Automates d'arbres

http://list.mpri.master.univ-paris7.fr/wws/info/cours-1-18

cours-1-18@mpri.master.univ-paris7.fr

Cours: Florent Jacquemard
TD: Ștefan Ciobâcă

INRIA Saclay & LSV (CNRS-ENS Cachan)

florent.jacquemard@inria.fr
ciobaca@lsv.ens-cachan.fr

1er octobre 2009

Bibliographie

- **TATA book** (Tree Automata Theory and Application)
  Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, Marc Tommasi
  `http://tata.gforge.inria.fr`
  chapitres 1, 2, 3, 7, 8

Arbres

- arbres finis d'arité fixée (termes en logique du premier ordre)
- arbres finis avec d'arité variable, ordonnés
- arbres finis avec arité variable, non ordonnés
- arbres infinis...

$\Rightarrow$ plusieurs classes d'automates d'arbres.

Propriétés

- déterminisme,
- clôture Booléennes,
- clôture par transformations
  (homomorphismes, transducteurs, systèmes de réécriture...)
- minimisation,
- problèmes de décision, complexité,
  - appartenance,
  - vide,
  - universalité,
  - inclusion, équivalence,
  - vide de l'intersection,
  - finitude...
- pompage et itération,
- expressivité, correspondance avec des logiques.

Plan

1. automates d'arbres finis (arité fixe)
2. correspondance avec logiques monadiques du second ordre (théorème de Thatcher et Wright).
3. automates d'arbres d'arité variable et ordonnés
   = Hedge automata, XML.
4. automates arbres d'arité variable non-ordonnés
   = automates de Presburger, XML et comptage.

- automates alternants et bidirectionnels (mots et arbres)
- représentation d'automates par clauses de Horn
- automates de mots infinis (conditions de Müller et Büchi), automates d'arbres infinis (automates de Rabin).

Motivations

Motivation : vérification

Vérification de systèmes infinis. **Regular model checking.**

- configuration / état = arbre
  - processus,
  - message échangé dans un protocole,
  - réseau local de forme arborescente,
  - structure d'arbre en mémoire avec pointeurs (par ex. arbres binaires de recherche).
  - ...
- ensemble de configurations = langage d'arbres L
- relation de transition entre configuration (post)
- sûreté : $\text{post}^*(L_{\text{init}}) \cap L_{\text{error}} = \emptyset$.

Motivation : déduction automatique

arbres $=$ termes en logique du premier ordre.

Automatisation de la récurrence et réductibilité inductive

Exemple :

axiomes : $0 + x = x$, $s(x) + y = s(x + y)$.
$x + 0 = x$ est un théorème inductif : valide dans le modèle initial

- domaine du modèle initial : $0, s(0), s(s(0)), \ldots$
- $=$ termes irréductibles (formes normales)
- termes engendré par la grammaire d'arbres $\mathsf{N} := 0 \mid s(\mathsf{N})$.

$x + y$ est inductivement réductible.

Motivation : déduction automatique

Exemple :

axiomes : $p(s(x)) = x, \quad s(p(x)) = x$
$0 + x = x, \quad s(x) + y = s(x + y), \quad p(x) + y = p(x + y).$
$x + 0 = x$ est aussi un théorème inductif.

Formes normales = Domaine du modèle initial, engendrés par :

$$\begin{array}{lcl} \mathsf{N}_0 & := & 0, \\ \mathsf{N}_s & := & s(\mathsf{N}_0) \mid s(\mathsf{N}_s), \\ \mathsf{N}_p & := & s(\mathsf{N}_0) \mid s(\mathsf{N}_p). \end{array}$$

Motivation : stratégies d'évaluation

évaluation dans les langages fonctionnels

Stratégies de réduction par réécriture

- termes en forme normale (irréductibles)
- termes normalisables
- existence de redex nécessaire

Exemple :

$\vee(\top, x_2) = \top$, $\vee(x_1, \top) = \top$: pas de redex nécessaire.
$\exists x_1, \forall x_2, \vee(x_1, x_2) = \top$ (évaluation de x_2 pas nécessaire)
$\exists x_2, \forall x_1, \vee(x_1, x_2) = \top$ (évaluation de x_1 pas nécessaire)

test : existence d'un $\vee(x_1, \Omega)$ et d'un $\vee(\Omega, x_2)$ s'évaluant à $\top$.
par construction d'automate d'arbres et décision du vide.

