UNIVERSITE DE NANTES
Master 1 Informatique

Nantes le 03/05/2005

Projet
Principes d'optimisations discrètes

Abdeslam MOKRANI

SOMMAIRE

I. Introduction

II. Rappels

1. Appellations et exposé du problème

- *Mot p :*
  Une séquence finie *p = p1p2 ... pm* de lettres prises dans un alphabet Σ.

- *Facteur x* de *p* :
  Mot sur l'alphabet Σ tel que *p = uxv* avec *u, v* des mots (éventuellement vides) sur Σ.

- *Suffixe x* de *p* :
  Facteur de *p* tel que *p = ux*, avec *u* un mot (éventuellement vide) sur Σ.

Les algorithmes que nous allons voir permettent de rechercher un mot donné dans un texte donné. Afin d'augmenter leur performance, ceux-ci stockent le texte dans une structure appelée *index*. Dans celui-ci, la recherche et les insertions de mot sont nettement plus rapides. En revanche, il en résulte un certain taux d'erreur dans les réponses données. En effet, un mot n'existant pas dans le texte peut tout de même être trouvé dans l'indexe. Par contre, si un mot n'est pas trouvé dans l'indexe, alors il est certain qu'il n'existe pas dans le texte. Le but est d'avoir le moins d'erreurs possibles dans les réponses données par ces algorithmes.

Un mot qui n'existe pas dans le texte est appelé *intrus*. Tous les mots qui ne sont pas trouvés dans l'indexe sont donc des intrus. L'efficacité d'un algorithme se calcule alors par le nombre d'intrus reconnus.

Dans ce projet, nous allons étudier deux algorithmes. Ce sont deux automates appelés respectivement Alpha et Beta.

2. Les algorithmes à étudier

a. L'algorithme `ConstructionAlpha`

```
Entrées : Σ, s ∈ Σ*
Sortie : Alpha(s)

Début
   Créer l'état initial e0
   Ss(e0) ← e-1

   Pour i de 1 à |s| Faire
      Créer l'état ei
      Construire ei-1 –s[i]→ ei
      Soit k l'indice de l'état Ss(ei-1)

      Tant Que k > -1 et ! ∃ ek –s[i]→ Faire
         Construire ek –s[i]→ ei
         K ← l'indice de l'état Ss(ek)
```

```
      Fin Tant Que

      Si (k = -1) Alors
         Ss(e0) ← e0
      Sinon
         Soit ex tel que ek –s[i]→ ex
         Ss(ei) ← ex
      Fin si
   Fin Pour
Fin
```

b. L'algorithme `ConstructionBeta`

```
Entrées : Σ, s ∈ Σ*
Sortie : Beta(s)

Début
   Créer l'état initial e0
   Ss(e0) ← e-1
   Ls(e0) ← -1

   Pour i de 1 à |s| Faire
      Créer l'état ei
      Construire ei-1 –s[i]→ ei
      l ← Ls(ei-1)
      Soit k l'indice de l'état Ss(ei-1)

      Tant Que k > -1 et ! ∃ ek –s[i]→ Faire
         Construire ek –s[i]→ ei
         ValeurDeGarde(ekei) = l
         l ← Ls(ek)
         K ← l'indice de l'état Ss(ek)
      Fin Tant Que

      Si (k = -1) Alors
         Ss(e0) ← e0
      Sinon
         Soit ex tel que ek –s[i]→ ex
         Ss(ei) ← ex
         Ls(ei) ← l + 1
         si x > k + 1 alors /* arc externe */
            l' ← ValeurDeGarde(ekex)
            ValeurDeGarde(ekex) ← max(l, l')
         Fin Si
      Fin si
   Fin Pour
Fin
```

III. Réponses aux questions

1. Question 1.

La chaîne de caractères utilisée est **gaccattctc**

a. L'automate Alpha de la chaîne

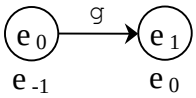
g	a	c	c	a	t	t	c	t	c
1	2	3	4	5	6	7	8	9	10

Tous les états sont terminaux. En dessous de chaque état, la valeur de S_s.

