INF 232
Automates et langages

Yassine Lakhnech

Yassine.Lakhnech@imag.fr

Bibliographie

- J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages and Computation, 2nd edition, Addison-Wesley, 2001
- P. Wolper. Introduction à la calculabilité - Paris : InterEditions, 1991.
- Cl.Benzaken, Systèmes Formels, Masson, 1991.

Auomates d'états finis

Les objectifs de ce cours

- Méthodologies et approche scientifique:
  - La modélisation mathématiques de problèmes informatiques.
  - La recherche de solutions.
  - L'analyse des solutions.
  - La présentation des solutions.
- Connaissances spécifiques:
  - Différents formalismes pour la définition des langages formelles: automates, expressions régulières et grammaires.
  - Leur étude d'un point de vue algorithmique: problèmes de décision.
  - L'étude de leur pouvoir expressif.

L’exemple d’une transaction éléctronique

Nous voulons modéliser une transaction à laquelle participent:

- un client,
- un marchand et
- une banque.

Le client veut acheter une marchandise chez la marchand et la paye à l’aide d’une monnaie (chèque) éléctronique.

Une monnaie éléctronique est tout simplement une sorte de chèque qu’on peut envoyer par email.

Les actions

Les actions de ces trois participants sont les suivantes:

- Le client peut payer sa marchandise en envoyant au marchand l’argent sous la forme d’un message éléctronique ($paye$).
- Il peut aussi abandonner la transaction et récupérer son argent (abd).
- Le marchand peut envoyer la marchandise au client (env).
- Le marchand peut solder le chèque éléctronique (sol).
- La banque peut transférer l’argent au marchand (tra).

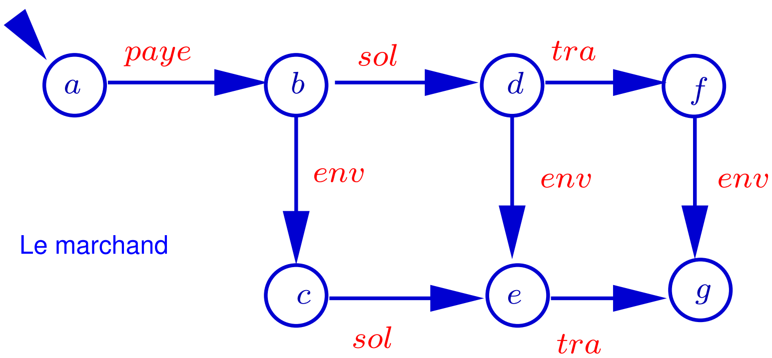
Hypothèses sur le comportement des participants

On va supposer que

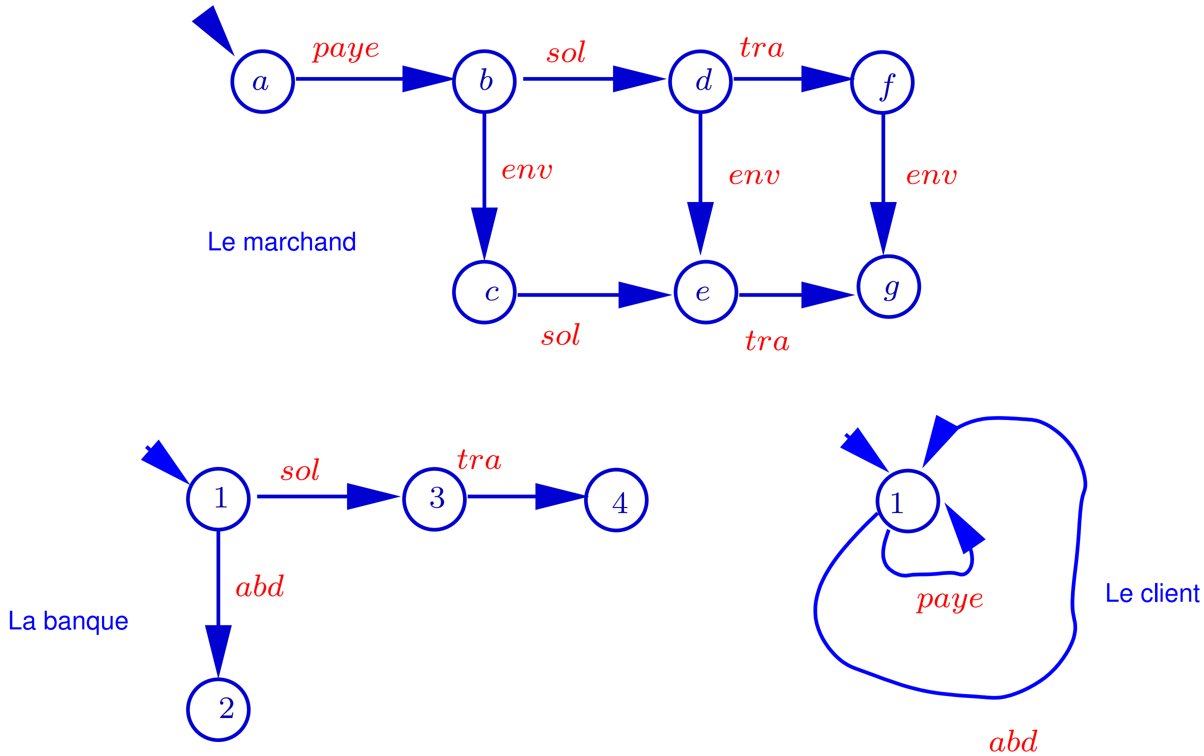
- la banque se comporte de manière honnête.
- le marchand doit faire attention à ne pas livrer la marchandise sans être payé.
- Le client va essayer de recevoir la marchandise tout en récupérant son argent.

On veut donc modéliser le comportement des trois participants et voir s'il y a un moyen pour le client de recevoir la marchandise sans payer.

Modélisation des participants



Modélisation des participants



Vérification du modèle

Approche:

- On calcule un automate qui modélise les comportements globaux des trois participants c'est-à-dire la composition de leur comportements.
- On utilise un algorithme pour savoir si certains comportements indésirables apparaissent dans l'automate produit. Par exemple, si des états indésirables sont accessibles à partir de l'état initial.

Exercice: Dennez un automate qui modélise la composition des trois participants.

Étude des automates

Pour faire ceci nous avons besoin d'étudier les automates pour savoir:

- Ce qu'on peut et ce qu'on ne peut pas décrire avec un automate.
- Définir des opération pour composer des comportements.
- Développer des algorithmes sur les automates.

Autres motivations pour étudier les automates

Une question fondamentale en informatique est la suivante:
Quels problèmes peut-on résoudre à l'aide d'une "machine à calculer"?

Autrement dit: Existe-t-il une limite à ce qu'on peut programmer?

Quelle est cette limite si elle existe?

Quelques réponses

Alan Turing a étudié cette question en 1930 avant même qu'il existe le premier ordinateur. Il a étudié une machine abstraite, un automate appelé Machine de Turing. Sa réponse à ces questions s'applique à tous les modèles d'ordinateur qu'on connait aujourd'hui.
Les réponses sont:

- Oui il y a des problèmes qu'on ne peut pas résoudre à l'aide d'un ordinateur même si on suppose une mémoire non-bornée (illimitée).
- Cette limite est caratérisée de plusieurs manières équivalentes:
  - Automates.
  - Classes de fonctions.
  - Logiques.

Compléxité des problèmes et algorithmes

S. Cook a re-considéré en 1969 les mêmes questions que A. Turing mais en s'interessant à ce qui peut être calculé de manière efficace.

Les notions "d'efficacité" ainsi que les réponses à ces questions ont été développées à l'aide d'automates: on parle de problèmes P et de problèmes NP.
Une question qui reste aujourd'hui ouverte est:

$$P = NP.$$

Une petite introduction historique

Ces automates ont été définis dans les années 40 et 50. La motivation au départ était l'étude du cerveau humain.
Aujourd'hui ils sont utilisés dans les domaines suivants:

- Conception de matériels informatiques.
- Compilation des langages.
- Reconnaissance de texte.
- Conception des protocoles.
- Synthèse des programmes.
- Vérification des programmes.

Les Grammaires et les automates

Fin des années 50, N. Chomsky, un linguiste, a commencé l'étude des grammaires formelles.
Les liens entre les grammaires et les automates ont été étudiés par la suite. Nous verrons quelques résultats en cours.

Il existe aussi des résulats (surprenant) concernant des équivalences entre automates et logique (Büchi 1962, Elgot 1962, McNaughton 1966, M.O. Rabin 1969).

Automates déterministes

Alphabets, symboles

Definition
Un *alphabet* est un ensemble fini dont les éléments sont appelés *symboles*. □

Exemple Dans l'exemple de la transaction éléctronique l'alphabet est:

$$\{abd, paye, env, sol, tra\}.$$

□

On va utiliser la lettre Σ pour désigner un alphabet.

Mots

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot de longueur* $n \in \mathbb{N}$ est un application de $\{0, \cdots, n-1\}$ vers Σ.
- On note par $|u|$ la longueur du mot u.
- Le *mot vide* qui est la fonction de l'ensemble vide vers Σ est noté ϵ.
- L'ensemble de tous les mots sur Σ est noté Σ^*.
- Un *langage sur* Σ est un sous-ensemble de Σ^*.

□

Des exemples

Exemple Considérons l'alphabet $\Sigma = \{0.1\}$. Alors,

- ϵ est le mot de longueur 0.
- 0 et 1 sont les mots de longueur 1.
- 00, 01, 10 et 11 sont les mots de longueur 2.
- $\emptyset$ est un langage sur Σ. Il est appelé le *langage vide*.
- Σ^* est un langage sur sur Σ. Il est appelé le *langage universel*.
- $\{\epsilon\}$ est un langage sur Σ.
- $\{0, 00, 001\}$ est aussi un langage sur Σ.
- L'ensemble des mots qui contiennent un nombre impair de 0 est un langage sur Σ.
- L'ensemble des mots qui contiennent autant de 0 que de 1 est un langage sur Σ.

□

Concaténation de mots

Intuitivement, la concaténation des mots 01 et 10 est le mot 0110.
La concaténation du mot vide ϵ et le mot 101 est le mot 101.
Definition La *concaténation* est une application de $\Sigma^* \times \Sigma^*$ vers Σ^*.
La concaténation de deux mots u et v dans Σ^* est le mot $u \cdot v : \{0, \cdots, |u| + |v| - 1\} \to \Sigma$ tel que:

- $(u \cdot v)(i) = u(i)$, si $i \in \{0, \cdots, |u| - 1\}$ et
- $(u \cdot v)(i) = v(i - |u|)$, si $i \in \{|u|, \cdots, |u| + |v| - 1\}$.

$\square$

Concaténation de Langages

On peut étendre la concaténation des mots aux langages de la manière suivante:

$$\cdot : \mathcal{P}(\Sigma^*) \times \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^*)$$
$$L_1 \cdot L_2 = \{u_1 \cdot u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\}$$

- $L \cdot \emptyset = \emptyset$
- $L \cdot \Sigma^* \neq \Sigma^*$
- Si $\epsilon \in L$ alors $L \cdot \Sigma^* = \Sigma^*$
- $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$

Automates d'états finis déterministes

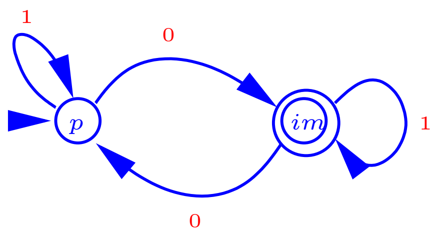
Definition Un *automate d'états finis déteministes (ADEF)* est donné par un quintuplet $(Q, \Sigma, q_0, \delta, F)$ où

- Q est un ensemble fini d'*états*.
- Σ est l'alphabet de l'automate.
- $q_0 \in Q$ est l'*état initial*.
- La fonction $\delta : Q \times \Sigma \to Q$ est la *fonction de transition*.
- $F \subseteq Q$ est l'ensemble des *états accepteurs*.

Un automate détérministe est appelé *complet* si sa fonction de transition δ est une *application*. □

Exemple : Nombre pair de 0

Exemple Ci-dessous un automate déterministe et complet qui reconnaît l'ensemble des mots qui contiennent un nombre impair de 0.



□

Configuration et exécutions

Soit $A = (Q, \Sigma, q_0, \delta, F)$.
Une *configuration* de l'automate A est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.
On définit la relation $\rightarrow$ de *dérivation* entre configurations:

$$(q, a \cdot u) \rightarrow (q', u) \text{ ssi } \delta(q, a) = q'.$$

Une *exécution de l'automate* A est une séquence de configurations

$$(q_0, u_0) \cdots (q_n, u_n) \text{ telle que}$$

$$(q_i, u_i) \rightarrow (q_{i+1}, u_{i+1}), \text{ pour } i = 0, \cdots, n-1.$$

Exemple

Exemple Soit $\Sigma = \{0, 1\}$. Donner un automate qui accepte tous les mots qui contiennent un nombre de 0 multiple de 3. Donnez une exécution de cet automate sur 1101010. □

Langage reconnu par un automate

Definition

- Un mot $u \in \Sigma^*$ est *accepté* par A, s'il existe une exécution $(q_0, u_0) \cdots (q_{n-1}, u_{n-1})$ de A telle que $u = u_0$, $u_{n-1} = \epsilon$ et $q_{n-1} \in F$.
- Le *langage reconnu par* A, qu'on note par $L(A)$, est l'ensemble $\{u \in \Sigma^* \mid u$ *est accepté par* $A\}$.
- un langage $L \subseteq \Sigma^*$ est appelé *langage d'états finis*, s'il existe un automate d'états finis déterministe qui reconnait L. La classe des langages d'états finis est dénotée EF.

□

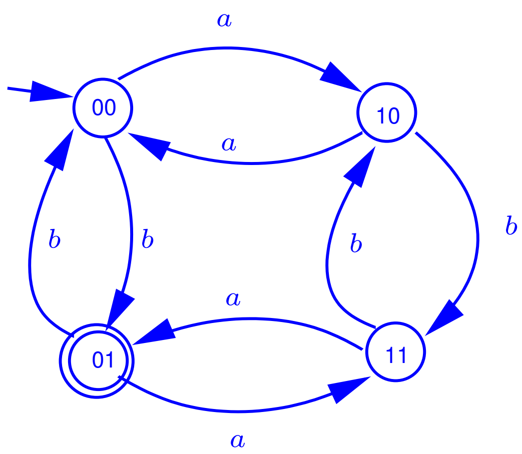
Exemples

Soit $\Sigma = \{a, b\}$.

- L'ensemble des mots dans lesquel b ne précède jamais a est-il un langage d'états finis?
- L'ensemble des mots dans lesquel A est toujours immédiatement suivi de b est-il un langage d'états finis?

Plus d'exemples

- Quel est le langage reconnu par l'automate suivant:



- Pour $k \in I\!N$ soit L_k l'ensemble des mots u telque $|u| < k$ et u contient le même nombre de a et de b.
    - L_k est-il un langage d'états finis?
    - $\bigcup_{k \in I\!N} L_k$ est-il un langage d'états finis?

Quelques Problèmes de décision

Nous allons commencer par considerer deux problèmes.

Problème 1:
Etant donné un ADEF A. Le langage reconnu par A est-il vide?

$$L(A) = \emptyset.$$

Problème 2:
Etant donné un ADEF A. Le langage reconnu par A est-il fini?

$$|L(A)| \in I\!N.$$

Décision du problème du langage vide

Théorème

- Le problème de l'accessibilité est décidable pour les graphes finis.
- Le problème du langage vide est décidable pour les automates déterministes d'états finis.
- Le problème de la finitude du langage est décidable pour les automates déterministes d'états finis.

Fermeture de EF par intersection et complémentation

Soit A un ADEF.

1. le langage $\Sigma^* \setminus L(A)$ est -il reconnaissable par un ADEF?
2. si oui peut-on construire de manière effective un automate qui reconnaît $\Sigma^* \setminus L(A)$?

