Sur les arbres de rang non bornés avec données

Jérémie Dimino

22 août 2010

Laboratoire Spécification et Vérification
ENS de Cachan

Résumé

Ce document constitue le rapport de mon stage que j'ai effectué au Laboratoire Spécification et Vérification à l'ENS de Cachan, sous la direction de Luc Segoufin et Florent Jacquemard.

Durant ce stage nous nous sommes intéressés à différents modèles d'automates sur les arbres de rang non bornés avec données.

Table des matières

1 Introduction

Nous nous proposons ici d'étudier des arbres de rang non bornés avec données. Les arbres de rang non bornés sont des arbres dont les noeuds peuvent avoir un nombre arbitraire de fils. Les arbres avec données sont des arbres où chaque noeud, en plus d'être étiqueté par une lettre provenant d'un alphabet fini, contient une valeur provenant d'un ensemble infini de données.

De tels abres apparaissent naturellement dans divers domaines de l'informatique. Par exemple un document XML est un arbre de rang non bornés avec données.

Nous allons nous pencher sur différents modèles pour définir des langages d'arbres de rang non borné avec données. En particulier en utilisant des automates d'arbres à registres ou des formules de logique. Un automate à registres est un automate muni d'un nombre fini de registres qui peuvent être utilisés pour stocker et comparer des données. Nous étudierons les lines qui existent entre ces différents modèles et la décidabilité de ceux-ci.

Concernant les mots avec données, on sait aujourd'hui que trois modèles différents sont équivalents et décidables : le modèle en utilisant des formules d'une certaine logique ($FO_2(+1, <, \sim)$), les automates à données et les machines à compteurs (les *VASS*).

Notre but original était de chercher un résultat similaire pour les abres de rang non borné à données. Nous n'y sommes pas parvenu mais avons obtenu plusieurs résultats sur ces modèles.

Nous commencerons tout d'abord par donner quelques définitions, puis nous rappellerons les résultats existants sur les mots. Nous définirons ensuite des modèles d'automates pour les arbres et enfin nous comparerons ces modèles.

2 Préliminaires

Dans ce rapport, nous utiliserons la notation Σ pour désigner un alphabet fini et D pour désigner un ensemble infini de données. On notera $\mathbb{B}$ l'ensemble des booléens et $\binom{n}{k}$ (lu "k parmis n") les coefficients binomiaux.

2.1 Mots et arbres à données

Nous commençons tout d'abord par donner quelque définitions sur les mots et arbres à données.

Définition 1 (Mot à données)**.** *Un mot à données sur l'alphabet Σ et l'ensemble de données D est un une suite finie d'éléments de $\Sigma \times D$. On notera ces mots sous la forme : $(a_1, d_1) \dots (a_n, d_n)$.*

La propection d'un mot à données w est le mot de Σ^* obtenu en retirant toutes les données de w ; ainsi la projection de $(a_1, d_1) \dots (a_n, d_n)$ est $a_1 \dots a_n$.

Définition 2 (Arbre de rang non borné avec données)**.** *Un arbre de rang non borné t est composé d'un sous ensemble fini non-vide de $\mathbb{N}^*$, noté $Dom(t)$, vérifiant les deux propriétés suivantes :*

– *pour toute suite finie d'entiers naturels u, si u appartient à $Dom(t)$, alors tous les préfixes de u sont également dans $Dom(t)$,*
– *pour toute suite finie d'entiers naturels u et tout entier $k > 0$, si $uk \in Dom(t)$, alors $u(k-1) \in Dom(t)$.*

Les suites maximales de $Dom(t)$ (pour le pré-ordre préfixe) sont appelées les feuilles de t, et le mot vide est appelé la racine de t.

Les noeuds de l'arbre peuvent être étiquetés par des lettres provenant d'un ensemble fini Σ. Un étiquetage de t est une fonction label : $Dom(t) \to \Sigma$.

Enfin, l'abre peut contenir des données provenant d'un ensemble infini D, dont l'assignement est donné par une fonction data : $Dom(t) \to D$.

Remarque 1. *Un arbre est dit binaire si tous ses noeuds internes (i.e. tous les noeuds qui ne sont pas des feuilles) ont exactement deux fils.*