Logique

- automates = outils sémantiques pour décider des logiques, (par ex. logiques monadiques du second ordre).
- caractérisation de modèles par automates.
  = "compilation" formule $\longrightarrow$ automate
- décision de la satisfiabilité
  $\equiv$ décision du vide de l'automate

Documents XML

= arbres d'arité variable étiquetés

Typage (documents conformes) par définition de **schémas** (DTD, XML Schema, Relax NG...).

Tous les formalismes de schéma en usage actuellement sont équivalents aux automates d'arbres.

Arbres finis orientés ordonnés étiquettés par des symboles de fonction d'arité déterminée.

= termes (en logique du premier ordre)

Signature

Définition : Signature

Une signature Σ est un ensemble fini de symboles de fonctions avec arité supérieure ou égale à 0.

On note Σ_i l'ensemble des symboles d'arité i.

Exemple :

$\{+ : 2, s : 1, 0 : 0\}$, $\{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$.

On considère aussi un ensemble dénombrable $\mathcal{X}$ de symboles de variables.

Termes

Définition : Terme

L'ensemble des termes sur la signature Σ et $\mathcal{X}$ est le plus petit ensemble $\mathcal{T}(\Sigma, \mathcal{X})$ tel que :

- $\Sigma_0 \subseteq \mathcal{T}(\Sigma, \mathcal{X})$,
- $\mathcal{X} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$,
- si $f \in \Sigma_n$ et si $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$, alors $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})$.

L'ensemble des termes clos (sans variables, i.e. $\mathcal{T}(\Sigma, \emptyset)$) est noté $\mathcal{T}(\Sigma)$.

Exemple :

x, $\neg(x)$, $\wedge\big(\vee(x, \neg(y)), \neg(x)\big)$.

Termes (2)

Un terme où chaque variable apparaît au plus une fois est appelé *linéaire*. Un terme sans variable est appelé *clos*.

Hauteur $\mathsf{h}(t)$:

- $\mathsf{h}(a) = \mathsf{h}(x) = 0$ si $a \in \Sigma_0$, $x \in \mathcal{X}$,
- $\mathsf{h}\big(f(t_1, \ldots, t_n)\big) = \max\{\mathsf{h}(t_1), \ldots, \mathsf{h}(t_n)\} + 1$.

Positions

Un terme $t \in \mathcal{T}(\Sigma, \mathcal{X})$ peut aussi être vu comme une fonction de l'ensemble de ses positions $\mathcal{P}os(t)$ vers $\Sigma \cup \mathcal{X}$.
Position vide (racine) notée ε.

$\mathcal{P}os(t)$ est un sous ensemble de $\mathbb{N}^*$ qui satisfait les propriétés suivantes :

- $\mathcal{P}os(t)$ est clos par préfixe,
- pour tout $p \in \mathcal{P}os(t)$ tel que $t(p) \in \Sigma_n$ $(n \geq 1)$, $\{pj \in \mathcal{P}os(t) \mid j \in \mathbb{N}\} = \{p1, ..., pn\}$,
- tout $p \in \mathcal{P}os(t)$ tel que $t(p) \in \Sigma \cup \mathcal{X}$ est maximal dans $\mathcal{P}os(t)$ pour l'ordre préfixe.

La taille de t est définie par $\|t\| = |\mathcal{P}os(t)|$.

Sous-terme $t|_p$ à la position $p \in \mathcal{P}os(t)$: $t|_\varepsilon = t$,
$f(t_1, \ldots, t_n)|_{ip} = t_i|_p$.

Le remplacement dans t de $t|_p$ par s est noté $t[s]_p$.

Positions (exemple)

Exemple :

$t = \wedge(\wedge(x, \vee(x, \neg(y))), \neg(x))$,
$t|_{11} = x$, $t|_{12} = \vee(x, \neg(y))$, $t|_2 = \neg(x)$,
$t[\neg(y)]_{11} = \wedge(\wedge(\neg(y), \vee(x, \neg(y))), \neg(x))$.

Substitutions, Contextes

Définition : Substitution

Une *substitution* est une fonction de $\mathcal{X}$ vers $\mathcal{T}(\Sigma, \mathcal{X})$ de domaine fini.
On étend la définition à $\mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\Sigma, \mathcal{X})$ par :

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma) \quad (n \geq 0)$$

Définition : Contexte

Un *contexte* est un terme linéaire.

