Chapitre 2 : Automates asynchrones et expressions rationnelles

I. Automates asynchrones

Souvenons-nous que jusqu'à ce point nous avons évité **le mot vide** comme étiquette de transition**.**

Un automate fini dans lequel certaines flèches sont étiquetées par le mot vide (« ***ε-transitions*** ») s'appelle **automate asynchrone**. L'ensemble des flèches vérifie donc $E \subset Q \times (A \cup \{\varepsilon\}) \times Q$.

Proposition : Pour tout automate asynchrone, il existe un automate fini ordinaire (c.à.d. sans ε-transitions) qui reconnaît le même langage.
Comme tout automate non déterministe est équivalent à un automate déterministe, il en suit que **tout automate asynchrone est équivalent à un automate déterministe**.

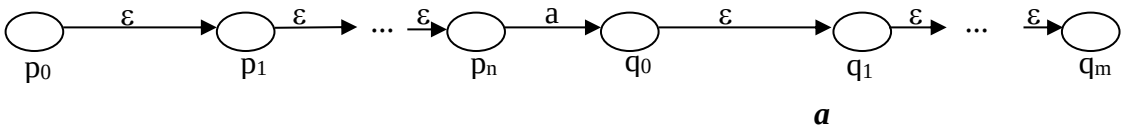
Il existe un algorithme de suppressions des ε-transitions.

Pour un automate asynchrone $C = (Q, I, T, E)$ avec $E \subset Q \times (A \cup \{\varepsilon\}) \times Q$ nous allons construire un automate $B=(Q, I, T, F)$ qui ne diffère de C que par l'ensemble de ses flèches (F) et ce, suivant une règle bien déterminée :

par définition on a $(p.a.q) \in F$ s'il existe un chemin

$c : \; p_0 \xrightarrow{\varepsilon} p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_n \xrightarrow{a} q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_m$

et $p_0 = p, q_m = q$

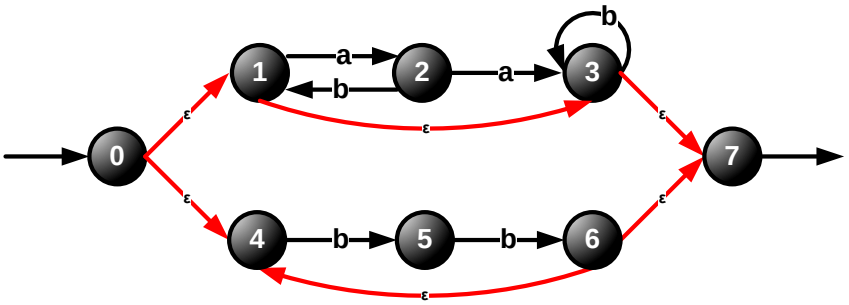


a

Le nombre de ε-transitions à droite ou à gauche de la transition $p_n \xrightarrow{} q_0$ peut être nul (dans ce cas $p_n = p_0$ ou $q_0=q_m$**).**
(Clarification : on identifie donc tout **chemin** de l'automate C n'impliquant qu'une seule transition étiquetée par un caractère autre que ε, à une transition de l'automate B marquée par ce caractère).

Ex. Prenons un automate asynchrone :

A1



1a) Elimination des ε-transitions

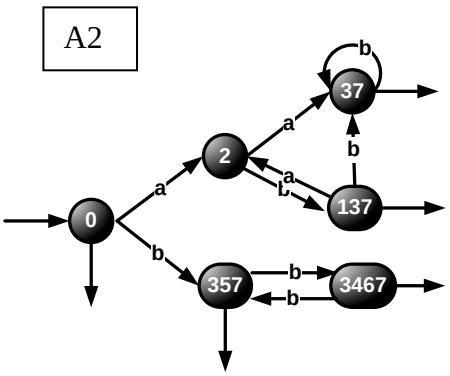
0ε1ε3ε7 signifie que 0 est une sortie 0ε1a2 → 0a2 0ε1ε3b3→ 0b3 0ε1ε3b3ε7→ 0b7 0ε4b5→ 0b5	1ε3ε7 signifie que 1 est une sortie 1a2 reste tel quel 1ε3b3→ 1b3 1ε3b3ε7→ 1b7	2a3 reste tel quel 2a3ε7 →2a7 2b1 reste tel quel 2b1ε3 →2b3 2b1ε3ε7 →2b7	3ε7 signifie que 3 est une sortie 3b3 reste tel quel 3b3ε7 →3b7
4b5 reste tel quel	5b6 reste tel quel 5b6ε4→ 5b4 5b6ε7→5b7	6ε7 signifie que 6 est une sortie 6ε4b5→ 6b5	7 : pas de transitions une sortie

On obtient une TT :

	état	a	b
E/S	0	2	3,5,7
S	1	2	3,7
	2	3,7	1,3,7
S	3	--	3,7
	4	--	5
	5	--	4,6,7
S	6	--	5
S	7	--	--

1b) Déterminisation (sans compléter):

	état	a	b
ES	0	2	357
	2	37	137
S	357	--	3467
S	137	2	37
S	37	--	37
S	3467	--	357



2) Déterminisation en un seul pas :

Nous introduisons la notion des ε-clôtures :

Une **ε-clôture** de l'état S est l'ensemble d'états consistant en cet état S lui-même et aussi en tous les états où on peut aller depuis S en suivant exclusivement des ε-transitions**.**

ε-clôtures :

de 0: **0**1347 de 3: **3**7
de 1 : **1**37 de 5: 5
de 2 : 2 de 6 : **4**6**7**

Notons 0'=01347, 1'=137, 2'=2, 3'=37, 5'=5 , 6'=467 (notation utile mais pas obligatoire).