Soient A et B deux ADEFs.
On se pose les questions suivantes:

1. le langage $L(A) \cap L(B)$ est-il reconnaissable par un ADEF?
2. si oui peut-on construire de manière effective un automate qui reconnait $L(A) \cap L(B)$?

Fermeture de EF par intersection et complémentation

Soit A un ADEF.

1. le langage $\Sigma^* \setminus L(A)$ est -il reconnaissable par un ADEF?
2. si oui peut-on construire de manière effective un automate qui reconnaît $\Sigma^* \setminus L(A)$?

Soient A et B deux ADEFs.
On se pose les questions suivantes:

1. le langage $L(A) \cap L(B)$ est-il reconnaissable par un ADEF?
2. si oui peut-on construire de manière effective un automate qui reconnait $L(A) \cap L(B)$?

Nous allons voir que nous pourrons répondre de manièrre affirmative à toutes ces questions.

Complétion d'automates

Soit $A = (Q, \Sigma, q_0, \delta, F)$ un ADEF.
On peut construire un ADEF complet qui reconnait $L(A)$ en ajoutant un nouveau état puit à Q.
Soit $C(A) = (Q \cup \{q_p\}, \Sigma, q_0, C(\delta), F)$ où $q_p \notin Q$ et tel que $C(\delta) : Q \times \Sigma \to Q$ est une application défini par:

$$\begin{array}{ll} C(\delta)(q,a) = \delta(q,a), & \textit{pour tout } (q,a) \in \mathcal{D}(\delta) \\ C(\delta)(q,a) = q_p, & \textit{sinon} \end{array}$$

On montre en TD qu'on a $L(A) = L(C(A))$. On montre aussi que pour chaque mot $u \in \Sigma^*$ il existe une exécution unique de $C(A)$ sur u.
Soit $A = (Q, \Sigma, q_0, \delta, F)$ un ADEF complet et soit $A^c = (Q, \Sigma, q_0, \delta, Q \setminus F)$. Alors,

$$L(A^c) = \Sigma^* \setminus L(A).$$

Procédure de complémetation

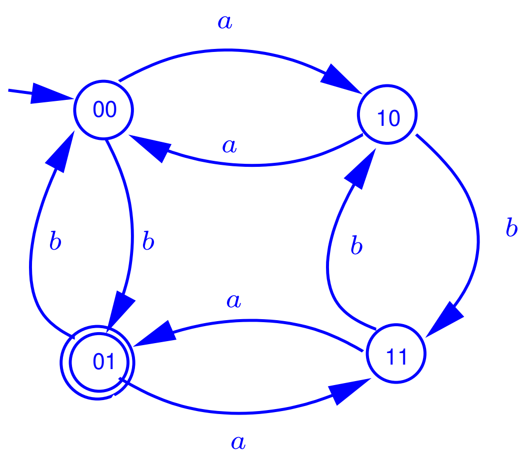
Donné un ADEF $A = (Q, \Sigma, q_0, \delta, F)$. Pour construire un automate qui reconnaît $\Sigma^* \setminus L(A)$, on suit les pas suivant:

- On construit $C(A)$.
- On obtient l'automate voulu en inversant les états accepteurs et non-accepteurs dans $C(A)$.

Exemple $\Sigma = \{a, b, c\}$. □

Procédure de complémetation

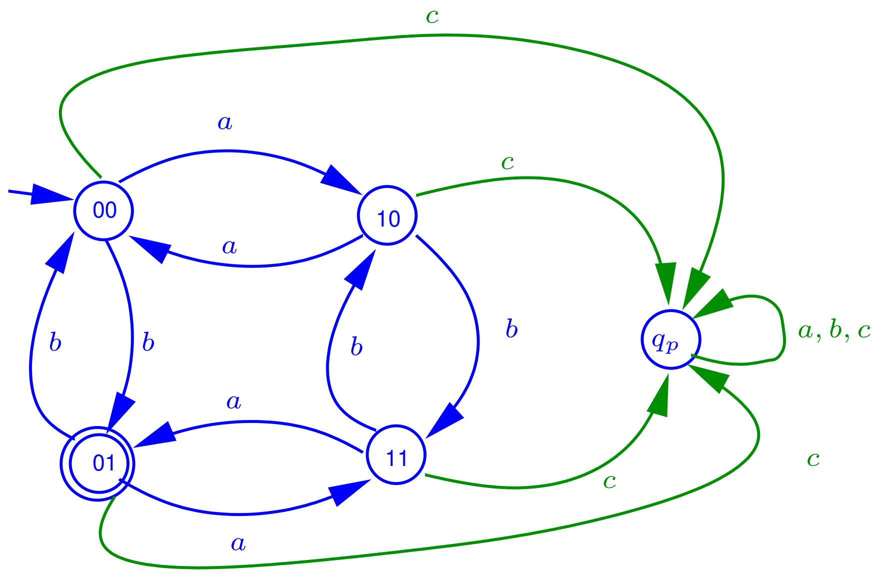
Exemple $\Sigma = \{a, b, c\}$.



□

Procédure de complémetation

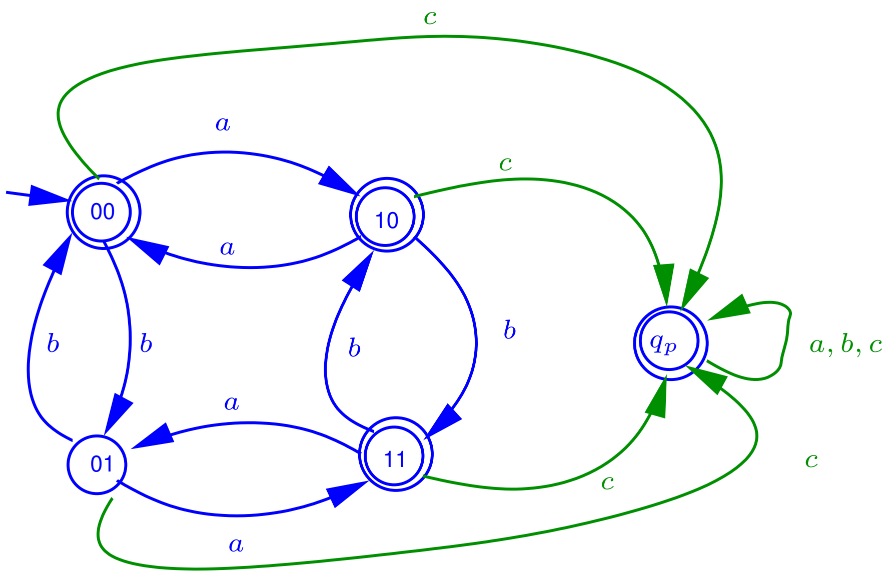
Exemple $\Sigma = \{a, b, c\}$.



□

Procédure de complémetation

Exemple $\Sigma = \{a, b, c\}$.



□

Fermeture par complément

Théorème

- La classe EF des langages d'états finis est fermée par complémentation.
- Il existe une procédure effective qui associe à un automate A un automate qui reconnait $\Sigma^* \setminus L(A)$.

Produit d'automates

Soient $A = (Q^A, \Sigma, q_0^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_0^B, \delta^B, F^B)$ deux ADEFs.
Definition *L'automate produit de A et de B* est donné par $A \times B = (Q, \Sigma, q_0, \delta, F)$ où:

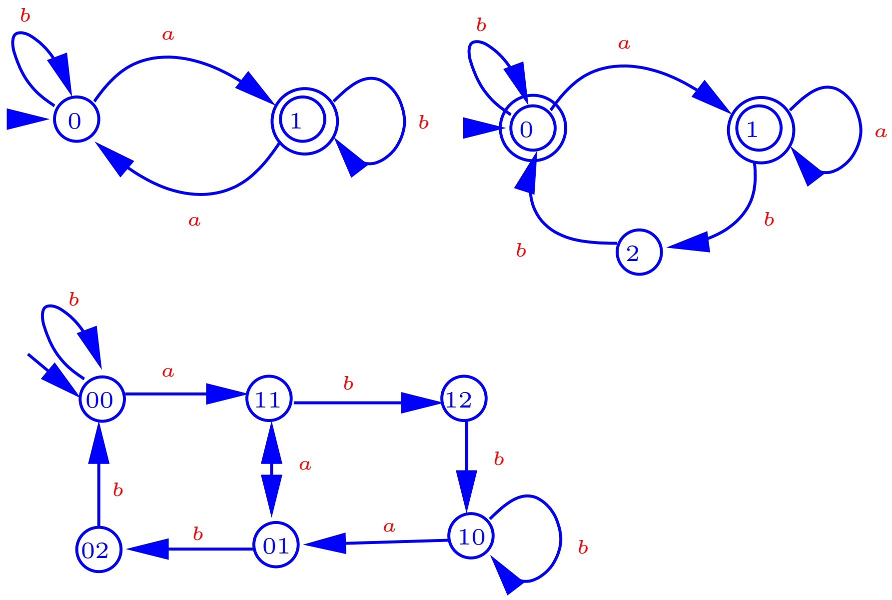
- $Q = Q^A \times Q^B$.
- $q_0 = (q_0^A, q_0^B)$
- $\delta : (Q^A \times Q^B) \times \Sigma \to (Q^A \times Q^B)$ est telle que

$$\delta((q^A, q^B), a) = (\delta^A(q^A, a), \delta^B(q^B, a)).$$

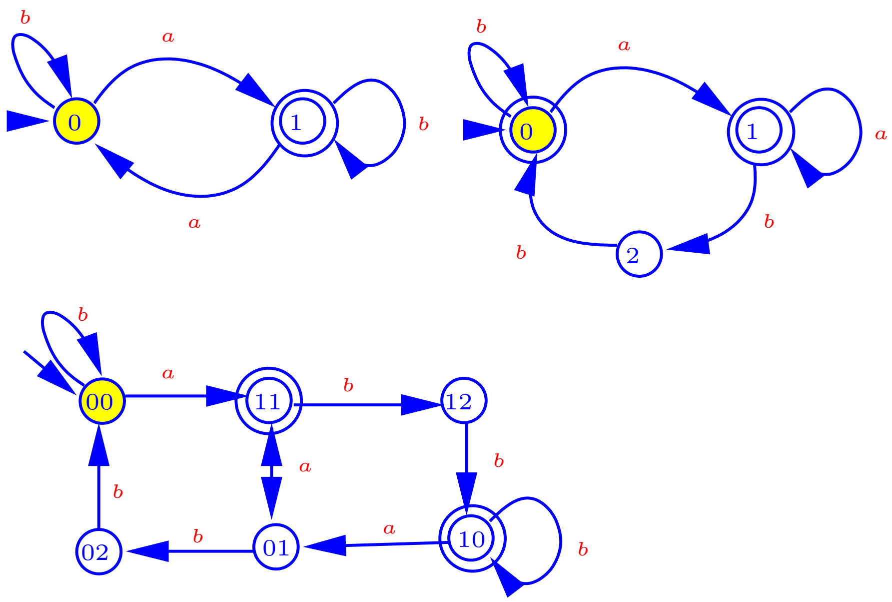
- $F = F^A \times F^B$.

$\square$

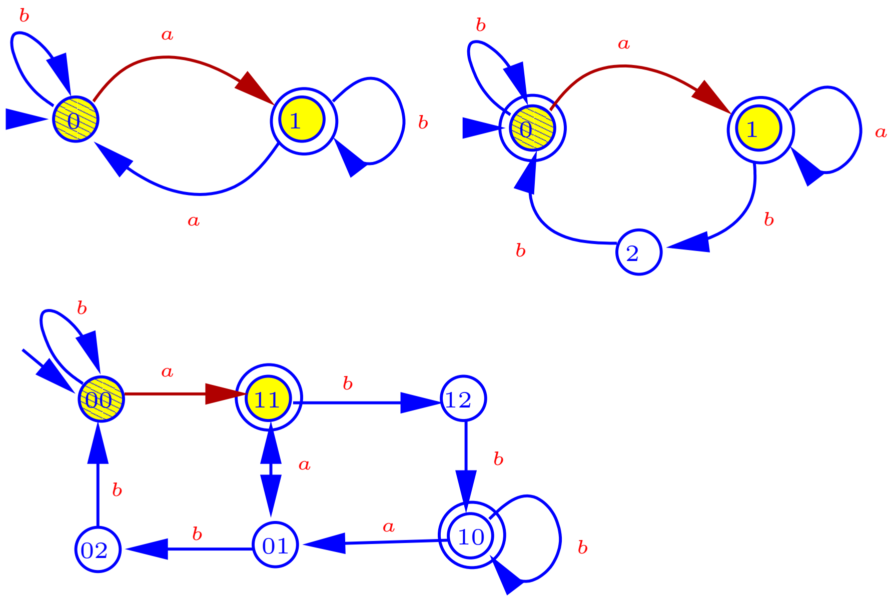
Exemple de produit d'automates



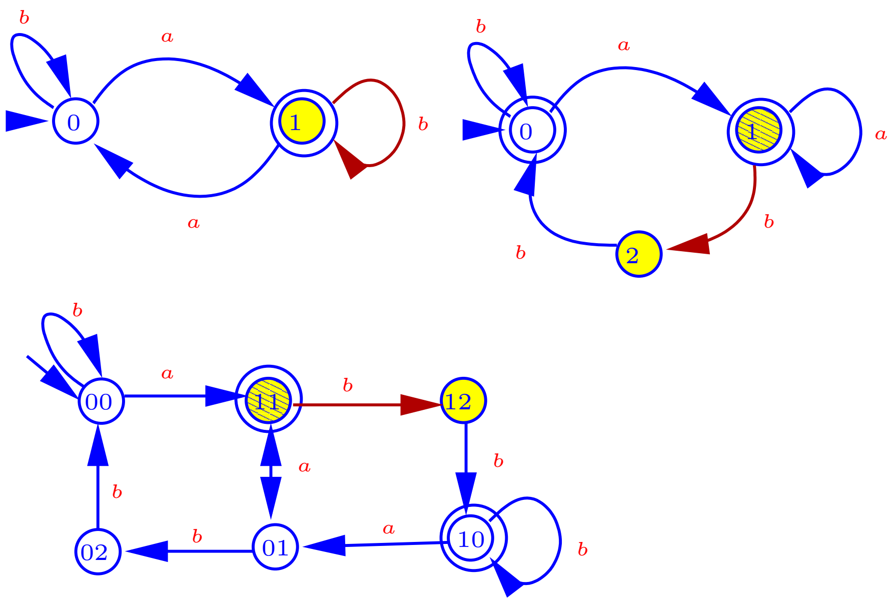
Exemple de produit d'automates



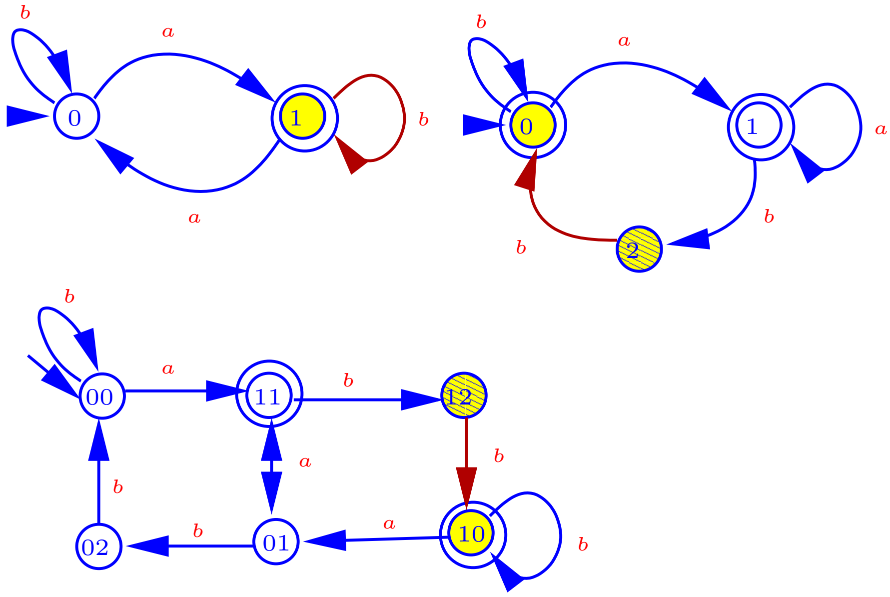
Exemple de produit d'automates



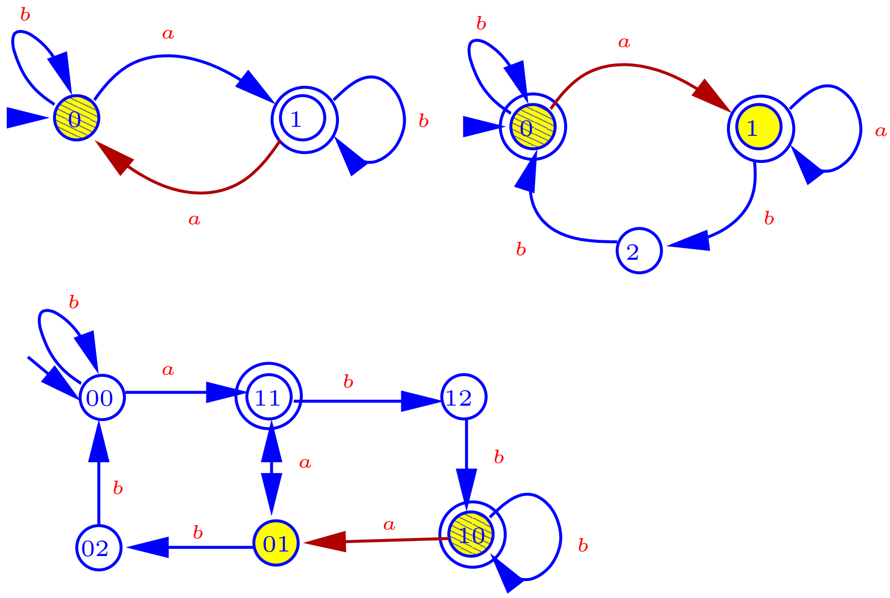
Exemple de produit d'automates



Exemple de produit d'automates



Exemple de produit d'automates



Automates et Langages Start – p.36/102

Fermeture par intersection

Théorème Soient $A = (Q^A, \Sigma, q_0^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_0^B, \delta^B, F^B)$ deux ADEFs.