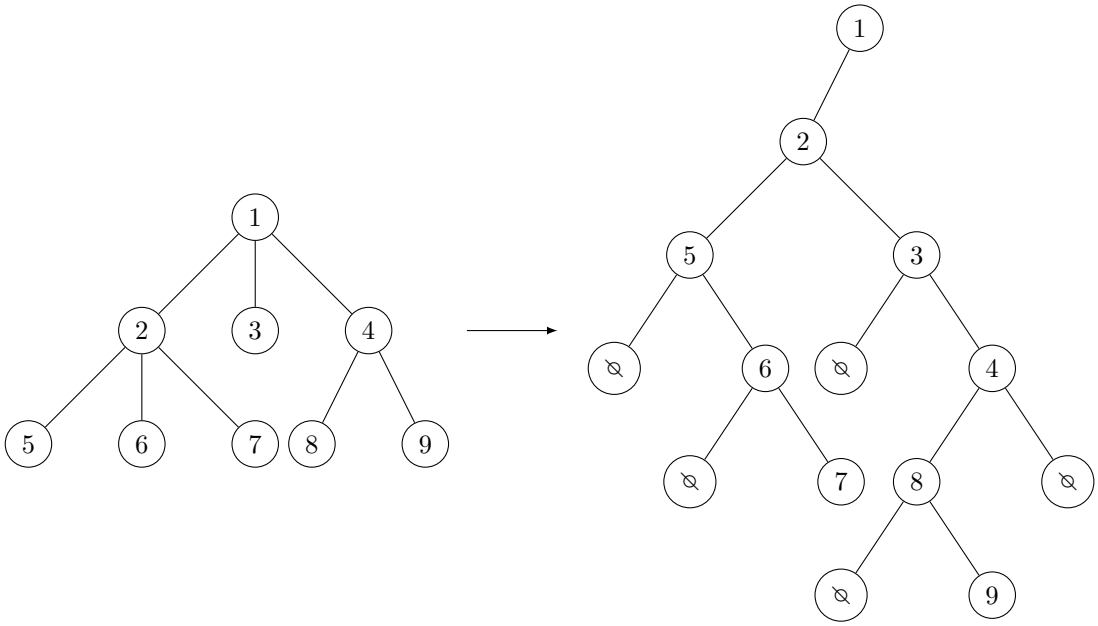
2.2 Codage fils gauche frère droit

Le codage *fils gauche frère droit* est une injection des arbres de rang non bornés dans les arbres binaires, définie comme suit :

Définition 3. *Étant donné un arbre de rang non borné t, on définit son codage fils gauche frère droit comme l'abre binaire t' construit de la manière suivante : la racine de t est conservée et chaque noeud n de t qui devient le noeud n' de t' vérifie :*
– *si n est une feuille de t alors n' est une feuille de t',*
– *si n a au moins un fils dans t, alors le premier de ceux-ci est le fils gauche de n' dans t', sinon le fils gauche de n' dans t' est $\diagdown$,*
– *si n à un frère droit dans t, alors celui-ci est le fils droit de n' dans t', sinon le fils droits de n' dans t' est $\diagdown$.*

Où $\diagdown$ est une nouvelle étiquette qui n'apparait dans l'alphabet des étiquettes de t.

Par exemple :



2.3 Logique FO_2 sur les mots et arbres à données

Définition 4. *On définit $FO_2(+1, <, \sim)$sur les mots avec données comme étant la logique du premier ordre restreinte à deux variables avec les prédicats suivant : le prédicat unaire a pour chaque lettre $a \in \Sigma$ et les prédicats binaires $+1$, $<$, $\sim$.*

Étant donné un mot w, la sémantique est la suivante :

- *le champ des variables est l'ensemble des positions dans w,*
- *$a(x)$ est vrai ssi l'étiquette à la position x dans w est a,*
- *$+1(x, y)$ (aussi noté $x = y + 1$) est vrai ssi y est la position suivante de x dans w,*
- *$x \sim y$ est vrai ssi les valeurs de données au position x et y dans w sont égales.*

Définition 5. *On définit la logique $FO_2(+1, <, \sim)$sur les arbres avec données de la manière suivante : étant donné un arbre t, on a :*

- *le champs des variables est l'ensemble des noeuds de t, i.e. $Dom(t)$,*
- *pour toute lettre a, il y a un prédicat unaire noté a qui est vrai pour tous les noeuds étiquetés par a, et faux pour les autres,*
- *il y a un prédicat $E_{\rightarrow}$ tel que $E_{\rightarrow}(x, y)$ est vrai ssi x et y ont le même père est y est le frêre immédiat de x. Formellement il existe un noeud $n \in Dom(t)$ tel que $x = nk$ et $y = n(k + 1)$ pour un certain $k \in \mathbb{N}$,*
- *$E_{\Rightarrow}$ est la clôture transitive de $E_{\rightarrow}$,*
- *il existe un prédicat $E_{\downarrow}$ tel que $E_{\downarrow}(x, y)$ est vrai ssi x est le père de y,*
- *$E_{\Downarrow}$ est la clôture transitive de $E_{\downarrow}$.*

3 Sur les mots

Dans cette partie nous décrivons les résultats existant sur les mots avec données. Il existe deux types d'automates pour les mots avec données : les automates à registres et les automates à données.

Définition 6 (Automates à registres)**.** *Un automate à registre sur les mots est un automate fini muni d'un nombre fini de registres contenant des données provenant d'un ensemble infini D. À chaque transition, en plus de tenir compte de l'étiquette courante et de son état, l'automate peut comparer la valeur de donnée courante avec celles contenues dans ses registres (la comparaison s'effectue uniquement avec l'égalité). Il peut également choisir de stocker la valeur courante dans un de ses registres.*

Avant d'introduire les automates à données, on définit la notion de classe de donnée pour un mots avec données.

Définition 7 (Classes de données)**.** *Soit w un mot avec données et d une donnée. La classe de w pour la donnée d est la projection du sous-mot de w obtenue en retirant toute les position ayant une valeur de donnée différente de d.*

Par exemple la classe de 1 pour le mot avec données $\mathbf{(a,\ 1)}(b,2)(c,2)\mathbf{(d,\ 1)}(a,3)$ est ad.