L'application d'un contexte $C \in \mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$ à n termes $t_1, \ldots, t_n$, notée $C[t_1, \ldots, t_n]$ est $C\sigma$ avec $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Réécriture

Un *système de réécriture* $\mathcal{R}$ est un ensemble fini de règles de réécritures de la forme $\ell \to r$ avec $\ell, r \in \mathcal{T}(\Sigma, \mathcal{X})$.

La relation $\xrightarrow[\mathcal{R}]{}$ est la plus petite relation binaire contenant $\mathcal{R}$, et fermée par application de contextes et substitutions.
i.e. $s \xrightarrow[\mathcal{R}]{} t$ ssi $\exists p \in \mathcal{P}os(s), \ell \to r \in \mathcal{R}, \sigma,\ s|_p = \ell\sigma$ et $t = s[r\sigma]_p$.

On note $\xrightarrow[\mathcal{R}]{*}$ la clôture réflexive et transitive de $\xrightarrow[\mathcal{R}]{}$.

Exemple :

$\mathcal{R} = \{+(0, x) \to x, +(s(x), y) \to s(+(x, y))\}$.

$$
\begin{aligned}
+\big(s(s(0)), +(0, s(0))\big) &\xrightarrow[\mathcal{R}]{} +\big(s(s(0)), s(0)\big) \\
&\xrightarrow[\mathcal{R}]{} s\big(+(s(0), s(0))\big) \\
&\xrightarrow[\mathcal{R}]{} s\big(s\big(+(0, s(0))\big)\big) \\
&\xrightarrow[\mathcal{R}]{} s(s(s(0)))
\end{aligned}
$$

Automates d'arbres finis ascendants

Définition : Automates d'arbres

Un *automate d'arbres* (TA) sur une signature Σ est un tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ où Q est un ensemble fini d'*états*, $Q^{\mathsf{f}} \subseteq Q$ est le sous-ensemble des états finaux et Δ est un ensemble de règles de transition de la forme : $f(q_1, \ldots, q_n) \to q$ avec $f \in \Sigma_n$ $(n \geq 0)$ et $q_1, \ldots, q_n, q \in Q$.

L'état q est appelé la tête de la règle.

On dit qu'un terme clos $t \in \mathcal{T}(\Sigma)$ est accepté par $\mathcal{A}$ dans l'état q ssi $t \xrightarrow[\Delta]{*} q$.

Le langage de $\mathcal{A}$ dans l'état q est $L(\mathcal{A}, q) := \left\{ t \in \mathcal{T}(\Sigma) \;\middle|\; t \xrightarrow[\Delta]{*} q \right\}$.

Le langage de $\mathcal{A}$ est $L(\mathcal{A}) := \bigcup_{q^{\mathsf{f}} \in Q^{\mathsf{f}}} L(\mathcal{A}, q^{\mathsf{f}})$ (langage régulier).

Automates d'arbres : exemple 1

Exemple :

$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$,

$$\mathcal{A} = \left(\Sigma, \{q_0, q_1\}, \{q_1\}, \left\{ \begin{array}{rclrcl} \bot & \to & q_0 & \top & \to & q_1 \\ \neg(q_0) & \to & q_1 & \neg(q_1) & \to & q_0 \\ \vee(q_0, q_0) & \to & q_0 & \vee(q_0, q_1) & \to & q_1 \\ \vee(q_1, q_0) & \to & q_1 & \vee(q_1, q_1) & \to & q_1 \\ \wedge(q_0, q_0) & \to & q_0 & \wedge(q_0, q_1) & \to & q_0 \\ \wedge(q_1, q_0) & \to & q_0 & \wedge(q_1, q_1) & \to & q_1 \end{array} \right\} \right)$$

$$\begin{array}{ll} & \wedge(\wedge(\top, \vee(\top, \neg(\bot))), \neg(\top)) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(\top, \vee(\top, \neg(\bot))), \neg(q_1)) \\ \xrightarrow[\mathcal{A}]{} & \wedge(\wedge(q_1, \vee(q_1, \neg(q_0))), \neg(q_1)) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(q_1, \vee(q_1, \neg(q_0))), q_0) \\ \xrightarrow[\mathcal{A}]{} & \wedge(\wedge(q_1, \vee(q_1, q_1)), q_0) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(q_1, q_1), q_0) \xrightarrow[\mathcal{A}]{} \wedge(q_1, q_0) \xrightarrow[\mathcal{A}]{} q_0 \end{array}$$

