant permis la définition de $\mathcal{H}_S$ (*i.e.* chaque boucle de l'automate $\mathcal{A}_S$) on regarde l'ensemble des boucles de $\mathscr{T}$ compatibles avec s. Intuitivement, cette étape est une projection de la représentation statistique (d'entrée) du XSL-Transducteur :
  - le sous-espace d'entrée est restreint aux valeurs des statistiques d'entrée d'un ensemble de boucles compatibles de $\mathcal{A}_S$
  - les décompositions admises (pour cet ensemble de boucles compatibles) sont restreintes aux combinaisons de boucles de $\mathscr{T}$ produisant cette statistique d'entrée.
- Cette procédure produit une représentation statistique (restreinte) dont la projection P dans l'espace statistique de sortie est encore représentée comme un système d'inéquations linéaires.
- On teste finalement qu'un vecteur satisfaisant P est toujours bien ε'-proches de $\mathcal{H}_{\mathbf{K}_T}$, on accepte si c'est le cas et on rejette sinon.

Chapitre 5

Applications



Une boite à outils a été conçue pour implémenter « l'échange de données ». Elle permet la transformation et la visualisation de structures formelles et est accessible à l'adresse http://info-atome.com. Elle a permis la réalisation en ligne de l'ensemble des exemples proposés dans ce chapitre.

Sur les langages de mots, elle intègre la transformation par un transducteur, potentiellement non déterministe et/ou pondéré, ainsi que les opérations classiques sur le langage obtenu (comparaison à un langage-cible, recherche de la transformation la moins coûteuse, minimisation...). On peut donc l'utiliser pour tester la consistance, la sûreté, et construire automatiquement les automates décrits dans le chapitre 4.

Sur les arbres, l'outil permet la transformation par des processeurs XSLT 1.0 et 2.0 et la confrontation à une DTD en utilisant Xsltproc, Xalan et Xmllint, tous complètement paramétrables depuis l'interface. On peut ainsi modéliser la consistance et la sûreté dans le cas des arbres. On peut aussi modéliser des transformations particulières de schémas. Les schéma-cibles utilisés pour ces exemples sont des restrictions simples de SVG ou de KML.

Ce choix est essentiellement motivé par le fait que ces formats sont :

- Structurés : ce sont des cas particuliers d'arbres (des fichiers XML particuliers).
- Normés : le W3C et les divers travaux de normalisation tendent à faire apparaître des standards adaptés et concourent à leur inter-opérabilité.
- « Scalable » : On peut les visualiser à toutes les échelles.

Sommaire

Un format de graphe étant déjà intégré pour l'affichage des automates, l'outil propose aussi la visualisation de graphes (décrits dans le langage dot) sous une vingtaine de formats. La définition d'un modèle de transduction de graphes serait alors l'étape suivante d'une généralisation passionnante de l'échange de données ; elle dépasse cependant le cadre de cette thèse.

Organisation :

Sur les Mots, l'application intègre le cadre des transducteurs potentiellement non déterministes et/ou pondérés. En donnant les schémas et le transducteur, le langage (éventuellement pondéré) des mots consistants, celui des mots sûrs, ou des mots non-sûrs peut être calculé et visualisé dans une vingtaine de formats. Cet outil permet aussi de calculer certaines distances entre langages. Le calcul de consistance généralisé avec des poids, peut en effet simuler, pour un transducteur adapté, le calcul de la distance entre langages. On peut par exemple calculer la distance de Hamming, la distance d'édition, ainsi que leurs versions généralisées et/ou asymptotiques. On verra également comment rendre efficaces des calculs de distance dès lors qu'un testeur de proximité est à notre disposition.

Sur les Arbres, des transformations particulières de schémas d'arbres nous permettrons de mettre en valeur l'intérêt pratique des transductions.

$XML \rightarrow SVG$: On crée des représentations graphiques (histogrammes, « camemberts » ...) à partir de données unidimensionnelles bruitées. On profite alors des avantages du langage-cible particulièrement bien adapté à la géométrie et aux aspects visuels (polytopes classiques, couleurs continues ...). Ces transformations sont ensuite utilisées pour visualiser[39] automatiquement les réponses numériques d'une requête OLAP [Chaudhuri et Dayal, 1997].

$XML \rightarrow KML$: Un fichier de logs est transformé en carte (représentant les connections). Le trafic d'un site peut alors être visualisé à l'échelle planétaire, ou à une échelle locale, sur la même carte.

[39] avec la possibilité de « zoomer autant qu'on veut »

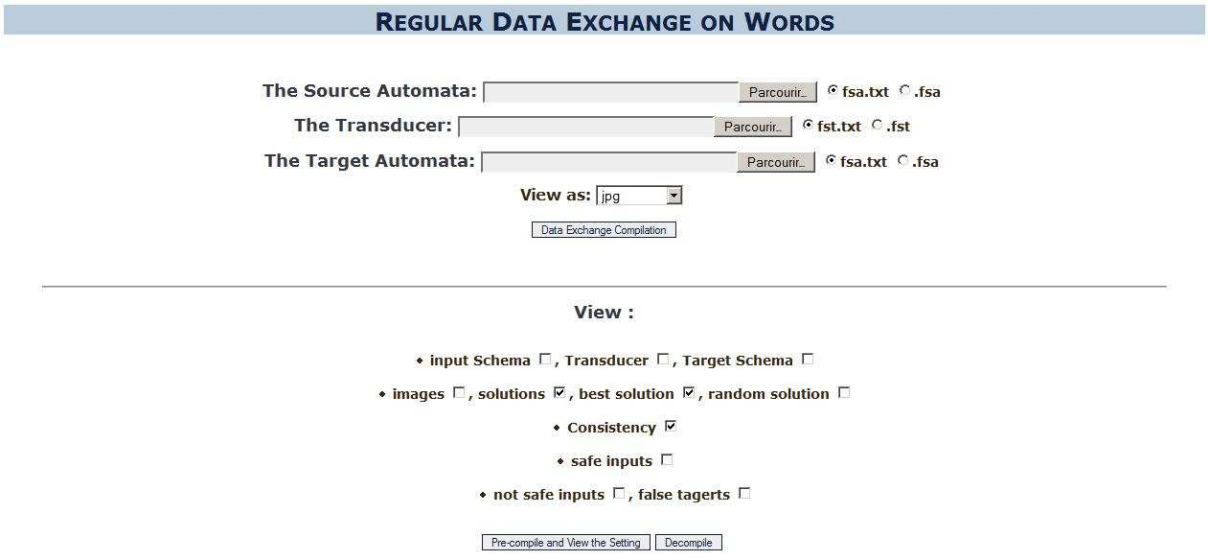


FIG. 5.1 – *L'interface pour l'échange de données sur les langages de mots : http ://info-atome.com/*

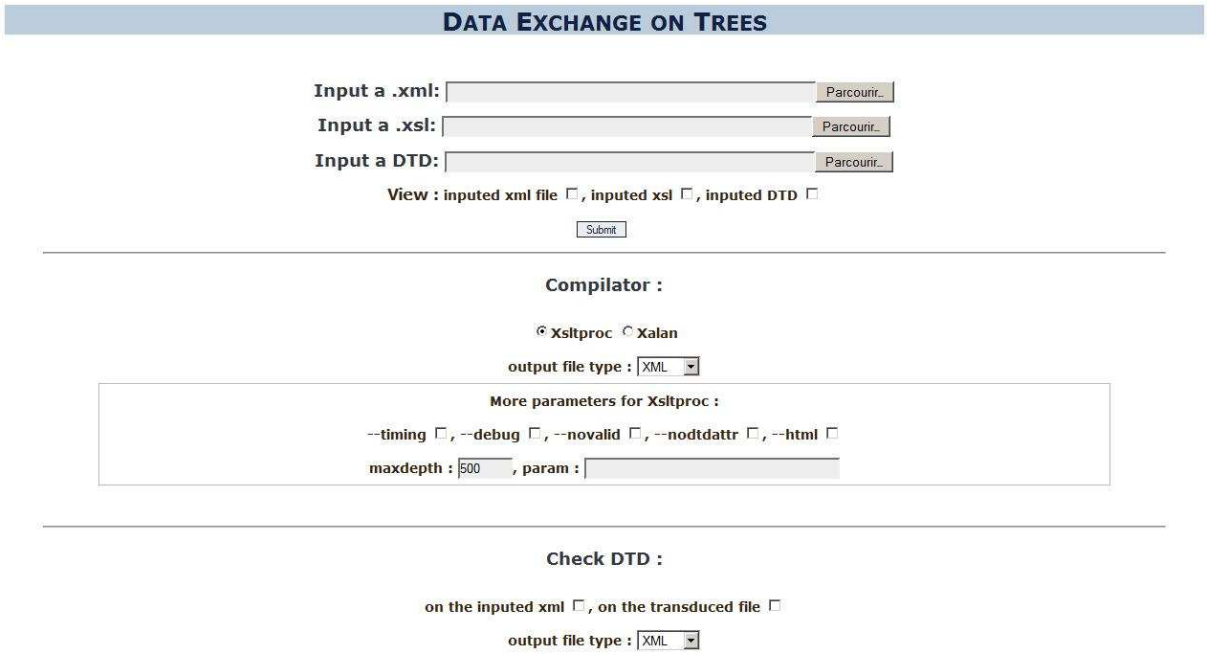


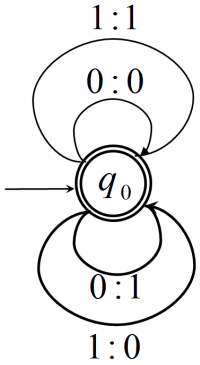
FIG. 5.2 – *L'interface pour l'échange de données sur les arbres*

5.1 Distances

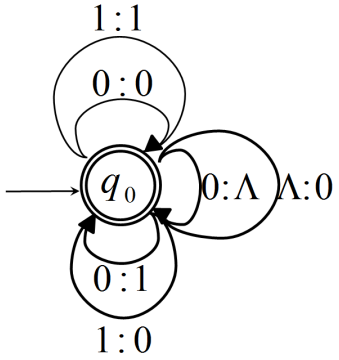
5.1.1 Calcul Exact

Nous nous sommes déjà intéressés aux distances entre instances, langages et transducteurs ; cette section montre que l'on peut aussi calculer certaines distances entre instances et entre langages avec une transduction adaptée. En autorisant les pondérations (coûts) sur les arêtes du transducteur, on peut représenter les opérations élémentaires de certaines distances, par exemple la distance de Hamming. En autorisant les transitions d'entrée ou de sortie vides, on modélise facilement l'insertion, la suppression, et donc la distance d'édition (sans déplacements).