Définition 8 (Automates à données)**.** *Un automate à données sur les mots avec données est un couple $\langle A, B\rangle$ où A est un transducteur lettre-à-lettre et B est un automate fini.*

Une exécution d'un automate à données $\langle A, B\rangle$ sur un mot $w = (a_1, d_1)\dots(a_n, d_n)$ s'effectue comme suit :

- A est appliqué sur la projection de w, et donne en sortie un mot $w' = a'_1 \dots a'_n$ (si w est accepté),
- B est appliqué sur chacune des classes de $(a'_1, d_1)\dots(a'_n, d_n)$.

w est accepté ssi il est accepté par A et si toutes les classes de $(a'_1, d_1)\dots(a'_n, d_n)$ sont acceptées par B.

Nous rappelons maintenant les relations entre ces différents modèles :

Proposition 1. *[BS10] Les automates à registres peuvent être simulés par des automates à données.*

Proposition 2. *Pour tout langage L reconnu par une formule de $FO_2(+1, <, \sim)$, il existe un automate à données qui reconnaît L.*

Il existe plusieurs preuve pour cette proposition. La plus élégante utilise la proposition 1. C'est pourquoi nous chercherons un résultat similaire pour les arbres.

Proposition 3. *Les automates à données sont inclus dans les machines à compteurs (les VASS).*

Proposition 4. *Les machines à compteurs sont incluses dans* $FO_2(+1, <, \sim)$.

Finalement on a le résultat de décidabilité suivant :

Proposition 5. *[KF94] Le vide des automates à registres est décidable.*

D'où l'on déduit que ces trois modèles (automates à registres, logique FO_2 et automates à compteurs) sont décidables.

4 Automates d'arbres à registres

Dans cette partie nous décrivons les automates à registres opérant sur des arbres avec données.

4.1 Automates d'abres binaires avec données

Définition 9. *Un automate d'abres binaires à registres Bottom-Up est un tuple* $\langle Q, q_0, F, \delta, \Sigma, D, k\rangle$ *où :*

- Q *est un ensemble fini d'états*
- q_0 *est l'état initial*
- $F \subseteq Q$ *est l'ensemble des états finaux*
- $k \in \mathbb{N}$ *est le nombre de registres de l'automate*
- Σ *est un alphabet fini d'étiquettes*
- D *est un ensemble infini de données*
- $\delta \subseteq Q \times Q \times \Sigma \times \mathbb{B}^{\binom{2k+1}{2}} \times Q \times ((\{right, left\} \times \{1 \dots k\}) \uplus \{current, new\})^k$

Une exécution sur un arbre t est une paire de fonctions $\lambda_s : Dom(t) \to Q$ (états assignés aux noeuds de l'arbre) et $\lambda_r : Dom(t) \to D^k$ (contenus des registres pour chaque noeud de l'arbre), vérifiant :

- pour toute feuille f de t et $\lambda_s(f) = q_0$ et $\lambda_r(f)$ est arbitraire.
- pour tout noeud n de t, étiqueté par a et avec une valeur de donné d, il existe une règle de transition $(q_1, q_2, b, tests, q, assign)$ dans δ vérifiant :
  - $q_1 = \lambda_s(n1)$, $q_2 = \lambda_s(n2)$ et $q = \lambda_s(n)$
  - $b = a$
  - les registres de $n1$ et $n2$ ainsi que la donnée d vérifient tous les tests décrit par $tests$
  - pour tout i compris entre 1 et k, on a :
    - si $assign[i] = (left, j)$, alors $\lambda_r(n)[i] = \lambda_r(n1)[j]$
    - si $assign[i] = (right, j)$, alors $\lambda_r(n)[i] = \lambda_r(n2)[j]$
    - si $assign[i] = current$, alors $\lambda_r(n)[i] = d$

    Si $assign[i] = new$, il n'y a aucune condition sur le contenu de $\lambda_r(n)[i]$.

Une exécution est acceptante si $\lambda_s(\varepsilon) \in F$.

Définition 10. *Un automate d'abres binaires à registres Top-Down est un tuple* $\langle Q, q_0, F, \delta, \Sigma, D, k\rangle$ *où*

- Q *est un ensemble fini d'états*
- q_0 *est l'état initial*

- $F \subseteq Q$ *est l'ensemble des états finaux*
- $k \in \mathbb{N}$ *est le nombre de registres*
- Σ *est un alphabet fini de label*
- D *est un ensemble infini de données*
- $\delta \subseteq Q \times \Sigma \times \mathbb{B}^{\binom{k+1}{2}} \times Q \times Q \times (\{1 \dots k\} \uplus \{current, new\})^k \times (\{1 \dots k\} \uplus \{current, new\})^k$

Une exécution sur un arbre t est une paire de fonction $\lambda_s : Dom(t) \to Q$ et $\lambda_r : Dom(t) \to D^k$, vérifiant :

- $\lambda_s(\varepsilon) = q_0$ et $\lambda_r(\varepsilon)$ est arbitraire
- pour tout noeud n de t, étiqueté par a et avec une valeur de donné d, il existe une règle de transition $(q, b, tests, q_1, q_2, b, assign_{left}, assign_{right})$ dans δ vérifiant :
  - $q_1 = \lambda_s(n1)$, $q_2 = \lambda_s(n2)$ et $q = \lambda_s(n)$
  - $b = a$
  - les registres de n ainsi que la donnée d vérifient tous les tests décrit par $tests$
  - pour tout i compris entre 1 et k, on a :
    - si $assign_{left}[i] = j$, alors $\lambda_r(n1)[i] = \lambda_r(n)[j]$
    - si $assign_{left}[i] = current$, alors $\lambda_r(n1)[i] = d$

    Si $assign_{left}[i] = new$, il n'y a aucune condition sur le contenue de $\lambda_r(n1)[i]$. De même pour n2.