Automates d'arbres : exemple 2

Exemple :

$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$,
TA reconnaissant les instances closes de $\neg(\neg(x))$:

$$\mathcal{A} = \left(\Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \left\{ \begin{array}{rcl@{\qquad}rcl} \bot & \to & q & \top & \to & q \\ \neg(q) & \to & q & \neg(q) & \to & q_\neg \\ \neg(q_\neg) & \to & q_\mathsf{f} & & & \\ \vee(q,q) & \to & q & \wedge(q,q) & \to & q \end{array} \right\} \right)$$

Proposition :

Étant donné un terme linéaire $t \in \mathcal{T}(\Sigma, \mathcal{X})$, il existe un TA $\mathcal{A}$ reconnaissant l'ensemble des instances closes de t : $L(\mathcal{A}) = \left\{ t\sigma \;\middle|\; \sigma : \mathcal{X} \to \mathcal{T}(\Sigma) \right\}$.

Exemple :

Termes clos contenant le motif $\neg(\neg(x))$: $\mathcal{A} \cup \{\neg(q_\mathsf{f}) \to q_\mathsf{f}, \vee(q_\mathsf{f}, q_*) \to q_\mathsf{f}, \vee(q_*, q_\mathsf{f}) \to q_\mathsf{f}, \ldots\}$ (propagation q_f).

Langages réguliers

Autre définition (récursive) de $L(\mathcal{A}, q)$:

$$\begin{array}{rcl} L(\mathcal{A}, q) & = & \left\{ a \in \Sigma_0 \;\middle|\; a \to q \in \Delta \right\} \\ & \cup & \displaystyle\bigcup_{f(q_1,\ldots,q_n)\to q \in \Delta} f\big(L(\mathcal{A}, q_1), \ldots, L(\mathcal{A}, q_n)\big) \end{array}$$

avec $f(L_1, \ldots, L_n) := \left\{ f(t_1, \ldots, t_n) \;\middle|\; t_1 \in L_1, \ldots, t_n \in L_n \right\}$.

Calculs (runs)

Définition : Calcul

Un *calcul* d'un TA $(\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ sur un terme $t \in \mathcal{T}(\Sigma)$ est une fonction $r : \mathcal{P}os(t) \to Q$ pour tout $p \in \mathcal{P}os(t)$, si $t(p) = f \in \Sigma_n$, $r(p) = q$ et $r(pi) = q_i$ pour tout $1 \leq i \leq n$, alors $f(q_1, \ldots, q_n) \to q \in \Delta$.

Calcul r *acceptant* si $r(\varepsilon) \in Q^{\mathsf{f}}$.
$L(\mathcal{A})$ est l'ensemble des termes de $\mathcal{T}(\Sigma)$ pour lesquels il existe un calcul acceptant.

Lemme de pompage

Lemme : Lemme de pompage

Soit $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$.
$L(\mathcal{A}) \neq \emptyset$ ssi il existe $t \in L(\mathcal{A})$ tel que $h(t) \leq |Q|$.

Lemme : Lemme d'itération

Pour tout TA $\mathcal{A}$, il existe $k > 0$ tel que pour tout terme $t \in L(\mathcal{A})$ avec $h(t) > k$, il existe deux contextes $C, D \in \mathcal{T}(\Sigma, \{x_1\})$ avec $D \neq x_1$ et un terme $u \in \mathcal{T}(\Sigma)$ tels que $t = C\big[D[u]\big]$ et pour tout $n \geq 0$, $C\big[D^n[u]\big] \in L(\mathcal{A})$.

usage : pour montrer qu'un langage n'est pas régulier.

Langages non réguliers

On montre avec les lemmes de pompage ou d'itération que les langages d'arbres suivants ne sont pas réguliers :

- $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$,
- $\{f(g^n(a), h^n(a)) \mid n \geq 0\}$,
- $\{t \in \mathcal{T}(\Sigma) \mid |\mathcal{P}os(t)| \text{ est premier}\}$.

Epsilon-transitions

On étend les TA en TAε avec l'ajout d'un autre type de règles de transition $q \xrightarrow{\varepsilon} q'$.
Sans augmentation d'expressivité.

