Apprentissage d'un langage régulier avec L^*

Sebastien Bonduelle, Pablo Espana Gutierrez*

À rendre avant le vendredi 29 novembre à 13h16 UTC+01:00

Ce devoir porte sur une méthode pour apprendre automatiquement un langage régulier. Plus précisément, nous allons étudier et implémenter un algorithme capable d'apprendre efficacement un langage $\mathcal{L}$ en utilisant deux oracles simples donnant un minimum d'information sur ce langage : un premier oracle permettra de savoir si un mot donné est dans le langage ; un second permet de tester si un automate a pour langage $\mathcal{L}$, et d'obtenir un mot contre-exemple si ce n'est pas le cas.

On se donne un alphabet Σ fixé et connu de l'apprenant(e). On suppose fixé un langage régulier $\mathcal{L}$, inconnu de l'apprenant(e).

Pour l'implémentation, l'utilisateur jouera le rôle de l'enseignant (les oracles) et la machine exécutera l'algorithme L^* et jouera le rôle de l'apprenant. Il y a trois fichiers fournis :

— `automator.mli` (qui donne les types des fonctions et leurs spécifications), à ne pas modifier ;

— `automator.cmi` qui est le fichier `automator.mli` après compilation, à ne pas toucher ;

— `automator.ml` qui suit la spécification de l'interface `automator.mli` ; certaines fonctions sont déjà implémentées, d'autres seront à implémenter au long du sujet.

Pour exécuter le programme `automator.ml`, il faut le compiler (après chaque modification) avec la commande `ocamlc -o automator automator.ml` ce qui produit l'exécutable `automator`, que vous pouvez exécuter avec la commande `./automator`.

*D'après un sujet de David Baelde

1 Représentation des automates

Nous allons représenter certains automates par des paires (S, T) où S et T sont des ensembles finis de mots, avec S clos par préfixe [1] et T clos par suffixe. Deux mots $u, v \in \Sigma^*$ sont dits T-équivalents si, pour tout $w \in T$, on a $uw \in \mathcal{L}$ ssi $vw \in \mathcal{L}$. Une paire (S, T) est dite correcte quand S ne contient pas deux mots distincts qui soient T-équivalents. Elle est complète quand, pour tous $u \in S$ et $a \in \Sigma$, il existe un mot $v \in S$ qui est T-équivalent à ua.

Question 1
Dans cette question on suppose que $\Sigma = \{a, b\}$ et $\mathcal{L}$ est le langage des mots ne contenant pas le facteur ab. Pour chaque (S_i, T_i) ci-dessous, indiquer si la paire est correcte, et si elle est complète.

1. $S_1 = T_1 = \emptyset$.
2. $S_2 = \{\epsilon, a\}$ et $T_2 = \{b, \epsilon\}$.
3. $S_3 = \{\epsilon, a, b\}$ et $T_3 = \{b, \epsilon\}$.

Question 2
Implémenter le type des langages finis `language`, puis la fonction `make_t_equiv` qui prend en argument un oracle pour l'appartenance et un langage fini `t`, et qui renvoie la fonction qui à une paire de mots `(u, v)` associe `true` si `u` et `v` sont `t`-équivalents, `false` sinon.

Étant donné une paire (S, T) on construit un automate $\mathcal{A}(S, T)$ comme suit :

— Les états de l'automate sont les mots de S.
— L'état initial est ϵ, les états finaux sont les mots $u \in S \cap \mathcal{L}$.
— On a une transition $u \xrightarrow{a} v$ quand v est T-équivalent à ua.

Question 3
Donner l'automate $\mathcal{A}(S_i, T_i)$ pour chacune des paires de la question précédente.

Question 4
Expliquer comment construire $\mathcal{A}(S, T)$ à l'aide d'appels à l'oracle. Donner le nombre d'appels faits ainsi que le nombre d'appels différents.

1. Pour tout $w \in S$ et tout préfixe w' de w, on a aussi $w' \in S$.

Question 5
Implémenter le type des automates finis `fa` ainsi que les tests `is_init` et `is_final` et les itérateurs `iter_states, iter_trans`. Cela vous permettra d'utiliser la fonction `display` pour visualiser des automates. On pourra utiliser des listes d'association [2].

Question 6
Implémenter la fonction `guess` qui prend en argument un oracle pour l'appartenance, l'alphabet `sigma` et les langages finis `s` et `t` et renvoie l'automate $\mathcal{A}(S, T)$.

Question 7
Soit (S, T) une paire correcte et complète. Montrer que $\mathcal{A}(S, T)$ est déterministe et complet.