Dans les termes de ces ε-clôtures qui deviennent des états composés, nous pouvons déterminiser tout de suite :

	état	a	b
E/S	01347	2	357
	2	37	--
S	357	--	3467
S	37	--	37
S	3467	--	357

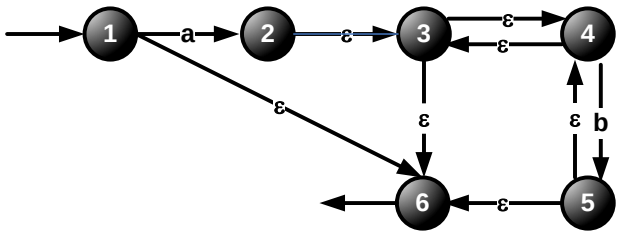
ou bien en notation "prime":

	état	a	b
E/S	0'	2'	3'5'
	2'	3'	--
S	3'5'	--	3'6'
S	3'	--	3'
S	3'6'	--	3'5'

Exemple plus compliqué

L'exemple qu'on vient de considérer n'est pas assez général. Prenons un exemple plus complexe :

A3



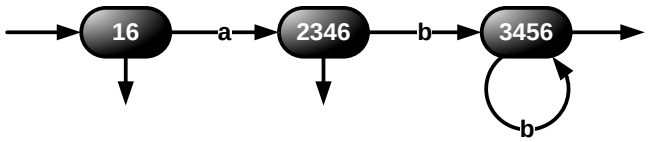
Déterminisons cet automate en considérant tous les états qu'on peut atteindre en lisant un caractère comme un état composé ; l'état initial c'est l'état où on peut arriver en ne lisant que le mot vide, donc ici c'est l'état composé (1,6). En lisant ***a*** à partir de l'état 1, on arrive en 2, mais en lisant ***aε=a***, on arrive en 3, en lisant ***aεε=a***, on arrive en 4 et en 6, donc on a l'état composé (2,3,4,6) etc.

			où, utilisant les ε-clôtures:		
état	**a**	**b**	**état**	**a**	**b**
1 6	2 3 4 6	--	1'	2'	--
2 3 4 6	--	3 4 5 6	2'	--	5'
3 4 5 6	--	3 4 5 6	5'	--	5'

Tous les états composés contenant l'état terminal 6 sont terminaux, donc ici tous les trois états sont terminaux.

On obtient :

A4



Il existe d'autres techniques équivalentes pour éliminer les ε-transitions[1].

II. Expressions rationnelles

Il existe un autre moyen de définir les langages reconnaissables. Sont reconnaissables les langages qui peuvent être exprimés par les expressions rationnelles (voir Théorème de Kleene ci-dessous).

Expression rationnelle, définition :

Soit *A* un alphabet fini, non vide. Nous avons déjà défini *A** comme ensemble de tous les mots sur l'alphabet *A* plus le mot vide. Nous verrons que cette définition suit naturellement de la définition d'une des opérations que nous allons décrire.

Nous construisons un ER de façon suivante, sans oublier que toute ER représente *un ensemble* :
On définit deux *formules atomiques :* ∅ et ε, où ∅ correspond à un langage vide, et ε est le mot vide (qu'on confond avec le langage qui contient un seul mot, ε).

- ∅ est une expression rationnelle
- le mot vide ε est une expression rationnelle
- toute lettre $a \in A$ est une expression rationnelle
- si X, Y sont des expressions rationnelles, alors X + Y, XY, X* sont des expressions rationnelles.

Notation

On définit trois opérations :

- X + Y désigne l'union des expressions X et Y.
  $$\mathbf{X + Y = \{u |\ u \in X \text{ ou } u \in Y\}}$$
- XY désigne le produit (concaténation, parfois notée avec un point comme dans l'algèbre normale)
  $$\mathbf{XY = \{uv |\ u \in X,\ v \in Y\}}$$
- L'étoile désigne une séquence de longueur arbitraire (y compris zéro) d'éléments de l'expression étoilée :
  $$\mathbf{X^* = \{u_1 u_2 \ldots u_n |\ n \geq 0,\ u_i \in X\}}$$
  (On convient ici que le mot vide appartient à **X*** - cela correspond au cas n=0).
- Règles de priorité : * a priorité sur · qui a priorité sur +. Autrement, on utilise les parenthèses comme dans l'arithmétique.
  Ainsi, a·b + c* = ((a·b) + (c*)), a·(b+c)*=(a·((b+c)*), (a·b+c)*=((a·b)+c)*).

[1] Par exemple : Patrice Séébold, Théorie des automates, Vuibert Informatique 1999, pp. 165-167.

Nous avons dit que des ER représentent des ensembles de mots. Mais la notation est différente de la notation habituelle pour les ensembles, elle est algébrique. En fait, on parle de **l'algèbre d'expressions rationnelles**. Si l'on veut être précis, il faut distinguer si on considère un objet comme un ensemble ou comme une expression. Par exemple, le langage consistant en deux mots « **papa** » et « **maman** » construit sur l'alphabet français s'écrit L={ **papa**, **maman** }. L'expression rationnelle lui correspondant s'écrit **papa** + **maman**. Nous allons confondre les deux concepts et nous allons nous permettre d'écrire L = **papa** + **maman.**

Exemple 1 Si **X = ab+ba,** alors **X*** désigne l'ensemble des mots de la forme $w=\prod_{i=1}^{n} u_i$ avec $\mathbf{u_i = ab}$ ou $\mathbf{u_i = ba}$ pour $\forall n\ 0 \le n < \infty$. Les mots appartenant à cet ensemble sont, par exemple, **ε, ab, ba, abba, abab, baba, baab**, mais non pas **aabb** ni **abaa**, donc on ne peut pas simplifier l'écriture de cet ensemble comme **(a+b)***.

Exemple 2 Les éditeurs de textes permettent l'usage d'expressions rationnelles pour rechercher une chaîne de caractères dans un fichier.

Dans EMACS on écrit :
[a-c][a-b]* au lieu de **(a+b+c)(a+b)***
. au lieu de la somme de tous les caractères sauf new-line.
x$ au lieu de x suivi du caractère new-line.

Quelques identités rationnelles
L'algèbre des expressions rationnelles est très compliquée. Assez souvent on découvre que deux expressions rationnelles qui ne se ressemblent pas du tout, sont égales. Pour l'établir, on doit le plus fréquemment passer par des automates (voir ci-dessous). Mais on peut quand-même établir quelques identités suffisamment simples. Dans ce qui suit, E, F etc. sont des expressions rationnelles.