Proposition : Suppression des ε-transitions

Pour tout TAε $\mathcal{A}_\varepsilon$, il existe un TA (sans ε-transition) $\mathcal{A}'$ tel que $L(\mathcal{A}) = L(\mathcal{A}_\varepsilon)$. La taille de $\mathcal{A}$ est polynomiale en la taille de $\mathcal{A}_\varepsilon$.

pr.: On part de $\mathcal{A}_\varepsilon$ et on ajoute $f(q_1, \ldots, q_n) \to q'$ si il existe $f(q_1, \ldots, q_n) \to q$ et $q \xrightarrow{\varepsilon} q'$.

Automates d'arbres descendants

Définition : Automates d'arbres descendants

Un automate d'arbres descendant sur une signature Σ est un tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{init}}, \Delta)$ où Q est un ensemble fini d'*états*, $Q^{\mathsf{init}} \subseteq Q$ est le sous-ensemble des états initiaux et Δ est un ensemble de règles de transition de la forme : $q \to f(q_1, \ldots, q_n)$ avec $f \in \Sigma_n$ $(n \geq 0)$ et $q_1, \ldots, q_n, q \in Q$.

Un terme clos $t \in \mathcal{T}(\Sigma)$ est accepté par $\mathcal{A}$ dans l'état q ssi $q \xrightarrow[\Delta]{*} t$.

Le langage de $\mathcal{A}$ depuis l'état q est
$L(\mathcal{A}, q) := \left\{ t \in \mathcal{T}(\Sigma) \;\middle|\; q \xrightarrow[\Delta]{*} t \right\}$.

Le langage de $\mathcal{A}$ est $L(\mathcal{A}) := \bigcup_{q^{\mathsf{i}} \in Q^{\mathsf{init}}} L(Q, q^{\mathsf{i}})$.

Automates d'arbres descendants (expressivité)

Proposition : Expressivité

L'ensemble des langages automates d'arbres descendants est exactement l'ensemble des langages d'arbres réguliers.

Déterminisme

Définition : Déterministe

Un TA $\mathcal{A}$ est *déterministe* si pour tout $f \in \Sigma_n$, pour tous états $q_1, \ldots, q_n$ de $\mathcal{A}$, il y a au plus un état q de $\mathcal{A}$ tel que $\mathcal{A}$ contient une transition $f(q_1, \ldots, q_n) \to q$.

Si $\mathcal{A}$ est déterministe, alors pour tout $t \in \mathcal{T}(\Sigma)$, il y a au plus un état q de $\mathcal{A}$ tel que $t \in L(\mathcal{A}, q)$. Il est noté $\mathcal{A}(t)$ ou $\Delta(t)$.

Définition : Complétude

Un TA $\mathcal{A}$ est dit *complet* si pour tout $f \in \Sigma_n$, pour tous états $q_1, \ldots, q_n$ de $\mathcal{A}$, il y a au moins un état q de $\mathcal{A}$ tel que $\mathcal{A}$ contient une transition $f(q_1, \ldots, q_n) \to q$.

Si $\mathcal{A}$ est complet, alors pour tout $t \in \mathcal{T}(\Sigma)$, il y a au moins un état q de $\mathcal{A}$ tel que $t \in L(\mathcal{A}, q)$.

Complétion

Proposition : Complétion

Pour tout TA $\mathcal{A}$, il existe un TA complet $\mathcal{A}_c$ tel que $L(\mathcal{A}_c) = L(\mathcal{A})$ et de plus, si $\mathcal{A}$ est déterministe, alors $\mathcal{A}_c$ est déterministe. La taille de $\mathcal{A}_c$ est polynomiale en la taille de $\mathcal{A}$, sa construction est PTIME.

pr.: ajout d'un état poubelle $q_\perp$.

Déterminisation

Proposition : Déterminisation

Pour tout TA $\mathcal{A}$, il existe un TA déterministe $\mathcal{A}_{det}$ tel que $L(\mathcal{A}_{det}) = L(\mathcal{A})$ et de plus, si $\mathcal{A}$ est complet, alors $\mathcal{A}_{det}$ est complet. La taille de $\mathcal{A}_{det}$ est exponentielle en la taille de $\mathcal{A}$, sa construction est EXPTIME.

pr.: construction sous-ensemble.