Question 8
Soit (S, T) une paire correcte. Montrer que le cardinal de S ne peut excéder le nombre de résiduels du langage $\mathcal{L}$.

Question 9
Soit une paire correcte (S, T). Montrer que l'on peut calculer une extension $S' \supseteq S$ tel que (S', T) est correcte et complète.

Question 10
Implémenter la fonction `extend_s` qui prend en argument un oracle pour l'appartenance, l'alphabet `sigma` et les langages finis `s` et `t` et renvoie un langage `s'` tel que décrit dans la question précédente.

Question 11 – *Non nécessaire pour la suite.*
Soit (S, T) une paire correcte et complète. Montrer que $\mathcal{A}(S, T)$ est (isomorphe à) l'automate minimal reconnaissant son langage.

2. Association Lists

2 Algorithme d'apprentissage

On se donne deux oracles ayant accès à $\mathcal{L}$. On a d'abord un oracle d'appartenance, qui prend en entrée un mot w et indique si $w \in \mathcal{L}$. On considère d'autre part un oracle d'équivalence, qui prend en entrée un automate et indique si le langage de cet automate est $\mathcal{L}$. Si ce n'est pas le cas, l'oracle fournit un mot pour lequel l'automate et $\mathcal{L}$ sont en désaccord [3]. On considère l'algorithme suivant, utilisant ces deux oracles :

1. Initialiser $(S, T) := (\{\epsilon\}, \{\epsilon\})$.
2. Si (S, T) n'est pas complète étendre $S := S'$ comme dans la question 9.
3. Faire une requête d'équivalence pour $\mathcal{A}(S, T)$.
4. Si la requête réussit, terminer : on a un automate reconnaissant $\mathcal{L}$.
5. Sinon, on obtient un contre-exemple w : ajouter w et tous ses suffixes à T et retourner à l'étape 2.

Question 12
Montrer que la paire (S, T) est correcte tout au long de l'exécution de l'algorithme.

Question 13
Implémenter la fonction `extend_t` qui prend en argument un langage fini `t` et un mot `w` et renvoie l'union de `t` et de l'ensemble des suffixes de `w`.

Question 14
Dans cette question on prend $\Sigma = \{a, b\}$ et $\mathcal{L} = \{w \in \Sigma^* \mid |w|_b = 3 \mod 4\}$. Montrer que l'algorithme permet de déterminer l'automate minimal de $\mathcal{L}$ en deux itérations (deux requêtes d'équivalence). On supposera que l'oracle d'équivalence donne des contre-exemples de longueur minimale.

Question 15
Implémenter la fonction `l_star` qui déroule une exécution interactive de L^* entre l'apprenant (la machine) et l'enseignant (l'utilisateur, jouant le rôle d'oracle). On utilisera `get_sigma` pour donner à l'apprenant l'alphabet utilisé. On utilisera `print_word` et `ask_in` comme oracle d'appartenance et `display` et `ask_equiv` comme oracle d'équivalence.

3. Soit le mot est accepté par l'automate mais n'est pas dans $\mathcal{L}$, soit il est dans $\mathcal{L}$ mais n'est pas accepté par l'automate.

Question 16
Vous remarquez probablement que vous devez répondre beaucoup de fois à la même question. Implémenter une fonction `memo` qui prend en argument une fonction `f` et renvoie une fonction qui se comporte comme `f` mais qui se souvient des réponses aux appels à `f`, et l'utiliser pour éviter le problème mentionné plus tôt.

Question 17
Considérons un passage par l'étape 5 avec un contre-exemple $w = w_1 \dots w_n$, qui va provoquer l'extension de T en $T' = T \cup \{w_i \dots w_n \mid 1 \leq i \leq n+1\}$. On suppose que (S, T') est complète, afin de montrer par l'absurde que ce n'est pas possible.

1. Montrer que $\mathcal{A}(S, T)$ et $\mathcal{A}(S, T')$ sont identiques.
2. Montrer que $\epsilon \xrightarrow{w} w'$ avec $w' \in \mathcal{L}$ ssi $w \in \mathcal{L}$.
3. Conclure.

Question 18
Montrer que l'algorithme termine, en donnant une borne sur le nombre de passages par l'étape 2.

Question 19
En déduire une nouvelle méthode de minimisation d'automate.

Question 20
Montrer que la complexité globale de l'algorithme est polynomiale en le nombre de résiduels de $\mathcal{L}$ et la taille des contre-exemples donnés par l'oracle d'équivalence, en bornant le nombre total d'appels aux deux oracles. Peut-on limiter la longueur des contre-exemples pour obtenir une complexité polynomiale en le nombre de résiduels de $\mathcal{L}$?