- Une loi commutative pour + : E + F = F + E
  (Attention : il n'y a pas de loi commutative pour la concaténation ! EF≠FE *!)*
- Une loi associative pour + : (E + F) + G=E + (F + G)
- Une loi associative pour ⋅: (E F) G=E (F G)
- Une loi distributive : E(F + G)=EF + EG et (E + F)G = EG + FG
- E + ∅ = ∅ + E = E
- E∅=∅E=∅
- Eε=εE=E
- E + E = E
- (E*)*=E*

Quelques identités (liste très partielle !) qui relient l'étoile et les autres opérations :

- $(E + F)^* = (E^*F)^*E^* = E^*(FE^*)^*$ (1)
- $(EF)^* = \varepsilon + E(FE)^*F$ (2)
- $E^* = \varepsilon + EE^* = \varepsilon + E^*E$ (3)

  Remarque : souvent on note $EE^* = E^*E = E^+$. Dans cette notation, on a $E^* = \varepsilon + E^+$).
  (il est très facile de montrer (2) et (3), tandis que (1) est moins évident)

En combinant ces identités, on peut obtenir d'autres. Par exemple, on peut obtenir $(a^*b)^*a^+ + \varepsilon = (b^*a)^*$ de façon suivante :

$(a^*b)^*a^+ + \varepsilon = (a^*b)^*a^*a + \varepsilon$	utilisons (1) avec E=a, F=b
$= (a+b)^*a + \varepsilon = (b+a)^*a + \varepsilon$	et encore une fois
$= (b^*a)^*b^*a + \varepsilon$	utilisons (3)
$= (b^*a)^*$	

Résolution d'équations

Lemme d'Arden, sans démonstration

Soit Y et Z deux parties de A* où Y ne contient pas ε. Alors X=Y*Z est l'unique solution de l'équation

$X = YX + Z$ (on peut écrire $X = YX + Z \Leftrightarrow X = Y^*Z$) (4a)

et X=ZY* est l'unique solution de l'équation

$X = XY + Z$ (on peut écrire $X = XY + Z \Leftrightarrow X = ZY^*$) (4b)

Remarque Si Y contient ε, une solution de (4a) et de (4b) est donnée par A* ; Y*Z (pour 4a) [ou ZY* (pour 4b)] n'est plus l'unique solution, c'est la plus petite solution de (4a) [ou (4b)].

Théorème de KLEENE (les années 50).

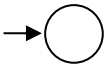
Un langage est reconnaissable par automate fini *ssi* il peut être décrit par une expression rationnelle.

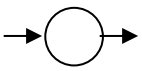
La preuve doit se faire séparément dans chacune des directions en donnant un algorithme pour passer des expressions rationnelles aux automates finis et inversement.

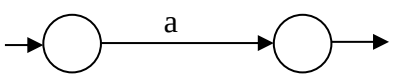
Expression rationnelle → Automate fini

On construit pour chaque expression rationnelle un automate asynchrone qui reconnaît le langage défini par cette expression.

Eléments de base :

Pour x=∅, on marque  (cet automate ne reconnaît *aucun* mot)

Pour x = ε, on marque : 

Pour x = a, a∈A, on marque : 

En ce qui concerne le reste des opérations, on doit choisir un jeu de règles bien défini, sinon on ne pourra pas programmer la construction d'un automate suivant une expression rationnelle. Le choix de ce jeu de règles est assez libre, mais une fois fait, il faut le respecter. Dans ce cours, on impose la règle selon laquelle

1. tout automate a un unique état initial
2. tout automate a un unique état terminal.

Tout automate construit selon nos règles, aura aussi des propriétés suivantes :

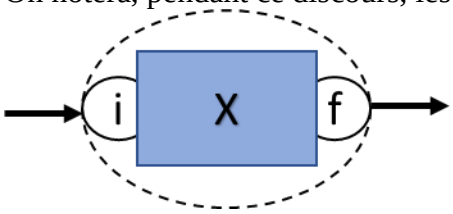
1. dans tout état entrent deux flèches au maximum,
2. de tout état sortent deux flèches au maximum,

La construction d'un AF suivant les règles est basée sur la traduction des trois opération figurant dans l'algèbre de ER en termes d'automates. Cette construction est modulaire et itérative (ou récursive): du point de vue itératif, nous construisons d'abord les éléments les plus petits et nous les combinons en unités plus grandes, et du point de vue récursif, nous considérons d'abord les opérateurs les plus externes de l'ER et nous descendons, détaillant chaque unité interne jusqu'à ce que nous atteignions les éléments de base. Cette phrase deviendra plus claire à mesure que nous décrirons la procédure en détail.

Opérations +, produit (concaténation) et * :

Soient X, Y deux expressions rationnelles. On suppose déjà construits pour X et Y les automates A et B qui vérifient les trois règles que nous venons de spécifier. On suppose qu'aucune flèche étiquetée ne sort de l'état terminal (ici on ne parle pas de la flèche sans étiquette qu'on utilise pour marquer l'état final, dont la signification est tout à fait différente).

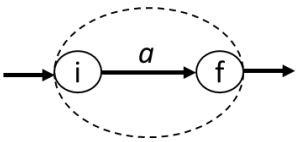
On notera, pendant ce discours, les automates (par exemple A_X) de façon suivante :



où le rectangle bleu représente l'automate X déjà construit, différent pour des X différents, sauf ses états initial et final qu'on dessine explicitement.

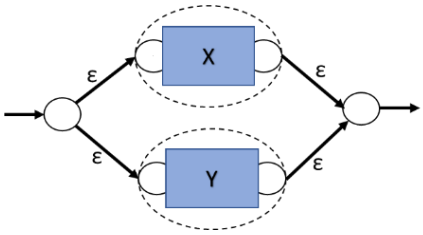
Pour que vous puissiez mieux comprendre de quoi il s'agit, voici le seul exemple qu'on peut donner à ce stade car pour l'instant on n'a pas construit aucun automate sauf les éléments de base :

Si X=*a*, le résultat est

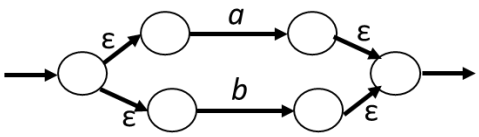


Maintenant nous sommes prêts à construire des automates résultant des trois opérations que nous avons introduites : X+Y, XY et X*.

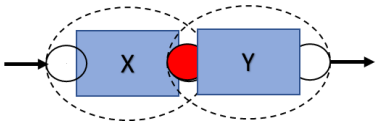
Opération + : X+Y est représenté par l'automate asynchrone suivant:



Par exemple, $a+b$ résulte en l'automate suivant :

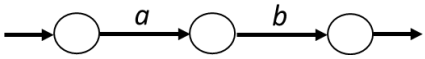


Opération $\cdot$: XY=X$\cdot$Y est représenté par l'automate asynchrone suivant :

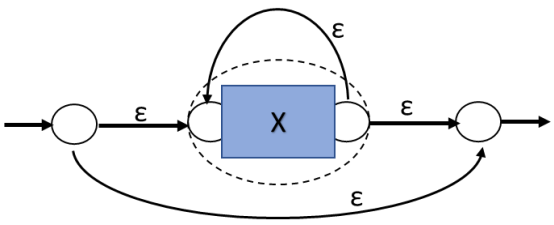


où l'état final de A_X est ***identifié*** avec l'état initial de A_Y (le cercle rouge).