Init.

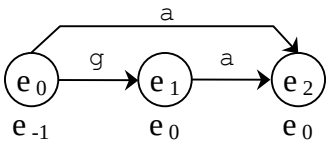
---

i = 1
k = -1 : $S_s(e_1) = e_0$



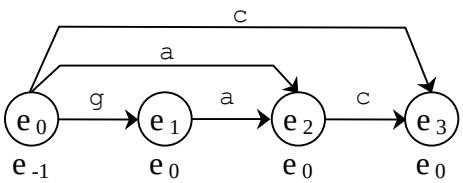
---

i = 2
k = 0 : $e_0 \xrightarrow{a} e_2$
k = -1 : $S_s(e_2) = e_0$

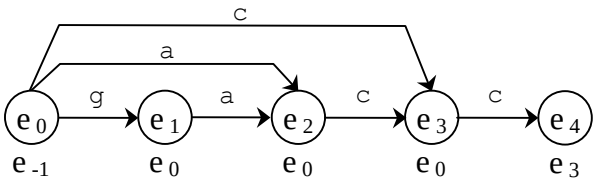


---

i = 3
k = 0 : $e_0 \xrightarrow{c} e_3$
k = -1 : $S_s(e_3) = e_0$

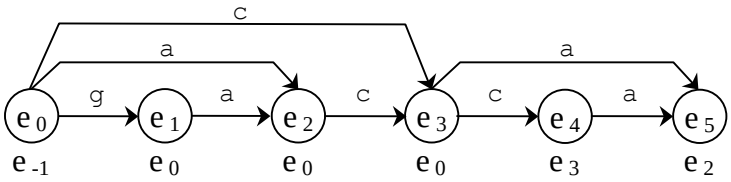


i = 4
k = 0 : □
$e_0 \xrightarrow{-c} e_3 : S_s(e_4) = e_3$



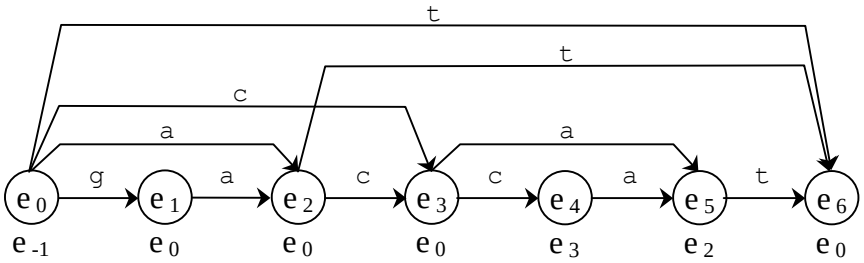
---

i = 5
k = 3 : $e_3 \xrightarrow{-a} e_5$
k = 0 : □
$e_0 \xrightarrow{-a} e_2 : S_s(e_5) = e_2$



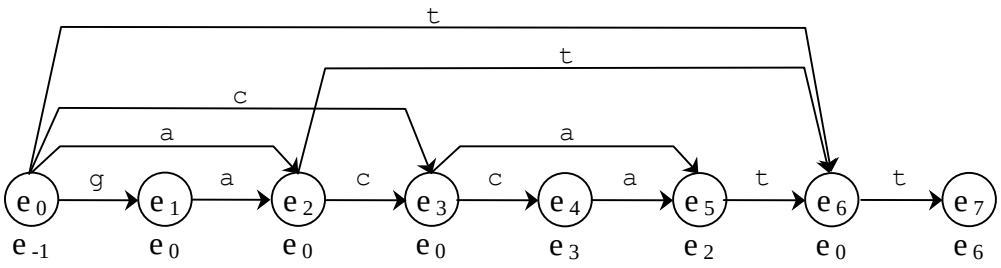
---

i = 6
k = 2 : $e_2 \xrightarrow{-t} e_6$
k = 0 : $e_0 \xrightarrow{-t} e_6$
k = -1 : $S_s(e_6) = e_0$