Exercice :

Déterminiser et compléter le TA précédent (pattern matching de $\neg(\neg(x))$) :

$$\mathcal{A} = \left(\Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \left\{ \begin{array}{rclrcl} \bot & \to & q & \top & \to & q \\ \neg(q) & \to & q & \neg(q) & \to & q_\neg \\ \neg(q_\neg) & \to & q_\mathsf{f} & \neg(q_\mathsf{f}) & \to & q_\mathsf{f} \\ \vee(q,q) & \to & q & \wedge(q,q) & \to & q \\ \vee(q_\mathsf{f}, q_*) & \to & q_\mathsf{f} & \vee(q_*, q_\mathsf{f}) & \to & q_\mathsf{f} \end{array} \right\} \right)$$

Automates d'arbres descendants : déterminisme

Définition : Déterministe

Un automate d'arbres descendant $(\Sigma, Q, Q^{\text{init}}, \Delta)$ est *déterministe* si $|Q^{\text{init}}| = 1$ et pour tout état $q \in Q$ et $f \in \Sigma$, Δ contient au plus une règle de membre gauche q et symbole f.

Les automates descendants ne sont en général pas déterminisables.

Proposition :

Il existe un langage d'arbres régulier qui n'est pas reconnaissable par un automate d'arbres descendant déterministe.

Automates d'arbres descendants : déterminisme

Définition : Déterministe

Un automate d'arbres descendant $(\Sigma, Q, Q^{\text{init}}, \Delta)$ est *déterministe* si $|Q^{\text{init}}| = 1$ et pour tout état $q \in Q$ et $f \in \Sigma$, Δ contient au plus une règle de membre gauche q et symbole f.

Les automates descendants ne sont en général pas déterminisables.

Proposition :

Il existe un langage d'arbres régulier qui n'est pas reconnaissable par un automate d'arbres descendant déterministe.

pr.: $L = \{f(a,b), f(b,a)\}$.

Clôture Booléenne des réguliers

Proposition : Clôture

La classe des langages d'arbres réguliers est close par union, intersection et complément.

op.	technique	temps de calcul et taille des automates
$\cup$	$\cup$ disjointe	linéaire
$\cap$	produit Cartésien	quadratique
$\neg$	déterminisation, complétion, inversion état finaux / non-finaux	exponentiel (borne inférieure)

Remarque :

Pour les TA déterministes, la construction pour le complément est polynomiale.

Nettoyage

Définition : Nettoyé

Un état q d'un TA $\mathcal{A}$ est dit *habité* si il existe au moins un $t \in L(\mathcal{A}, q)$. Un TA est dit *nettoyé* si tous ses états sont habités.

Proposition : Nettoyage

Pour tout TA $\mathcal{A}$, il existe un TA nettoyé $\mathcal{A}_{net}$ tel que $L(\mathcal{A}_{net}) = L(\mathcal{A})$. La taille de $\mathcal{A}_{net}$ est inférieure à la taille de $\mathcal{A}$, sa construction est PTIME.

pr.: algorithme de marquage d'états, en temps $O\big(|Q| \times \|\Delta\|\big)$.

Problème de l'appartenance

Définition : Appartenance

INPUT : un TA $\mathcal{A}$ sur Σ, un terme $t \in \mathcal{T}(\Sigma)$.
QUESTION : $t \in L(\mathcal{A})$?

Proposition : Appartenance

Le problème de l'appartenance est décidable en temps polynomial.

Complexité exacte :

- ascendants non-déterministes : LOGCFL-complet
- ascendants déterministes : inconnue (LOGDCFL)
- descendants déterministe : LOGSPACE-complete.

Problème du vide

Définition : Vide

INPUT : un TA $\mathcal{A}$ sur Σ.
QUESTION : $L(\mathcal{A}) = \emptyset$?

Proposition : Vide

Le problème du vide est décidable en temps linéaire.

Problème du vide

Définition : Vide

INPUT : un TA $\mathcal{A}$ sur Σ.
QUESTION : $L(\mathcal{A}) = \emptyset$?

Proposition : Vide

Le problème du vide est décidable en temps linéaire.

pr.:
quadratique : nettoyer, regarder si l'automate clean contient un état final.
linéaire : réduction à HORN-SAT propositionnel.
linéaire bis : optimisation des structures de données pour le nettoyage (exo).

Remarque :

Le problème du vide est PTIME-complet.

Problème du vide de l'intersection

Définition : Vide de l'intersection

INPUT : n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ sur Σ.
QUESTION : $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$?

Proposition : Vide de l'intersection

Le problème du vide de l'intersection est EXPTIME-complet.