Par exemple, *ab* résulte en l'automate suivant (qui, exceptionnellement, n'est pas asynchrone)



Operation * : est représenté par l'automate asynchrone suivant :



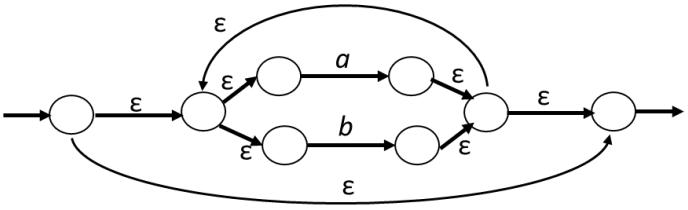
On peut décrire cette construction de façon suivante nous ajoutons

- deux ε-transitions à gauche et à droite de A_X, ce qui veut dire que nous ajoutons deux états (qui deviennent l'état initial and l'état final de A_{X^*}),

- une ε-transition partant de l'état final de A_X vers son état initial (une ε-transition inverse), et
- une ε-transition partant du nouvel état initial vers le nouvel état final (une ε-transition directe).

La transition ε inverse est toujours complètement à l'intérieur de la transition ε directe.

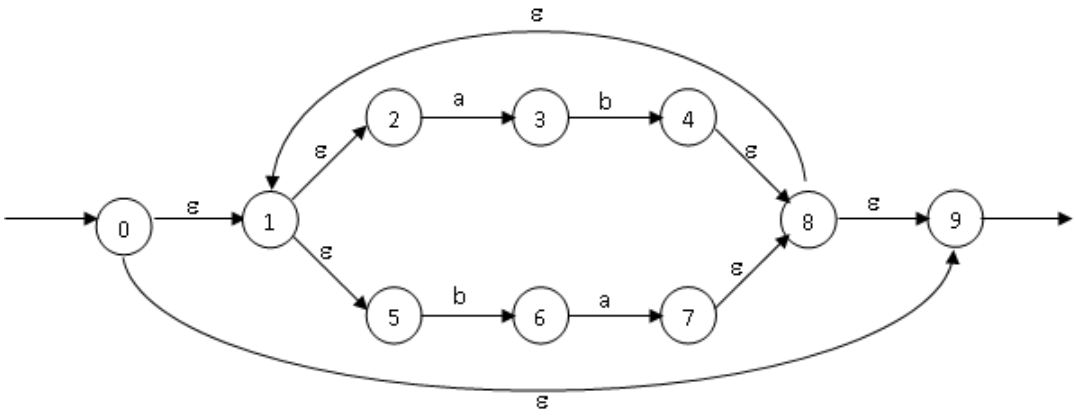
Par exemple, $(a+b)$ résulte en l'automate suivant:



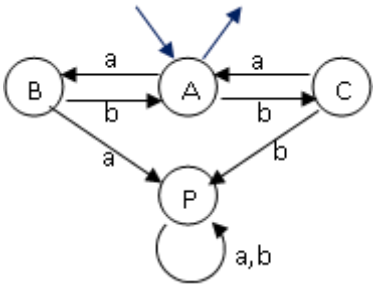
Prenons un exemple un peu plus compliqué et poussons l'analyse jusqu'au bout, c.à.d. jusqu'à l'obtention de l'automate minimal reconnaissant le langage donné :

$X=(ab + ba)^*$

En suivant les règles, on obtient



Cet automate asynchrone peut être déterminisé et minimisé, avec l'ADC suivant comme résultat :



Voici le détail des opérations :

(i) Déterminisation

L'état initial de l'automate déterministe est composé d'états 0,1,2,5,9 (les états où on peut arriver en lisant le mot vide). Les états terminaux sont ceux qui contiennent l'état 9. On introduit une poubelle pour que l'automate déterministe obtenu soit complet.

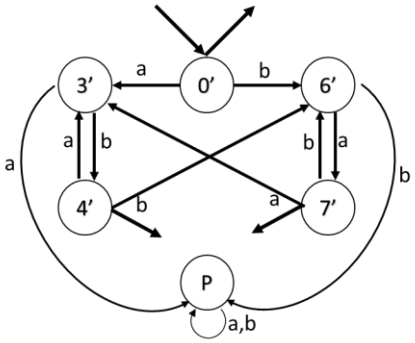
Ou bien, en utilisant les ε-clôtures,

Etat	a	b
(01259)	3	6
3	P	(124589)
6	(125789)	P
(124589)	3	6
(125789)	3	6
P	P	P

Etat	a	b
0'	3'	6'
3'	P	4'
6'	7'	P
4'	3'	6'
7'	3'	6'
P	P	P

L'état initial : i = (01259) = 0', les états terminaux T = {(01259),(124589),(125789)}={0',4',7'}.

Voici l'image graphique de l'automate déterminisé :



(ii) Minimisation

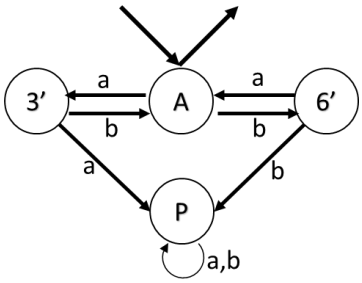
Groupe d'états non terminaux : NT = {3', 6', P}. Groupe d'états terminaux : T = {0',4', 7'}.