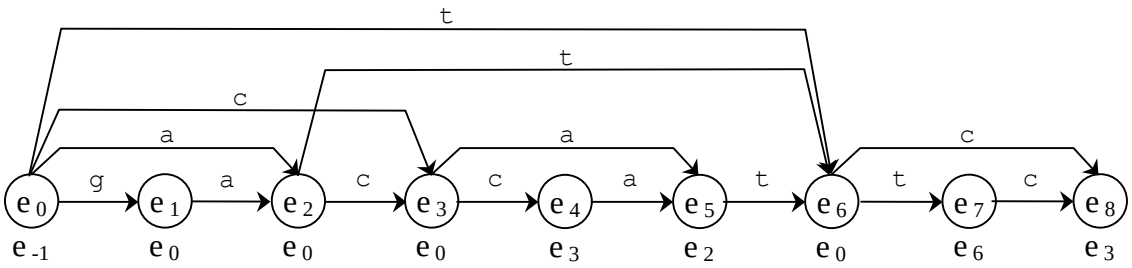


---

i = 7
k = 0 : □
$e_0 \xrightarrow{-t} e_6 : S_s(e_7) = e_6$

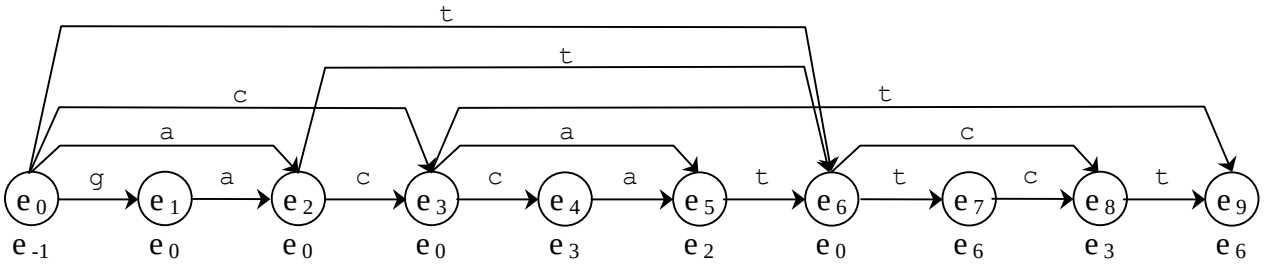


i = 8
 k = 6 : $e_6 \xrightarrow{c} e_8$
 k = 0 : □
 $e_0 \xrightarrow{c} e_3$: $S_s(e_8) = 3$



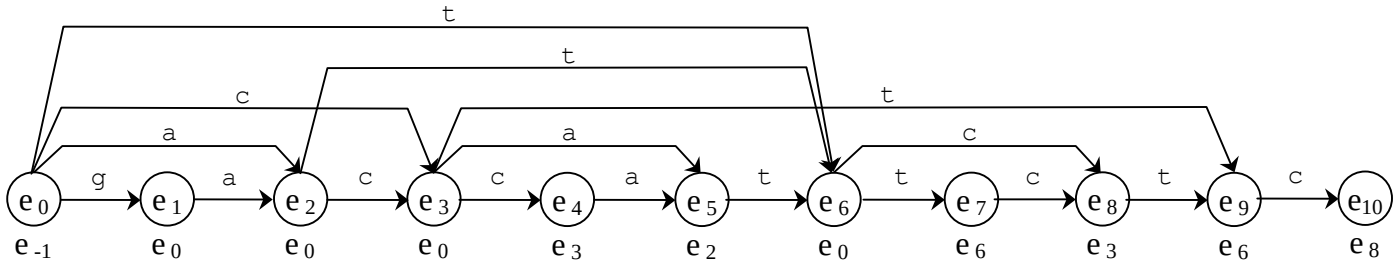
---

i = 9
 k = 3 : $e_3 \xrightarrow{t} e_9$
 k = 0 : □
 $e_0 \xrightarrow{t} e_6$: $S_s(e_8) = e_6$