Pour la distance de hamming : $\mathscr{T}_{Ham}$.



Pour la distance d'édition : $\mathscr{T}_{Ed}$.



Les arêtes fines sont celles de coût 0 et les épaisses celles de coût 1.

Fig. 5.3 – *Les Transducteurs de Distances pour* $\Sigma = \{0, 1\}$

Les transducteurs à considérer pour $\Sigma = \{0, 1\}$, sont illustrés par la Figure 5.3. Dans le cas général d'un alphabet fini Σ :

- Pour la distance de Hamming : $\mathscr{T}_{Ham} = (\Sigma, \Sigma, \{0\}, \{0\}, \{0\}, \delta_{ham})$ avec pour tout $a, b \in \Sigma$, $\delta(a, b)$ est pondérée par 0 si $a = b$ et 1 sinon. On a donc Σ^2 arêtes, une arête par couple de lettres, pondérée par 0 si l'entrée est égale à la sortie et 1 sinon.
- Pour la distance d'édition : on ajoute, au transducteur pondéré précédent, une arête d'insertion (Λ, a) et une arête de suppression (a, Λ) pour chaque lettre a de Σ, chacune pondérée par 1.

Pour calculer la distance de Hamming entre deux langages $\mathbf{K}_S$ et $\mathbf{K}_T$, reconnus respectivement par les automates $\mathcal{A}_S$ et $\mathcal{A}_T$:

1. On compose $\mathcal{A}_S$ avec $\mathscr{T}_{Ham}$, le transducteur pondéré de la distance de Hamming (Figure 5.3). Le résultat est un automate pondéré $\mathcal{A}'$.
2. On calcule ensuite l'intersection de $\mathcal{A}'$ et de $\mathcal{A}_T$. $\mathcal{A}'$ étant pondéré alors que $\mathcal{A}_T$ ne l'est pas, cette intersection s'effectue en conservant les pondérations issues de $\mathcal{A}'$ et produit un automate pondéré $\mathcal{A}'' = \mathcal{A}_S \otimes_{Ham} \mathcal{A}_T$.
3. la distance de Hamming entre $\mathbf{K}_S$ et $\mathbf{K}_T$ est le poids (la somme des poids de ses arêtes) minimal d'un chemin acceptant de $\mathcal{A}''$.

Si on cherche la « distance asymptotique » entre les deux langages, *i.e.* la distance restreinte à des longueurs arbitrairement grandes, il suffit de connaître la boucle simple de $\mathcal{A}'' = \mathcal{A}_S \otimes_{\mathsf{dist}} \mathcal{A}_T$ accessible et co-accessible de poids minimal. Changer le coût des opérations élémentaires revient seulement à changer les pondérations du transducteur. On peut aussi facilement généraliser la modélisation pour des coûts non-uniformes, non-unitaires, ou dépendants de l'état du transducteur, en adaptant ce dernier.

Exemple 5.1

Distance entre langages : La figure 5.4 (*page* 143) illustre la construction de l'automate pondéré $\mathcal{A}'' = \mathcal{A}_S \otimes_{Ham} A_T$.

L'état initial est le gros sommet noir (en haut à gauche) ; l'état final est le gros sommet cerclé noir (en bas à droite). Le type d'arête représente l'étiquette et sa largeur correspond à son poids. Les arêtes pleines sont celles étiquetées par 0, et les discontinues celles étiquetées par 1. Les arêtes fines sont celles de coût 0 et les épaisses celles de coût 1.

- La distance de Hamming entre $\mathbf{K}_S$ et $\mathbf{K}_T$ est lue comme la somme du poids des arêtes à extrémités triangulaires (le chemin le moins coûteux) : c'est 2.
- La distance de Hamming asymptotique est lue sur la boucle de gauche (celle de « poids/longueur » minimal) : 1/6.

♪

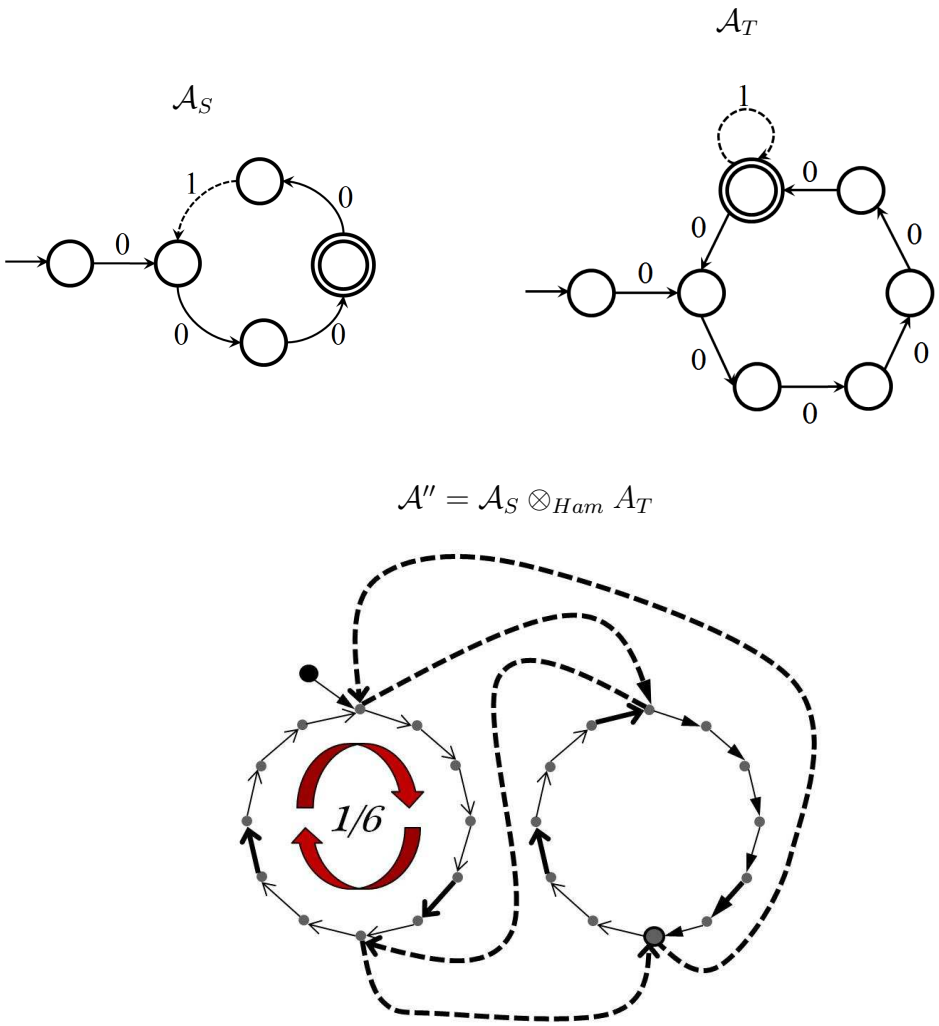


Fig. 5.4 – *Distances de Hamming entre Automates*

Le point négatif est que la taille de l'automate pondéré obtenu peut être exponentiellement plus grande que la taille de l'automate-source. On relativise ce problème en évitant la construction explicite de $\mathcal{A}''$, puisque le plus court chemin suffit. On explore alors les stratégies de transduction, en suivant le coût le plus faible d'abord. On s'arrête dès qu'un chemin acceptant (respectivement une boucle) est construit. En bornant le coût maximal autorisé lors de la transduction, la complexité devient exponentielle en la distance calculée, mais polynomiale en la taille des automates.

D'une façon générale, on peut obtenir la distance par un calcul de plus courts chemins, et la distance asymptotique par la recherche de la boucle simple dont le rapport poids/longueur est minimal. Cette étape est polynomiale en temps ; la complexité résulte bien de la taille de $\mathcal{A}''$. Pour limiter l'impact de cette explosion combinatoire, on pourra borner le coût de transduction maximal autorisé et/ou calculer les plus courts chemins (ou boucle dans le cas asymptotique) « on the fly » sur une construction partielle de cet automate pondéré.

Ceci vaut pour toute distance représentable par un transducteur et on pourra entre autre traiter la distance de Hamming, la distance d'édition, mais aussi leurs versions généralisées et/ou asymptotiques.

5.1.2 Calcul Approché

Les calculs précédents se heurtent rapidement au problème de l'explosion combinatoire et ne sont applicables en pratique que sur de petites instances. Calculer la distance d'édition avec déplacements est par exemple un problème NP-difficile. Un intérêt de la représentation statistique proposée est qu'elle permet également d'approximer efficacement cette distance. En effet, cette distance peut être approximée par dichotomie dans laquelle une étape consiste à approximer les statistiques (en temps constant qui ne dépend que de ε') et à décider de quel côté continuer la recherche en calculant la valeur de la norme $\mathbf{L}_1$ entre ces statistiques.

Algorithme 13 : Approximation de Distances : $D_{app}(I, I', \varepsilon, f)$

Données : Deux instances I et I', $\varepsilon > 0$, $f :]0,1[\rightarrow]0,1[$

Résultat : L'approximation de $\mathsf{dist}(I, I')$

début

  $Res := 1/2$; $\varepsilon' := 1/2$;

  tant que $\varepsilon' > \varepsilon$ **faire**

    $x := \mathsf{ustat}_{1/\varepsilon'}(I)$; $x' := \mathsf{ustat}_{1/\varepsilon'}(I')$;

    si $|x - x'| < f(\varepsilon')$ **alors** $\varepsilon' := \varepsilon'/2$; $x := x - \varepsilon'$; **sinon** $\varepsilon' := \varepsilon'/2$; $x := x + \varepsilon'$;

  fin

  retourner x

fin

Proposition 5.2

Sur les mots et les arbres, il existe f tel que $\Pr\left(|\mathsf{dist}(I, I') - D_{app}(I, I', \varepsilon, f)| > \varepsilon\right) < 2/3$

$\star$

Un testeur de proximité pour une distance particulière (ici la distance d'édition avec déplacements) fournit alors indirectement une méthode d'approximation efficace de cette distance en obtenant un algorithme de complexité constante, *i.e.* ne dépendant que du seuil d'approximation souhaité et non de la taille des instances considérées.