Etat	a	b	a	b
3'	P	4'	NT	T
6'	7'	P	T	NT
P	P	P	NT	NT
0'	3'	6'	NT	NT
4'	3'	6'	NT	NT
7'	3'	6'	NT	NT

Les états 3', 6' et P se séparent. Les états 0' , 4' et 7' ne peuvent jamais se séparer car ils mènent aux mêmes états de l'automate initial (3' en lisant a, et C'6' en lisant b).
On obtient donc 4 groupes de la partition finale : 3', 6', P et (0'4'7'). Pour simplifier la notation, on va utiliser A au lieu de (0'4'7').
La table de transitions de l'automate minimal est donc

avec un seul état initial et un seul état terminal A. On retrouve donc l'image graphique suivant :

	Etat	a	b
E/S	A	3'	6'
	3'	P	A
	6'	A	P
	P	P	P



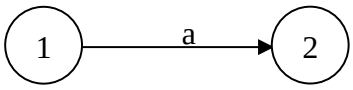
III. Automate fini → Expression rationnelle

Il y a plusieurs méthodes pour déterminer le langage reconnu par un automate. Ici nous allons présenter trois méthodes, dont deux basées sur la construction d'un système d'équations pour les ER et l'utilisation du lemme d'Arden dans une de ses deux formes. Les deux systèmes d'équations en résultant s'appellent le **système de l'arrivée** et **le système du départ.**

Tout en présentant les deux méthodes bas »es sur le lemme d'Arden, nous n'allons utiliser que le système de l'arrivée en TD.

On n'a pas besoin de déterminiser l'automate pour utiliser ces méthodes. Mais elles ne sont pas directement appicables à des automates asynchrones : il faut se débarasser des transitions ε d'abord.

Méthode 1 : Système de l'arrivée

- Chaque état représente une expression rationnelle (***langage de l'arrivée de cet état)*** correspondant à l'ensemble des mots qu'on peut lire pour l'atteindre à partir d'un état initial). Par exemple, si on a
- 

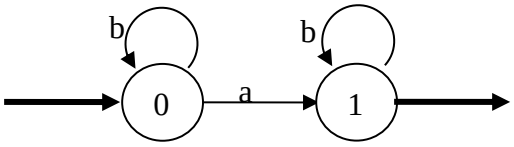
et l'état 1 correspond à l'expression rationnelle X_1 et l'état 2 à X_2, alors, s'il n'y a pas d'autre transition arrivant en 2, $X_2 = X_1a$. *(Cela veut dire : l'état 2 lit tous les mots lus par l'état 1 suivis de a).*

- La flèche non étiquetée qui entre dans un état initial correspond à ε. *(Un état d'entrée lit certainement le mot vide !)*
- On peut utiliser la loi distributive (du genre (a+b)c=ac+bc).
- εX=Xε=X
- On peut utiliser la règle (4b) (le lemme d'Arden) :

$$X = XY + Z \Leftrightarrow X = ZY^* \quad (*)$$
$$\varepsilon \notin Y$$

- Le langage reconnu par l'automate est la somme des langages reconnus par ses états terminaux.

Exemple A1 :



Le premier état, qu'on note X0, reçoit les flèches suivantes :

- Une flèche initiale (ε) (il lit le mot vide),
- Une flèche partant de lui-même, marquée b, ce qui donne X_0b.

Le second état, noté X_1, reçoit une flèche partant de X_0, marquée a, ce qui donne X_0a, et une partant de lui-même, marquée b, ce qui donne X_1b.

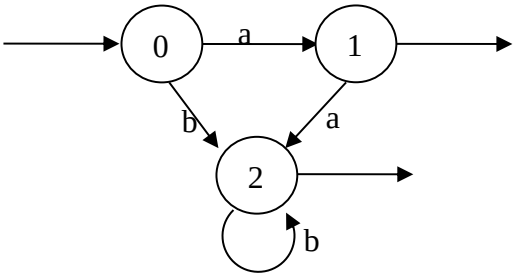
On a donc les équations suivantes :

$X_0 = \varepsilon + X_0b$ (0)

$X_1 = X_0a + X_1b$ (1)

Le lemme d'Arden appliqué à (0) donne $X_0 = \varepsilon b^* = b^*$, puis $X_1 = X_0a + X_1b = b^*a + X_1b$, et une nouvelle utilisation du lemme d'Arden donne $X_1 = b^*ab^*$; 1 étant le seul état terminal, le langage reconnu par l'automate est $L = b^*ab^*$.

Exemple A2:



Les équations ici sont :

$X_0 = \varepsilon$, (3)

$X_1 = X_0a$, (4)

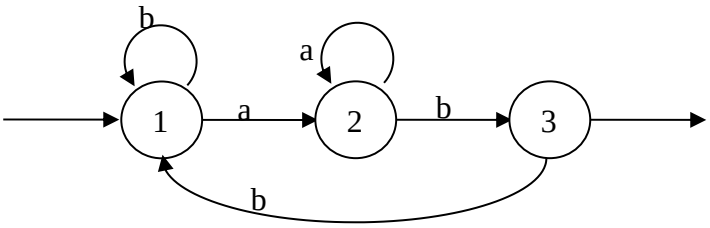
$X_2 = X_0b + X_1a + X_2b$. (5)

On obtient : $X_1 = X_0a = \varepsilon a = a$,

$X_2 = X_0b + X_1a + X_2b = \varepsilon b + aa + X_2b = X_2b + (b+aa) = (b+aa)b^*$, où on a appliqué le lemme d'Arden.

Le langage reconnu par l'automate est $L = X_1 + X_2 = a + (b+aa)b^*$.

Exemple A3, plus compliqué :



Le premier état (état 1) reçoit des flèches suivantes :

- Une flèche initiale (ε),
- Une flèche partant de lui-même, marquée b, ce qui donne X_1b,
- Une flèche partant de l'état 3, marquée b, ce qui donne X_3b.

On obtient donc une équation
$X_1 = X_1b + X_3b + \varepsilon$.
En continuant de la même façon pour deux autres états, on obtient un système d'équations :

$X_1 = X_1b + X_3b + \varepsilon$. (6)
$X_2 = X_1a + X_2a$ (7)
$X_3 = X_2b$ (8)

Remplaçant (8) dans (6), on obtient
$X_1 = X_1b + X_2bb + \varepsilon$ (9)

Appliquant la règle (*) à (7), on obtient
$X_2 = X_1a + X_2a \Leftrightarrow X_2 = X_1aa^*$ (10)
Remplaçant X_2 dans (9), on obtient
$X_1 = X_1b + X_1aa^*b^2 + \varepsilon$
$= X_1(b + aa^* b^2) + \varepsilon$ (11)
Appliquant la règle (*) à (11), on obtient
$X_1 = \varepsilon (b + aa^*b^2)^* = (b + aa^*b^2)^*$ (12)
et puis, de (10)
$X_2 = (b+aa^*b^2)^*aa^*$ (13)
et de (8)
$X_3 = (b + aa^*b^2)^*aa^*b$ (14)

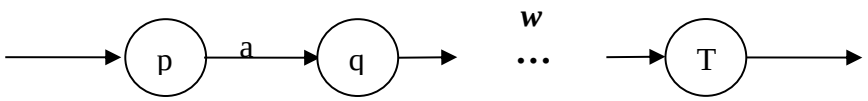
L'état 3 étant l'état terminal, l'automate reconnait le langage décrit par l'expression rationnelle $(b + aa^*b^2)^*aa^*b$.
Ce n'est pas la seule forme possible de cette expression! En effectuant les opération dans un ordre différents, on aurait pu obtenir une autre expression équivalente (exprimant le même langage).

