Université de Lille 1 SCIENCES ET TECHNOLOGIES

TDM 6 - Automates et Langages
Implémentation d'automates

novembre 2015

1 Description du travail à réaliser

Copiez et ouvrez l'archive avec les sources java.
Vous y trouverez notamment
- l'interface `Automaton` : définit un ensemble de méthodes destinées à l'utilisation d'un automate existant
- l'interface `AutomatonBuilder` : définit un ensemble de méthodes destinées à la construction d'un automate
- la classe `NDAutomaton` : implémente un automate non déterministe.

Voici ce que vous aurez à implémenter
1. Dans la classe `NDAutomaton`, la méthode `accept` n'est pas «réellement» implémentée. Donnez-lui une implémentation correcte.
2. Ajoutez à cette classe une méthode `deterministic` qui calcule le déterminisé de l'automate. Cet automate déterminisé pourra être une instance de `NDAutomaton` (un automate déterministe peut être vu comme un cas particulier d'automate non-déterministe).
3. Écrivez une classe `Dautomaton` qui implémente un automate déterministe.
4. Écrivez une classe `AhoCarosick` qui étend la classe précédente. Elle aura pour constructeur un tableau de chaînes de caractères non vides qui seront les mots à rechercher dans un texte (voir chapitre ci-dessous)

2 La construction de Aho Corasick

Cet algorithme dû à Alfred Aho et Margaret Corasick permet de construire un automate de recherche de mots dans un texte. Il s'agit d'une généralisation du problème que nous avons rencontré lors d'un exercice précédent : l'automate alors nommé «AutomateMotif» permettait de rechercher **un** mot alors que de celui de Aho-Corasick permet d'en chercher **plusieurs** en même temps.
La donnée de l'algorithme est donc un ensemble de M de n mots non vides : $M = \{u_0, ..., u_{n-1}\}$, son résultat est un automate déterministe reconnaissant $L = X^*.M$.

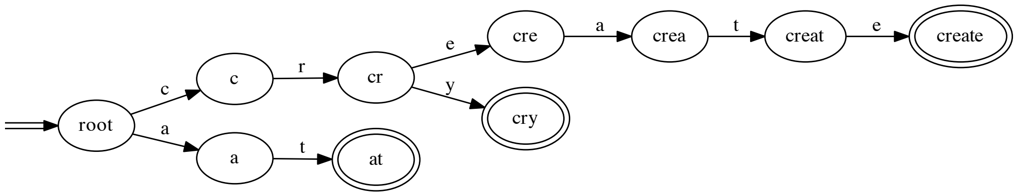
2.1 Caractéristiques de l'automate

Un automate déterministe reconnaissant L possède au moins un état pour chaque résiduel de L. Intéressons-nous à ces résiduels.
- Soit un mot z tel qu'aucun suffixe non vide de z n'est préfixe de M. En d'autres termes, z ne se termine pas par le début d'un mot u_i.
  Montrons tout d'abord que $L/z \subseteq L$.
  $v \in L/z \implies z.v \in X^*.M \implies \exists w \in X^*, \exists u_i, z.v = w.u_i$
  Aucun suffixe de z n'étant préfixe d'un u_i, il résulte que $\exists w' \in X^*, v = w'.u_i$
  donc $v \in X^*M = L \quad \implies L/z \subseteq L$.
  Par ailleurs $z.L \subseteq X^*.L = L \implies L \subseteq L/z$ et donc
  $$L/z = L = L/\epsilon$$
- Soit un mot w possédant un suffixe non vide préfixe de M. Appelons p le **plus long** suffixe de w tel que p est préfixe de M, et appelons z le mot tel que $w = z.p$
  z vérifie les caractéristiques du cas précédent, donc $L/z = L$.
  $$L/w = L/(z.p) = (L/z)/p = L/p$$

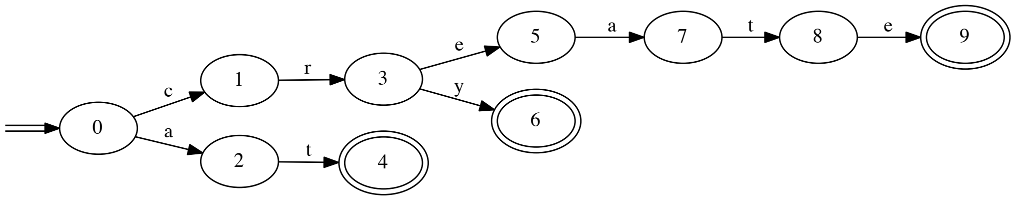
Chaque état de l'automate à construire correspondra au résiduel de L par l'un des préfixes p (vide ou non) de M. Un état sera acceptant si et seulement si p possède pour suffixe un mot de M.

2.2 Le squelette de l'automate

La première étape consiste à construire un automate reconnaissant M. Un tel automate possède une forme d'arbre (chaque nœud possède un et un seul antécédent à l'exception d'un nœud,la racine de l'arbre, qui ne possède aucun antécédent). Il s'agit de l'arbre des préfixes de M : à chaque préfixe de M correspond un et un seul état de l'automate. L'état initial est la racine et chaque état correspondant à un mot u_i est acceptant.
Prenons l'exemple de 3 mots à rechercher : **create**, **at**, et **cry**.