5.2 Visualisation

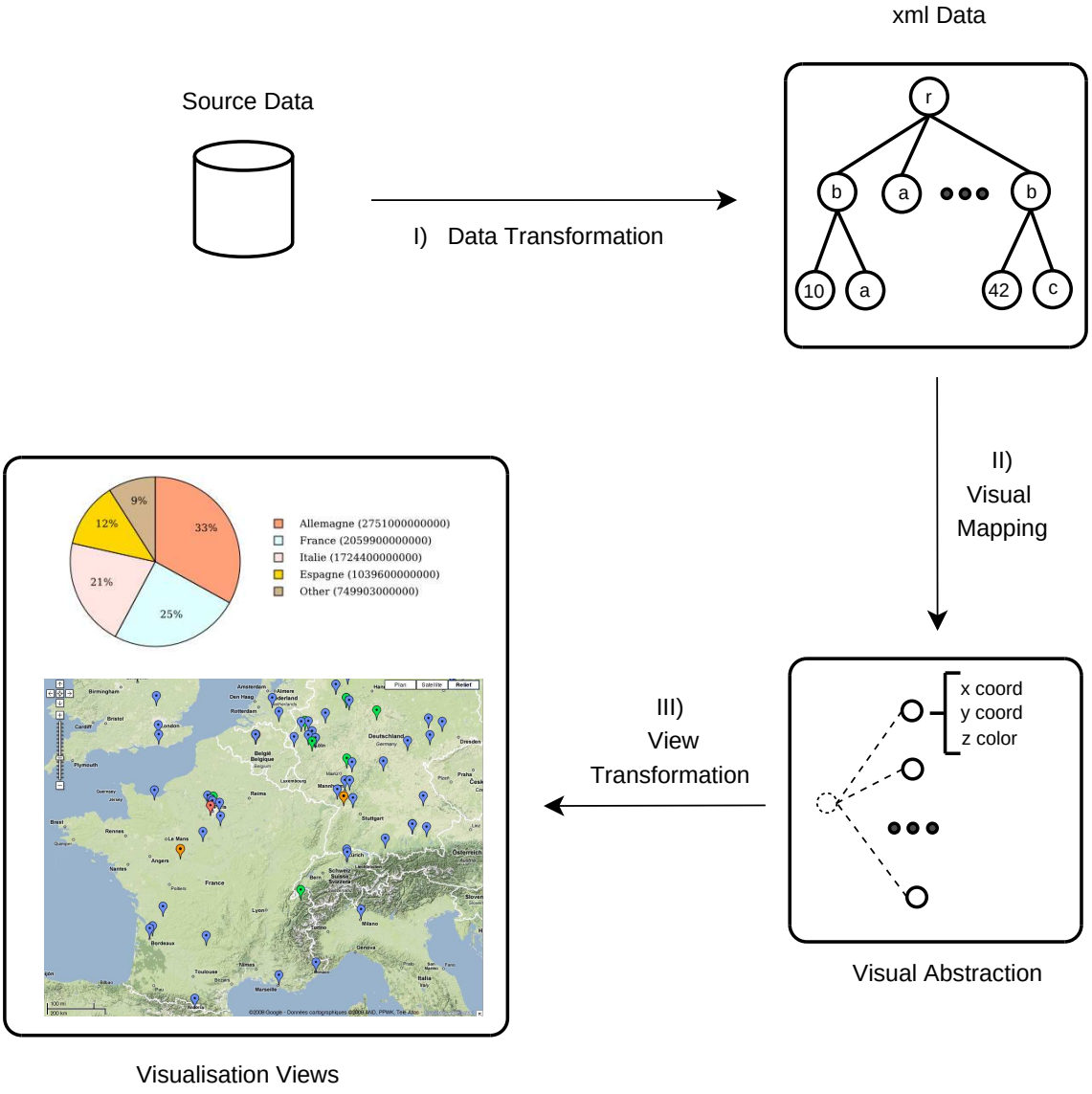


Fig. 5.5 – *Les trois grandes étapes de la visualisation*

Comme proposé dans [Card et al., 1999], la visualisation peut se décomposer en trois grandes étapes représentées par la figure 5.5 (*page* 146). Nous allons maintenant donner deux exemples de telles transformations. Les transformations considérées produisent des documents structurés ; nous utiliserons ainsi des formats vectoriels comme SVG ou KML pour représenter les représentations graphiques de sortie. Pour modéliser ces transformations, les XSL-Transducteurs ne suffisent pas à visualiser des valeurs (PCDATA) dans des domaines potentiellement infinis (entiers, chaînes de caractères ...). Les XSLA-Transducteurs sont des généralisations qui permettent cette manipulation de valeurs.

XSLA-Transducteur

Soit Σ un alphabet fini, A_S et A_T deux ensembles finis de noms d'attributs. On notera $H_{\Sigma,\mathsf{A}_S,\mathsf{A}_T}[Q]$ les haies de $H_{\Sigma_T}[Q]$ dont les noeuds internes[40] peuvent admettre un ensemble d'égalités du type $a \leftarrow b$ avec $a \in \mathsf{A}_T$ et $b \in \mathsf{A}_S$ (définissant une fonction partielle[41] de A_T dans A_S).

Définition 5.3

Un *XSLA-Transducteur* d'arbres entre (Σ_S, A_S) arbres et (Σ_T, A_T) arbres est donné par (Q, q_0, δ) où : δ est un ensemble de règles de la forme $(\Sigma_S, Q) \rightarrow H_{(\Sigma_T,\mathsf{A}_S,\mathsf{A}_T)}[Q]$. ✓

Soit $\mathcal{V}ar$ un ensemble (infini dénombrable) de valeurs nulles. La sémantique des égalités est la suivante : un noeud $e_{\{a\leftarrow b\}}$ (en partie droite de règle) produit un noeud étiqueté $e \in \Sigma_T$ et possédant comme attribut a :

- soit la valeur de l'attribut b du noeud courant.
- soit $v \in \mathcal{V}ar$ lorsque le noeud courant ne possède pas de valeur pour l'attribut b.

Exemple 5.4 (Transduction de mots avec attributs)

Soit $\Sigma_S = \{0,1\}$, $\mathsf{A}_S = \{N, M\}$, $\mathcal{S} = \{a, c\}$, $\Sigma_T = \{b, c, d\}$, $\mathsf{A}_T = \{P\}$, et le XSLA-Transducteur (Q, q, δ) défini par : $Q = \{q\}$ et $\delta = \big\{(0, q) \rightarrow c_{\{P\leftarrow N\}}.d.[q]\ ;\ (1, q) \rightarrow b.d.[q]\big\}$. L'image d'un mot $1.0_{[N=a]}.1_{[N=c,M=c]}.0_{[M=c]}.1.0_{[N=a]}$ —étiqueté sur $\{0,1\}$ avec des attributs N, M— est un mot étiqueté sur $\{a, b, c\}$ avec attribut P dont les valeurs d'attributs sont dans $\mathcal{S} \cup \mathcal{V}ar$, *i.e.* $b.d.c_{[P=a]}.d.b.d.c_{[P=V_1]}.b.d.c.d$ ♪

[40] ceux étiquetés dans Σ_T

[41] *i.e.* a détermine b au sens où au plus un b peut être mis en relation avec un a donné.

5.2.1 $XML \rightarrow SVG$

Ce premier exemple, construit depuis une base de données, produit une représentation SVG des entrées sous forme de pourcentages. Prenons le fichier XML suivant, répertoriant les PIB de quelques pays, fourni par exemple par la transformation d'une table relationnelle simple :

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance">
    <component name="France">2059900000000</component>
    <component name="Espagne">1039600000000</component>
    <component name="Italie" capital="Rome">1724400000000</component>
    <component name="Suisse">358610000000</component>
    <component name="Allemagne">2751000000000</component>
    <component name="Luxembourg">33593000000</component>
    <component name="Belgique">357700000000</component>
    <error/>
</components>
```

Ce fichier est proche de la DTD-Source :