A. Système du départ (ne pas traiter en cours !)

- Chaque état représente maintenant une expression rationnelle (***langage du départ de cet état)*** correspondant au langage qui serait reconnu par l'automate si cet état était la seule entrée. **Attention ! On ne parle plus du langage « lu par l'état » !**

- Dans cette optique, une sortie correspond au mot vide : si l'état terminal en question est initial, l'automate lit certainement le mot vide.

Le langage correspondant à un état est la somme des mots qui peuvent arriver à une sortie si cet état est initial ; si l'état ***p*** est lié à un autre état, ***q***, par une transition ***p.a.q***, alors, si le mot ***w*** fait partie du langage X_q correspondant à ***q,*** le mot ***aw*** fait partie du langage X_p correspondant à ***p***.



(Ici, ***q*** et ***T*** sont liés par un ***chemin*** $q \xrightarrow{w} T$, ce chemin pouvant consister en plusieurs transitions).

On peut exprimer ceci par une formule :

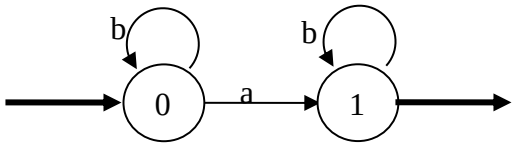
- Dans la méthode du langage du départ, si l'état ***p*** a les transitions $p.a_i.q_i$,,l'expression rationnelle lui correspondant est égale à $\sum_i a_i q_i$. Evidemment, cela s'applique, en particulier, dans le cas où un des q_i est l'état ***p*** lui-même. Ainsi, une boucle en ***a*** sur l'état ***p*** correspond au terme ***ap***. (Attention ! Dans cette méthode, tous les caractères sont situés - des RE correspondant à des états !)

- Comme dans la méthode de l'arrivée, on peut utiliser le lemme d'Arden, mais cette fois dans sa version (4a) :

$$X = YX + Z \Leftrightarrow X = Y^*Z \qquad (**)$$
$$\varepsilon \notin Y$$

- Le langage reconnu par l'automate est la somme des langages reconnus par les états **initiaux**.

Exemple B1 : on prend le même automate que celui de l'exemple A1 :



Le système d'équation s'écrit :

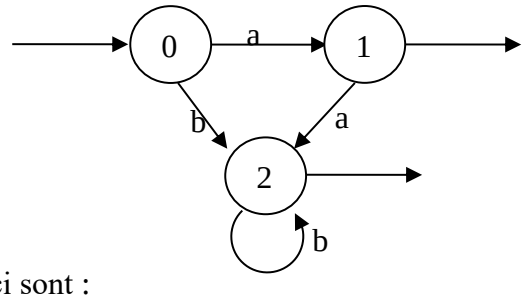
$X_0 = aX_1 + bX_0$ (15)

$X_1 = \varepsilon + bX_1$ (16)

(16) résulte en $X_1 = b^*\varepsilon = b^*$, et en le mettant dans (15), on obtient $X_0 = ab^* + bX_0$.

En appliquant le lemme d'Arden (**) encore une fois, on a $X_0 = b^*ab$. L'état 0 étant la seule entrée, c'est le langage reconnu par l'automate : $L = b^*ab$. Nous avons obtenu exactement la même chose qu'avec la méthode de l'arrivée.

Exemple B2 : on prend le même automate que celui de l'exemple A2 :



Les équations ici sont :

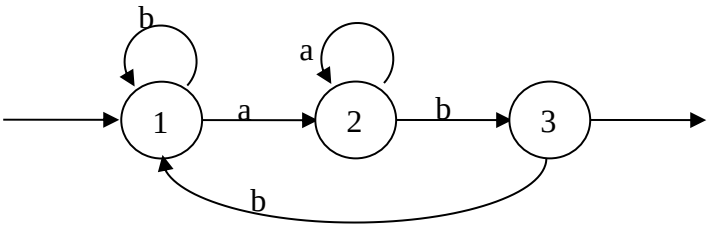
$X_0 = aX_1 + bX_2$, (17)

$X_1 = \varepsilon + aX_2$, (18)

$X_2 = \varepsilon + bX_2$. (19)

(19) résulte en $X_2 = b^*\varepsilon = b^*$, en le mettant dans (18) on obtient $X_1 = \varepsilon + ab^*$, et finalement $X_0 = a(\varepsilon + ab^*) + bb^* = a + aab^* + bb^*$, ce qui est la même expression que celle qu'on a obtenu avec la méthode de l'arrivée.

Exemple B3 : on prend le même automate que celui de l'exemple A3 :



Les équations:

$X_3 = \varepsilon + bX_1$, (20)

$X_2 = aX_2 + bX_3$, (21)

$X_1 = bX_1 + aX_2$. (22)

En mettant (20) dans (21), on obtient $X_2 = aX_2 + b + bbX_1$ (23)

En utilisant le lemme d'Arden sur (22), on obtient $X_1 = b^*aX_2$ (24)

En utilisant (24) dans (23), on obtient $X_2 = aX_2 + b + bbb^*aX_2$ (25)

Réécrivant (25) comme $X_2 = (a + bbb^*a)X_2 + b$ et en utilisant le lemme d'Arden, on obtient $X_2 = (a + bbb^*a)^*b$.

Donc, $X_1 = b^*a(a + bbb^*a)^*b$, et l'état 1 étant la seule entrée, $L = b^*a(a + bbb^*a)^*b$. (26)

On a obtenu une expression rationnelle différente de celle obtenue dans l'exemple A3 (éq. 14 : $L = (b + aa^*b^2)^*aa^*b$.)