---

i = 10
 k = 6 : □
 $e_6 \xrightarrow{c} e_8$: $S_s(e_{10}) = e_8$



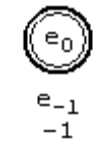
Version suffixes :

Seules les états e_3, e_8 et e_{10} sont terminaux.

b. L'automate Beta de la chaîne

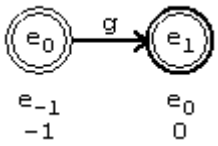
Les états terminaux sont entourés par un double cercle. En dessous de chaque état, les valeurs de S_s et L_s.

Init.



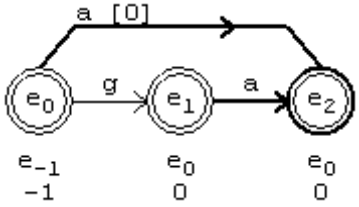
---

i = 1
k = -1 : $S_s(e_1) = e_0$



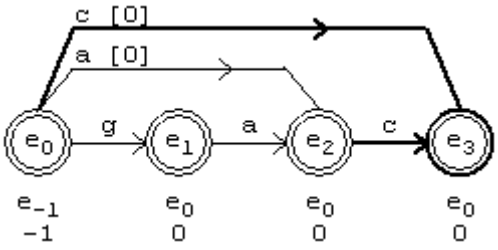