```
<!ELEMENT components (#PCDATA | component)*>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component name CDATA #IMPLIED>
```

En effet, il suffit de retirer le noeud <error> et l'attribut « capital » de l'Italie pour satisfaire cette DTD.

Nous allons construire le « camembert » des pourcentages correspondants sous la forme d'une image vectorielle SVG[42]. Nous choisissons une restriction de SVG comme DTD-cible :

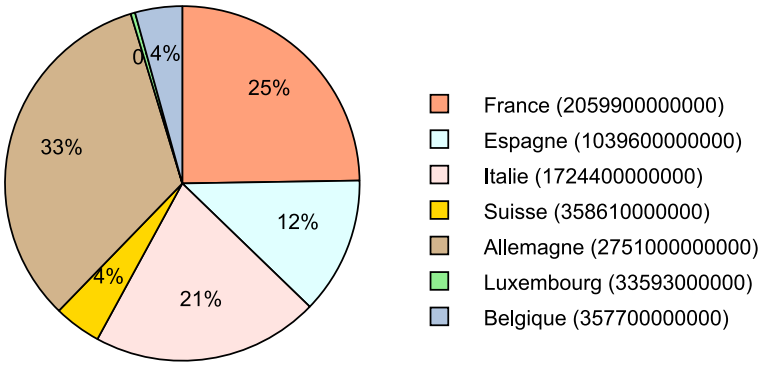
```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text transform CDATA #IMPLIED>
<!ATTLIST text style CDATA #IMPLIED>
<!ATTLIST text x CDATA #IMPLIED>
<!ATTLIST text y CDATA #IMPLIED>
<!ELEMENT path (#PCDATA)>
<!ATTLIST path transform CDATA #IMPLIED>
<!ATTLIST path style CDATA #IMPLIED>
<!ATTLIST path d CDATA #IMPLIED>
<!ELEMENT svg (#PCDATA | path | text)*>
<!ATTLIST svg viewBox CDATA #IMPLIED>
<!ATTLIST svg height CDATA #IMPLIED>
<!ATTLIST svg width CDATA #IMPLIED>
```

[42]initiales de « Scalable Vector Graphics » : http://fr.wikipedia.org/wiki/SVG

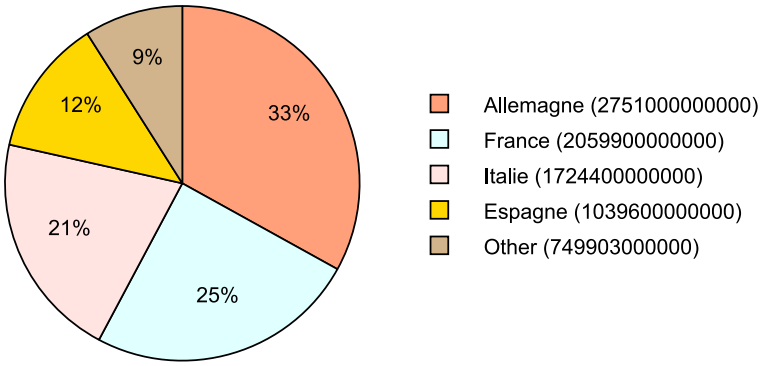
Le transducteur peut être précédé de deux pré-calculs facultatifs :

- la correction du fichier-source par rapport à la DTD-source.
- l'émondage des plus petites valeurs[43].
- le tri par valeurs décroissantes[44].

L'application du transducteur sur l'entrée corrigée génère le fichier svg correspondant[45] et la figure suivante :



Avec le pré-traitement émondant et le tri, on obtient la figure suivante[46] :



L'émondage et le tri peuvent être vus comme des cas particuliers de « Visual Mapping », alors que la production réelle du camembert est la « View Transformation ».

[43] les valeurs de moins de 5% de la somme des valeurs sont négligées : http://info-atome.com/xsl/bdRemoveSmall.xsl

[44] http://info-atome.com/xsl/bdTri.xsl

[45] http://info-atome.com/svg/pie1.svg

[46] http://info-atome.com/svg/pie2.svg

5.2.2 Application à OLAP

$$\text{Requête OLAP} \xrightarrow{Mondrian} \text{Cube (XML)} \xrightarrow{XSLT} \text{Rendu (SVG)}$$

Mondrian[47] est un outil développé en java qui permet, à partir d'une requête OLAP, de construire le cube-résultat sous la forme d'un fichier XML. En adaptant le transducteur précédent pour l'affichage d'un résultat sur une dimension, on obtient un histogramme (SVG) des données du type de ceux de la figure 5.6. Les valeurs exactes restent également accessibles d'un simple clic.

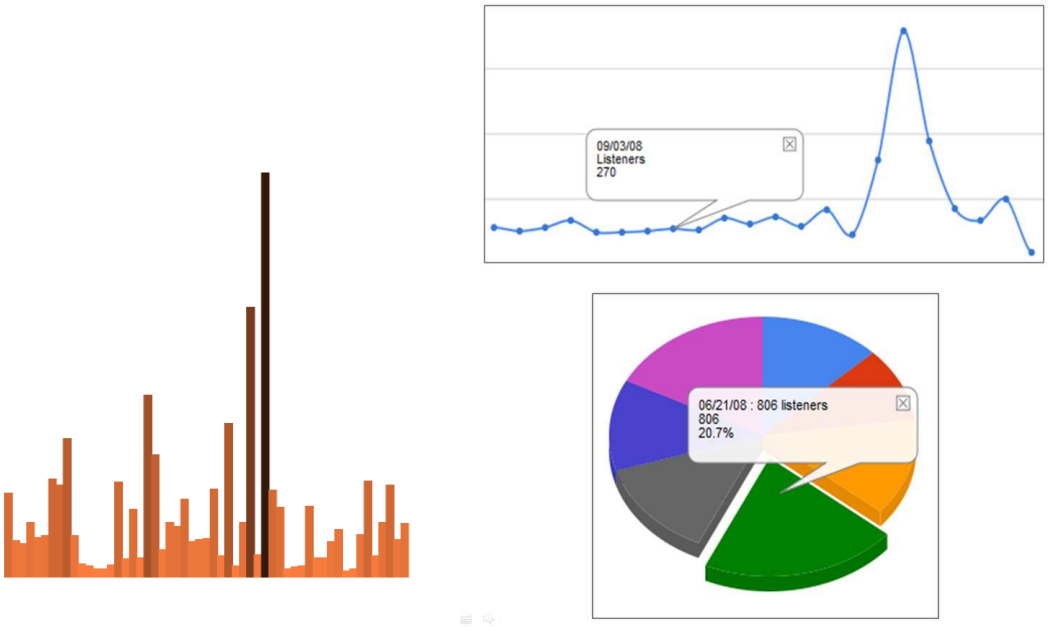


Fig. 5.6 – *Visualisation : une dimension*

[47] http://sourceforge.net/projects/mondrian

5.2.3 $XML \rightarrow KML$

Dans cet exemple, la source est constituée d'un fichier de logs d'une web-radio dont quelques lignes sont données ci-dessous.

```
<05/05/08@11:24:21> [dest: 91.50.204.92] starting stream (UID: 602)
<05/05/08@11:24:21> [dest: 217.116.248.9] starting stream (UID: 603)
<05/05/08@11:24:21> [dest: 84.186.219.192] starting stream (UID: 604)
<05/05/08@11:24:21> [dest: 91.50.204.92] starting stream (UID: 605)
<05/05/08@11:24:39> [dest: 77.0.244.210] connection closed (283 seconds) (UID: 601)
<05/05/08@11:26:29> [yp_tch] yp.shoutcast.com touched!
<05/05/08@11:28:21> [dest: 91.50.204.92] connection closed (240 seconds) (UID: 605)
<05/05/08@11:28:22> [dest: 217.116.248.9] connection closed (242 seconds) (UID: 603)
<05/05/08@11:28:24> [dest: 84.186.219.192] connection closed (244 seconds) (UID: 604)
<05/05/08@11:29:42> [dest: 77.0.244.210] connection closed (2000 seconds) (UID: 401)
...
```

Mise en forme Arborescente

Ce fichier est d'abord parsé pour être structuré sous une forme xml. On ne retiendra que les évènements du type de la dernière ligne qui nous fournit des temps de connexion à la radio.

```
<?xml version="1.0" encoding="UTF-8"?>
<ips>
    ....
    <ecoute ip="77.0.244.210" quand="05/05/08@11:24:39" duree="283"/>
    <ecoute ip="91.50.204.92" quand="05/05/08@11:28:21" duree="240"/>
    ....
    <ecoute ip="91.50.204.92" quand="05/05/08@11:29:42" duree="2000"/>
    ...
</ips>
```

Pour afficher ces visiteurs sur une carte du monde, on les regroupe par ip et on cumule les temps d'écoute. On en profite pour ajouter des informations et localiser l'évènement sur la planète, par exemple à l'aide de « GeoLite Country »[48] ; on déduit de l'ip, la latitude et la longitude associées. On obtient alors un fichier xml de la forme :

```
<ip>
  ...
  <ecoute ip="77.0.244.210" quand="05/05/08@11:24:39" duree="283"
           pays="Sweden" latitude="55.7" longitude="13.1833" occ="1"/>
  <ecoute ip="91.50.204.92" quand="05/05/08@11:29:42" duree="2240"
          pays="Sweden" latitude="55.7" longitude="13.1833" occ="2"/>
  ...
</ip>
```

Association Visuelle

Intuitivement, les positions x et y sont les coordonnées associées à l'ip, et la couleur z est par exemple la durée ou le nombre d'occurrences.

Transformation d'Affichage

Pour chaque feuille de l'arbre précédent, nous allons générer un point KML associé aux coordonnées déjà calculées et à une icône dont la couleur est déterminée par une condition sur les attributs (la durée par exemple). Le transducteur utilisé produit un fichier KML valide, qui appartient également à la DTD restreinte donnée par la figure 5.7. Le format permet ainsi d'agréger plusieurs points de même localisation sans perdre la possibilité de les distinguer (voir Figure 5.8 (*page* 154)) Les informations explicites d'un point sont conservées et peuvent être obtenues via l'interaction avec la carte comme le montre la figure 5.11 (*page* 156).

[48] http://www.maxmind.com/app/geolitecountry