Peut-on obtenir la même expression que celle de l'éq. 14 ? Dans ce cas particulier, ce n'est pas trop compliqué : il suffit de commencer par l'application du lemme d'Arden à (21) au lieu de (22) :
$X_2 = a^*bX_3$, et en replaçant X_3 par (20), $X_2 = a^*b + a^*bbX_1$.
Maintenant remplaçant X2 dans (22), on obtient
$X_1 = bX_1 + a(a^*b + a^*bbX_1) = bX_1 + aa^*b + aa^*bbX_1 = (b + aa^*bb)X_1 + aa^*b$,
d'où

X1 = (b+aa*bb)*aa*b, et on retrouve l'expression (14).

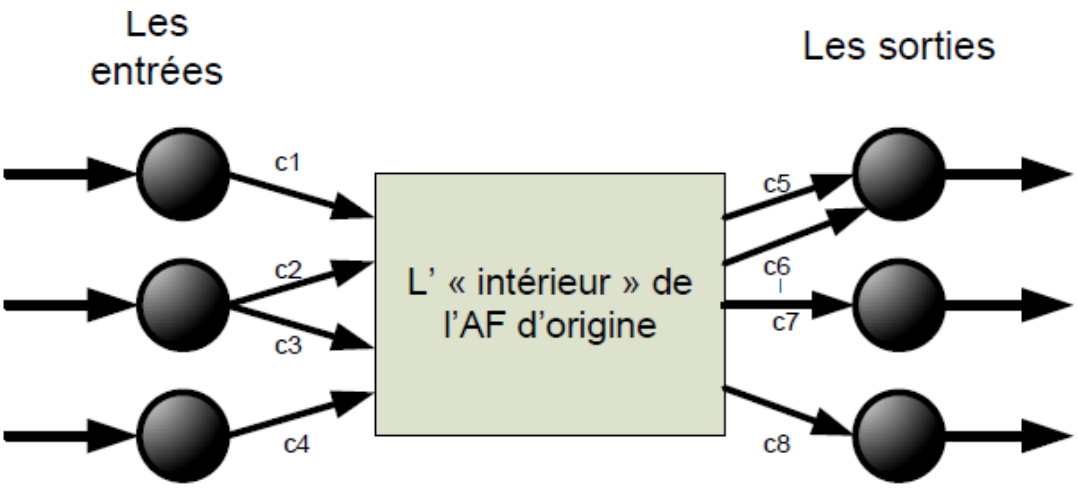
En résolvant un système d'équations sur des expressions rationnelles dans un ordre différent, ou par des méthodes differentes, on peut obtenir un résultat différent. De tels résultats différents seront, bien sûr, équivalents, mais il est difficile de le voir immédiatement car l'algèbre des expressions rationnelles est fort compliquée (nous en avons vu un seul exemple – les règles (4a,b)). Une des meilleures façons de voir que deux expressions rationnelles sont équivalentes est de construire à partir d'elles deux automates asynchrones, les déterminiser et minimiser et arriver au même automate minimal.

B. Méthode d'élimination d'états

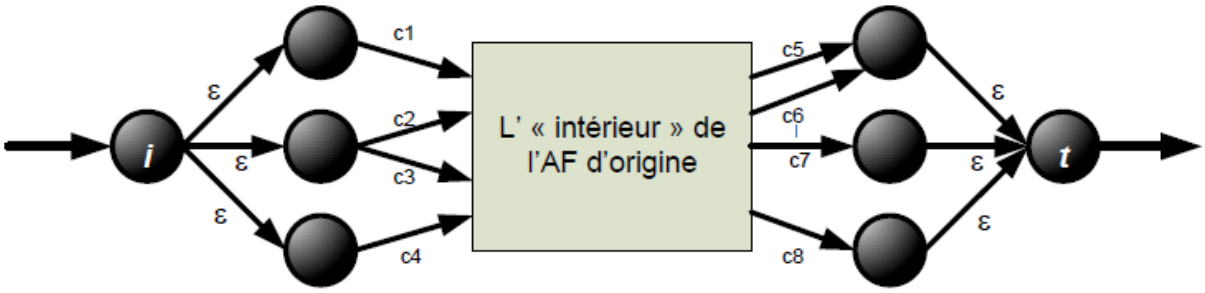
On définit d'abord un **Automate Fini Généralisé** : la seule différence avec un AF « normal » consiste en ce que les transitions peuvent être marquées par n'importe quelle expression rationnelle.

Voici la méthode :

1) On prend l'AF dont le langage on veut obtenir en tant qu'expression rationnelle. On y ajoute un état d'entrée ***i*** et un état de sortie ***t***, on crée des ε-transitions partant de l'état ***i*** vers toutes les entrées de l'automate d'origine, et arrivant dans l'état ***t*** de toutes les sorties de l'automate d'origine :

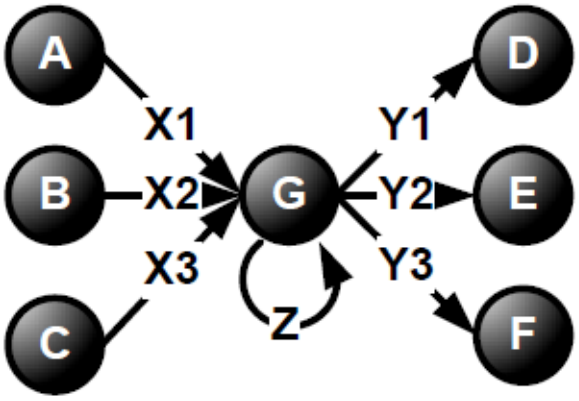


Devient



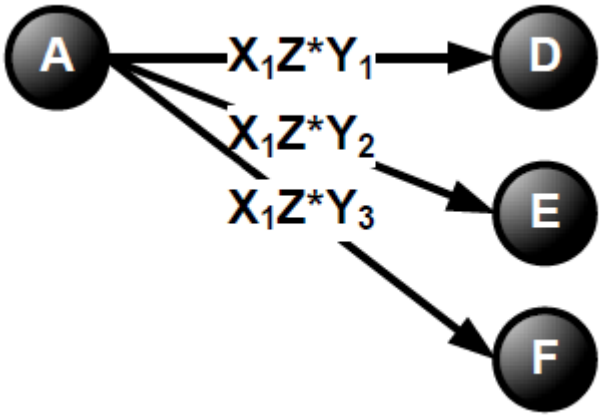
2) On « élimine » tous les états sauf ***i*** et ***t*** l'un après l'autre, dans un ordre quelconque, de la façon suivante : soit l'état G qu'on veut éliminer, avec des transitions entrantes, transitions sortantes et éventuellement avec une boucle. Dans le cadre d'un AF généralisé, je marque toutes ces transitions par des expressions rationnelles X_i, Y_i, Z :