---

i = 2
k = 0 : $e_0 \xrightarrow{a} e_2$
k = -1 : $S_s(e_2) = e_0$



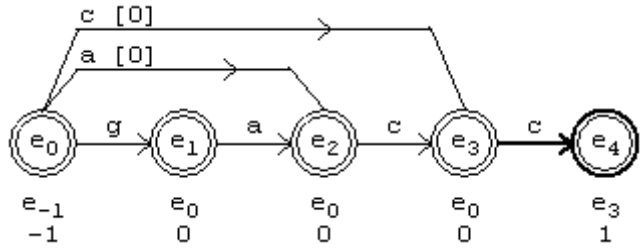
---

i = 3
k = 0 : $e_0 \xrightarrow{c} e_3$
k = -1 : $S_s(e_3) = e_0$



---

i = 4
k = 0 : □
$e_0 \xrightarrow{c} e_3$: $S_s(e_4) = e_3$

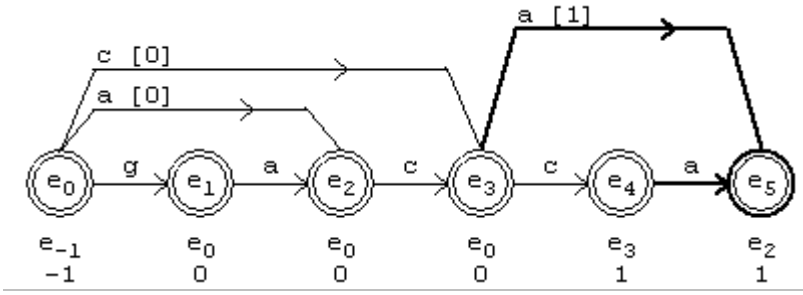


i = 5

k = 3 : $e_3 \xrightarrow{a} e_5$

k = 0 : □

$e_0 \xrightarrow{a} e_2$: $S_s(e_5) = e_2$



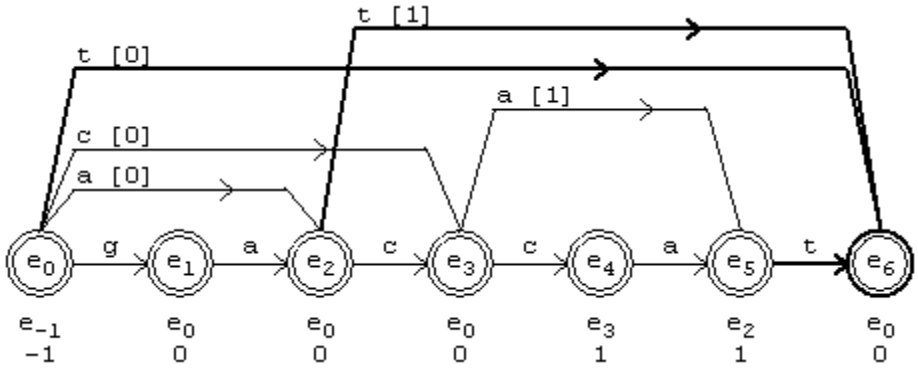
---

i = 6

k = 2 : $e_2 \xrightarrow{t} e_6$