```
<!ELEMENT kml (Document)>
<!ATTLIST kml  xmlns CDATA  #IMPLIED>
<!ELEMENT Document (Style | Folder)*>
<!ELEMENT Style (LineStyle | PolyStyle | IconStyle)*>
<!ATTLIST Style id CDATA #IMPLIED>
<!ELEMENT Folder (name | open | visibility | description |
                  Folder | Style | Placemark)*>
<!ELEMENT Icon (#PCDATA | href)*>
<!ELEMENT link (#PCDATA | href)*>
<!ELEMENT href (#PCDATA)>
<!ELEMENT open (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT Point (#PCDATA | coordinates)*>
<!ELEMENT description (#PCDATA | link | br)*>
<!ELEMENT scale (#PCDATA)>
<!ELEMENT br (#PCDATA)>
<!ELEMENT PolyStyle (#PCDATA | color)*>
<!ELEMENT IconStyle (#PCDATA | scale | Icon)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT coordinates (#PCDATA)>
<!ELEMENT Placemark (name | styleUrl | Point | description)*>
<!ELEMENT styleUrl (#PCDATA)>
<!ELEMENT visibility (#PCDATA)>
<!ELEMENT LineStyle (color | width)*>
<!ELEMENT width (#PCDATA)>
```

Fig. 5.7 – *DTD pour une restriction KML*

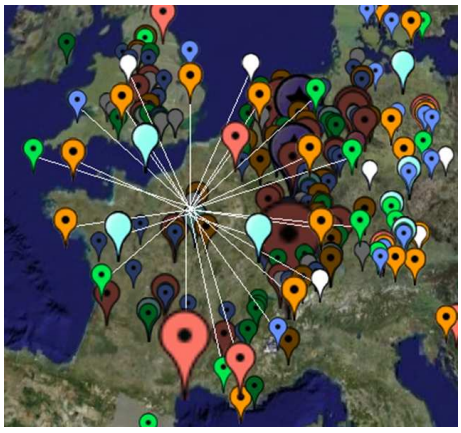


Fig. 5.8 – *Interaction KML*

Statistiques Correspondantes

Le fichier-source est essentiellement « plat » : ses statistiques sont issues de la relation de frère entre noeuds d'étiquette 'ecoute' et se concentrent donc, à l'ordre k, sur le type 1^{k-1}. Le fichier-cible est, lui, bien plus structuré, bien que sa matrice statistique reste creuse (ce phénomène s'amplifiant même lorsque l'ordre augmente). A l'ordre 8 :

- plus de 90% des statistiques squelétiques sont couvertes par seulement 11 des 1024 types possibles (cf. Figure 5.9).
- plus de 97% de chaque statistique squelétique se décompose sur moins de 4 séquences d'étiquettes sur les $16^8 = 4\ 294\ 967\ 296$ possibles (cf. Figure 5.10).

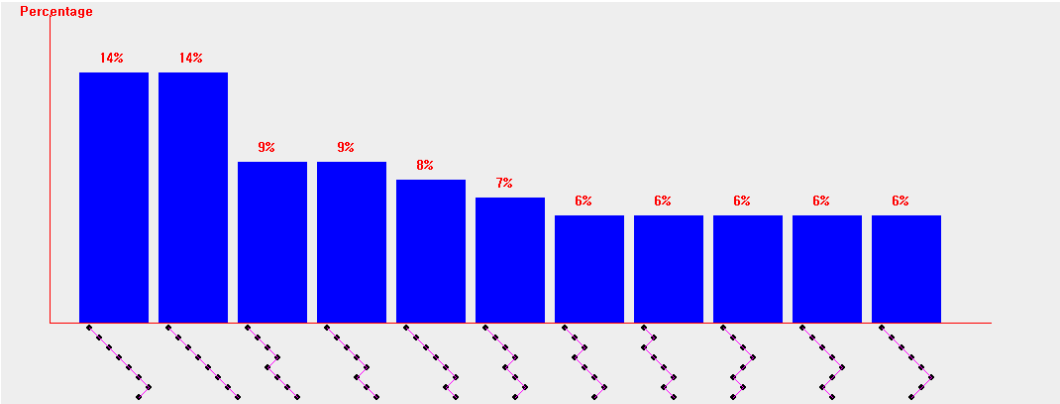


Fig. 5.9 – *Statistiques Squelettiques du KML-cible*

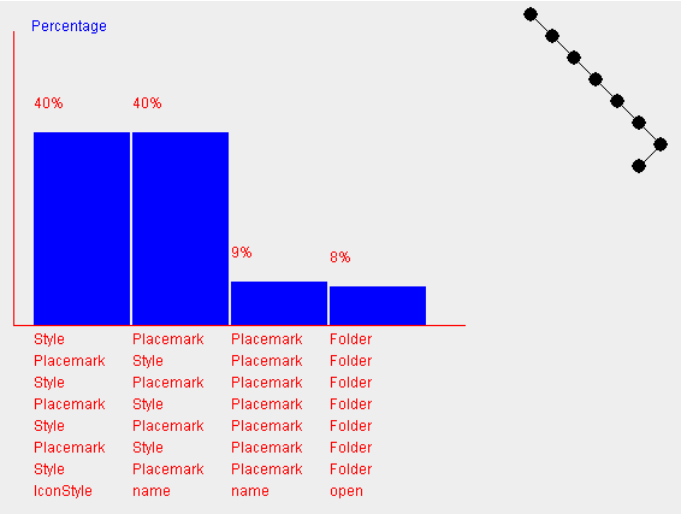


Fig. 5.10 – *Décomposition d'une Statistique Squelettique du KML-cible*

FIG. 5.11 – *Visualisation Géographique de Logs*

Chapitre 6

Conclusion

Dans cette thèse, différents aspects de l'approximation dans l'échange de données ont été introduits. Pour ce faire, une méthode d'approximation des instances, des langages et des transducteurs a été développée. Plusieurs propriétés rendent ces statistiques particulièrement intéressantes puisqu'il est possible d'approcher ces objets en temps constant (pour un mot ou un arbre), ou de les calculer exactement en temps linéaire.

Équivalence

La propriété centrale de la représentation statistique proposée est de pouvoir lier la distance entre mots (respectivement arbres) à la distance (géométrique) entre vecteurs (resp. matrices) statistiques. Cette propriété a été étendue pour les transducteurs et conduit à un algorithme qui permet de décider de l'équivalence approchée. L'idée fondamentale est de comprendre comment une erreur ε sur les transducteurs se traduit par une erreur ε' sur leurs représentations.

Cette méthode permet de traiter des transducteurs non-déterministes (quand la plupart des résultats de décidabilité portent sur les transformations déterministes).
La nouveauté principale vis-à-vis des travaux existants est qu'elle introduit la notion de robustesse : l'ε-équivalence va pouvoir être testée quelque soit ε strictement positif. Ce moyen de décider la proximité de programmes peut être utilisé en vérification puisque l'on en déduit un moyen de déterminer si des programmes se comportent « essentiellement de la même manière », et ainsi de savoir si un programme est une abstraction « suffisamment précise » d'un autre programme.

Échange de données

La consistance est une propriété régulière et décidable en temps linéaire en la taille de l'instance-source. La consistance approchée est, quant à elle, résolue en temps constant : un nombre fixé d'échantillons permet de prendre la bonne décision avec grande probabilité. Déjà dans le cadre des mots, la complexité de la sûreté dans sa version exacte (PSPACE), contraste avec la décision efficace de sa version approchée (PTIME), motivant ainsi la pertinence de la relaxation. Cette complexité est issue du non-déterminisme des transductions considérées, et disparaît pour les transducteurs déterministes puisque la sûreté se traite alors comme la consistance.
Dans le cas non déterministe, réduire la dépendance de la complexité des constructions proposées, (vis-à-vis des schémas et du transducteur) est à nouveau la question essentielle pour la mise en oeuvre d'une implémentation effective qui « passe à l'échelle ». Les extensions évoquées pour l'équivalence (retour à une interpolation linéaire et prise en compte des distorsions arbitraires) sont à nouveau deux voies (d'amélioration ou de généralisation) à creuser.

Applications

L'utilisation de transducteurs non déterministes et/ou pondérés fournit un cadre général intéressant, par exemple pour le calcul de certaines distances. Une étude à mener est l'évaluation expérimentale des décisions approchées proposées. Les transformations particulières de schémas proposés (pour les arbres) peuvent être généralisées et appliquées de nombreuses manières. La visualisation multi-dimensionnelle des réponses à une requête OLAP est une des voies en cours de développement. Poursuivant l'intérêt théorique des transductions de forêts et leurs applications à la publication multi-fichiers, la prochaine étape pourrait être la généralisation de ces techniques aux transducteurs de graphes où une notion de robustesse apparaît sous la dénomination de « test de propriétés ». Une représentation statistique robuste des transducteurs étendue à ce cas promettrait alors une généralisation passionnante de l'échange de données approchées.

Bibliographie

[Abe et al., 2002] Abe, K., Kawasoe, S., Asai, T., Arimura, H., et Arikawa, S. 2002. Optimized substructure discovery for semi-structured data. In : PKDD. pages 1–14.

[Alon et al., 2003] Alon, N., Milo, T., Neven, F., Suciu, D., et Vianu, V. 2003. Typechecking XML views of relational databases. ACM Transactions on Computational Logic, 4(3) :315–354.

[Andre et Bossut, 1993] Andre, Y. et Bossut, F. 1993. Decidability of equivalence for linear letter to letter top-down tree transducers. In : Fundamentals of Computation Theory. 710 :142–151.

[Andre et Bossut, 1995] Andre, Y. et Bossut, F. 1995. The equivalence problem for letter-to-letter bottom-up tree transducers is solvable. In : Theory and Practice of Software Development. 915 :155–171.

[Andre et Bossut, 1998] Andre, Y. et Bossut, F. 1998. On the equivalence problem for letter-to-letter top-down tree transducers. Theoretical Computer Science, 205 :207–229.

[Arenas et al., 2004] Arenas, M., Barcelo ;, P., Fagin, R., et Libkin, L. 2004. Locally consistent transformations and query answering in data exchange. In : ACM Principles on Databases Systems. pages 229–240.

[Arenas et Libkin, 2005] Arenas, M. et Libkin, L. 2005. Xml data exchange : Consistency and query answering. In : ACM Principles on Databases Systems. pages 13–24.

[Arora et Safra, 1998] Arora, S. et Safra, S. 1998. Probabilistic checking of proofs : a new characterization of np. Journal of the Association of Computing Machinery, 45(1) :70–122.

[Asai et al., 2002] Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H., et Arikawa, S. 2002. Efficient substructure discovery from large semi-structured data. In : SDM.

[Bex et al., 2005] Bex, G. J., Martens, W., Neven, F., et Schwentick, T. 2005. Expressiveness of xsds : from practice to theory, there and back again. In : WWW '05 : Proceedings of the 14th international conference on World Wide Web. ACM, pages 712–721.

[Bird, 1973] Bird, M. 1973. The equivalence problem for deterministic two-tape automata. Journal of Computer and System Sciences, 7(2) :218–236.

[Calvanese et al., 2006] Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., et Rosati, R. 2006. Data complexity of query answering in description logics. In : Principles of Knowledge Representation and Reasoning. pages 260–270.

[Calvanese et al., 1999] Calvanese, D., Giacomo, G. D., Lenzerini, M., et Vardi, M. Y. 1999. Rewriting of regular expressions and regular path queries. In : ACM Principles on Databases Systems. pages 194–204.

[Card et al., 1999] Card, S. K., Mackinlay, J. D., et Shneiderman, B. 1999. Readings in information visualization : using vision to think. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Champavère et al., 2008] Champavère, J., Gilleron, R., Lemay, A., et Niehren, J. 2008. Efficient inclusion checking for deterministic tree automata and dtds. In : Language and Automata Theory and Applications. pages 184–195.

[Chandra et al., 1981] Chandra, A. K., Kozen, D., et Stockmeyer, L. J. 1981. Alternation. Journal of the Association of Computing Machinery, 28(1) :114–133.