Problème du vide de l'intersection

Définition : Vide de l'intersection

INPUT : n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ sur Σ.
QUESTION : $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$?

Proposition : Vide de l'intersection

Le problème du vide de l'intersection est EXPTIME-complet.

pr.: EXPTIME : Théoreme de clôture Booléenne et décision du vide.

EXPTIME-difficulté : APSPACE $=$ EXPTIME
réduction du problème de l'existence d'un calcul réussit (à partir d'une configuration initiale) d'une machine de Turing alternante (A.T.M.) $M = (\Gamma, S, s_i, S_\mathsf{f}, \delta)$.

Problème de l'appartenance d'une instance

Définition : Appartenance d'une instance (AI)

INPUT : un TA $\mathcal{A}$ sur Σ, un terme $t \in \mathcal{T}(\Sigma, \mathcal{X})$.
QUESTION : existe-t-il $\sigma : vars(t) \to \mathcal{T}(\Sigma)$ t.q. $t\sigma \in L(\mathcal{A})$?

Proposition : Appartenance d'une instance

1. Le problème AI est décidable en temps polynomial quand t est linéaire.
2. Le problème AI est NP-complet quand $\mathcal{A}$ est déterministe.
3. Le problème AI est EXPTIME-complet en général.

Problème de l'universalité

Définition : Universalité

INPUT : un TA $\mathcal{A}$ sur Σ.
QUESTION : $L(\mathcal{A}) = \mathcal{T}(\Sigma)$

Proposition : Universalité

Le problème de l'universalité est EXPTIME-complet.

Problème de l'universalité

Définition : Universalité

INPUT : un TA $\mathcal{A}$ sur Σ.
QUESTION : $L(\mathcal{A}) = \mathcal{T}(\Sigma)$

Proposition : Universalité

Le problème de l'universalité est EXPTIME-complet.

pr.: EXPTIME : clôture Booléenne et décision du vide.

EXPTIME-difficulté : encore APSPACE = EXPTIME.

Remarque :

Le problème de l'universalité est décidable en temps polynomial pour les TA ascendants déterministes.

pr.: complétion et nettoyage.

Problèmes de l'inclusion, de l'équivalence

Définition : Inclusion

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Définition : Équivalence

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Équivalence

Les problèmes de l'inclusion et de l'équivalence sont EXPTIME-complet.

Problèmes de l'inclusion, de l'équivalence

Définition : Inclusion

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Définition : Équivalence

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Équivalence

Les problèmes de l'inclusion et de l'équivalence sont EXPTIME-complet.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ ssi $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)} = \emptyset$.

Problèmes de l'inclusion, de l'équivalence

Définition : Inclusion

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Définition : Équivalence

INPUT : deux TA $\mathcal{A}_1$ et $\mathcal{A}_2$ sur Σ.
QUESTION : $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Équivalence

Les problèmes de l'inclusion et de l'équivalence sont EXPTIME-complet.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ ssi $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)} = \emptyset$.
EXPTIME-hardness : l'universalité est $\mathcal{T}(\Sigma) = L(\mathcal{A}_2)$?

Remarque :

Si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont déterministes, c'est $O\big(\|\mathcal{A}_1\| \times \|\mathcal{A}_2\|\big)$.

Problèmes de la finitude

Définition : Finitude

INPUT : un TA $\mathcal{A}$
QUESTION : est-ce que $L(\mathcal{A})$ est fini ?

Proposition : Appartenance

Le problème de la finitude est décidable en temps polynomial.

Théorème de Myhill-Nerode

Définition :

Une *congruence* $\equiv$ sur $\mathcal{T}(\Sigma)$ est une relation d'équivalence telle que pour tout $f \in \Sigma_n$, si $s_1 \equiv t_1, \ldots, s_n \equiv t_n$, alors $f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n)$.

Étant donné $L \subseteq \mathcal{T}(\Sigma)$, la congruence $\equiv_L$ est définie par : $s \equiv_L t$ si pour tout contexte $C \in \mathcal{T}\big(\Sigma, \{x\}\big)$, $C[s] \in L$ ssi $C[t] \in L$.