k = 0 : $e_0 \xrightarrow{t} e_6$

k = -1 : $S_s(e_6) = e_0$

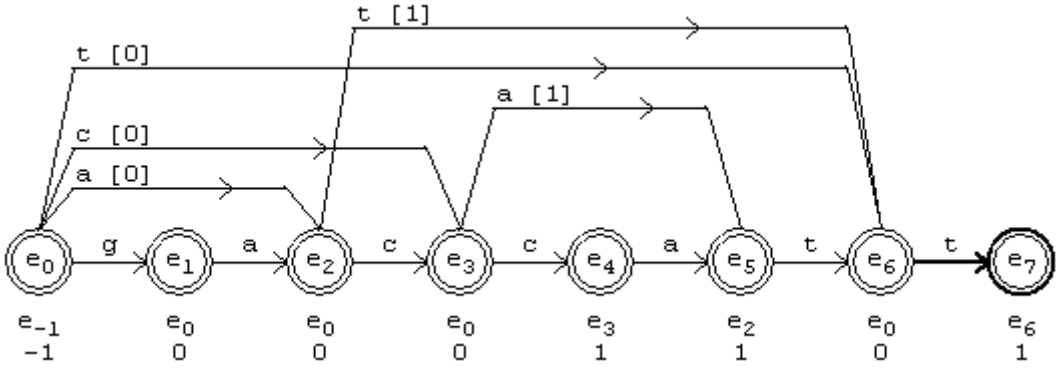


---

i = 7

k = 0 : □

$e_0 \xrightarrow{t} e_6$: $S_s(e_7) = e_6$

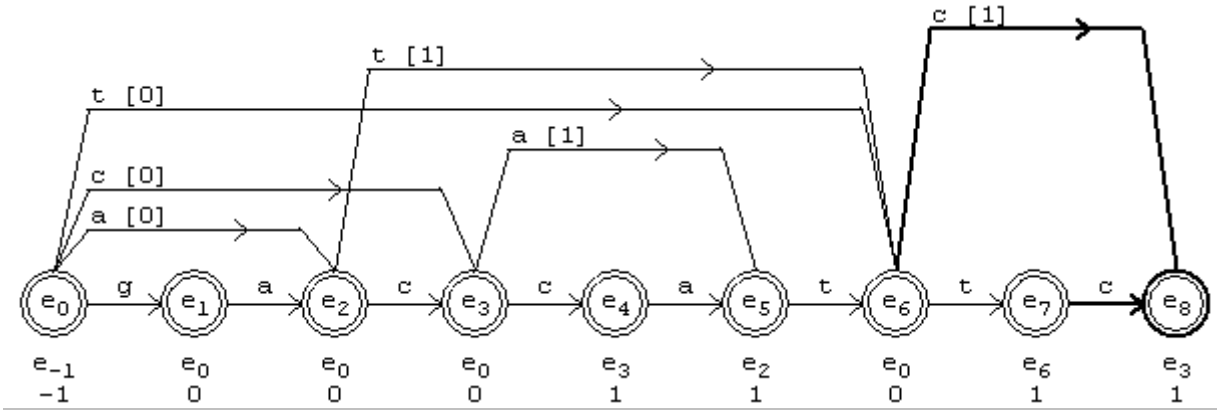


i = 8

k = 6 : $e_6 \xrightarrow{-c} e_8$

k = 0 : □

$e_0 \xrightarrow{-c} e_3$: $S_s(e_8) = 3$

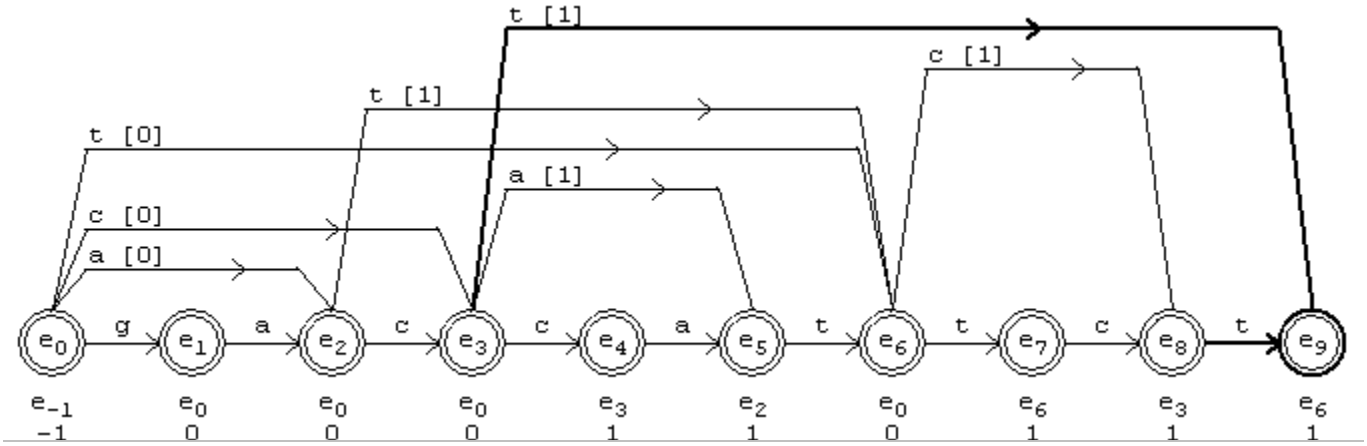


i = 9

k = 3 : $e_3 \xrightarrow{-t} e_9$

k = 0 : □

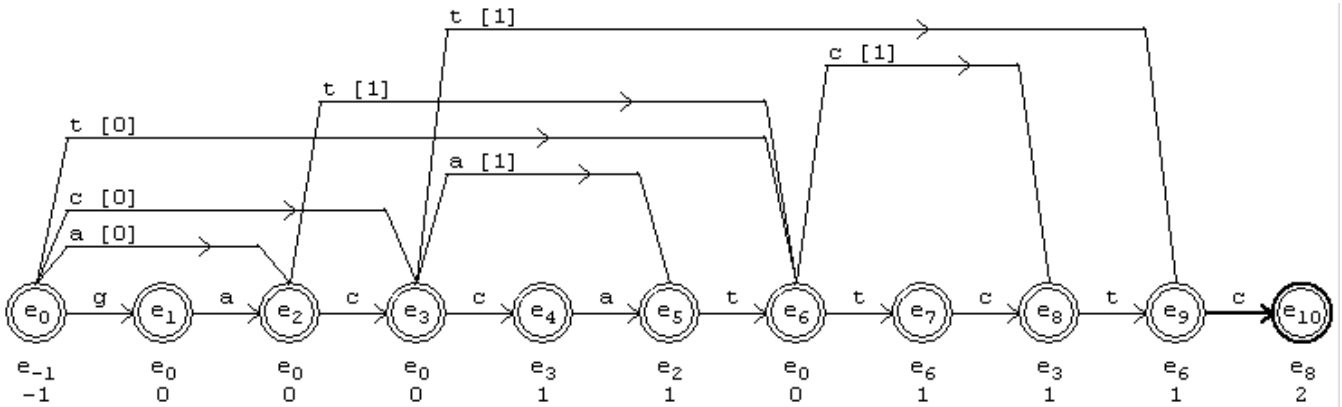
$e_0 \xrightarrow{-t} e_6$: $S_s(e_8) = e_6$