[Chaudhuri et Dayal, 1997] Chaudhuri, S. et Dayal, U. 1997. An overview of data warehousing and olap technology. ACM SIGMOD : international conference on Management of data, 26(1) :65–74.

[Cobham, 1965] Cobham, A. 1965. The intrinsic computational difficulty of functions. In : Bar-Hillel, Y. (Editor), Proceedings of the 1964 International Conference for Logic, Methodology, and Philosophy of Science. North-Holland, pages 24–30.

[Comon et al., 1997] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., et Tommasi, M. 1997. Tree automata techniques and applications.

[Comon et al., 2007] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., et Tommasi, M. 2007. Tree automata techniques and applications. Available on : http://www.grappa.univ-lille3.fr/tata. release October, 12th 2007.

[Cook, 1971] Cook, S. A. 1971. The complexity of theorem-proving procedures. In : ACM Symposium on Theory of Computing. ACM Press, pages 151–158.

[Cormen et al., 1990] Cormen, T. H., Leiserson, C. E., et Rivest, R. L. 1990. Introduction to Algorithms. MIT Press/McGraw-Hill.

[Cristau et al., 2005] Cristau, J., Löding, C., et Thomas, W. 2005. Deterministic automata on unranked trees. In : In Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT), LNCS. Springer, pages 68–79.

[de Bruijn, 1946] de Bruijn, N. G. 1946. A combinatorial problem. Nederlandsche Akademie van Wetenschappen, 49 :758–764. (Indagationes Math. **8** (1946), 461–467).

[Doner, 1965] Doner, J. 1965. Decidability of the weak second-order theory of two successors. Notices Amer. Math. Soc, 12 :819.

[Doner, 1970] Doner, J. 1970. Tree acceptors and some of their applications. Journal of Computer and System Sciences, 4 :406–451.

[dtd, 1995] dtd 1995. Document type definition.
http ://en.wikipedia.org/wiki/Document_Type_Definition.

[Duchamp et al., 2005] Duchamp, G. H. E., Kacem, H. H., et Laugerotte, E. 2005. Algebraic elimination of ϵ-transitions. Discrete Mathematics and Theoretical Computer Science, 7(1) :51–70.

[Engelfriet, 1975] Engelfriet, J. 1975. Bottom-up and top-down tree transformations a comparison. Math. Systems Theory, 9(3) :198–231.

[Engelfriet et Maneth, 2005] Engelfriet, J. et Maneth, S. 2005. The equivalence problem for deterministic mso tree transducers is decidable. In : Foundations of Software Technology and Theoretical Computer Science. 3821 :495–504.

[Fagin et al., 2002] Fagin, R., Kolaitis, P. G., Miller, R. J., et Popa, L. 2002. Data exchange : Semantics and query answering. In : International Conference on Database Theory. pages 207–224.

[Fagin et al., 2005] Fagin, R., Kolaitis, P. G., et Popa, L. 2005. Data exchange : getting to the core. Transactions on Database Systems, 30(1) :174–210.

[Feige et al., 1996] Feige, U., Goldwasser, S., Lovász, L., Safra, S., et Szegedy, M. 1996. Interactive proofs and the hardness of approximating cliques. Journal of the Association of Computing Machinery, 43(2) :268–292.

[Fischer et al., 2006] Fischer, E., Magniez, F., et Rougemont, M. 2006. Approximate satisfiability and equivalence. In : Logic in Computer Science. pages 421–430.

[Goldreich et al., 1998] Goldreich, O., Goldwasser, S., et Ron, D. 1998. Property testing and its connection to learning and approximation. Journal of the Association of Computing Machinery, 45(4) :653–750.

[Good, 1946] Good, I. 1946. Normal recurring decimals. Journal of the London Mathematical Society, 21 :167–169.

[Gurari, 1982] Gurari, E. M. 1982. The equivalence problem for deterministic two-way sequential transducers is decidable. SIAM Journal of Computing, 11(3) :448–452.

[Harju et Karhumäki, 1991] Harju, T. et Karhumäki, J. 1991. The equivalence problem of multitape finite automata. Theoretical Computer Science, 78(2) :347–355.

[Hopcroft et Ullman, 1969] Hopcroft, J. E. et Ullman, J. D. 1969. Formal Languages and Their Relation to Automata. Addison Wesley Longman Publishing Co., Inc.

[Hopcroft et Ullman, 1990] Hopcroft, J. E. et Ullman, J. D. 1990. Introduction To Automata Theory, Languages, And Computation. Addison-Wesley Longman Publishing.

[Hunt et al., 1976] Hunt, H. B., Rosenkrantz, D. J., et Szymanski, T. G. 1976. On the equivalence, containment, and covering problems for the regular and context-free languages. Journal of Computer and System Sciences, 12(2) :222–268.

[Karhumäki et Lisovik, 1999] Karhumäki, J. et Lisovik, L. P. 1999. On the equivalence of finite substitutions and transducers. In : Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa. pages 97–108.

[Khuller et Raghavachari, 1997] Khuller, S. et Raghavachari, B. 1997. Graph and network algorithms. In : The Computer Science and Engineering Handbook. pages 203–225.

[Kontorovich, 2004] Kontorovich, L. 2004. Uniquely decodable n-gram embeddings. Theoretical Computer Science, 329(1-3) :271–284.

[Lee et Yannakakis, 1996] Lee, D. et Yannakakis, M. 1996. Principles and methods of testing finite state machines - a survey. In : Proceedings of the IEEE. pages 1090–1123.

[Lohrey, 2001] Lohrey, M. 2001. On the parallel complexity of tree automata. In : Rewriting Techniques and Applications. pages 201–215.

[Makoto, 1995] Makoto, M. 1995. Hedge automata : A formal model for xml schemata. Fuji Xerox Information Systems.

[Maneth et al., 2005] Maneth, S., Berlea, A., Perst, T., et Seidl, H. 2005. XML type checking with macro tree transducers. In : ACM Principles on Databases Systems. pages 283–294.

[Maneth et Seidl, 2007] Maneth, S. et Seidl, H. 2007. Deciding equivalence of top-down xml transformations in polynomial time. In : PLAN-X. pages 73–79.

[Martens et Neven, 2002] Martens, W. et Neven, F. 2002. Typechecking top-down uniform unranked tree transducers. In : International Conference on Database Theory. pages 64–78.

[Martens et Neven, 2004] Martens, W. et Neven, F. 2004. Frontiers of tractability for typechecking simple XML transformations. In : ACM Principles on Databases Systems. pages 23–34.

[Martens et al., 2008] Martens, W., Neven, F., et Gyssens, M. 2008. Typechecking top-down xml transformations : Fixed input or output schemas. Inf. Comput., 206(7) :806–827.

[Martens et al., 2004] Martens, W., Neven, F., et Schwentick, T. 2004. Complexity of decision problems for simple regular expressions. In : Mathematical Foundations of Computer Science. pages 889–900.

[Martens et al., 2005] Martens, W., Neven, F., et Schwentick, T. 2005. Which xml schemas admit 1-pass preorder typing ? In : International Conference on Database Theory. pages 68–82.

[Martens et Niehren, 2005] Martens, W. et Niehren, J. 2005. Minimizing tree automata for unranked trees. In : Database Programming Languages. pages 232–246.

[Milo et al., 2000] Milo, T., Suciu, D., et Vianu, V. 2000. Typechecking for XML transformers. ACM Principles on Databases Systems, pages 11–22.

[Milo et al., 2003] Milo, T., Suciu, D., et Vianu, V. 2003. Typechecking for XML transformers. Journal of Computer and System Sciences, 66(1) :66–97.

[Rabin, 1960] Rabin, M. 1960. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report No. 2, Hebrew University, Branch of Applied Logic.

[Rabin et Scott, 1959] Rabin, M. O. et Scott, D. 1959. Finite automata and their decision problems. IBM Journal of Research and Development, 3 :114–125.

[relaxNG, 2001] relaxNG 2001. Relax ng. http ://relaxng.org/.

[Sakarovitch, 2003] Sakarovitch, J. 2003. Éléments de théorie des automates. Éditions Vuibert. Table of Contents, preface and introductions to chapters available at http ://perso.enst.fr/∼jsaka/ETA/.

[Savitch, 1970] Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences, 4(2) :177–192.

[schematron, 2004] schematron 2004. schematron. http ://www.schematron.com/.

[Segoufin, 2003] Segoufin, L. 2003. Typing and querying xml documents : some complexity bounds. In : ACM Principles on Databases Systems. pages 167–178.

[Seidl, 1990] Seidl, H. 1990. Equivalence of finite-valued bottom-up finite state tree transducers is decidable. In : Colloquium on Trees in Algebra and Programming. Springer-Verlag, pages 269–284.

[Shamir, 1990] Shamir, A. 1990. Ip=pspace. In : IEEE Symposium on Foundations of Computer Science. I :11–15.

[Stearns et Hunt, 1981] Stearns, R. E. et Hunt, H. B. 1981. On the equivalence and containment problems for unambiguous regular expressions, grammars, and automata. In : IEEE Symposium on Foundations of Computer Science. pages 74–81.

[Stearns et Hunt, 1985] Stearns, R. E. et Hunt, H. B. 1985. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM Journal of Computing, 14(3) :598–611.

[Stockmeyer et Meyer, 1973] Stockmeyer, L. J. et Meyer, A. R. 1973. Word problems requiring exponential time(preliminary report). In : ACM Symposium on Theory of Computing. pages 1–9.