Une exécution est acceptante si pour toute feuille f, $\lambda_s(f)$ appartient à F.

Proposition 6. *Les automates à registres Top-Down et Bottom-Up ont le même pouvoir d'expression.*

4.2 Automates d'arbres de rang non borné

Définition 11 (UTA)**.** *Un automate d'arbres non bornés est un tuple* $\langle Q, \Sigma, \delta, F \rangle$*, ou :*

- Q *est un ensemble fini d'états*
- Σ *est un alphabet fini d'étiquettes*
- F *est un ensemble fini d'états finaux*
- $\delta : Q \times \Sigma \to 2^{Q^*}$ *est la fonction de transition, vérifiant que pour tout état* q *et toute lettre* a*,* $\delta(q, a)$ *est un langage régulier*

Une exécution sur un arbre t est une fonction $\lambda_s : Dom(t) \to Q$, tel que pour tout noeud n de t étiqueté par a et d'arité k, on a :

$$\lambda_s(n1) \dots \lambda_s(nk) \in \delta(\lambda_s(n), label(n))$$

En particulier, pour les feuilles, cette condition devient : $\varepsilon \in \delta(\lambda_s(n), label(n))$. Une exécution est acceptante si $\lambda(\varepsilon) \in F$.

4.3 Automates d'arbres de rang non borné à registres

Définition 12 (URTA). *Un automate d'arbres de rang non borné à registres est un tuple $\langle Q, \Sigma, D, D, \delta, F, k\rangle$, où :*

- *Q est un ensemble fini d'états*
- *Σ est un alphabet fini d'étiquettes*
- *D est un ensemble infini de données*
- *D est un ensemble inifini de données*
- *F est un ensemble fini d'états finanux*
- *$k \geq 0$ est le nombre de registres de l'automate*
- *$\delta : Q \times \Sigma \to 2^{Q^*} \times \Psi$ est la fonction de transition, vérifiant que pour tout état q et toute lettre a, si $\delta(q, a) = (L, \psi)$, alors L est un langage régulier et ψ est une formule sans variable libre, respectant la grammaire suivante :*

$$
\begin{array}{rl}
\psi ::= & \exists x.\psi \mid \forall y.\phi \\
\phi ::= & \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi \\
\mid & a(father) \\
\mid & a(z) \\
\mid & q(father) \\
\mid & q(z) \\
\mid & reg(father, i) = reg(father, j) \\
\mid & reg(father, i) = reg(z, j) \\
\mid & reg(x, i) = reg(x', j) \\
\mid & reg(x, i) = reg(y, j) \\
\mid & reg(y, i) = reg(y + p, j) \\
\mid & reg(x, i) = current \\
\mid & z < z' \\
\mid & x = x' + p
\end{array}
$$

où x désigne une variable liée existentiellement, y une variable liée universellement et z l'une où l'autre.

Une exécution d'un URTA à k registres sur un arbre t est une paire de fonctions $\lambda_s : Dom(t) \to Q$ et $\lambda_r : Dom(t) \to D^k$ telles que pour tout noeud n de t étiqueté par a et d'arité k, avec $\delta(\lambda_s(n), a) = (L, \psi)$ on a :

- L est régulier
- $\lambda_s(n1) \dots \lambda_s(nk) \in L$
- la formule ψ est vérifié au noeud n, en considérant que :
  - le champ des quantificateurs $\forall$ et $\exists$ est l'ensemble des fils de n,
  - $a(z)$ (resp. $a(father)$) signifie que le fils z (resp. n) a pour étiquette a,
  - $q(z)$ (resp. $q(father)$) signifie que le fils z (resp. n) est dans l'état q,
  - $reg(z, i)$ (resp. $reg(father, i)$) dénote le contenue du i-ème registres de z (resp. n).
  - *current* désigne la valeur de donnée de n

Une exécution est acceptante si $\lambda_s(\varepsilon) \in F$.

4.4 Lien entre les différents modèles

Nous explicitons maintenant les liens qui existent entre les différents modèles d'automates à registres.

Proposition 7. *Pour tout URTA $\mathcal{A}$, il existe un automate d'arbres binaires avec données à registres $\mathcal{A}'$, tel que un arbre t est reconnu par $\mathcal{A}$ ssi son codage fils gauche frère droit est reconnu par $\mathcal{A}'$*

Démonstration. Soit $\mathcal{A} = \langle Q, \Sigma, D, \delta, F, k\rangle$ un URTA. On construit un automate d'abres binaires à registres bottom-up $\mathcal{A}' = \langle Q', q'_0, F', \delta', \Sigma, k'\rangle$ de la manière suivante : lorsque l'on se trouve sur le fils le plus à droite d'un noeud, on devine l'état et l'étiquette du père, on en déduit :

$$\delta(q, a) = (L, \exists x_1 \dots x_n.\forall y.\phi)$$

On devine alors le contenue des registres des n témoins (en utilisant $n.k$ registres), puis on vérifie que ce qu'on a deviné est correct.

En parallèle on doit également vérifier que le mot formé par les états est dans L. On ne décrit ici que la véfification de la formule. Il suffit ensuite de faire le produit de l'automate obtenu avec l'autotmate qui vérifie la condition sur les états.