Théorème : Myhill-Nerode

Les trois propositions suivantes sont équivalentes :

1. L est régulier
2. L est une union de classes d'équivalence pour une congruence $\equiv$ d'index fini
3. $\equiv_L$ est une congruence d'index fini

Minimisation

Corollaire :

Pour tout DTA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$, il existe un unique DTA $\mathcal{A}_{\mathsf{min}}$ dont le nombre d'états est l'index de $\equiv_{L(\mathcal{A})}$ et tel que $L(\mathcal{A}_{\mathsf{min}}) = L(\mathcal{A})$.

Caractérisation algébrique des réguliers

Théorème :

Un ensemble $L \subseteq \mathcal{T}(\Sigma)$ est régulier ssi il existe
une Σ-algèbre $\mathcal{Q}$ de domaine fini Q,
un homomorphisme $h : \mathcal{T}(\Sigma) \to \mathcal{A}$ et
un sous-ensemble $Q' \subseteq Q$ tels que $L = h^{-1}(Q')$.

Homomorphismes d'arbres

- formalismes de **transformation** de termes (langages) : systèmes de réécriture, homomorphisme d'arbres...
  - $=$ transitions dans un système d'états infinis,
  - $=$ évaluation de programmes,
  - $=$ transformation de documents,..
- problème de la **vérification de type** :
  - étant donné $L_{\text{in}} \subseteq \mathcal{T}(\Sigma)$ langage (régulier) d'entrée,
  - h transformation $\mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$,
  - et $L_{\text{out}} \subseteq \mathcal{T}(\Sigma')$ langage (régulier) de sortie,
  - a-t-on $h(L_{\text{in}}) \subseteq L_{\text{out}}$?

Homomorphismes d'arbres

Définition :

$$h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$$
$$h\big(f(t_1, \ldots, t_n)\big) := t_f\big\{x_1 \leftarrow h(t_1), \ldots, x_n \leftarrow h(t_n)\big\}$$
pour $f \in \Sigma_n$, avec $t_f \in \mathcal{T}\big(\Sigma', \{x_1, \ldots, x_n\}\big)$.

h est dit *linéaire* si pour tout $f \in \Sigma$, t_f est linéaire.

Exemple : arbres ternaires $\longrightarrow$ arbres binaires

Soit $\Sigma = \{a : 0, b : 0, g : 3\}$, $\Sigma' = \{a : 0, b : 0, f : 2\}$ et $h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$ défini par $t_a = a$, $t_b = b$, $t_g = f(x_1, f(x_2, x_3))$.
$h\big(g(a, g(b, b, b), a)\big) = f(a, f(f(f(b, f(b, b))), a))$

Exemple : élimination du $\wedge$

Soit $\Sigma = \{0 : 0, 1 : 0, \neg : 1, \vee : 2, \wedge : 2\}$, $\Sigma' = \{0 : 0, 1 : 0, \neg : 1, \vee : 2\}$ et $h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$ avec $t_\wedge = \neg(\vee(\neg(x_1), \neg(x_2)))$.

Clôture des réguliers par homomorphismes linéaires

Théorème :

Si L est régulier et h est un homomorphisme linéaire, alors $h(L)$ est régulier.

Ce n'est pas vrai pour les homomorphismes non-linéaires.

Clôture des réguliers par homomorphismes linéaires

Théorème :

Si L est régulier et h est un homomorphisme linéaire, alors $h(L)$ est régulier.

Ce n'est pas vrai pour les homomorphismes non-linéaires.

Exemple : Homomorphismes non linéaires

$\Sigma = \{a : 0, g : 1, f : 1\}$, $\Sigma' = \{a : 0, g : 1, f' : 2\}$, $h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$ avec $t_a = a$, $t_g = g(x_1)$, $t_f = f'(x_1, x_1)$.
Soit $L = \left\{ f\left(g^n(a)\right) \;\middle|\; n \geq 0 \right\}$, $h(L) = \left\{ f'\left(g^n(a), g^n(a)\right) \;\middle|\; n \geq 0 \right\}$ n'est pas régulier.

Clôture des réguliers par homomorphismes inverses

Théorème :

Pour tout langage L régulier et tout homomorphisme h, $h^{-1}(L)$ est régulier.

Clôture par homomorphismes

Théorème :

La classe des langages d'arbres régulier est la plus petite classe non triviale d'ensembles d'arbres close par homomorphismes linéaires et homomorphismes inverses.

Problème dont la question de la décidabilité a été ouverte pendant 35 ans :

INPUT : un TA $\mathcal{A}$, un homomorphisme h
QUESTION : est-ce que $h(L(\mathcal{A}))$ est régulier ?