[Thatcher et Wright, 1968] Thatcher, J. W. et Wright, J. B. 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory, 2 :57–81.

[Trang, 2004] Trang 2004. Trang. http ://www.thaiopensource.com/relaxng/trang.html.

[Valiant, 1974] Valiant, L. G. 1974. The decidability of equivalence for deterministic finite-turn pushdown automata. In : ACM Symposium on Theory of Computing. pages 27–32.

[Valiant, 1979] Valiant, L. G. 1979. The complexity of computing the permanent. Theoretical Computer Science, 8 :189–201.

[Vardi, 1998] Vardi, M. Y. 1998. Automata-theoretic approach to automated verification. Technical report, Rice University.

[Wang et al., 2004] Wang, C., Hong, M., Pei, J., Zhou, H., Wang, W., et Shi, B. 2004. Efficient pattern-growth methods for frequent tree pattern mining. In : PAKDD. pages 441–451.

[World Wide Web Consortium, 1999] World Wide Web Consortium 1999. XSL Transformations (XSLT) Version 1.0. http ://www.w3.org/TR/xslt.

[World Wide Web Consortium, 2006] World Wide Web Consortium 2006. XML Version 1.0 (Fourth Edition). http ://www.w3.org/TR/2004/REC-xml-20040204/.

[xmlSchema, 2001] xmlSchema 2001. XML Schema. http ://www.w3.org/XML/Schema.

[Yang et al., 2004] Yang, L. H., Lee, M. L., Hsu, W., et Guo, X. 2004. 2pxminer : an efficient two pass mining of frequent xml query patterns. In : Knowledge Discovery and Data Mining. pages 731–736.

[Zachar, 1978] Zachar, Z. 1978. The solvability of the equivalence problem for deterministic frontier-to-root tree transducers. Acta Cybernetica, 4 :167–177.

[Zaki, 2002] Zaki, M. J. 2002. Efficiently mining frequent trees in a forest. In : Knowledge Discovery and Data Mining. pages 71–80.

Annexe A

Appendices

A.1 Classes de Complexité

NP :

Une « machine NP » est une machine de Turing non-déterministe, polynomiale en temps. La classe NP est la classe des problèmes de décision résolubles en temps polynomial par une machine NP tels que :

- Si la réponse est oui, au moins un calcul est accepté.
- Si la réponse est non, tous les calculs sont rejetés.

De façon équivalente, NP est la classe de problèmes pour lesquels :

- Si la réponse est oui, il y a une preuve de longueur polynomiale qui, de ce fait, peut être vérifiée dans PTIME.
- Si la réponse est non, l'algorithme doit réfuter toute proposition de preuve qui affirmerait que la réponse est oui.

Un problème de décision est NP-complet si (1) il est dans NP, et (2) tout problème de NP peut se réduire à lui (par une réduction polynomiale). Le problème SAT, qui consiste à décider si une formule booléenne est satisfiable, a été le premier problème démontré NP-complet : Il est en effet dans NP puisqu'une assignation satisfaisant la formule est une preuve de taille polynomiale que l'on peut vérifier en temps polynomial (évaluation d'un circuit). Il est aussi NP-dur comme l'a démontré Cook [Cook, 1971]

PTIME :

Introduite entre autre par [Rabin, 1960, Cobham, 1965], PTIME est la classe des problèmes de décision résolubles en temps polynomial par une machine de Turing déterministe. Un problème est dit PTIME-complet s'il est dans PTIME, et si tout problème de PTIME peut se réduire à lui en espace logarithmique (LogSpace). Un problème standard PTIME-complet est l'évaluation d'un circuit : étant donné un circuit booléen et une entrée, la question est de décider la valeur de sortie de ce circuit sur cette entrée.

EXPTIME :

La classe EXPTIME regroupe les problèmes résolubles en temps (déterministe) exponentiel *i.e.*

$$\text{EXPTIME} = \bigcup_{p \ polynome} DTIME(2p(n))$$

PSPACE :

La classe PSPACE est celle des problèmes de décision résolubles par une machine de Turing en espace polynomial. Elle est égale à NPSPACE [Savitch, 1970], AP [Chandra et al., 1981] et IP [Shamir, 1990].

LIN, NLIN :

LIN (respectivement NLIN) est la classe des problèmes de décision résolubles en temps linéaire par une machine de Turing déterministe (resp. non déterministe).

LogSpace :

La classe LogSpace (aussi appelée **L**) est la classe des problèmes de décision résolubles par une machine de Turing utilisant une quantité de mémoire logarithmique en la taille de l'entrée. La connexité d'un graphe non orienté est un problème de LogSpace. LogSpace contient NC^1

LOGDCFL :

La classe LOGDCFL est celle des problèmes réductibles, en LogSpace, au problème de l'appartenance à un langage hors contexte. C'est aussi celle des problèmes de décision résolubles par une famille uniforme de circuits AC^1, dans lesquels aucune porte ET n'a plus de deux entrées.

NC :

On parle d'un circuit de degré entrant k quand chaque porte du circuit a un nombre constant (k) d'entrées. Typiquement $k = 2$. NC^i est alors la classe de problèmes de décision résolubles par une famille uniforme de circuits booléens de taille polynomiale, de profondeur $O(\log^i(n))$ et de degré entrant 2. La classe NC est alors définie comme l'union pour $i \in \mathbb{N}$ des classes NC^i.

$\#P$:

Définie dans [Valiant, 1979], cette classe regroupe des problèmes de calcul de la forme « calcule $f(x)$ », où f est le nombre de chemins acceptants d'une machine NP. Un problème canonique $\#P$-complet est le calcul, étant donnée une formule booléenne, du nombre d'assignations satisfaisant la formule. Calculer le nombre de couplages parfaits dans un graphe biparti, ou évaluer le permanent d'une matrice 0,1 sont aussi des problèmes $\#P$-complets.

A.2 Opérations sur les Automates de Mots

A.2.1 Notions élémentaires de théorie de Graphes

On rappelle qu'un graphe orienté $G = (V, E)$ est composé d'un ensemble de sommets V et d'un ensemble d'arêtes orientées E. Un chemin π est une suite $(x_0, x_1, ..., x_{n-1}, x_n)$ de sommets de G telle que deux sommets consécutifs quelconques x_i et x_{i+1} sont reliés par un arc de $G : \forall i,\ 0 \leq i \leq n-2,\ (x_i, x_{i+1}) \in E$. Les sommets x_0 et x_n sont respectivement l'origine et l'extrémité du chemin π. Un chemin est simple quand tous ses sommets sont deux à deux distincts. On appelle circuit un chemin de longueur non nulle et dont l'origine et l'extrémité sont identiques. Une boucle élémentaire est un chemin de s à s dont tous les arcs sont deux à deux distincts et dont tous les sommets sont distincts, sauf pour s qui apparaît exactement deux fois.

Une composante fortement connexe C d'un graphe $G = (V, E)$ est un sous-ensemble maximal de sommets tels que deux quelconques d'entre eux soient reliés par un chemin :

- si $x \in C$, alors $\forall y \in C$, il existe un circuit passant par x et y
- $\forall z \in S \setminus C$, il n'existe pas de circuit passant par x et z.

A.2.2 Construction du Graphe des Composantes Fortement Connexes

Une étape souvent utilisée dans cette thèse est l'emploi de la décomposition d'un automate ou d'un transducteur en l'ensemble de ses composantes fortement connexes. Ce pré-traitement ne tient absolument pas compte des étiquettes présentes sur les arêtes et on peut donc appliquer un traitement de décomposition sur les graphes orientés en oubliant les étiquettes.

Proposition A.1

On sait obtenir l'ensemble des composantes fortement connexes d'un graphe (orienté) à n sommets et m arêtes en temps $\Theta(n+m)$. ⋆

Démonstration : Il s'agit en effet d'un problème classique de théorie des graphes qui peut se résoudre par l'application de deux parcours en profondeur du graphe qui coûtent chacun un temps $\Theta(n+m)$. Les détails de la preuve que l'algorithme 14 calcule bien les composantes fortement connexes peuvent se trouver par exemple dans la section 23.5 de la première édition de [Cormen et al., 1990].

□

Algorithme 14 : Le Calcul des Composantes Fortement Connexes

début

1 Appeler $DFS(G)$ en marquant pour chaque nœud u son temps de fin $f(u)$;

2 Calculer G^T, le graphe G dans lequel on a inversé le sens de toutes les arêtes;

3 Appeler $DFS(G^T)$ en imposant un parcours des noeuds dans l'ordre décroissant pour les $f(u)$ fournis par la ligne 1 ;

4 Sortir les sommets de chaque arbre de la forêt (en profondeur d'abord) de l'étape 3 comme une composante fortement connexe séparée;

fin

A.2.3 Décision du Vide

Pour un automate Λ-libre, on peut, sans perte de généralité, négliger les étiquettes des arêtes utilisées et on peut donc se limiter au problème où toutes les arêtes sont étiquetées par un même symbole a. Nous allons donner des méthodes de décision du vide d'un automate $\mathcal{A} = (\{a\}, Q, S_0, F, \delta)$ suivant quelques modèles de complexité (non déterministe, déterministe et parallèle) dont les résultats sont résumés ci-dessous.

Dans un modèle non déterministe

Algorithme 15 : Approche Non Déterministe de la Connection

Données : Un graphe orienté, S_0 un ensemble d'états initiaux, F un ensemble d'états finaux

Résultat : La décision de la connexion entre S_0 et F