On note :
- N le nombre maximum de variables liées existentiellement dans les formules de Ψ. Pour simplifier on supposera que toutes les formules de Ψ en contiennent exactement N,
- P le maximum des p apparaissant dans les sous-formules $z = z' + p$ et $reg(y, i) = reg(y, i + p)$

On devine alors les informations suivante :
- l'état, l'étiquette, la valeur de donnée et les registres du père,
- les états, les étiquettes et les registres des variables liées existentiellement,
- l'ordre dans lequel ces variables apparaissent (et lesquelles sont égales),
- toutes les égalitées $x = x' + p$ entre les variables liées existentiellement.

On définit Q' de la manière suivante :

$Q' =$	Σ	*étiquette du père*
$\times$	Σ^N	*étiquettes des variables liées existentiellement*
$\times$	Q	*état du père*
$\times$	Q^N	*états des variables liées existentiellement*
$\times$	$\mathcal{O}(\{1 \dots N\})$	*ordre d'apparition des variables liées existentiellement*
$\times$	$\mathcal{P}(\{1 \dots N\} \times \{1 \dots N\} \times \{1 \dots P\})$	*ensemble des égalités $x = x' + p$ vraies*
$\times$	$\mathcal{P}(\{1 \dots N\})$	*témoins déjà visités*
$\times$	$\mathcal{P}(\{1 \dots N\} \times \{1 \dots P\})$	*distances aux derniers témoins visités*

Dans les registres de $\mathcal{A}'$ on stocke :
- la valeur de donnée et les registres du père
- les registres des n témoins

– les contraintes sur les registres des P prochaines positions, induites par les formules de la forme $reg(y, i) = reg(y + p, j)$

On choisi ensuite de manière non déterministe les témoins et on vérifie que les états, les étiquettes et les registres des témoins concordent avec les valeurs devinées. De même lorqu'on arrive sur le fils le plus à gauche, on vérifie que les valeurs devinées pour le père son correcte. □

Proposition 8. *Le vide d'un URTA est décidable.*

Démonstration. D'après la proposition 7, il suffit de montrer que le vide pour les arbres binaires est décidable. Pour cela on montre tout d'abord qu'il suffit de tester les arbres avec un nombre borné de valeurs de données différentes.

Soit $\mathcal{A} = \langle Q, q_0, F, \delta, \Sigma, D, k\rangle$ un automate d'arbres binaires à registres Bottom-Up. Pour tout arbre binaire t, on construit un arbre binainre t' partageant la structure et étiquetage de t mais n'ayant au plus que $2k + 1$ valeur de données différentes. t' est construit inductivement de la manière suivante :

– si t est est composé d'une seule feuille, on prend $t' = t$
– si t est un noeud avec une valeur de donnée d et deux fils $t1$ et $t2$ pour lequels on a construit deux arbres $t1'$ et $t2'$ utilisant au plus $2k+1$ valeurs de données différentes, on construit $t1''$ identique à $t1'$ mais où les valeurs de données sont renommées de la manière suivante, où $(\lambda_{i,s}, \lambda_{i,r})$ est une exécution de $\mathcal{A}$ sur t'_i :
  – toutes les valeurs de données apparaissant dans $\lambda_{1,r}(\varepsilon)$ et $\{d\}$ sont conservées,
  – toutes les autres son renommées par des valeurs n'apparaissant pas dans $\lambda_{1,r}(\varepsilon)$, mais pouvant apparaître dans $t2'$, $\lambda_{2,r}(\varepsilon)$ ou $\{d\}$.

  À un renomage près, $(\lambda_{1,s}, \lambda_{1,r})$ est une exécution de $\mathcal{A}$ sur $t1''$. On effectue la même opération pour $t2'$ et on définit t' comme l'abre t où l'on a remplacé $t1$ par $t1''$ et $t2$ par $t2''$, et où la valeur de donnée à la racine est remplacée par une valeur n'apparaissant ni dans $\lambda_{1,r}(\varepsilon)$ ni dans $\lambda_{2,r}(\varepsilon)$.

L'arbre t' ainsi construit a au plus $2k + 1$ valeurs de données différentes, et est accepté pas $\mathcal{A}$ ssi t l'est. Le problème de la décidabilité se ramène donc à celui de la décidabilité sur des arbres sans valeurs de données, qui est vrai. □

5 Automates à données et FO_2

Dans cette partie nous définissons les automates à données sur les arbres de rang non bornés.

Définition 13 (DTA)**.** *Un automate à données sur les arbres de rang non bornés est un couple $\langle A, B\rangle$ où A est un transducteur d'arbres lettre à lettre et B un automate d'arbres (sans registre).*

Définition 14 (Classes d'un arbre à données)**.** *Étant donné un arbre à données t et une donnée d apparaissant dans t, on définit la classe de d dans t comme la forêt construite de la manière suivante :*

– *on supprime tous les noeuds de t dont la valeur de donnée est différente de d,*
– *les noeuds non supprimés pour lequels il existe un préfixe strict n'ayant pas été supprimé sont rattachés au plus grand de ces préfixes,*
– *les noeuds non supprimés dont tous les préfixes ont été supprimés deviennent les racines des arbre de la forêt.*

Un exemple de classe est donnée en figure 5.