- $L(A \times B) = L(A) \cap L(B)$.
- La classe EF des langages d'états finis est fermée par intersection.

Pour montrer $L(A \times B) = L(A) \cap L(B)$, on doit montrer:

1. $L(A \times B) \subseteq L(A)$,
2. $L(A \times B) \subseteq L(B)$ et
3. $L(A) \cap L(B) \subseteq L(A \times B$.

Pour 1.) et 2.) nous attendrons le prochain cours pour avoir un moyen élégant pour le faire.
3) sera fait sous-formes d'exercices en TD.

Minimisation d'automates déterministes

question: Etant donné un langage d'états finis. Existe-t-il un automate minimal qui reconnait ce langage?
Definition Soit $A = (Q, \Sigma, \delta, q_0, F)$ un ADEF dont tous les états sont accessibles.
On définit une relation d'équivalence $\approx$ sur Q:

$$p \approx q \textit{ ssi } \forall u \in \Sigma^* \cdot [\delta^*(p,u) \in F \Leftrightarrow \delta^*(q,u) \in F]$$

□

On Montre d'abord que $\approx$ est effectivement une relation d'équivalence. On note par $[q]$ la classe d'équivalence et par $Q_{/\approx}$ l'ensemble des classes d'équivalence.

Minimisation: Le Théorème

Théorème Soit $A = (Q, \Sigma, \delta, q_0, F)$ un ADEF complet dont tous les états sont accessibles.
Soit $A_{/\approx} = (Q_{/\approx}, \Sigma, [q_0], \delta_{/\approx}, F_{/\approx})$ où:

- $\delta_{/\approx} : Q_{/\approx} \times \Sigma \rightarrow Q_{/\approx}$ est l'application de transition avec $\delta_{/\approx}([q], a) = [\delta(q, a)]$.
- $F_{/\approx} = \{[q] \mid q \in F\}$.

Alors,

1. $L(A_{/\approx}) = L(A)$ et
2. $A_{/\approx}$ est minimal pour $L(A)$: il n'existe pas d'ADEF complet qui reconnaît $L(A)$ et contient moins d'états que $A_{/\approx}$.

La relation d'équivalence $\equiv_k$

Soit $A = (Q, \Sigma, \delta, q_0, F)$ un ADEF dont tous les états sont accessibles.
Pour chaque $k \in \mathbb{N}$, on introduit la relation $\equiv_k$ sur Q:

1. $q \equiv_0 q'$ ssi $q \in F \Leftrightarrow q' \in F$.
2. Pour $k \in \mathbb{N}$, $q \equiv_{k+1} q'$ ssi $q \equiv_k q'$ et $\forall a \in \Sigma \cdot [\delta(q, a) \equiv_k \delta(q', a)]$.

Construction de $A_{/\approx}$

Lemme Pour tout $k \in I\!N$, $q \equiv_k q'$ ssi

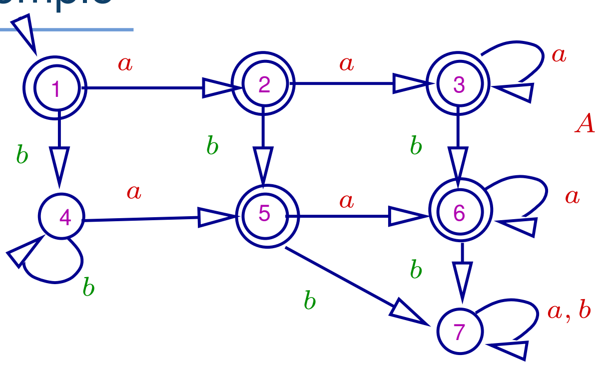
$$\forall u \in \Sigma^* \cdot |u| \leq k \Rightarrow (\delta^*(q,u) \in F \Leftrightarrow \delta^*(q',u) \in F).$$

$\square$

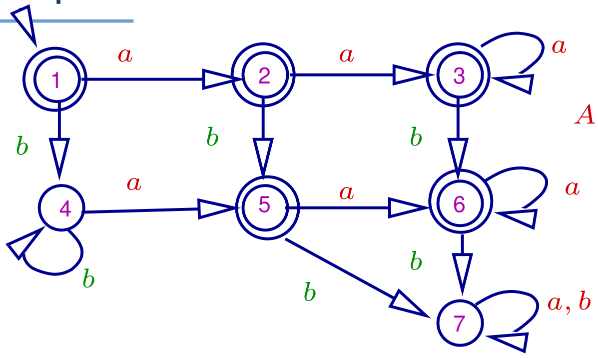
Corrollaire $\bigcap_{k \in I\!N} \equiv_k = \approx$. $\square$

$R := (F \times F) \cup [(Q \setminus F) \times (Q \setminus F)];$
$R_1 := \emptyset;$
While $R \neq R_1$ Do
    $R_1 := R;$
    $X := \{(p,q) \in R \mid \exists a \in \Sigma \cdot \neg(\delta(p,a) \in R \Leftrightarrow \delta(q,a) \in R)\};$
    $R := R \setminus X;$
Od; return X

Exemple

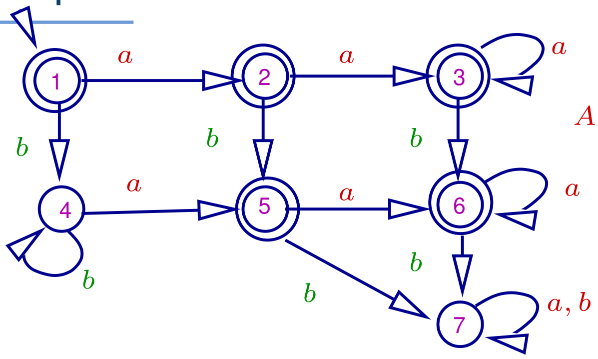


Exemple



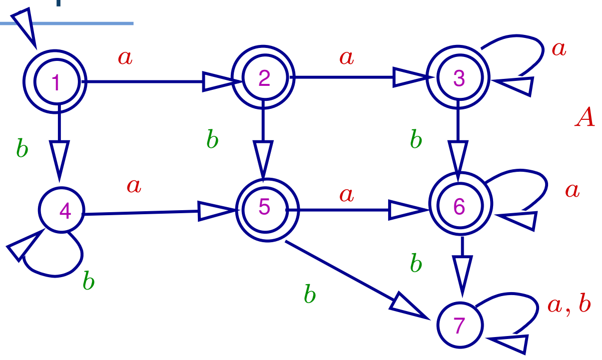
$\equiv_0$
1
2
3
5
6
4
7

Exemple



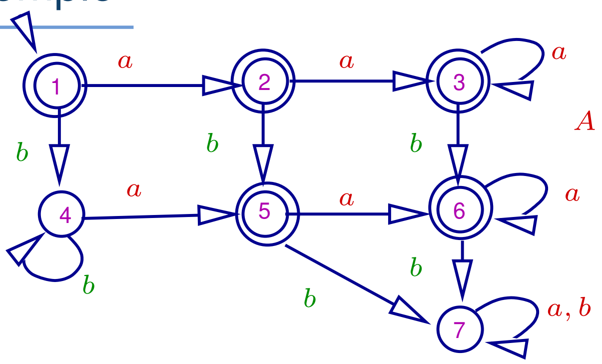
$\equiv_0$	$\equiv_{0,1}$
1	2
2	3
3	1
5	5
6	6
4	4
7	7

Exemple

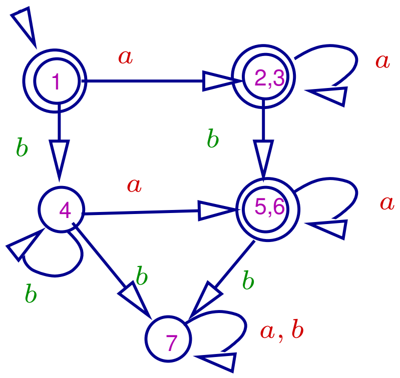


$\equiv_0$	$\equiv_{0,1}$	$\equiv_{0,1,2}$
1	2	2
2	3	3
3	1	1
5	5	5
6	6	6
4	4	4
7	7	7

Exemple



$\equiv_0$	$\equiv_{0,1}$	$\equiv_{0,1,2}$	$\equiv_{0,1,2,3}$
1	2	2	2
2	3	3	3
3	1	1	1
5	5	5	5
6	6	6	6
4	4	4	4
7	7	7	7



Preuves

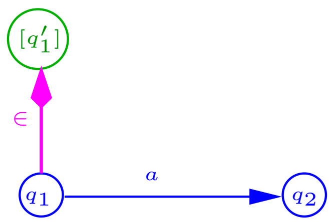
Il nous reste à montrer :

1. $L(A_{/\approx}) = L(A)$ et
2. $A_{/\approx}$ est minimal pour $L(A)$: il n'existe pas d'ADEF complet qui reconnaît $L(A)$ et contient moins d'états que $A_{/\approx}$.

Preuves

1. $L(A_{/\approx}) = L(A)$ et

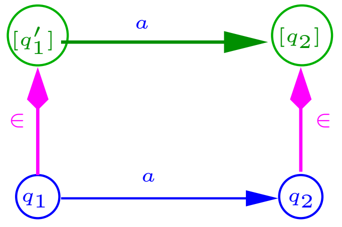
1. Pour montrer $L(A_{/\approx}) \subseteq L(A)$, il suffit de montrer que le diagramme suivant commute :



Preuves

1. $L(A_{/\approx}) = L(A)$ et

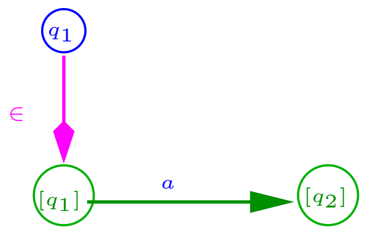
1. Pour montrer $L(A_{/\approx}) \subseteq L(A)$, il suffit de montrer que le diagramme suivant commute :



Preuves

1. $L(A_{/\approx}) = L(A)$ et

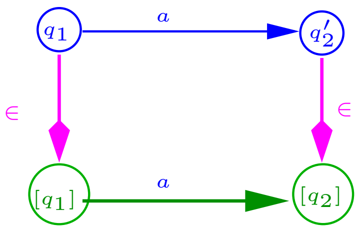
1. Pour montrer $L(A) \subseteq L(A_{/\approx})$, il suffit de montrer que le diagramme suivant commute :



Preuves

1. $L(A_{/\approx}) = L(A)$ et

1. Pour montrer $L(A) \subseteq L(A_{/\approx})$, il suffit de montrer que le diagramme suivant commute :



Preuve de la minimalité de $A_{/\approx}$

1. $L(A_{/\approx}) = L(A)$ et
2. $A_{/\approx}$ est minimal pour $L(A)$: il n'existe pas d'ADEF complet qui reconnaît $L(A)$ et contient moins d'états que $A_{/\approx}$.

Preuve de la minimalité de $A_{/\approx}$

1. $A_{/\approx}$ est minimal pour $L(A)$: il n'existe pas d'ADEF complet qui reconnaît $L(A)$ et contient moins d'états que $A_{/\approx}$.

Preuve de la minimalité de $A_{/\approx}$

1. $A_{/\approx}$ est minimal pour $L(A)$: il n'existe pas d'ADEF complet qui reconnaît $L(A)$ et contient moins d'états que $A_{/\approx}$.

Supposons que $A_{/\approx}$ a n états et soit B un automate telque : $L(A) = L(B)$ et B a $m < n$ état.

Alors, pour chaque état q de $A_{/\approx}$, il existe un mot u_q telque $\delta^*(q_0, u_q) = q$. Comme $m < n$, il existe deux états q_1, q_2 de $A_{/\approx}$ tels que u_{q_1} et u_{q_2} mênent au même état p dans B. Comme q_1 et q_2 ne sont pas équivalent, il existe un mot v qui est accepté à partir de q_1 mais pas à partir de q_2 (ou inversement). Alors, on a $u_{q_1} \cdot v \in L(A)$ et $u_{q_2} \cdot v \notin L(A)$.

Comme B est déterministe et
$\delta_B^*(q_0^B, u_{q_1} \cdot v) = \delta_B^*(p, v) = \delta_B^*(q_0^B, u_{q_2} \cdot v)$

nous donc avons une contradiction.

Automates non-déterministes

Automates d'états finis non-déterministes

Idée:

- Déterminisme: A chaque état et pour chaque symbole de l'alphabet, il existe au plus un état successeur.
- Non-déterminisme: Pour un état et un symbole, on peut avoir 0, 1 ou plusieurs états successeurs.

Motivation:

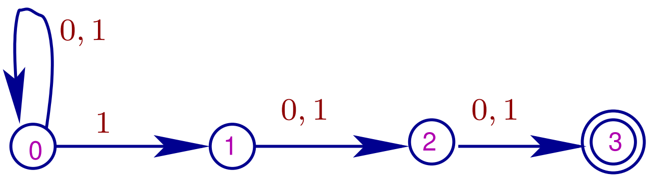
- Il est souvent plus facile de trouver un automate non-déterministe qui reconnait un langage L qu'un automate déterministe.
- Pour certains langages, on peut trouver un automate non-déterministe qui les reconnait et qui est plus petit que tout automate déterministe qui les reconnait.
- Mais, comme on verra, on ne pourra pas se passer des automate déterministes.

Exemple

Soit $\Sigma = \{0, 1\}$. Soit L_3 le langage constitué des mots de longueur ≥ 3 et dont le 3ème symbole de droite est 1.

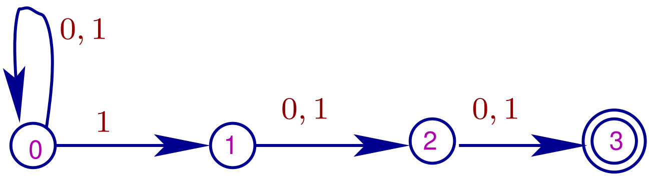
Exemple

Soit $\Sigma = \{0, 1\}$. Soit L_3 le langage constitué des mots de longueur ≥ 3 et dont le 3ème symbole de droite est 1.