```
1 Choisir s_0 ∈ S_0;
2 Soit s l'état courant  début
3 |   Choisir un état t tel que s est directement connecté à t ;
4 |   Si t ∈ F retourner Oui
5 fin
```

L'approche non déterministe proposée par l'algorithme 15 (*page* 172) est dans NLOG-SPACE et donc dans NC.

Dans un modèle déterministe : L'accessibilité dans un graphe peut être résolue par une recherche en profondeur (DFS) ou en largeur (BFS). Le temps et l'espace nécessaires sont alors linéaires en $||\mathcal{A}||$. Pour $\mathcal{A}$ à n_A états et a_A arêtes, on peut effectuer cette recherche en temps $O(n'_A + a_A)$ que l'on peut facilement ramener à un temps $O(a_A)$ puisque tous les (n'_A) sommets réellement considérés sont issus d'une arête (*i.e.* $n'_A \leq a_A$).

Lien avec la $s-t$ connection : Le problème du vide est une extension du problème de $s-t$ connection qui, étant donné un graphe (G,V) à n et deux de ses sommets s et t, on cherche à décider si t est connecté à s. Ce problème est connu comme étant NLOGSPACE-complet, on va montrer qu'il est dans $\mathsf{SPACE}(\log^2 n)$.

1. Soit (s,t,n) la racine de l'arbre.
2. Un nœud (x,y,k) est un nœud OU dont chaque fils est étiqueté par un couple $((x,z,\lfloor\frac{k}{2}\rfloor),(z,y,\lceil\frac{k}{2}\rceil))$
3. Un nœud $((x,z,\lfloor\frac{k}{2}\rfloor),(z,y,\lceil\frac{k}{2}\rceil))$ est un nœud ET possédant 2 fils étiquetés respectivement $((x,z,\lfloor\frac{k}{2}\rfloor)$ et $(z,y,\lceil\frac{k}{2}\rceil))$

On applique alors l'algorithme 16

Algorithme 16 : La s-t Connection version Arborescente

Soit (s,t,n) la racine de l'arbre;

début

Étiqueter un nœud (x,y,k) avec 1 si y est directement connecté à x;

Étiqueter un nœud OU avec 1 si un de ses enfants est étiqueté 1;

Étiqueter un nœud ET avec 1 si tous ses enfants sont étiquetés 1;

fin

La hauteur de l'arbre construit est inférieure à $\log(n)$ et le facteur de branchement est inférieur à n à chaque niveau. L'arbre est donc de taille $O(n^{\log n})$. On peut éviter la construction explicite de cet arbre en cherchant récursivement dans l'arbre : la longueur de la récursion est $\log n$, l'espace utilisé est $O(\log^2 n)$ et le temps $2^{\log^2 n} = n^{\log n}$.

Dans un modèle parallèle : Supposons que des processeurs $P(x,y)$ déterminent, pour chaque paire d'états, si y est directement connecté à x, et que l'on dispose aussi de processeurs $P(x,y,z)$ vérifiant si y est connecté à x et z à y. Avec $O(n^3)$ processeurs, le problème de s-t connection peut être résolu en $O(\log n)$ étapes.

A.2.4 Élimination de Transitions Spontanées

Une remarque théorique simple (proposition 1.17 de [Sakarovitch, 2003]) nous permet d'oublier temporairement la présence d'ϵ-transitions sans perdre la généralité des automates considérés.

Fait A.2 (Automates Λ-libres)
Tout NFA avec des Λ-transitions peut être effectivement transformé en un NFA équivalent sans transition spontanée avec le même ensemble d'états.

L'élimination des Λ-transitions peut se faire récursivement : Supposons une Λ-transition de a à b, et $b_1, ..., b_l$ accessible depuis b par les lettres $w_1, ..., w_l$. Cette Λ transition peut être remplacée par les transitions de a vers w_i étiquetées par w_i. Pour k Λ-transitions, on engendre moins de $|\text{Nombre d'arêtes }|^k$ nouvelles transitions. N'autoriser aucune Λ-transition ou en autoriser un nombre k fini ne provoque donc aucun changement vis-à-vis de la polynomialité des résultats obtenus.

Élimination efficace pour les automates pondérés : Dans le cadre des automates normaux, l'ajout d'une pondération non nulle de toutes les arêtes existantes permet d'appliquer des éliminateurs d'Λ-transitions plus généraux, et d'oublier les pondérations de l'automate généré pour obtenir un algorithme de même complexité permettant l'élimination des Λ-transitions.

Proposition A.3 ([Duchamp et al., 2005])
Soit k un semi-anneau. L'élimination des Λ-transitions est calculable en temps $O((|\Sigma| + 1) \times m^\omega$ si m est la dimension de l'automate pondéré. $\star$

m^ω désigne le coût de calcul de l'étoile droite (gauche) d'une matrice de taille m. ω est connu pour être 3 si k n'est pas un anneau, 2.808 si k est un anneau et 2.376 si k est un corps.

Pour des pondérations tropicales : Un automate, d'état Q et d'arêtes E (dont des Λ-transitions) pondérées sur un semi-anneau peut être transformé en un automate équivalent sans Λ-transition en temps $O(|Q||E| + |Q|^2 \log |Q|)$ pour le semi-anneau tropical[49].

[49] $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$

A.3 Opérations sur les Transducteurs de Mots

A.3.1 Construction du Graphe des Composantes Fortement Connexes

La décomposition en composantes fortement connexes est fournie par l'algorithme 14 (*page* 171) dans le cadre des graphes orientés. Comme l'étiquetage ne joue pas de rôle dans cette décomposition, on peut aussi l'utiliser pour les transducteurs. On fournit alors, de la même manière, une décomposition du transducteur en son graphe des composantes fortement connexes en temps $\Theta(n+m)$ où n est le nombre de sommets et m le nombre d'arêtes du transducteur.

A.3.2 Exponentiation d'un Transducteur

Définition A.4 (Exponentiation d'un transducteur)
Soit $\mathscr{T} = (\Sigma, \Sigma', Q, I, F, \delta)$ un transducteur Λ-libre et $k \in \mathbb{N}$. On notera $\mathscr{T}^k$ *la* $k^{ème}$*puissance de* $\mathscr{T}$ et définie par

$$\mathscr{T}^k = (\Sigma^k, \Sigma', Q, I, F, \delta^k)$$

où $(q_1\ ,\ a_1a_2...a_k\ ,\ w\ ,\ q_{k+1}) \in \delta^k$ si et seulement si il existe $q_1, ..., q_k \in Q$ et $w_1, ..., w_k \in (\Sigma')^*$ tels que $w = w_1.w_2...w_k$ et pour tout $i \in \{1, k\}$, $(q_i, a_i, w_i, q_{i+1}) \in \delta$. ✓

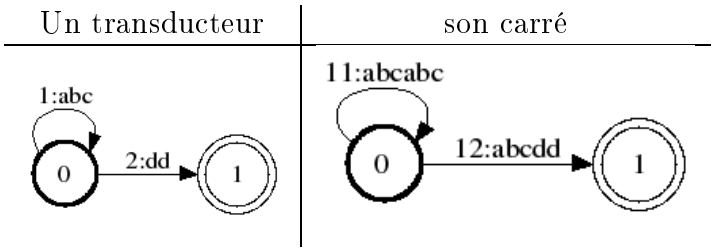


Fig. A.1 – *Une élévation à la puissance deux.*

On va maintenant considérer les boucles dans ce nouveau transducteur $\mathscr{T}^k$.

Proposition A.5 (Majoration du nombre de boucles)

Soit $\mathcal{A}$ un automate (respectivement $\mathscr{T}$ un transducteur) à a arêtes. Alors $\mathcal{A}^k$ (respectivement $\mathscr{T}^k$) possède au plus a^k boucles élémentaires. ⋆

Démonstration : Le résultat est immédiat pour $k = 1$, le pire des cas étant que chaque arête soit elle-même une boucle élémentaire. Pour k supérieur, le pire cas est que toutes les arêtes soient des boucles élémentaires bouclant sur le même état ; il y a dans ce cas a^2 boucles élémentaires dans $\mathcal{A}^2$ puisque tout couple d'arêtes (éventuellement identiques) produit une boucle élémentaire. Le cas le plus défavorable pour des k supérieurs est aussi celui présenté pour $k = 1$ ou 2. On obtient donc une borne supérieure de a^k boucles élémentaires dans $\mathcal{A}^k$, et cette borne est atteinte de cette façon. □

A.3.3 Transduction d'un Mot

Fait A.6 (Fait 3.4 (*page* 77))

Soit un mot w et un transducteur $\mathscr{T}$ normalisé à $n_{\mathscr{T}}$ états et $a_{\mathscr{T}}$ arêtes. On sait construire $\mathscr{T}(w)$, la transduction de w par $\mathscr{T}$, en temps $\mathcal{O}(|w|.n_{\mathscr{T}}.a_{\mathscr{T}})$. L'automate engendré (FSA) possède moins de $|w|.n_{\mathscr{T}}$ états et moins de $2.|w|.n_{\mathscr{T}}.a_{\mathscr{T}}$ arêtes.

Démonstration : Soit $\mathscr{T} = (\Sigma, \Sigma', Q_{\mathscr{T}}, I_{\mathscr{T}}, F, \delta_{\mathscr{T}})$. On associe à w l'automate $\mathbf{K}_w = (\Sigma, Q, I, F, \delta)$ où $Q = \{1, |w| + 1\}$, $I = \{1\}$, $F = \{|w| + 1\}$ dans lequel $\delta : (i \xrightarrow{w[i]} i + 1)_{i=1..|w|}$. Posons $\mathbf{K}' = (\Sigma', Q \times Q_{\mathscr{T}}, I \times I_{\mathscr{T}}, F \times F_{\mathscr{T}}, \delta')$ où δ' est obtenu par l'algorithme 17.

Par construction, l'algorithme produit l'automate de l'ensemble des images de w par $\mathscr{T}$. Il effectue $|w|$ boucles extérieures. Chaque boucle est composée de deux boucles internes, toutes deux effectuées $n_{\mathscr{T}}$ fois, et chacune pouvant être générée en regardant l'ensemble des arêtes de $\mathscr{T}$. La complexité totale est alors en temps $\mathcal{O}(|w|.n_{\mathscr{T}}.a_{\mathscr{T}})$. □

Algorithme 17 : Calcul de la fonction de transition de la transduction d'un mot

Données : Un mot w et un transducteur $\mathscr{T}$
Résultat : la fonction de transition de $\mathscr{T}(w)$
$Aux := \emptyset$; $Res := \emptyset$;
pour $i \in \{0, |w|\}$ **faire**
  pour $j \in \{1, n_{\mathscr{T}}\}$ **faire**
    $Aux \mathrel{+}= \{(x,k)$ tel que $j \xrightarrow{(\Lambda|x)} k\}$;
1   $Res \mathrel{+}= (\text{ pour } (x,k) \in Aux, \text{ créer les arêtes } (i,j) \xrightarrow{x} (i,k))$;
    $Aux := \emptyset$;
  fin
  Soit w_i la lettre tel que : $i \xrightarrow{w_i} i+1$;
  pour $j \in \{1, n_{\mathscr{T}}\}$ **faire**
    $Aux \mathrel{+}= \{(x,k)$ tel que $j \xrightarrow{(w_i|x)} k\}$;
2   $Res \mathrel{+}= (\text{ pour } (x,k) \in Aux, \text{ créer les arêtes } (i,j) \xrightarrow{x} (i,k))$;
    $Aux := \emptyset$;
  fin
fin
retourner Res ;

A.3.4 Élimination des Transitions Spontanées

Il n'est pas toujours possible d'éliminer complètement les Λ-transitions. En effet une boucle simple lisant Λ et écrivant $a \neq \Lambda$ n'a bien sûr pas d'équivalent sans Λ-transition entrante. Cette opération de suppression d'Λ-transitions a un équivalent algébrique permettant de traiter ce genre d'opération sur un semi-anneau. Elle est très liée aux problèmes de plus courts chemins puisqu'elle se base sur des généralisations des algorithmes classiques tels que Floyd-Warshall ou Dijkstra. Cette élimination des transitions spontanées, lorsqu'elle est possible, peut être effectuée efficacement :

Proposition A.7 (Coût de l'élimination des Λ-transitions)
Entrée : un transducteur de mots pondérés sur un semi-anneau avec des Λ-transitions
Sortie : un transducteur équivalent sans Λ-transition (en entrée, ou en sortie).
*Pour un transducteur et un semi-anneau quelconque, une généralisation de la proposition A.2.4 (*page *174) permet l'élimination des Λ transitions (en entrée ou en sortie) en temps* $O(|Q|^3(T_{\oplus} + T_{\otimes} + T_*))$ $\star$

Démonstration : Par une adaptation de la procédure classique d' Λ-fermeture (voir par exemple [Hopcroft et Ullman, 1990]), on obtient un transducteur équivalent, sur le même nombre d'états, libre d'Λ transitions en sortie.
Soit $\mathscr{T}$ le String-Transducteur donné par $\mathscr{T} = (\Delta, \Delta, S, \tau S_0, F)$. Posons $\xi_{\mathscr{T}}$ le graphe obtenu à partir de celui de $\mathscr{T}$ en ne conservant que les Λ-transitions, et $\mathscr{T}' = (\Delta, \Delta, S, \tau', S_0, F')$ où

$$F' = F \cup \{s : s \in S_0 \wedge F \cap \Lambda - fermeture(s) \neq \emptyset\}.$$

$$\tau = \left\{ \begin{array}{c} \{(s, R', t, R) : (v, R', t, R) \in \tau \wedge R \neq \Lambda\} \\ \bigcup \\ \{(s, W, t, R) : \exists v \in \Lambda\text{-fermeture}(s) \text{ tq } (v, R', t, R) \in \tau \text{ où } R \neq \emptyset\} \end{array} \right.$$

où W est un mot étiquetant le (ou un des) plus court(s) chemin(s) de s à v dans $\xi_{\mathscr{T}}$ ajouté au poids de la plus petite transition sans output étiquetée par R de l'état v à l'état t dans $\mathscr{T}'$. $\square$

Table des figures

Liste des Algorithmes

Index