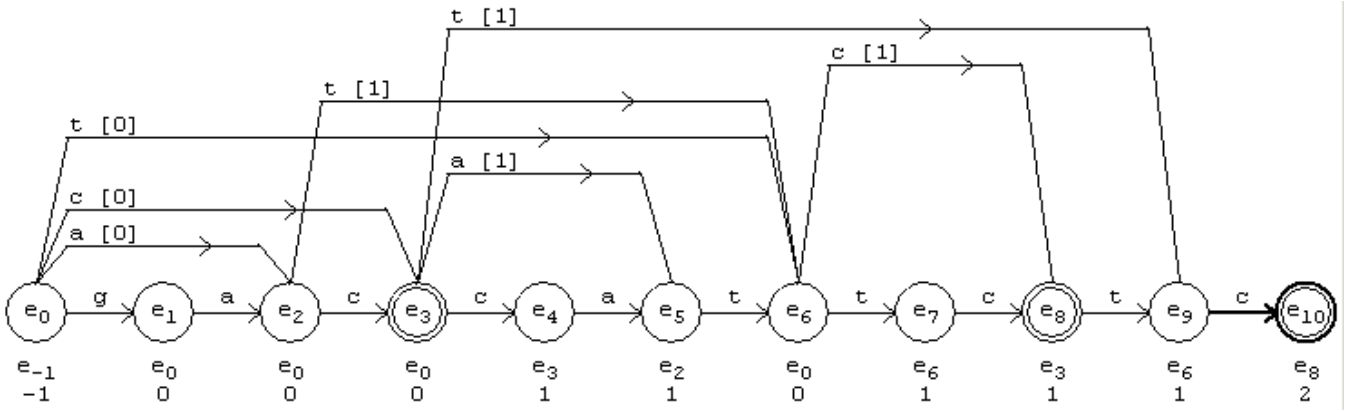
i = 10

k = 6 : □

$e_6 \xrightarrow{-c} e_8$: $S_s(e_{10}) = e_8$



Version suffixes :



2. Question 2

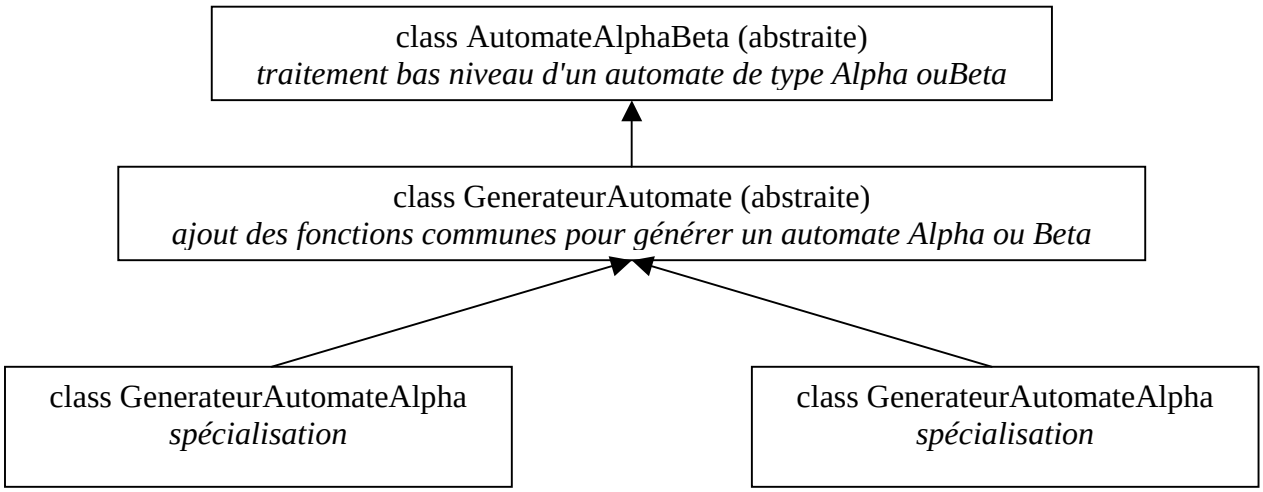
L'automate *Beta* correspond exactement à l'automate *Alpha* auquel on ajoute des contraintes (valeurs de gardes) sur les arcs extérieurs. Ces contraintes limitent (d'après l'algorithme ConstructionBeta) le nombre de mots reconnus. Or tous les facteurs (ou suffixes pour les automates $Alpha_s$ et $Beta_s$) du mot traité sont reconnus par les deux automates (d'après énoncé) Donc les mots reconnus par l'automate *Alpha* et non reconnus par l'automate *Beta* correspondant sont forcement des intrus.