Dans la suite de l'algorithme, il sera nécessaire de parcourir les états selon l'ordre de distance croissante séparant les états de la racine. On prendra donc soin de créer d'abord tous les états correspondant aux préfixes de longueur 1, puis ceux pour les préfixes de longueur 2, etc
Voici le même automate, avec les états numérotés par ordre de création



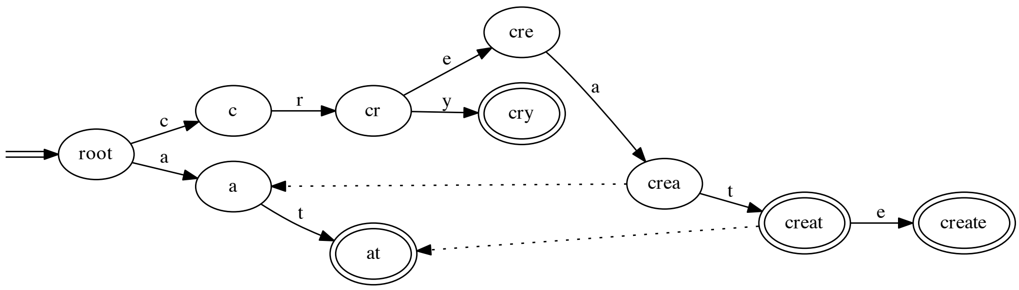
$racine \leftarrow nouvel\ état$;
for $i = 0$ **to** $n - 1$ **do**
  $finBranche[i] \leftarrow racine$;
end for
for $l = 0$ **to** $max(|u[i]|) - 1$ **do**
  for $i = 0$ **to** $n - 1$ **do**
    if $l < |u[i]|$ **then**
      $q \leftarrow nouvel\ état$;
      $définir\ \delta(finBranche[i], u[i][l]) \leftarrow q$;
      $finBranche[i] \leftarrow q$;
      if $l + 1 = |u[i]|$ **then**
        $finBranche[i]\ est\ acceptant$
      end if
    end if
  end for
end for

À cette étape, tous les états de l'automate à construire ont été créés.

2.3 Les états de repli

À chaque état, nous allons maintenant associer un «état de repli». Appelons q un état et v le mot correspondant à cet état (v est donc un préfixe de M).

L'état de repli de q est défini comme l'état correspondant au plus long suffixe de v, différent de v, qui soit également préfixe de M.
Par exemple le repli de l'état correspondant à `creat` est l'état correspondant à `at`.
Voici plus complètement ce qu'on obtient sur l'exemple. les flèches en pointillé représentent la relation de repli. (NB : pour des raisons de lisibilité, seules les flèches n'amenant pas à l'état racine ont été dessinées. En réalité pour tous les autres états q, $repli(q) = racine$, une flèche en pointillé devrait donc renvoyer à l'état de départ)



- L'état de repli de q correspond à un mot strictement plus court que v : il est donc strictement plus proche de la racine que ne l'est q.
- Tout état possède un état de repli (au «pire» l'état racine).
- L'état de repli de chaque successeur de la racine est la racine.
- Si q est un état qui n'est ni la racine ni l'un de ses successeurs. Supposons que l'on ait calculé l'état de repli de tous les états de rang inférieur à q. On dispose de l'algorithme suivant pour déterminer l'état de repli de q.

  $s \leftarrow parent(q)$
  $lettre \leftarrow l$ telle que $\delta(s, l) = q$
  repeat
    $s \leftarrow repli(s);$
    $e \leftarrow \delta(s, lettre)$
  until $e \neq null$ ou $s = racine$
  if $e \neq null$ **then**
    $repli(q) \leftarrow e$
  else
    $repli(q) \leftarrow racine$
  end if

Si l'état de repli de q est un état acceptant, cela signifie que le mot v possède un suffixe qui est l'un des mots de M. q doit donc également être un état acceptant.
Le calcul des états de repli pourra être intégré à la phase de création du squelette, puisque celui-ci est construit par éloignement croissant depuis la racine. Voilà donc une nouvelle version de l'algorithme de construction :

$racine \leftarrow nouvel\ \text{état}$;
for $i = 0$ **to** $n - 1$ **do**
  $finBranche[i] \leftarrow racine$;
end for
for $l = 0$ **to** $max(|u[i]|) - 1$ **do**
  for $i = 0$ **to** $n - 1$ **do**
    if $l < |u[i]|$ **then**
      $q \leftarrow cr\acute{e}erNouvelEtat(finBranche[i], u[i][l])$
      $finBranche[i] \leftarrow q$;
      if $l + 1 = |u[i]|$ **then**
        $finBranche[i]\ est\ acceptant$
      end if
    end if
  end for
end for

$créerNouvelEtat(parent, lettre)$:

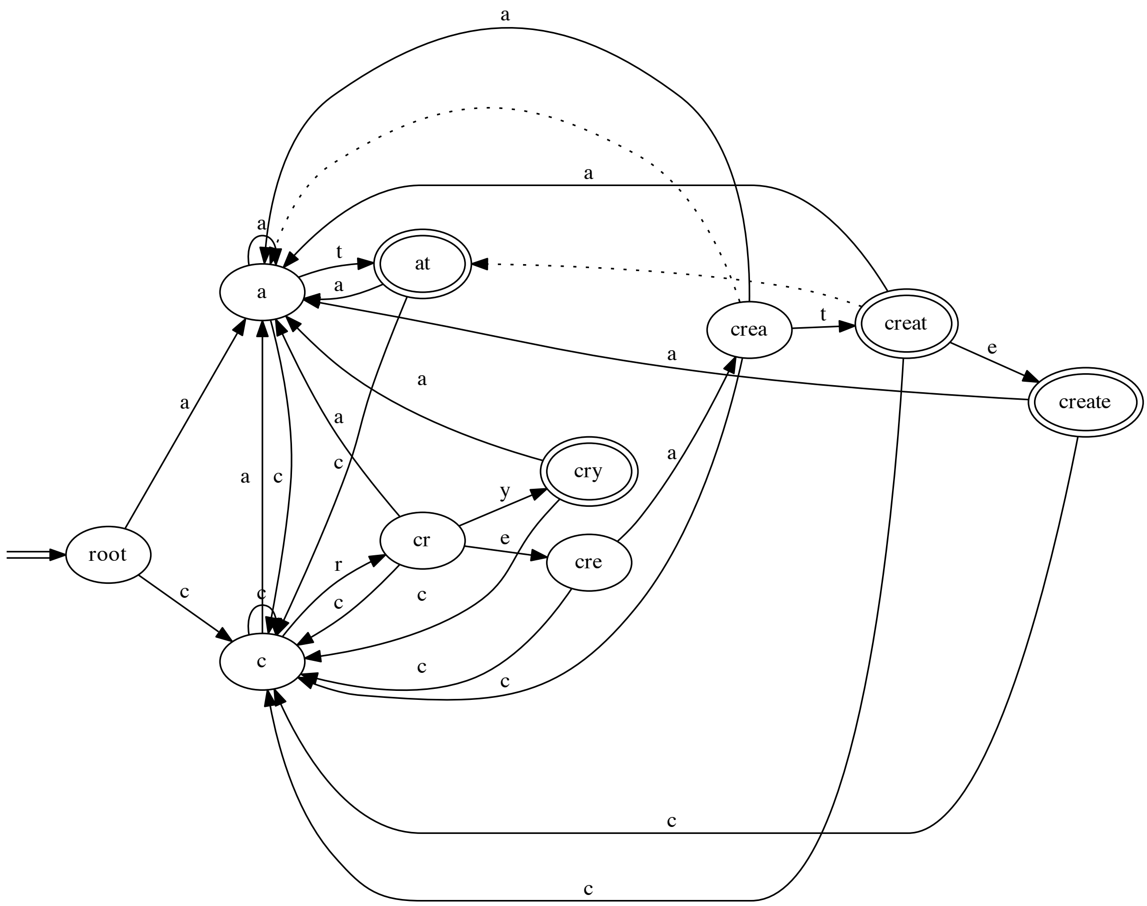
$q \leftarrow nouvel$ $état$;
if $parent = racine$ **then**
  $repli(q) \leftarrow racine$
else
  $s \leftarrow parent$
  repeat
    $s \leftarrow repli(s)$;
    $e \leftarrow \delta(s, lettre)$
  until $e \neq null$ ou $s = racine$
  if $e \neq null$ **then**
    $repli(q) \leftarrow e$
    if e est $acceptant$ **then**
      q est $acceptant$
    end if
  else
    $repli(q) \leftarrow racine$
  end if
end if
return q ;

2.4 Ajouter des transitions

Il nous reste à compléter l'automate par de nouvelles transitions. Pour chaque état q et chaque lettre x, si $\delta(q, x)$ n'est pas définie, alors on ajoute $\delta(q, x) = \delta(repli(q), x)$.
Là encore, il faudra avoir calculé au préalable $\delta(repli(q), x)$ ce qui peut être aisément réalisé en parcourant les états par éloignements croissants depuis la racine.
$completerAutomate()$:

for all état q **do**
  for all lettre **do**
    if $\delta(q, lettre) = null$ **then**
      if $q = racine$ **then**
        $\delta(q, lettre) \leftarrow racine$
      else
        $\delta(q, lettre) \leftarrow \delta(repli[q], lettre)$
      end if
    end if
  end for
end for

Voici en page suivante l'automate final obtenu pour notre exemple
Là encore, pour des raisons de lisibilité, seules les flèches n'amenant pas à l'état racine ont été dessinées. En réalité toutes les transitions non dessinées ramènent à l'état de départ.


root
a
at
c
cr
cry
cre
crea
creat
create