Exemple

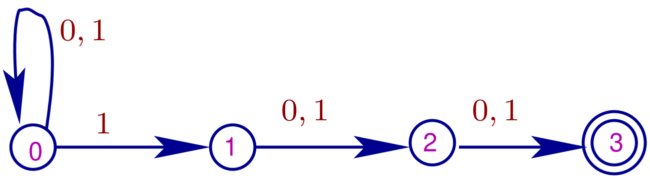
Soit $\Sigma = \{0, 1\}$. Soit L_3 le langage constitué des mots de longueur ≥ 3 et dont le 3ème symbole de droite est 1.



On verra que le plus petit automate déterministe qui reconnait L_3 a 8 états.

Exemple

Soit $\Sigma = \{0, 1\}$. Soit L_3 le langage constitué des mots de longueur ≥ 3 et dont le 3ème symbole de droite est 1.



On verra que le plus petit automate déterministe qui reconnait L_3 a 8 états.
Plus généralement, soit L_k le langage constitué des mots de longueur $\geq k$ et dont le kème symbole de droite est 1.

Aucun automate déterministe avec moins de 2^k états ne reconnait L_k.

Automates d'états finis non-déteministes

Definition Un *automate d'états finis non-déteministes (ANDEF)* est donné par un quintuplet $(Q, \Sigma, q_0, \Delta, F)$ où

- Q est un ensemble fini d'*états*.
- Σ est l'alphabet de l'automate.
- $q_0 \in Q$ est l'*état initial*.
- $\Delta \subseteq Q \times \Sigma \times Q$ est la *relation de transition*.
- $F \subseteq Q$ est l'ensemble des *états accepteurs*.

□

Configuration et exécutions

Soit $A = (Q, \Sigma, q_0, \Delta, F)$.

- Une *configuration* de l'automate A est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.
- On définit la relation $\rightarrow_\Delta$ de *dérivation* entre configurations:

$$(q, a \cdot u) \rightarrow_\Delta (q', u) \text{ ssi } (q, a, q') \in \Delta.$$

- Une *exécution de l'automate A* est une séquence de configurations $(q_0, u_0) \cdots (q_n, u_n)$ telle que

$$(q_i, u_i) \rightarrow_\Delta (q_{i+1}, u_{i+1}), \text{ pour } i = 0, \cdots, n-1.$$

On dénote par $\stackrel{*}{\longrightarrow}_\Delta$ la fermeture réflexive et transition de $\longrightarrow_\Delta$.

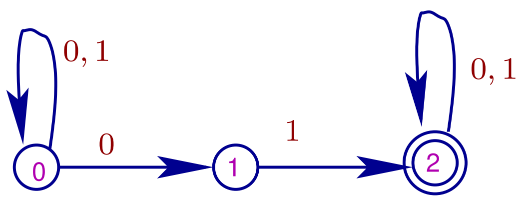
Langage reconnu par un ANDEF

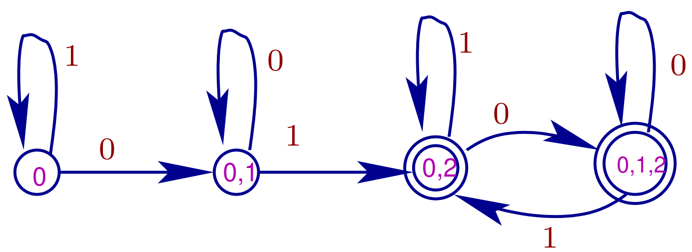
Soit $A = (Q, \Sigma, q_0, \Delta, F)$.

- Un mot $u \in \Sigma^*$ est *accepté* par A, s'**il existe une exécution** $(q_0, u_0) \cdots (q_{n-1}, u_{n-1})$ de A telle que
  1. $u = u_0$,
  2. $u_{n-1} = \epsilon$ et
  3. $q_{n-1} \in F$.
- Le *langage reconnu par* A, qu'on note par $L(A)$, est l'ensemble $\{u \in \Sigma^* \mid u$ est accepté par $A\}$.

Procédure de déterminisation (subset construction)

(Rabin & Scott 1959): On va coder dans un état accessible par un mot u dans l'automate déterministe tous les états qu'on peut atteindre avec u dans l'automate non-déterministe.
Soit $\Sigma = \{0, 1\}$.





Procédure de déterminisation.

Soit $A = (Q, \Sigma, q_0, \Delta, F)$ un ANDEF.
Definition Det(A) dénote l'automate déterministe détats finis dŕfini par:

$$(\mathcal{P}(Q), \Sigma, \{q_0\}, \delta, \mathcal{F}) \text{ où}$$

- $\delta(X, a) = \{q' \mid \exists q \in X \cdot (q, a, q') \in \Delta\}$ et
- $X \in \mathcal{F}$ ssi $X \cap F \neq \emptyset$.

$\square$

Nous avons le résultat suivant:

Correction de la procédure de déterminisation.

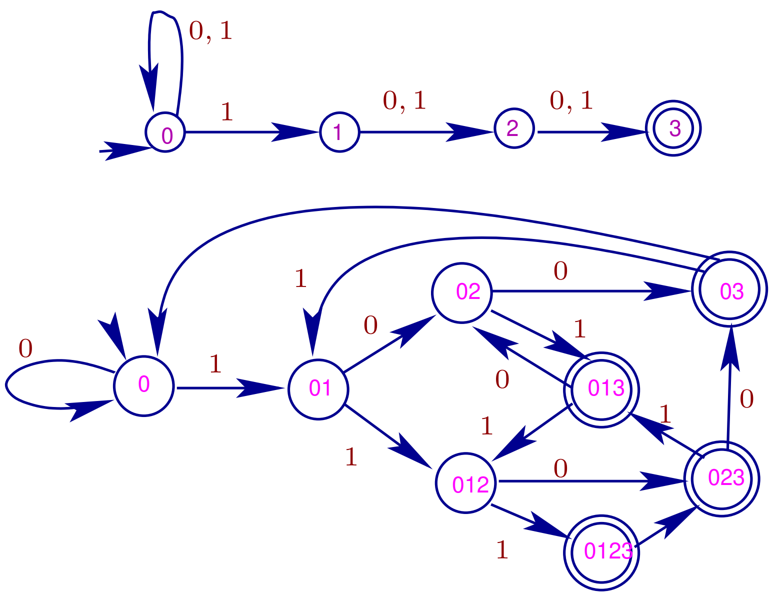
Théorème Soit $A = (Q, \Sigma, q_0, \Delta, F)$ un ANDEF. Alors, $L(\text{Det}(A)) = L(A)$.
Preuve

- On peut montrer $L(A) \subseteq L(\text{Det}(A))$ en montrant $A \sqsubseteq_{\in} \text{Det}(A)$.
- Pour montrer $L(\text{Det}(A)) \subseteq L(A)$, on montre:

  Pour tout $u \in \Sigma^*$, $\forall q \in \delta^*(\{q_0\}, u) \cdot (q_0, u) \xrightarrow{*} (q, \epsilon)$.

$\square$

Exemple



Sur la complexité de la déterminisation

Soit $\Sigma = \{0, 1\}$ et soit L_k le langage constitué des mots de longueur $\geq k$ et dont le kème symbole de droite est 1:

$$L_k = \{a_1 \cdots a_n \mid a_{n-k+1} = 1\}.$$

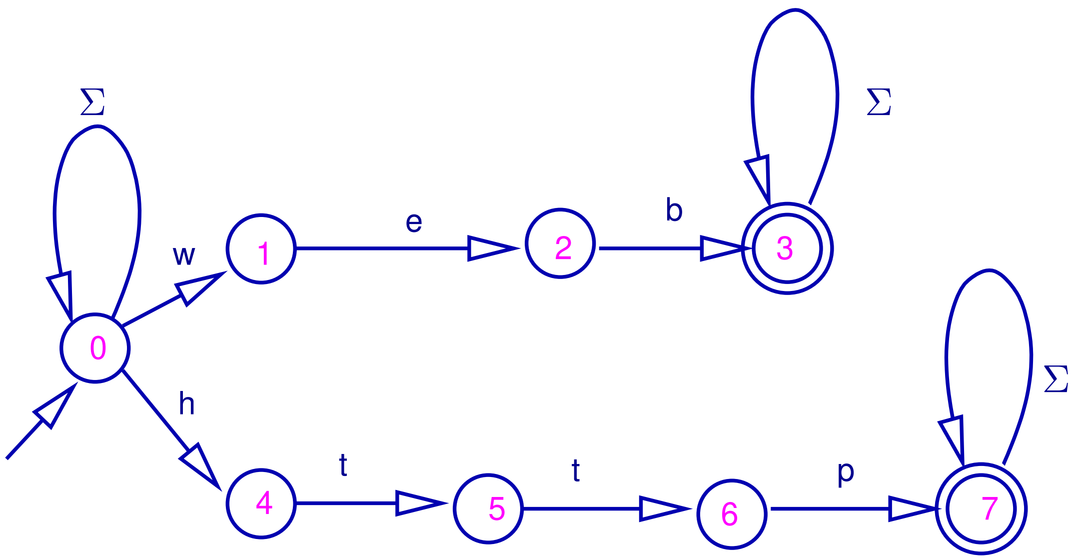
Lemme Aucun automate déterministe avec moins de 2^k états ne reconnaît L_k. $\square$

Preuve

Preuve Par contraposition.
Soit $A = (Q, \Sigma, q_0, \delta, F)$ un automate déterministe tel que $|Q| < 2^k$ et $L(A) = L$.
Soient $u = a_1 \cdots a_k$ et $v = b_1 \cdots b_k$ deux mots différents de longueur k tels que $\delta^*(q_0, u) = \delta^*(q_0, v)$. De tels mots doivent exister car ils existent 2^k differents mots de longueur k et seulement $|Q| < 2^k$ états.
Comme u et v sont differents il existe i tel que $a_i \neq b_i$ Par symétrie supposons $a_i = 1$ et $b_i = 0$.
Soient $u' = u0^{i-1}$ et $v' = v0^{i-1}$. Alors,
$u'(|u'| - k + 1) = u'(k + i - 1 - k + 1) = u'(i) = a_i = 1$ et $v'(|v'| - k + 1)) = b_i = 0$. Donc $u' \in L_k$ et $v' \notin L_k$. Ce qui contredit $\delta^*(q_0, u') = \delta^*(q_0, v')$.

$\square$

Reconnaissance de texte



Automates avec ϵ-transitions

Motivation

Definition Soit L un langage sur Σ.
La Fermeture de Kleene de L, dénoté L^*, est défini inductivement par :

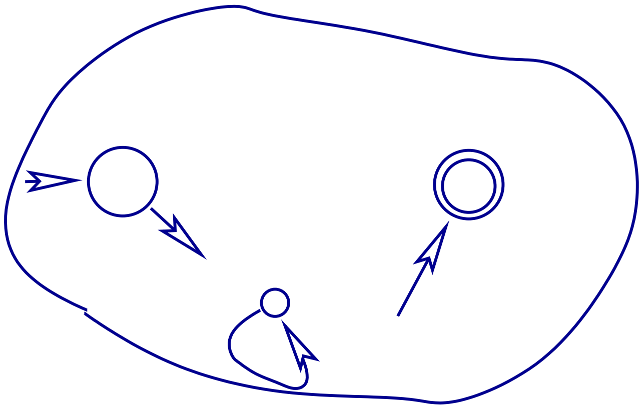
- $\epsilon \in L^*$ et
- $f(u, v) = \{uv\}$

□

On veut montrer que si L est EF alors aussi L^*.

Fermeture par l'opérateur de Kleene

Etant donné un automate A qui reconnait L peut on construire un automate qui reconnait L^*.



Motivation

Definition Soit L un langage sur Σ.
La Fermeture de Kleene de L, dénoté L^*, est défini inductivement par :

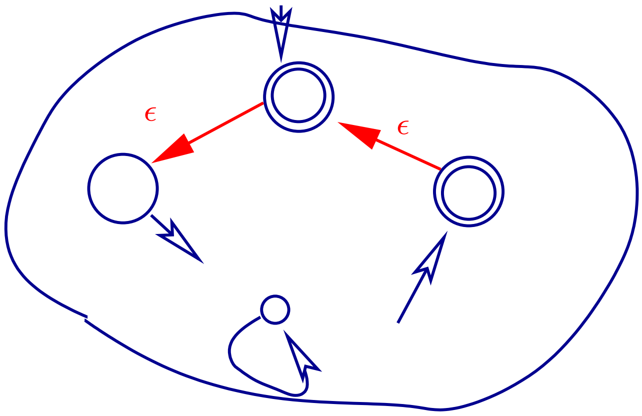
- $\epsilon \in L^*$ et
- $f(u, v) = \{uv\}$

□

On veut montrer que si L est EF alors aussi L^*.

Fermeture par l'opérateur de Kleene

Etant donné un automate A qui reconnait L peut on construire un automate qui reconnait L^*.



Automates avec ϵ-transitions

Definition Un *automate d'états finis non-déteministes (ϵ-ANDEF) avec ϵ-transitions* est donné par un quintuplet $(Q, \Sigma, q_0, \Delta, F)$ où

- Q est un ensemble fini d'*états*.
- Σ est l'alphabet de l'automate.
- $q_0 \in Q$ est l'*état initial*.
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ est la *relation de transition*.
- $F \subseteq Q$ est l'ensemble des *états accepteurs*.

$\square$

Configuration et exécutions

Soit $A = (Q, \Sigma, q_0, \Delta, F)$ un ϵ-ANDEF.

- Une *configuration* de l'automate A est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.
- On définit la relation $\rightarrow_\Delta$ de *dérivation* entre configurations: $(q, a \cdot u) \rightarrow_\Delta (q', u')$ ssi

$$[(q, a, q') \in \Delta \text{ et } u' = u] \text{ ou } [a \cdot u = u' \text{ et } (q, \epsilon, q') \in \Delta].$$

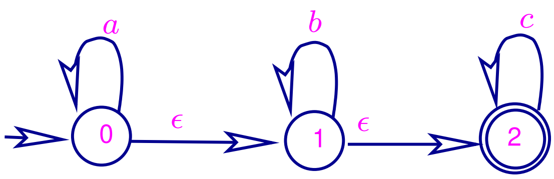
- Une *exécution de l'automate* A est une séquence de configurations $(q_0, u_0) \cdots (q_n, u_n)$ telle que

$$(q_i, u_i) \rightarrow_\Delta (q_{i+1}, u_{i+1}), \text{ pour } i = 0, \cdots, n-1.$$

- Les notions d'acceptation d'un mot et de langage reconnu sont définies comme dans le cas des ANDEF mutatis mutandis.

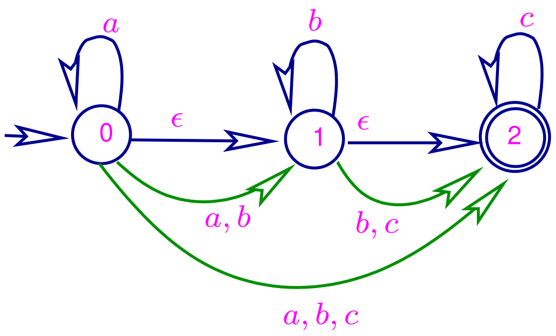
Exemple d'automate avec ϵ-transition

Soit $\Sigma = \{a, b, c\}$.



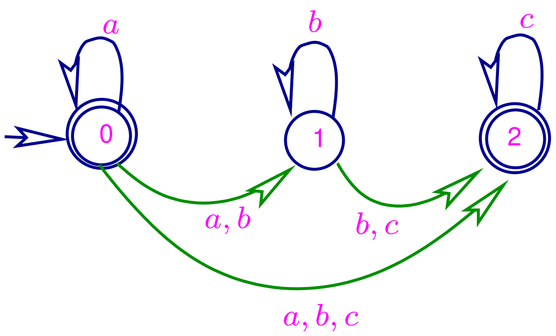
Exemple d'automate avec ϵ-transition

Soit $\Sigma = \{a, b, c\}$.