On déduit que l'automate *Alpha* reconnaît plus d'intrus que l'automate *Beta*.

3. Question 3

Le langage de programmation choisi est le C++. Pour visualiser les automates, la bibliothèque GTK a été utilisée. Pour les dessins d'automates, de fonctions primitives de dessins de GDK ont été utilisées.

Le code source a été factorisé en utilisant des l'héritage. Ainsi les fonctions de traitement d'automates et celles qui sont communes aux automates Alpha et Beta sont codés une seule fois dans une classe de base. Voici la hiérarchie de ces classes :



Voir le code source des fonctions de construction en annexe.

4. Question 4

Les automates de type Alpha ou Beta ont la même complexité. En effet, les conditions d'entré en boucle (*pour* et *tant que*) sont les même et les variables qui y figurent (*i, k,* …) sont traitées exactement de la même manière.

Soit *n* la longueur du mot traité. La boucle principale (*pour*) est exécuté exactement *n* fois.

5. Question 5

Voir annexe.

6. Question 6

Voir annexe.

7. Question 7

Tous les facteurs ou les suffixes d'un mot sont reconnus par l'automate Alpha ou Beta correspondant. Chaque mot est reconnu une seule fois car les automates en question sont déterministes. Le nombre de mot n'appartenant pas au mot traité et reconnus par un automate est donc égale au nombre total de mots reconnu moins le nombre de mots qu'il faut reconnaître.

Le nombre total de mots reconnus est calculé dans la question 5. Le nombre de suffixes qu'il faut reconnaître est égal à la longueur du mot traité. Celui des facteurs et calculé à l'aide de l'algorithme suivant :

```
/* calculer le nombre de facteur de taille n */
fonction calculer_nb_facteurs_aux (mot : TChaîne, n : entier) : entier
début
      liste ← liste_vide
      pour i ← 0 à taille(mot) - n faire
            si non appartient(liste, mot[i..i+n]) alors
                  ajouter (liste, mot[i..i+n])
            fin si
      fin pour
      retourner taille(liste)
fin.

fonction calculer_nb_facteur (mot : TChaîne)
début
      nb ← 0
      pour i ← 1 à taille(mot) faire
            nb ← nb + calculer_nb_facteurs_aux (mot, i)
      fin pour

      nb ← nb + 1 /* pour la chaîne vide */
      retourner nb;
fin.
```

On a la relation *nbIntrus = nbMotsReconnus - nbMotsQuiDoiventEtreREconnus*, on peut considérer la fonction qui retourne *nbIntrus* comme fonction de mesure de

performances d'un automate. En effet, plus le nombre *nbIntrus* est grand, plus l'automate est moins performant. 0 est la performance maximale : aucun intrus reconnus.

On peut effectuer cette opération sur une liste de mots générés aléatoirement. Dans ce cas, les valeurs *nbIntrus* des différents mots sont additionnées pour obtenir la performance de la liste des mots.

8. Question 8

Voir annexe.

9. Question 9

Voir annexe.

10. Question 10

dd

11. Question 11

dd