(Il faut remarquer que parmi les états D,E, F certains peuvent coïncider avec certains des états A, B, C)



On liste tous les chemins passant par G : AGD, AGE, AGF, BGD, BGE, BGF, CGD, CGE, CGF. Chaque chemin donne lieu à une ER du type $X_iZ^*Y_j$. On obtient des transitions généralisées $A.X_1Z^*Y_1.D$, $A.X_1Z^*Y_2.E$, $A.X_1Z^*Y_3.F$, $B.X_2Z^*Y_1.D$ etc. :

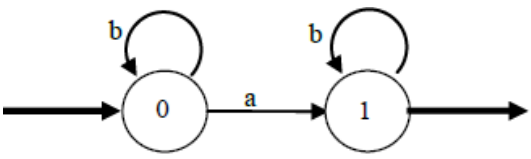
(je n'ai pas dessiné les états B et C et leurs transitions généralisées vers D, E et F pour ne pas encombrer le dessin).



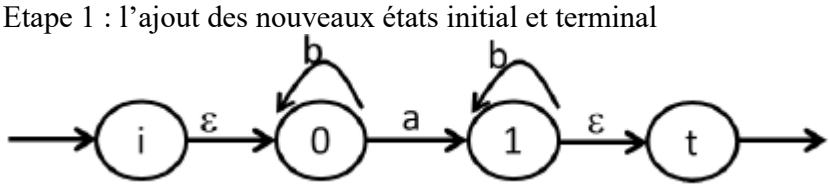
On procède ainsi jusqu'à l'élimination de tous les états sauf ***i*** et ***t*** . L'ER marquant la transition généralisée reliant ***i*** et ***t*** est le langage reconnu par l'automate.

Prenons les mêmes exemples qu'auparavant.

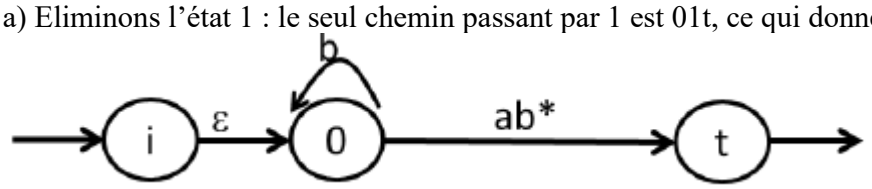
Exemple C1 : on prend le même automate que celui de l'exemple A1 :



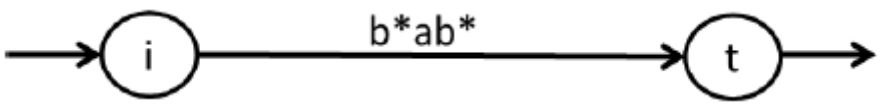
Etape 1 : l'ajout des nouveaux états initial et terminal



Etape 2, élimination itérative d'états :
a) Eliminons l'état 1 : le seul chemin passant par 1 est 01t, ce qui donne l'ER $ab^*\varepsilon=ab^*$:

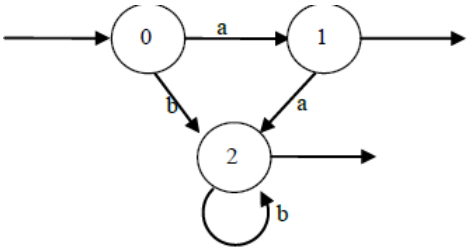


b) Eliminons l'état 0 : le seul chemin passant par 1 est i0t, ce qui donne l'ER $\varepsilon b^*ab^*=b^*ab^*$:

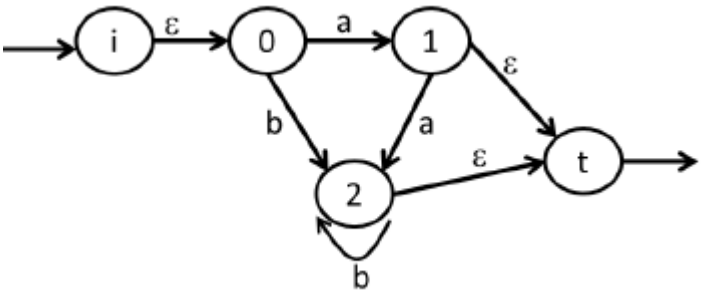


L = b*ab*.

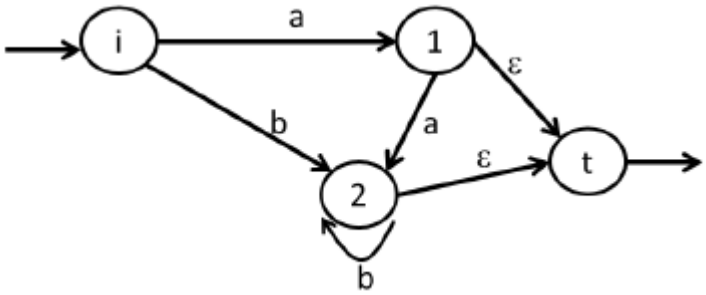
Exemple C2 : on prend le même automate que celui de l'exemple A2 :



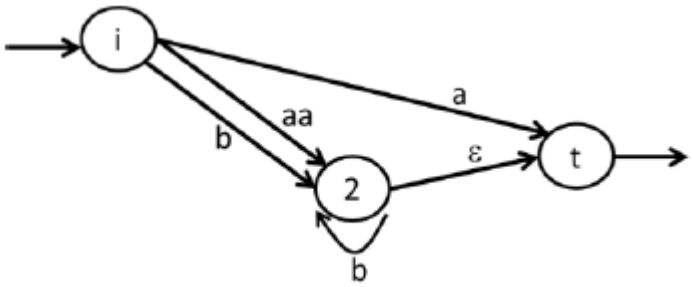
Etape 1 :



Etape 2 :
Eliminons 0 : il y a 2 chemins passant par 0, ce qui donne iεa1=ia1, iεb2=ib2



Eliminons 1: il y a 2 chemins passant par 1, ce qui donne iaεt=iat, iaa2



Eliminons 2: il y a un chemin passant par 2, ce qui donne i(aa+b)b*εt=i(aa+b)b*t. Avec la transition déjà existante iat cela donne i(a+(aa+b)b*)t et donc L = a+(aa+b)b* = a + aab* + bb*.