Exemple d'automate avec ϵ-transition

Soit $\Sigma = \{a, b, c\}$.



Elimination des ϵ-transition

Soit $A = (Q, \Sigma, q_0, \Delta, F)$ un ϵ-ANDEF.
On construit un automate $\epsilon\ell(A) = (Q, \Sigma, q_0, \epsilon\ell(\Delta), \epsilon\ell(F))$ d'états finis non-déterministe qui reconnait $L(A)$.
La relation de transition $\epsilon\ell(\Delta)$ est définie par: $(q, a, q') \in \epsilon\ell(\Delta)$ *ssi ils existent* $q_1, q_2 \in Q$ *tels que :*

1. $q \overset{*}{\rightarrow}^{\epsilon} q_1$
2. $(q_1, a, q_2) \in \Delta$ *et* $q_2 \overset{*}{\rightarrow}^{\epsilon} q'$.

L'ensembles des états accepteurs $\epsilon\ell(F)$ est défini par:
$\epsilon\ell(F) = \{q \in Q \mid \exists q' \in F \cdot q \overset{*}{\rightarrow}^{\epsilon} q'\}$

Correction de l'élimination des ϵ-transitions

Théorème Soit $A = (Q, \Sigma, q_0, \Delta, F)$ un ϵ-ANDEF. Alors,

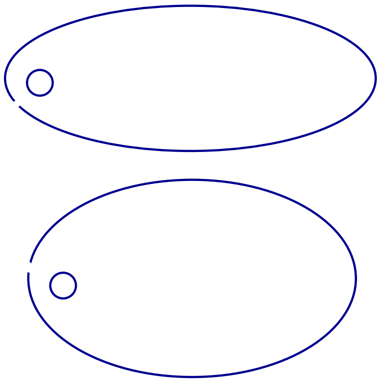
$$L(A) = L(\epsilon\ell(A))$$

On montre:

- Pour rout $u \in \Sigma^*$, si $(q, u) \longrightarrow^*_{\epsilon\ell(\Delta)} (q', u)$ alors $(q, u) \longrightarrow^*_{\Delta} (q', u)$.
- si $\epsilon \in L(A)$ alors $\epsilon \in L(\epsilon\ell(A))$ et
- pour rout $u \in \Sigma^*$ avec $|u| > 0$, si $(q, u) \longrightarrow^*_{\Delta} (q', u)$ alors $(q, u) \longrightarrow^*_{\epsilon\ell(\Delta)} (q', u)$.

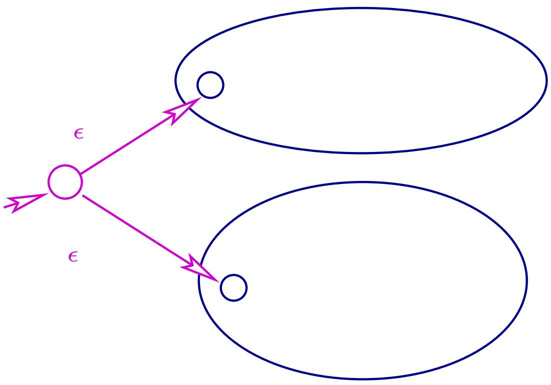
Fermeture de EF par Union et concaténaton

Union



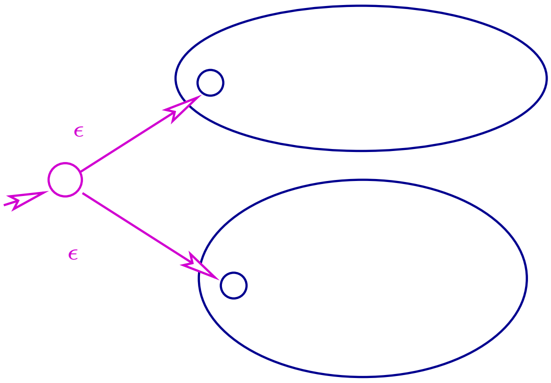
Fermeture de EF par Union et concaténaton

Union

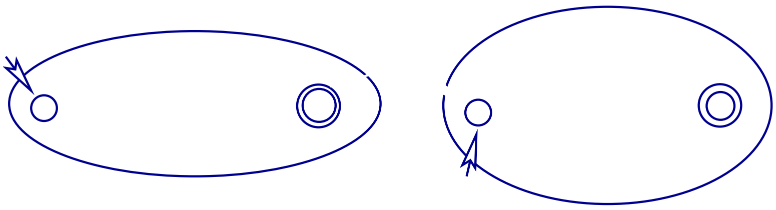


Fermeture de EF par Union et concaténaton

Union

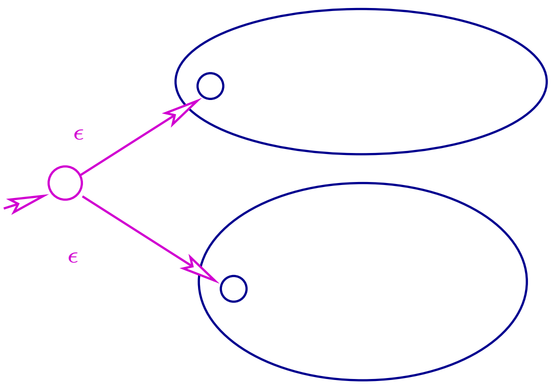


Concaténation

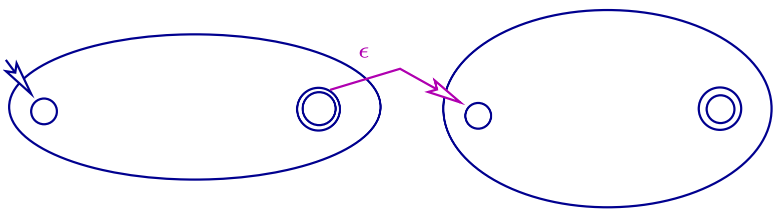


Fermeture de EF par Union et concaténaton

Union



Concaténation



Homomorphisme de mots

Soit Σ et Σ' deux alphabet. Une application $h : \Sigma \rightarrow \Sigma'^*$ induit un homomorphisme $\hat{h}$, de $\Sigma^* \rightarrow \Sigma'^*$ de la manière suivante:

- $\hat{h}(\epsilon) = \epsilon$ et
- $\hat{h}(u \cdot a) = \hat{h}(u) \cdot h(a)$.

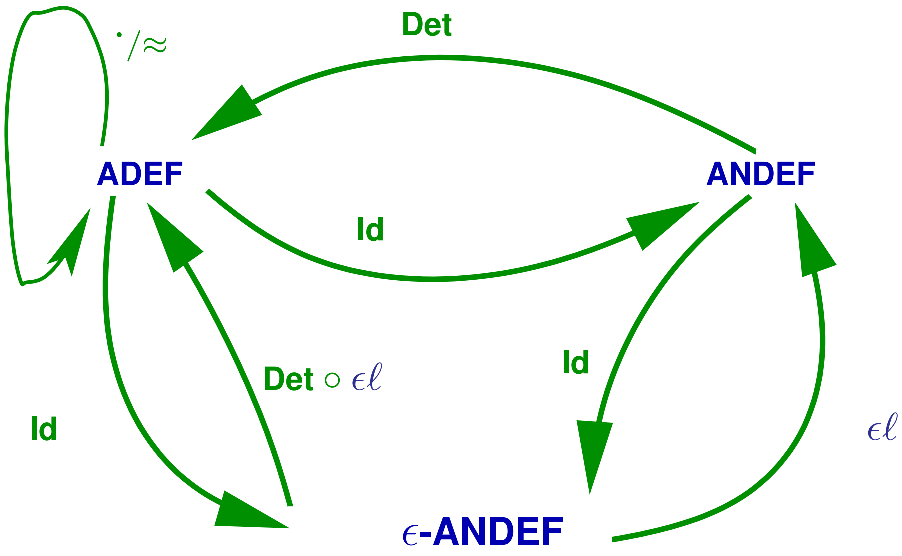
A partir de maintenant, on écrit h aulieu de $\hat{h}$.

Fermeture par homomorphisme et inverse homomorphisme

Théorème

- Si $L \subseteq \Sigma^*$ est détats finis alors ainsi est $h(L) = \{h(u) \mid u \in L\}$. Donc EF est fermé par homomorphisme.
- Si $L' \subseteq \Sigma'^*$ est détats finis alors ainsi est $h^{-1}(L') = \{u \in \Sigma^* \mid \exists u' \in L' \cdot h(u) = u'\}$. Donc EF est fermé par inverse homomorphisme.

Résumé I



Résumé II

Propriétés de fermeture: Les langages d'états finis sont fermés par

1. Union, intersection,
2. complément,
3. concaténation et
4. l'étoile de Kleene (L^*).
5. Ainsi que par homomorphisme, inverse homomorphisme et opération miroir.

Procédure de décision:

1. Langage vide.
2. Langege infini.
3. Inclusion de langages.
4. Egalité de langages.

Expresions régulières

Motivation

On cherche une notation plus concise que les automates pour décrire des langages d'états finis.
Par exemple, il est inconcevable que pour faire un **grep** sur Unix ou Linux on doit décrire un automate. De même pour utiliser des logiciels d'analyse lexicale comme Lex ou Flex on doit spécifier les lexiques (token). On prefère faire ceci de manière concise.

- Les automates offrent la possibilité de décrire des langages de manières opérationnelle: par une sorte de machine (l'automate).
- Les expressions régulières permettent de le faire de manière déclarative.

Expressions régulières: syntaxe

Soit Σ un alphabet.
Definition Les *expressions régulières sur* Σ sont définies inductivement:

- ϵ et $\emptyset$ sont des expressions régulières sur Σ.
- Si $a \in \Sigma$ alors a est une expression régulière sur Σ.
- Si e et e' sont des expressions régulières sur Σ alors $e + e'$ est une expression régulière.
- Si e et e' sont des expressions régulières sur Σ alors $e \underline{\cdot} e'$ est une expression régulière.
- Si e est une expression régulière sur Σ alors $e^{\underline{*}}$ est une expression régulière sur Σ.

L'ensemble des expressions régulières est dénoté par ER. □

Expressions régulières: sémantique

Definition La sémantique des expressions régulières est donnée par l'application $L : \mathrm{ER} \to \mathcal{P}(\Sigma^*)$ qui associe à e un langage $L(e)$. L'application L est définie inductivement:

- $L(\epsilon) = \{\epsilon\}$, $L(\emptyset) = \emptyset$ et $L(a) = \{a\}$.
- $L(e + e') = L(e) \cup L(e')$.
- $L(e \underline{\cdot} e') = L(e) \cdot L(e')$.
- $L(e^{\underline{*}}) = L(e)^*$.

□

On dit qu'un langage L est *régulièr* ssi il existe un expression régulière e telle que $L(e) = L$.

Convention et notation

- Nous ne ferons plus la distinction explicitement entre $\underline{\cdot}$ et $\cdot$; ni entre $\underline{*}$ et $*$.
- Nous voulons aussi pouvoir écrire des expressions comme $a+b+c$ à la place de $(a+b)+c$ et $a+b^*$ à la place de $(a+(b^*)$. Pour éviter les ambiguités nous permettons l'utilisation des parenthèses et admettons les priorités suivantes dans un ordre décroissant:
  1. $*$
  2. $\cdot$
  3. $+$.

Convention et notation

- Nous écrivons aussi LL' à la place de $L \cdot L'$.

Donc les expression $e_1 + e_2^*$ et $e_1 + (e_2)^*$ dénote les mêmes ensembles. Ainsi que $e_1 + e_2 \cdot e_3$ et $e_1 + (e_2 \cdot e_3)$.
Exemple Le langage constitué des mots sur $\{a, b\}$ avec nombre pair de a ou nombre impair de b peut être décrit par l'expression régulière:

$$((ab^*a+b)^* + (ba^*b+a)^*ba^*.$$

$\square$

Théorème de Kleene

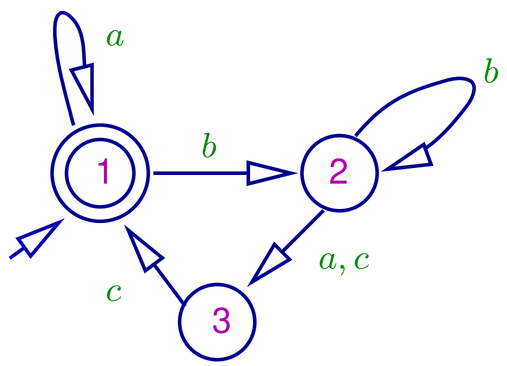
Théorème Soit Σ un alphabet et $L \subseteq \Sigma^*$.
L est régulier ssi L est détats finis.
Plus encore,

1. Il existe un algorithme qui transforme une expression régulère en un automate fini équivalent.
2. Inversement, il existe un algorithme qui transforme un automate fini en une expression régulière équivalente.

Preuve L'implication de droite à gauche et 2.) suivent des propriétés de fermeture des automates finis.
Nous allons montrer l'impliquation de gauche à droite et 1.). □

L'idée par un exemple

Soit $\Sigma = \{a, b, c\}$ et A l'automate suivant:

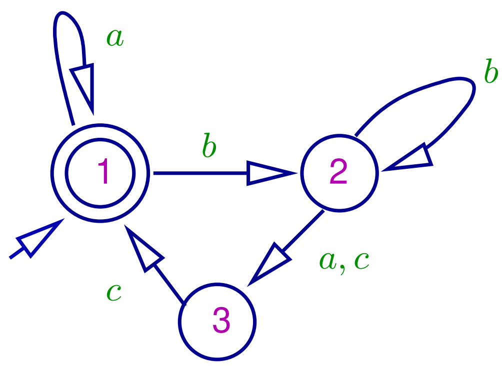


À A on peut associer le système d'équations suivant:

$$X_1 = \{a\} \cdot X_1 \cup \{b\} \cdot X_2 \cup \{\epsilon\}$$
$$X_2 = \{b\} \cdot X_2 \cup \{a, c\} X_3$$
$$X_3 = \{c\} \cdot X_1$$

Intituitivement, X_i décrit les mots acceptés à partir de l'état i.

L'idée par un exemple



si on utilise les expressions régulières comme notation, on peut écrire ce système d'équations de la manière suivante:

$$X_1 = aX_1 + bX_2 + \epsilon$$
$$X_2 = bX_2 + (a + c)X_3$$
$$X_3 = cX_1$$

Question: Comment résoudre un tel système déquations.

Résolution d'équations linéaires

Lemme Soient $A, b \subseteq \Sigma^*$ des langages.