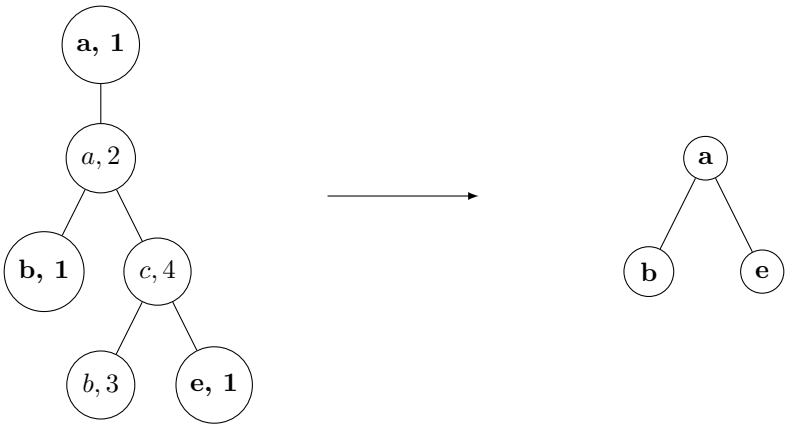


FIGURE 1 – Exemple : classe de 1

Une exécution d'un automate à données sur un arbre t s'effectue en deux étapes. La première est un passage de t par A, qui donne un nouvel arbre t' ayant la même structure que t mais avec des étiquettes différentes. On applique ensuite B sur chacune des classes de t' et on accepte ssi B accepte toutes les classes.

Nous définissons maintenant un modèle un peu plus puissant que les automates à données :

Définition 15 (Classes marquées d'un arbre à données)**.** *Une classe marquée d'un arbre à données se construit de la même façon qu'un classe standard, excepté que les étiquettes sont remplacées par des triplets définis de la manière suivante : soit t un arbre à données, $n \in Dom(t)$ un noeud de t étiquetté par x et ayant la valeur de donnée d. n est remplacé par le triplet (x, s, p) dans la classe de d où :*
– *x est l'étiquette de n dans t,*
– *s vaut 1 si le frère droit de n dans t à pour valeur de donnée d et 0 sinon,*
– *p vaut 1 si le père de n dans t à pour valeur de donnée d et 0 sinon.*

Pour les noeuds n'ayant pas de frère droit, s vaut 1. Pour la racine p vaut 1.

Un exemple est donné en figure 5.

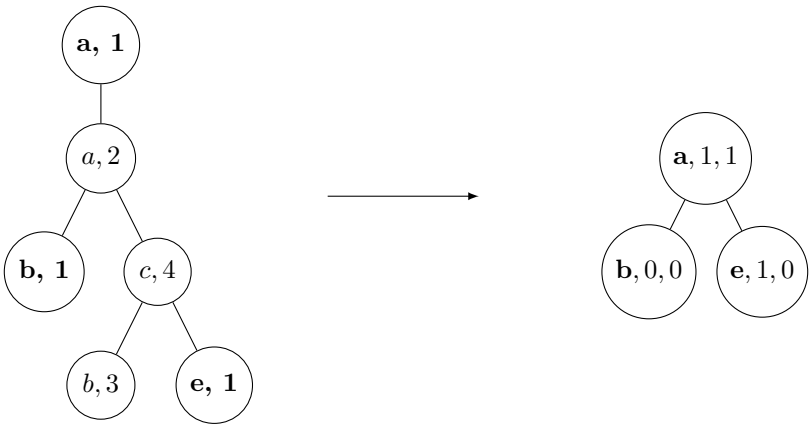


FIGURE 2 – Exemple : classe marquée de 1

Définition 16 (DTA^m)**.** *Un automate à données marqué sur les arbres de rang non bornés est un couple $\langle A, B\rangle$ où A est un transducteur d'arbres lettre à lettre et B un automate d'arbres (sans registre) opérant sur des classes marquées.*

Une exécution d'un automate à données marqué sur un arbre t s'effectue en deux étapes comme pour les automates standards, excepté que B est appliqué sur les classes marquées de l'abre t' obtenue en sortie de A. L'automate accepte t si B accepte toutes les classes marquées de t'.

Cepandant ces automates ne sont pas assez puissant pour simuler les automates à registres :

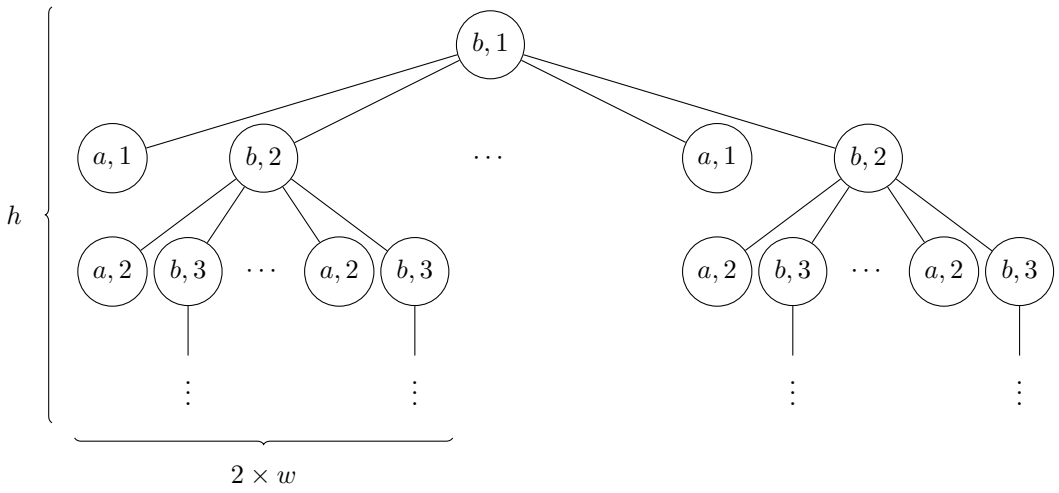
Proposition 9. *Les $URTAs$ ne sont pas inclus dans les DTA^m.*