1. Le langage A^*B est une soulution de l'équation

$$X = AX + B$$

2. (Lemme d'Arden) : Si $\epsilon \notin A$ alors A^*B est la solution unique de

$$X = AX + B$$

☐ **Attention: Pour résoudre un système d'équations correspondant à un automate, on applique le Lemme que dans le deuxième cas.**

Exemple de résolution de système d'équations

Considérons

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= bX_2 + (a+c)X_3 \\ X_3 &= cX_1 \end{aligned}$$

Exemple de résolution de système d'équations

Considérons

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= bX_2 + (a+c)X_3 \\ X_3 &= cX_1 \end{aligned}$$

On remplace X_3 par cX_1 dans la deuxième équation:

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= bX_2 + (a+c)cX_1 \\ X_3 &= cX_1 \end{aligned}$$

Exemple de résolution de système d'équations

On remplace X_3 par cX_1 dans la deuxième équation:

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= bX_2 + (a+c)cX_1 \\ X_3 &= cX_1 \end{aligned}$$

on applique le Lemme sur la deuxième équation:

Exemple de résolution de système d'équations

On remplace X_3 par cX_1 dans la deuxième équation:

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= bX_2 + (a+c)cX_1 \\ X_3 &= cX_1 \end{aligned}$$

on applique le Lemme sur la deuxième équation:

$$\begin{aligned} X_1 &= aX_1 + bX_2 + \epsilon \\ X_2 &= b^*(a+c)cX_1 \\ X_3 &= cX_1 \end{aligned}$$

Exemple de résolution de système d'équations-suite

On remplace X_2 par $b^*(a+c)cX_1$

Exemple de résolution de système d'équations-suite

On remplace X_2 par $b^*(a+c)cX_1$

$$\begin{aligned} X_1 &= (a + bb^*(a+c)c)X_1 + \epsilon \\ X_2 &= b^*(a+c)cX_1 \\ X_3 &= cX_1 \end{aligned}$$

Exemple de résolution de système d'équations-suite

On remplace X_2 par $b^*(a+c)cX_1$

$$\begin{array}{rcl} X_1 & = & (a+bb^*(a+c)c)X_1+\epsilon \\ X_2 & = & b^*(a+c)cX_1 \\ X_3 & = & cX_1 \end{array}$$

et on applique le Lemme sur la première équation:

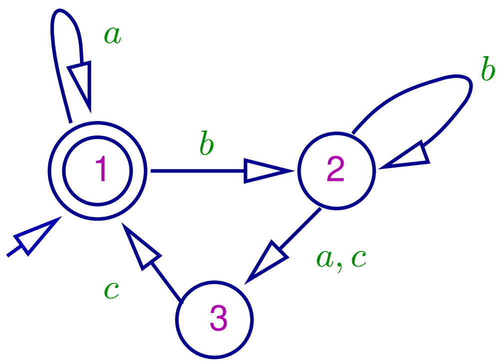
Exemple de résolution de système d'équations-suite

On remplace X_2 par $b^*(a+c)cX_1$

$$\begin{array}{rcl} X_1 & = & (a+bb^*(a+c)c)X_1+\epsilon \\ X_2 & = & b^*(a+c)cX_1 \\ X_3 & = & cX_1 \end{array}$$

et on applique le Lemme sur la première équation:

$$\begin{array}{rcl} X_1 & = & (a+bb^*(a+c)c)^* \\ X_2 & = & b^*(a+c)cX_1 \\ X_3 & = & cX_1 \end{array}$$



Système d'équations associé à un automate

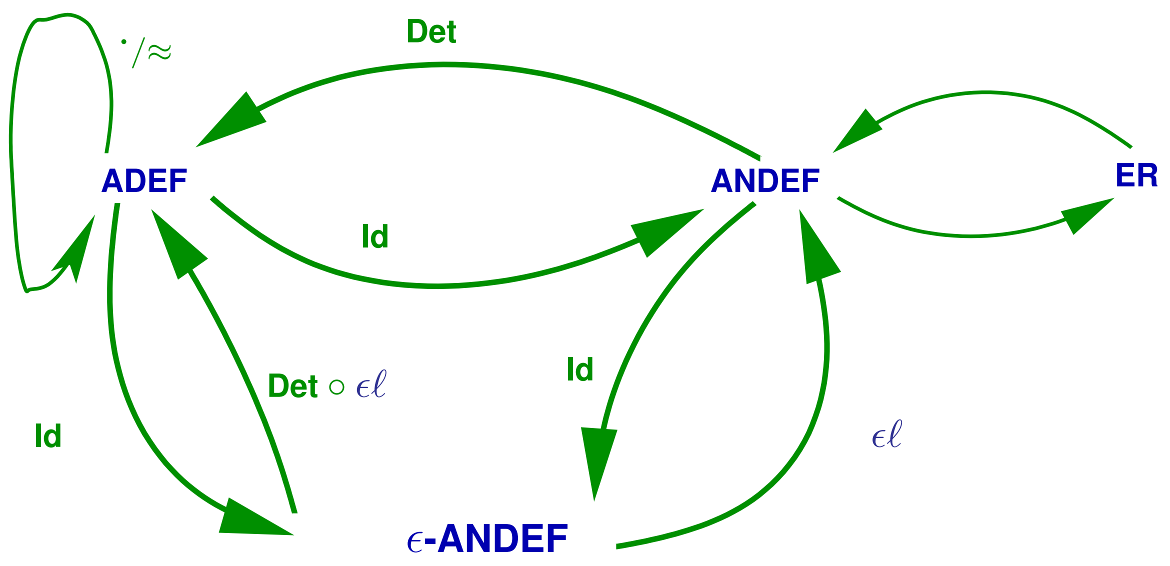
Soit $A = (Q, \Sigma, q_0, \delta, F)$ un automate fini.
Soit $SE(A)$ le système d'équations donné par:

$$X_q = \sum_{\delta(q,a)=q'} aX_{q'} + (\text{ si } q \in F \text{ alors } \epsilon \text{ sinon } \emptyset)$$

Théorème Soit $A = (Q, \Sigma, q_0, \delta, F)$ un automate fini. Soit $(L_q \mid q \in A)$ la plus petite solution de $SE(A)$. Alors,

$$L(A) = L_{q_0}.$$

Résumé



Langages non-réguliers et Lemme de l'itération

Comment montrer qu'un langage L n'est pas régulier?

Théorème (Lemme de l'itération, Pumping lemma)
Soit L un langage régulier. Alors, il existe $n \in I\!N$ tel que pour tout mot $w \in L$ avec $|w| \geq n$, on peut trouver $x, y, z \in \Sigma^*$ tels que $w = xyz$ et

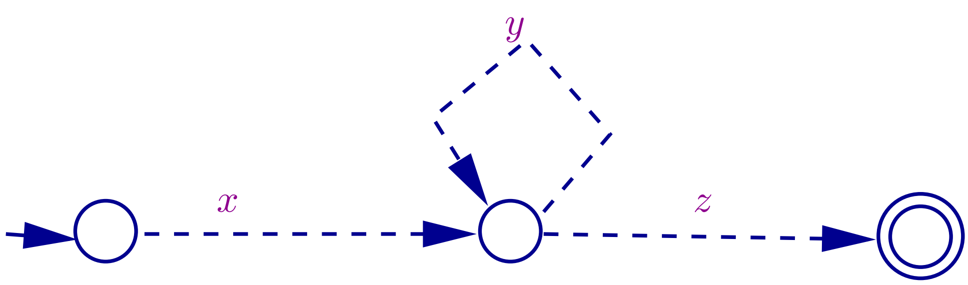
1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. pour tout $k \in I\!N$, $xy^k z \in L$.

Langages non-réguliers et Lemme de l'itération

Comment montrer qu'un langage L n'est pas régulier?

Théorème (Lemme de l'itération, Pumping lemma)
Soit L un langage régulier. Alors, il existe $n \in I\!N$ tel que pour tout mot $w \in L$ avec $|w| \geq n$, on peut trouver $x, y, z \in \Sigma^*$ tels que $w = xyz$ et

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. pour tout $k \in I\!N$, $xy^k z \in L$.

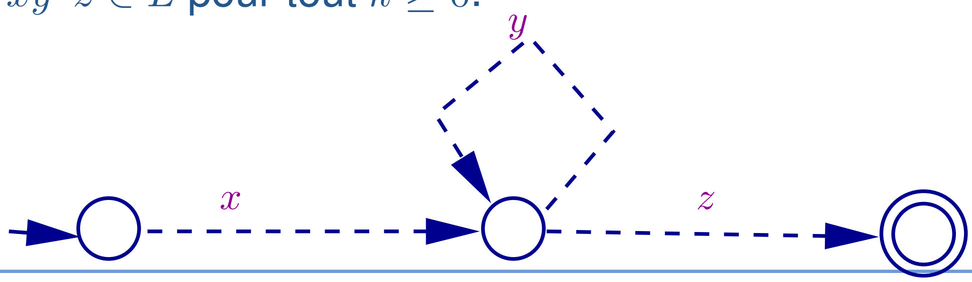


Démonstration du Lemme de l'itération

Soit L un langage régulier. Alors, il existe un automate déterministe $A = (Q, \Sigma, q_0, \delta, F)$ tel que $L(A) = L$. Soit $n = |Q|$ et soit $w = a_1 \cdots a_m \in L$ tel que $|w| = m \geq n$.
Soit $p_i = \delta^*(q_0, a_1, \cdots a_i)$ pour $i \leq m$. Alors, ils existent i et j avec $0 \leq i < j \leq n$ tels que $p_i = p_j$. On pose $x = a_1 \cdots a_i$, $y = a_{i+1} \cdots a_j$ et $z = a_{j+1} \cdots a_m$. Alors, on a:

1. $w = xyz$.
2. $|xy| \leq n$.
3. $\delta^*(p_i, y) = p_j = p_i$ et donc $\delta^*(p_i, y^k) = p_i$, pour tout $k \geq 0$.

Donc, $xy^k z \in L$ pour tout $k \geq 0$.



Application du Lemme de l'itération

Soit $L = \{a^i b^i \mid i \geq 0\}$. On veut montrer que L n'est pas régulier.
Preuve par contradiction
Supposons que L est régulier. Alors, on sait par le Lemme de l'itéraion, qu'il existe $n \geq 0$ tel que pour tout $w \in L$ avec $|w| \geq n$ ils existent $x, y, z \in \Sigma^*$ avec:

1. $w = xyz$.
2. $y \neq \epsilon$.
3. $|xy| \leq n$.
4. $xy^k z \in L$, pour tout $k \geq 0$.

Soit $w = a^n b^n$ (où n est le n du Lemme de l'itération). Soient $x, y, z \in \Sigma^*$ comme ci-dessus. Alors, comme $|xy| \leq n$ et $y \neq \epsilon$, on a $y = a^i$ avec $i > 0$. Soit $w' = xy^2 z = a^{n+i} b^n$. Alors, d'un côté on a $w' \in L$ mais aussi $w' \notin L$ car $n + i > n$. Ce qui est une contradiction. Donc L n'est pas régulier.

Grammaires formelles et langages hors-contexte

Grammaires: Motivation et exemple

- Les automates permettent de décrire des langages de manière operationnelle.
- Les expressions régulières permettent de décrire des langages de manière déclarative.
- Les grammaires permettent de décrire des langages de manière *inductive*.

Exemple

Exemple

- 0 et 1 sont des termes.
- Si t_1 et t_2 sont des termes alors ainsi sont $(t_1 + t_2)$ et $(t_1 * t_2)$.

De manière plus concise:

$$\begin{array}{llll} T \rightarrow 0 & T & \rightarrow (T + T) \\ T \rightarrow 1 & T & \rightarrow (T * T) \end{array}$$

On peut alors montrer que $(0 + (1 * 1))$ est un terme:

$$\begin{array}{l} T \vdash (T + T) \\ \quad \vdash (0 + T) \\ \quad \vdash (0 + (T * T)) \ \vdash \ (0 + (T * 1)) \ \vdash \ (0 + (1 * 1)) \end{array}$$

Grammaires: Motivation et exemple-suite

$$\begin{array}{ll} \text{Fact} & \rightarrow 0 \\ \text{Fact} & \rightarrow 1 \\ \text{Fact} & \rightarrow \text{Fact} * \text{Fact} \\ \text{Fact} & \rightarrow (\text{Term}) \\ \text{Term} & \rightarrow \text{Fact} \\ \text{Term} & \rightarrow \text{Term} + \text{Fact} \end{array}$$

On peut montrer:

$$\text{Fact} \overset{*}{\vdash} (0 + 1 * 1)$$

Fact et *Term* sont appelée les *symboles non-terminaux*, *Fact* est l'*axiome*, 0 et 1 forment les *symboles términals*.

Grammaires: Définition

Definition Une *grammaire* G est donnée par un quadruplet (N, T, P, S) où

- N est un ensemble fini. Les éléments de N sont appelés les *symboles non-términals*.
- T est un ensemble fini tel que $N \cap T = \emptyset$. Les él'ements de N sont appelés les *symboles terminals*. T est appelé l'alphabet.
- $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ est un ensemble fini de *règles de production*.
- $S \in N$ est appelé l'*axiome*.

□

Notation

On note $V = N \cup T$ et $\alpha \rightarrow \beta$ aulieu de $(\alpha, \beta) \in P$. On va utiliser A, B, ... pour désigner des symboles non-terminals; α, β, ... des mots sur V qu'on appelle formes sententielles, a, b, ... des symboles terminals; et u, v, ... des mots sur T.

Relation de dérivation

Definition
Soient $\alpha, \beta \in V^*$.
$\alpha \vdash_G \beta$ ssi ils existent $\gamma, \delta, \alpha_1, \beta_1$ telsque $\alpha = \gamma\alpha_1\delta$, $\beta = \gamma\beta_1\delta$, et $\alpha_1 \rightarrow \beta_1 \in P$. □
Exemple $\alpha = (0 + (T * 1)) \vdash \beta = (T + (T * 1))$. On pose: $\gamma = ($, $\alpha_1 = 0$, $\delta = +(T * 1))$ et $\beta_1 = T$. □

Notation: On note par $\vdash_G^*$ la fermeture réflexive et transitive de $\vdash_G$.

On note aussi $\alpha \vdash_G^n \beta$ s'ils existent $\alpha_0, \cdots \alpha_n$ telsque $\alpha_0 = \alpha$, $\alpha_n = \beta$ et $\alpha_i \vdash_G \alpha_{i+1}$, pour $0 \leq i < n$.

Langage généré par une grammaire

Definition Soit $G = (N, T, P, S)$ une grammaire.
Le langage $L(G)$ généré par G est défini par

$$L(G) = \{u \in T^* \mid S \vdash_G^* u\}.$$

□

Exemple

La grammaire G donnée par $S \to aSb \mid \epsilon$ génére le langage $L = \{a^n b^n \mid n \in I\!N\}$.
$L(G) \subseteq L$ en démontrant que $S \vdash^n \alpha$ implique $\alpha = a^i S b^i$ ou $\alpha = a^i b^i$ avec $i \in I\!N$. Ceci est invariant des règles de production qu'on peut montrer par récurrence:

- Base: S est de la forme $a^i S b^i$ avec $i = 0$.
- Pas de la récurrence: On montre que les règles préservent cet invariant. C'est-à-dire qu'appliquer une des règles sur α de la forme $a^i S b^i$ ou $\alpha = a^i b^i$ donne une forme sententielle de la même forme.
  D'abord on remarque que si $\alpha = a^i b^i$ alors on ne peut appliquer aucune règle.
  Soit $\alpha = a^i S b^i$. La première règle donne $a^{i+1} S b^{i+1}$ et la deuxième $\alpha = a^i b^i$. Donc l'invariant est préservé.

L'autre inclusion

Pour montrer $L \subseteq L(G)$ on montre comment G génère $a^i b^i$, pour $i \in I\!N$. On observe qu'en appliquant i fois la règle $S \to aSb$ on a $S \vdash^i a^i S b^i$. Il suffit donc d'appliquer ensuite $S \to \epsilon$ pour avoir $S \vdash^{i+1} a^i b^i$. Donc $a^i b^i \in L(G)$.

Grammaires: Classification

Definition Une grammaire $G = (N, T, P, S)$ est appelée de

1. *type 0*, en général.
2. *type 1* ou *sous-contexte*, si pour tout $\alpha \rightarrow \beta \in P$, on a $|\alpha| \leq |\beta|$ ou $\alpha = S$ et $\beta = \epsilon$.
3. *type 2* ou *hors-contexte*, si toutes les règles sont de la forme $A \rightarrow \alpha$.
4. *type 3* ou *linéaire à droite*, si toutes les règles sont de la forme $A \rightarrow a$, $A \rightarrow aB$ ou $S \rightarrow \epsilon$.

□

Remarque

- Toute grammaire de type 3 est de type 2.
- Toute grammaire de type 2 est de type 2.
- Toute grammaire de type 1 est de type 0.