Démonstration. On considère le langage L des arbres pour lesquels tous les noeuds étiquetés par a d'une même fratrie ont la même valeur de donnée. L se définit par la formule suivante :

$$\forall x.\forall y.[a(x) \wedge a(y) \wedge E_{\Rightarrow}(x, y)] \Rightarrow x \sim y$$

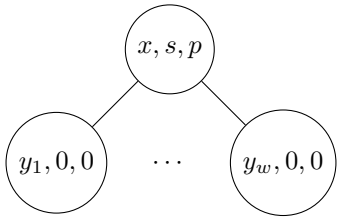
L est trivialement reconnaissable par un automate à registres. On montre maintenant que L n'est pas reconnaissable par un automate à données marqué.

Supposons que L soit reconnut par un tel automate $\mathcal{A} = \langle A, B\rangle$. On considère l'arbre $t_{w,h}$ suivant, où w et h sont des entiers naturels non nuls :



Nous allons maintenant modifier $t_{w,h}$ de telle sorte à obtenir un arbre qui n'appartient pas à L mais qui est reconnu par $\mathcal{A}$.

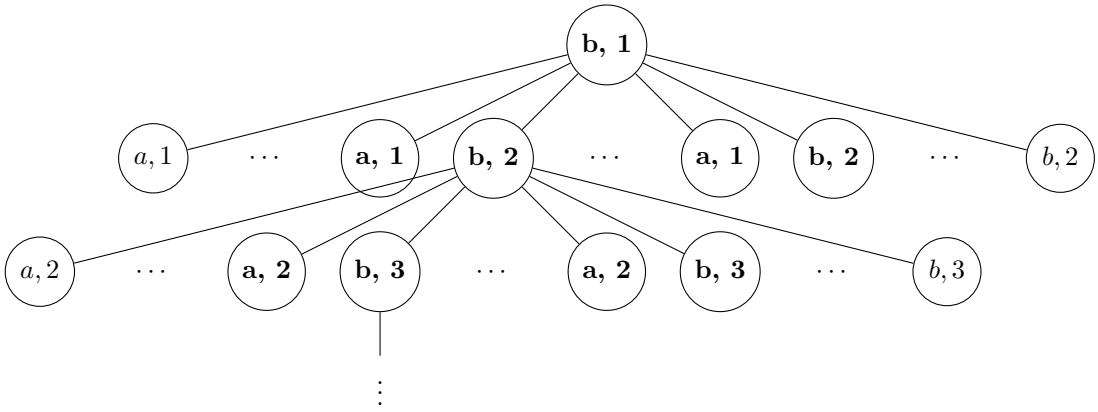
Soit $d \geq 1$ une donnée. La classe pour d de l'arbre obtenu en sortie de A est de la forme :



Où s et p valent 0 ou 1 suivant la position du b correspondant dans l'arbre.

On considère maintenant l'état q_{tr} dans lequel se trouve le transducteur A lorsqu'il lit un a, et l'état q_{cl} de l'automate de classe B lorsqu'il lit la lettre y_i correspondante. En prenant w suffisamment grand, on peut trouver sur chaque fratrie deux paires (q_{tr}, q_{cl}) égales pour deux positions différentes.

On construit maintenant le chemin suivant dans $t_{w,h}$: on part de la racine, et pour chaque fraterie on choisit une paire $(q_{tr,n}, q_{cl,n})$ qui se répète pour deux positions différentes étiquetées par a ; on continue ensuite en prenant le b se trouvant juste après la première de ces deux positions :



En prenant h suffisamment grand on peut trouver deux niveaux i et j avec $i < j$ tels que $(q_{tr,i}, q_{cl,i}) = (q_{tr,j}, q_{cl,j})$. On remplace alors dans $t_{w,h}$ au niveau j la section correspondante par celle du niveau i, mais en remplacent les noeuds étiquetés par b par les mêmes noeuds que ceux du niveau j. On note t' l'arbre obtenue. On peut alors vérifier que le transducteur accepte t', et que B accepte toutes les classes de l'arbre obtenue en sortie de A. □

Corollaire 1. *Les $URTAs$ ne sont pas inclus dans les DTA.*

Nous introduisont maintenant une nouvelle variante des automates à données où les classes conservent la structure originale de l'arbre :

Définition 17 (Classes avec dièses)**.** *Une classe avec dièses pour la valeur de donnée d d'un arbre t se définit de la manière suivante :*

- *tous les noeuds ayant pour valeur de donnée d conserve leur étiquette,*
- *tous les autres sont remplacé par $\sharp$, où $\sharp$ est une nouvelle étiquette.*

Un exemple est donné en figure 5.

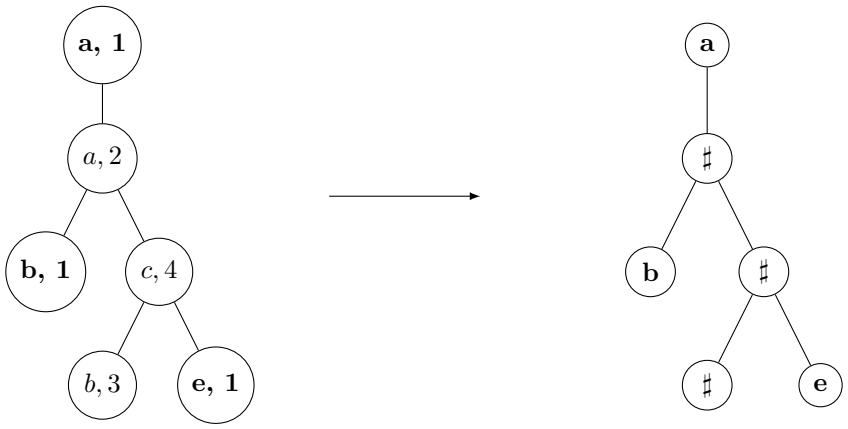


FIGURE 3 – Exemple : classe avec dièses de 1

Définition 18 ($DTA^\sharp$)**.** *Un automate à données avec dièses sur les arbres de rang non bornés est un couple $\langle A, B\rangle$ où A est un transducteur d'arbres lettre à lettre et B un automate d'arbres (sans registre) opérant sur des classes avec dièses.*

Ce modèle d'automates est suffisament puissant pour reconnaître les langages d'arbres reconnus par des formules de $FO_2(+1, <, \sim)$:

Proposition 10. *FO_2 est inclu dans $DTA^\sharp$*

Cependant ce modèle n'est pas décidable :

Proposition 11. *$DTA^\sharp$ est indécidable.*

Pour démontrer cela, on introduit le modèle d'automate sur les mots avec données suivant :

Définition 19 (CA)**.** *Un automate de classes sur les mots est une paire $\langle A, B\rangle$ où A est un transducteur lettre-à-lettre et B un automate finit qui s'applique sur les mots du langage $\Gamma \times \{0, 1\}$ où Γ est le langage de sortie de A.*

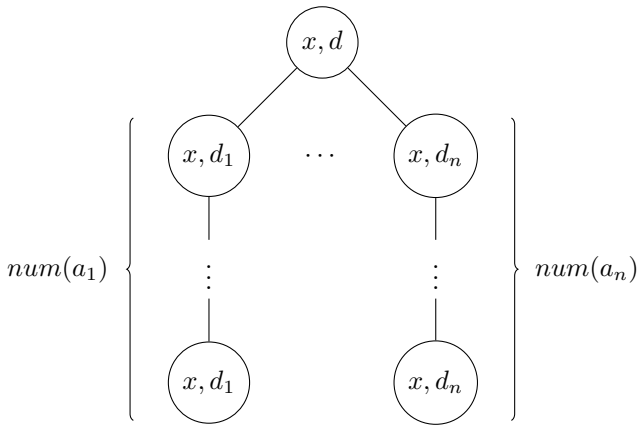
Une exécution d'un automate de classes $\langle A, B\rangle$ sur un mot avec données w consiste à exécuter A sur w (sans les données), puis à tester chacune des classes du mot obtenue en sortie de A avec B. Une classe d'un mot $w = (a_1, d_1)(a_2, d_2)\dots(a_n, d_n)$ pour une donnée d étant le mot $(a_1, b_1)(a_2, b_2)\dots(a_n, b_n)$, où :

- $b_i = 0$ si $d_i \neq d$
- $b_i = 1$ si $d_i = d$

Théorème 1. *[BL10] Le vide des automates de classes sur les mots (CA) est indécidable.*

Preuve de la proposition 11. Pour cela on montre que l'on peut encoder un automate de classes sur les mots en un automate à données avec dièses sur les arbres.

Soit $\langle A_w, B_w \rangle$ un automate de classes sur les mots. On associe de manière unique un nombre entier non nulle à chaque lettre de l'alphabet : $num : \Sigma \to \mathbb{N} \setminus \{0\}$. À chaque mot $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$ on associe l'abre suivant, où x est un symbole quelconque de l'aphabet de sortie de A_w et d une donnée quelconque :



On définit l'automate à données avec dièses sur les arbres $\langle A_t, B_t \rangle$ de la manière suivante : le transducteur A_t ne faire que recopier l'entrée sur la sortie. B_t reconstruit les lettres du mot d'entré en comptant le nomber de x ou de $\sharp$, puis applique B_w sur le résultat.

Un mot w est accepté par $\langle A_w, B_w \rangle$ ssi son codage est accepté par $\langle A_t, B_t \rangle$. Le vide des automates de classes étant indécidable, celui des $DTA^\sharp$ l'est aussi. □

6 Conclusion

Nous avons ainsi défini et étudié plusieurs modèles d'automates opérant sur des arbres de rang non borné à données, et obtenu des résultats d'indédicabilités et d'inclusion entre les différents modèles. La prochaine étape consisterais à trouver le bon modèle d'automate à données.

Références

[BL10] Mikołaj Bojańczyk and Sławomir Lasota. An extension of data automata that captures xpath. In *Proc. LICS'10*, 2010.

[BS10] Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5) :702–715, 2010.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2) :329–363, 1994.