□

Types de langages

Definition *Un langage $L \subseteq T^*$ est de type i*, s'il existe une grammaire $G = (N, T, P, S)$ de type i avec $L = L(G)$. □

Exemple

1. $\{a^n b^n c^n \mid n \in I\!N\}$ est de type 1.

$$\begin{array}{llll} S & \rightarrow abc \mid aAbc & \qquad Ab & \rightarrow bA \\ Ac & \rightarrow Bbcc & \qquad bB & \rightarrow Bb \\ aB & \rightarrow aaA \mid aa & & \end{array}$$

2. $\{a^n b^n \mid n \in I\!N\}$ est de type 2.
3. $\{a^n b^m \mid n, m \in I\!N\}$ est de type 3.

$$\begin{array}{ll} S & \rightarrow aS \mid bB \mid b \mid \epsilon \\ B & \rightarrow bB \mid b \end{array}$$

□

Equivalence langages linéaires à droite et langages réguliers

Théorème L est linéaire à droite (type 3) ssi L est régulier.
Cette équivalence est effective.

Preuve

- On montre d'abord "$\Rightarrow$". Soit $G = (N, T, P, S)$ une grammaire linéaire à droite telle que $L = L(G)$.
  On définit l'automate $A = (N \cup \{q_f\}, T, S, \Delta, F)$ tel que:
  - (A, a, B) ssi $A \rightarrow aB \in P$.
  - (A, a, q_f) ssi $A \rightarrow a \in P$.
  - $F = \{q_f\} \cup \{S \mid S \rightarrow \epsilon \in P\}$.
- On montre "$\Leftarrow$". Soit $A = (Q, \Sigma, q_0, \delta, F)$ un ADEF qui reconnait L.
  On définit la grammaire $G = (Q, \Sigma, P, q_0)$ avec:
  - $q \rightarrow a \in P$ ssi $\delta(q, a) \in F$.
  - $q \rightarrow aq' \in P$ ssi $\delta(q, a) = q'$.
  - $q_0 \rightarrow \epsilon \in P$ ssi $q_0 \in F$.

Exemple

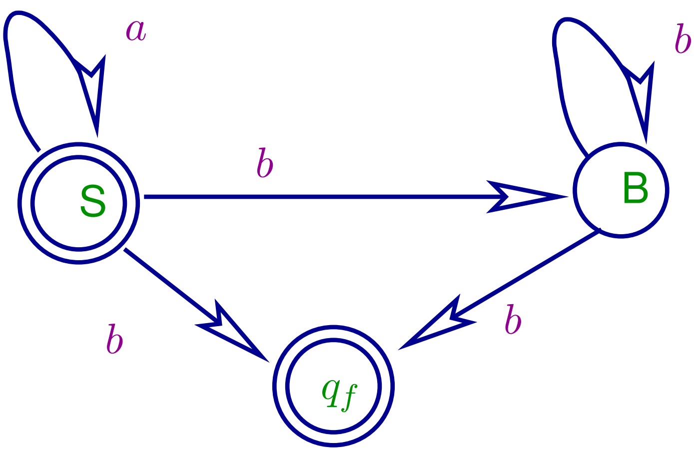
On considere la grammaire G donnée par: $\{a^n b^m \mid n, m \in \mathbb{N}\}$ est de type 3.

$$S \rightarrow aS \mid bB \mid b \mid \epsilon$$
$$B \rightarrow bB \mid b$$

Exemple

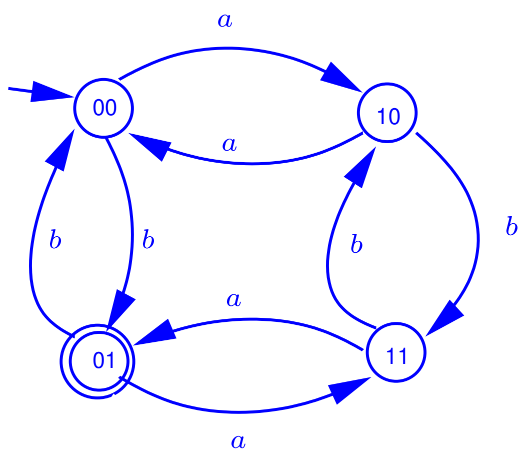
On considere la grammaire G donnée par: $\{a^n b^m \mid n, m \in \mathbb{N}\}$ est de type 3.

$$S \rightarrow aS \mid bB \mid b \mid \epsilon$$
$$B \rightarrow bB \mid b$$

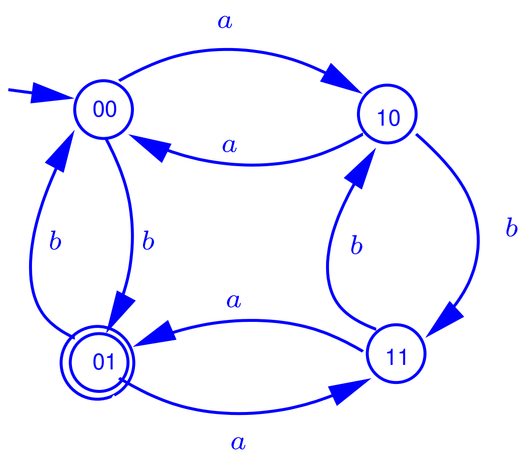


Yassine Lakhnech, Langages formels et automates Cours 6. Start – p.101/102

Exemple



Exemple



La grammaire associé:

$$
\begin{aligned}
S &\rightarrow aA_{10} \mid bA_{01} & \qquad A_{10} &\rightarrow aS \mid aA_{11} \\
A_{01} &\rightarrow bS \mid aA_{11} \mid \epsilon & \qquad A_{11} &\rightarrow bA_{10} \mid aA_{01}
\end{aligned}
$$

Yassine Lakhnech, Langages formels et automates Cours 6. Start – p.102/102

Eléments de logique

Nous allons dans les transparents qui suivent introduire les expressions arithmétiques et les prédicats. Le but est d'avoir un langage concis et rigoureux qui nous permettra de décrire et spécifier les programmes.
En L3, vous aurez un cours plus complet dédié à la logique et à l'étude des formules logiques.

Expressions arithmétiques

Nous supposons un ensemble dénombrable $\mathcal{X}$ de variables $x, y, z, \cdots, x_0, x_1, \cdots$.
Pour l'instant pour simplifier les choses, nous supposons que toutes les variables sont de type $\boldsymbol{Z}$.
Nous pouvons alors définir des expressions à partir de variables et de constantes.
L'ensemble des expressions, dénoté par Exp, est défini inductivement de la manière suivante :

- Cas de base : une variable dans $\mathcal{X}$ est une expression, une constante dans $\boldsymbol{Z}$ est une expression.
- Induction : Si e_1 et e_2 sont des expressions alors (e_1), $e_1 + e_2$, $e_1 - e_2$, $e_1 * e_2$ sont des expressions.

Expressions arithmétiques

L'ensemble Exp est donc le plus petit ensemble (par rapport à $\subseteq$) qui contient toutes les variables et les constantes et qui est fermé par les règles de construction suivantes :
A partir de e_1 et e_2 on peut construire les expressions (e_1), $e_1 + e_2$, $e_1 - e_2$, $e_1 * e_2$. Tacitement nous nous permettrons d'utiliser d'autre opération comme division, puissance, etc...
Exemple :

- $0 + x$ est une expression
- $(1 + 2) * x$ est une expression
- $(x + y) \wedge 3 - 4$ n'est pas une expression

Sémantique des expressions arithmétiques

Quelle valeur dénote l'expression $0 + x$?

Sémantique des expressions arithmétiques

Quelle valeur dénote l'expression $0 + x$?
Ceci dépend de la valeur de x.

Sémantique des expressions arithmétiques

Quelle valeur dénote l'expression $0 + x$?
Ceci dépend de la valeur de x.
Pour donner une sémantique à une expression, il faut donc supposer des valeurs associées aux variables. Une telle association c.a.d. une application $\sigma : \mathcal{X} \longrightarrow \boldsymbol{Z}$ est appelée état.
L'ensemble des états est dénoté Σ.
Pour un état $\sigma \in \Sigma$ et une expression e, on dénote par $[\![e]\!]\sigma$ la valeur de e dans l'état σ. Nous définissons l'application $[\![\ ]\!] : Exp \longrightarrow (\Sigma \longrightarrow \boldsymbol{Z})$:

- Cas de base :
  - pour une variable x, on a $[\![x]\!]\sigma = \sigma(x)$
  - pour un constante n, on a $[\![n]\!]\sigma = n$
- Pas d'induction : $[\![e_1\, op\, e_2]\!]\sigma = ([\![e_1]\!]\sigma)\ op\ ([\![e_2]\!]\sigma)$ et $[\![(e)]\!]\sigma = [\![e]\!]\sigma$.

Expressions - sémantique : exemples

Soit σ un état tel que $\sigma(x) = 1$, $\sigma(y) = -1$ et $\sigma(z) = 3$.
Quelle est la valeur des expressions suivantes dans σ :

- $x + y$
- $0 - y$
- $x + z + y$
- $x + y * z$ attention à l'ambiguité et aux priorités
- $(x + y) * z$

Notation- variation d'état

Soit $\sigma : \mathcal{X} \longrightarrow \mathbb{Z}$ un état et $n \in \mathbb{Z}$.
Alors, $\sigma[n/x]$ dénote l'état qui associe à toute variable y autre que x la valeur $\sigma(y)$ et à x la valeur n.
On étend cette notation à n variables disjointes :
$\sigma[n_1, \cdots, n_n/x_1, \cdots, x_n]$.
Quelques faits :

- $\sigma[n/x](x) = n$
- $\sigma[n/x](y) = \sigma(y)$, si y est différent de x
- $\sigma[\sigma(x)/x] = \sigma$
- $[\![e]\!]\sigma[n/x] = [\![e]\!]\sigma$, si x n'apparaît pas dans e

Prédicats et formules logiques

L'ensemble des prédicats (formules logiques) de 1er-ordre est défini inductivement par :

- Base :
  - Les constante V et F sont des prédicats.
  - Si e_1, e_2 sont des expressions alors $e_1 = e_2$, $e_1 < e_2$, $e_1 \leq e_2$, $e_1 > e_2$, $e_1 \geq e_2$ sont des prédicats.
- Induction : Si P_1 et P_2 sont des prédicats alors $\neg P_1$, $P_1 \wedge P_2$, $P_1 \vee P_2$, $\exists x \cdot P_1$ et $\forall x \cdot P_1$sont des prédicats.

Exemples :

- $\forall y \exists x \cdot y = 2 * x \vee y = 2 * x + 1$: toujours vrai
- $\forall x \exists y \cdot y < x \wedge y \geq 0$: toujours faux
- $\exists y \cdot x = 2 * y$: dépend de la valeur de x

Sémantique des prédicats

Pour chaque prédicat P et état σ, nous voulons définir quand σ satisfait P, dénoté par $\sigma \models P$. Cette définition est par induction sur la structure de P :

- Cas de base :
  - $\sigma \models$ V et $\sigma \not\models$ F
  - $\sigma \models e_1 \sim e_2$ ssi $[\![e_1]\!]\sigma \sim [\![e_2]\!]\sigma$, pour $\sim \in \{<, \leq, =, >, \geq, ...\}$.
- Induction :
  - $\sigma \models \neg P$ ssi $\sigma \not\models P$
  - $\sigma \models P_1 \wedge P_2$ ssi $\sigma \models P_1$ et $\sigma \models P_2$
  - $\sigma \models P_1 \vee P_2$ ssi $\sigma \models P_1$ ou $\sigma \models P_2$
  - $\sigma \models \exists x \cdot P$ ssi il existe $n \in \boldsymbol{Z}$ tel que $\sigma[n/x] \models P$
  - $\sigma \models \forall x \cdot P$ ssi pour tout $n \in \boldsymbol{Z}$, on a $\sigma[n/x] \models P$

Validité et satisfiabilité

- Un prédicat P est valide, si pour tout état σ on a $\sigma \models P$.
- Un prédicat P est satisfiable, s'il existe un état σ tel que $\sigma \models P$.

Nous montrons :

- Pour tout prédicat P, P est valide ssi $\neg P$ est insatisfiable.
- Il existe un prédicat P tel que P et $\neg P$ sont satisfiables.

Automates étendus, invariants et vérification de programmes

Un exemple

Exemple :

$$x, y, z \ : \ \boldsymbol{Z};$$
$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

On se pose la question suivante : quelle est la sémantique de ce programme? c.a.d. quelle fonction réalise-t-il?

Un exemple

Exemple :

$$x, y, z \ : \ \boldsymbol{Z};$$
$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

$$\uparrow$$

$$\begin{array}{ll} x & x_0 \\ y & y_0 \\ z & z_0 \end{array}$$

Un exemple

Exemple :

$$x, y, z \;:\; \boldsymbol{Z};$$

$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

		↑
x	x_0	x_0
y	y_0	y_0
z	z_0	0

Un exemple

Exemple :

$$x, y, z \;:\; \boldsymbol{Z};$$

$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

		↑
x	x_0	$2 * x_0$
y	y_0	y_0
z	z_0	0

Un exemple

Exemple :

$x, y, z \; : \; \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

		↑
x	x_0	$2 * x_0$
y	y_0	y_0
z	z_0	1

Un exemple

Exemple :

$x, y, z \; : \; \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

		↑
x	x_0	$4 * x_0$
y	y_0	y_0
z	z_0	1

Un exemple

Exemple :

$$x, y, z \ : \ \boldsymbol{Z};$$
$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

$$\begin{array}{lll} & & \uparrow \\ x & x_0 & 4 * x_0 \\ y & y_0 & y_0 \\ z & z_0 & 2 \end{array}$$

Un exemple

Exemple :

$$x, y, z \ : \ \boldsymbol{Z};$$
$$z := 0; \quad \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od}$$

$$\begin{array}{lll} & & \uparrow \\ x & x_0 & 4 * x_0 \\ y & y_0 & y_0 \\ z & z_0 & 2 \end{array}$$

Le programme sous forme d'automate étendu

Le flux de contrôle

$x, y, z \ : \ \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_2$ $\uparrow q_3$

Le programme sous forme d'automate étendu

Le flux de contrôle

$x, y, z \ : \ \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_2$ $\uparrow q_3$

On distingue deux aspects liés à l'exécution d'un programme :

1. L'évolution des valeurs des variables : aspect données
2. Quelle action on doit exécuter au prochain pas : aspect contrôle.

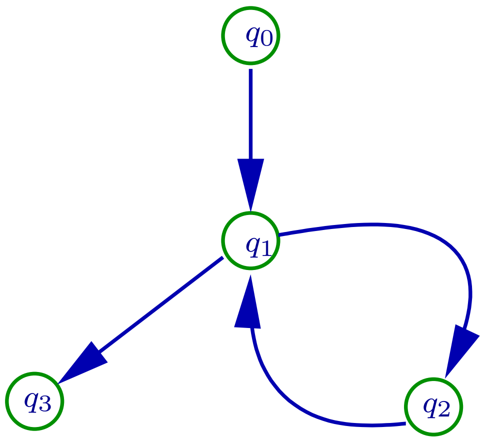
Le programme sous forme d'automate étendu

Le flux de contrôle

$x, y, z \ : \ \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_2$ $\uparrow q_3$



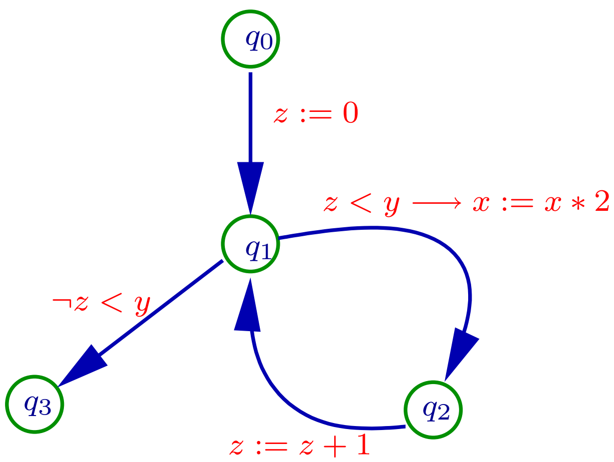
Le programme sous forme d'automate étendu

Le flux de contrôle

$x, y, z \ : \ \boldsymbol{Z};$

$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_2$ $\uparrow q_3$



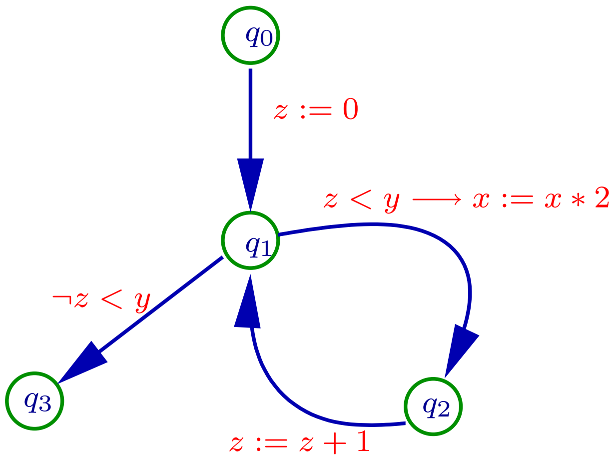
Le programme sous forme d'automate étendu

Le flux de contrôle

$x, y, z \;:\; \boldsymbol{Z};$

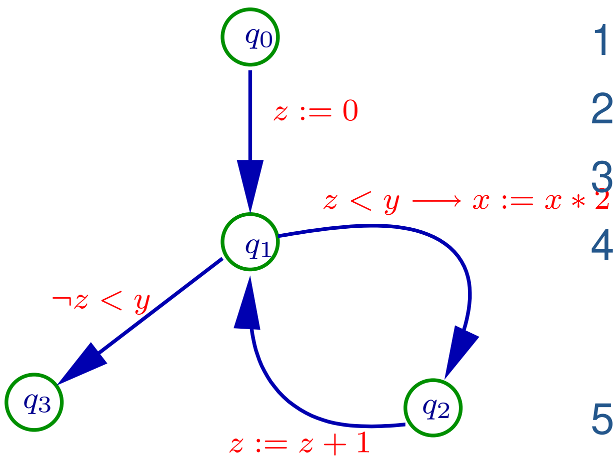
$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_2$ $\uparrow q_3$



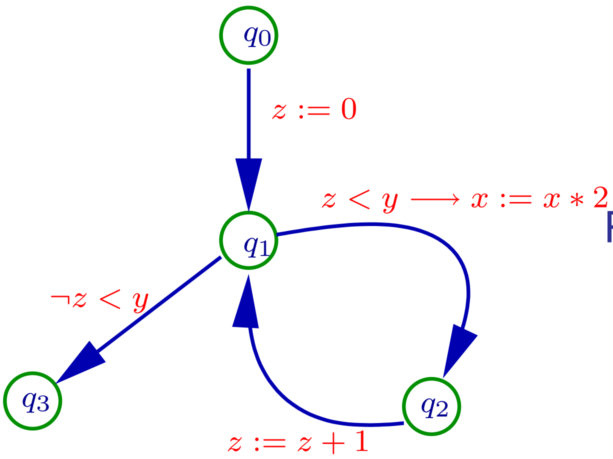
q_0, q_1, q_2, q_3 sont appelés état de contrôle. $(q_0, z := 0, q_1)$ est une transition.

Automates étendus : un exemple



1. Déclaration : $x : \boldsymbol{Z}, y : \boldsymbol{Z}, z : \boldsymbol{Z}$
2. Etats de contrôle : q_0, q_1, q_2, q_3
3. Etat de contrôle initial (de départ) : q_0
4. Transitions : $(q_0, z := 0, q_1)$, $(q_1, z < y \rightarrow x := x * 2, q_2)$, $(q_2, z := z + 1, q_1)$ et $(q_1, \neg z < y, q_3)$.
5. Etat final (terminal) : q_3

Automates étendus : un exemple



Remarque :

- Dans la transition $(q_1, z < y \to x := x * 2, q_2)$ $z < y$ est appelée la garde. On ne peut prendre cette transition que si $z < y$. Dans les autres transitions la garde est le prédicat vrai; dans ce cas on peut ne pas l'écrire explicitement.
- Dans la transition $(q_1, \neg z < y, q_3)$, il n'y a pas d'affectation donnée explicitement. C'est la même chose qu'avoir l'affectation $x := x$ (l'identité).

Automates étendus : la définition

Un automate étendu est donné par un quintuplet

$$(D, Q, q_0, \mathcal{T}, Q_t) \text{ où}$$

- D est une liste finie de déclarations.
- Q est un ensemble fini d'états de contrôle.
- $q_0 \in Q$ est l'état de contrôle de départ. On dit aussi état de contrôle initial.
- $\mathcal{T}$ est un ensemble fini de transitions de la forme $(q, g \to x := e, q')$ où g est un prédicat appelé garde. Uniquement des variables déclarées apparaissent dans la garde g et dans l'affectation $x := e$.
- $Q_t \subseteq Q$ est l'ensemble des états de contrôle terminaux (finals).

Etats et configuration

- Les valeurs des variables à chaque instant de l'exécution sont données par un état. Il faut distinguer entre état et état de contrôle. Un état $\sigma : \mathcal{X} \longrightarrow \mathbb{Z}$ est donc une application qui associe à chaque variable une valeur dans $\mathbb{Z}$.
- A chaque instant de l'exécution nous avons donc un état de contrôle q et un état σ. La paire (q, σ) est appelée configuration.

Pour un automate étendu A, on dénote par Conf_A l'ensemble des configurations de A.

Nous écrirons simplement Conf quand il n'y a pas d'ambiguité.

Exemples de configurations

Pour notre exemple, nous pouvons par exemple observer les configurations suivantes :

$$(q_0, [x \mapsto 3, y \mapsto 2, z \mapsto 10]) \to (q_1, [x \mapsto 3, y \mapsto 2, z \mapsto 0]) \to$$
$$(q_2, [x \mapsto 6, y \mapsto 2, z \mapsto 0]) \to (q_1, [x \mapsto 6, y \mapsto 2, z \mapsto 1]) \to$$
$$(q_2, [x \mapsto 12, y \mapsto 2, z \mapsto 1]) \to (q_1, [x \mapsto 12, y \mapsto 2, z \mapsto 2]) \to$$
$$(q_3, [x \mapsto 12, y \mapsto 2, z \mapsto 2])$$

Cette séquence de configurations représente l'exécution de notre programme exemple quand initialement nous avons $x = 3 \wedge y = 2 \wedge z = 10$.

Relation de transition

Dans ce qui suit nous supposons un automate étendu A donné.
Nous allons définir une relation entre configurations, *la relation de transition.*
Cette relation décrit quand on peut passer d'une configuration à une autre.
Nous définissons $\rightarrow \subseteq \mathsf{Conf} \times \mathsf{Conf}$:

$(q, \sigma) \rightarrow (q', \sigma')$ ssi il existe une transition $(q, g \rightarrow x := e, q') \in \mathcal{T}$ telle que

1. $\sigma \models g$ et
2. $\sigma' = \sigma[[\![e]\!]\sigma/x]$.

Exemple : Vérifier les transitions données dans le transparent précédent.

Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$
while $x \neq y$ do if $x \geq y$ then $x := x - y$
else $y := y - x$ fi
od

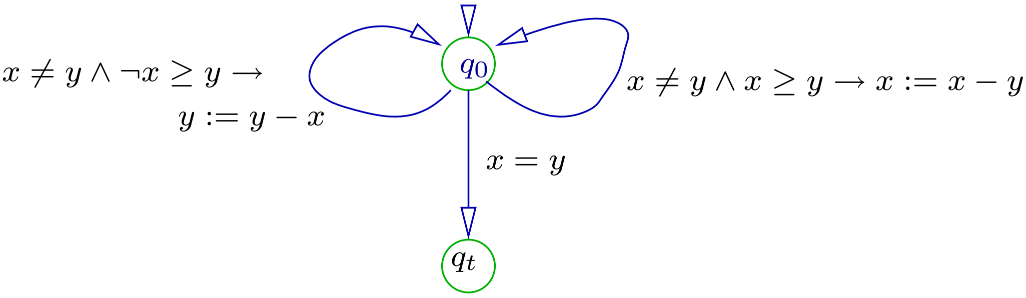
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

while $x \neq y$ do if $x \geq y$ then $x := x - y$

else $y := y - x$ fi

od



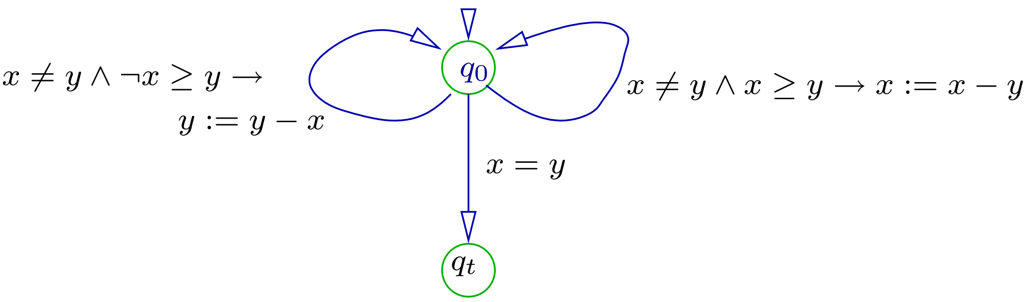
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

while $x \neq y$ do if $x \geq y$ then $x := x - y$

else $y := y - x$ fi

od



Pré-condition : $x = x_0 \wedge y = y_0 \wedge x_0 > 0 \wedge y_0 > 0$.

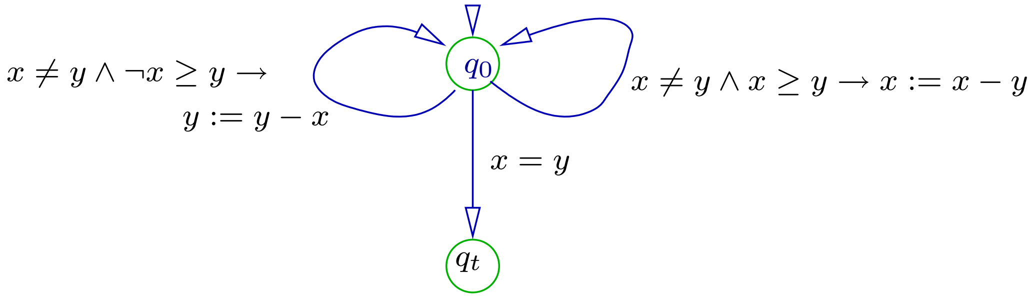
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

while $x \neq y$ do if $x \geq y$ then $x := x - y$

else $y := y - x$ fi

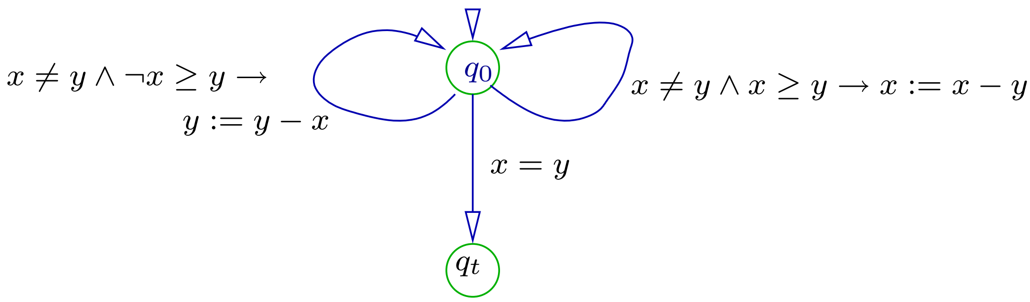
od



Pré-condition : $x = x_0 \wedge y = y_0 \wedge x_0 > 0 \wedge y_0 > 0$.

Post-condition : $x = y \wedge x = pgcd(x_0, y_0)$.

Exemples de traces d'execution



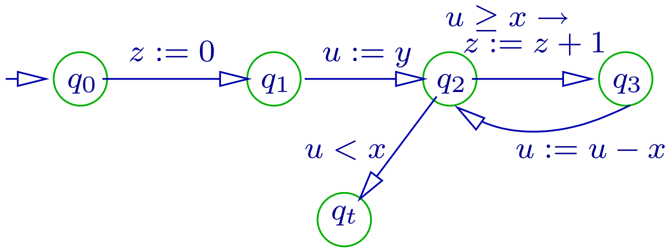
Développer (au tableau) les traces d'exécution avec une configuration initiale qui satisfait :

- $x = 2 \wedge y = 4$
- $x = 2 \wedge y = 5$
- $x = 5 \wedge y = 2$
- $x = 1 \wedge y = 3$
- $x = 0 \wedge y = 5$
- $x = -1 \wedge y = -2$

Exemples d'automates étendus-2

$x, y, z, u : \mathbb{Z}$

$z := 0; u := y;$ while $u \geq x$ do $z := z + 1; u := u - x$ od

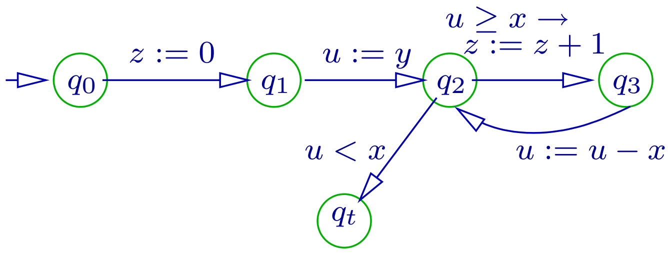


Pré-condition : $x > 0 \wedge y \geq 0$.

Exemples d'automates étendus-2

$x, y, z, u : \mathbb{Z}$

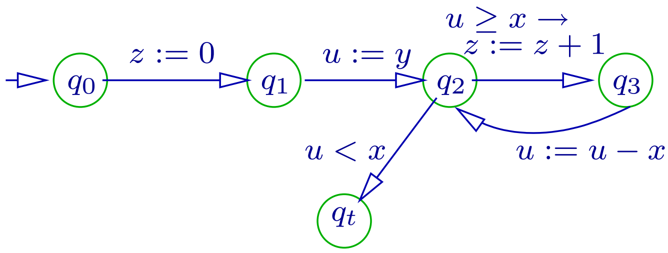
$z := 0; u := y;$ while $u \geq x$ do $z := z + 1; u := u - x$ od



Pré-condition : $x > 0 \wedge y \geq 0$.

Post-condition : $y = z * x + u \wedge u < x$.

Exemples de traces d'execution



Développer (au tableau) les traces d'exécution avec une configuration initiale qui satisfait :

- $x = 2 \wedge y = 3$
- $x = 3 \wedge y = 2$
- $x = 0 \wedge y = 2$
- $x = -2 \wedge y = -3$

Traces d'exécution

Soit A un automate étendu.
Une *trace finie* de A est une séquence de configurations :

$$(q_0, \sigma_0) \cdots (q_n, \sigma_n) \text{ où}$$

$n \in \mathbb{N}$ et telle que :

1. $q_0 \in Q_0$
2. pour tout $i \in \{0, \cdots, n-1\}$ on a

$$(q_i, \sigma_i) \longrightarrow (q_{i+1}, \sigma_{i+1})$$

L'entier n est la *longueur* de la trace.

L'ensemble de toutes les traces finies de A est dénoté $\mathrm{Tr}_f(A)$.

Traces maximales et traces terminales

Une trace finie

$$(q_0, \sigma_0) \cdots (q_n, \sigma_n)$$

est dite *maximale*, s'il n'existe pas de configuration (q, σ) telle que $(q_n, \sigma_n) \longrightarrow (q, \sigma)$.
Elle est *terminale*, si elle est maximale et $q_n \in Q_t$.
L'ensemble de toutes les traces finies maximales de A est dénoté $\text{Tr}_{fm}(A)$.

L'ensemble de toutes les traces finies terminales de A est dénoté

$\text{Tr}_{ft}(A)$.

Traces d'exécution infinies

Une *trace infinie* de A est une séquence infinie de configurations :

$$(q_0, \sigma_0) \cdots (q_i, \sigma_i) \cdots \text{ où}$$

$n \in I\!N$ et telle que :

1. $q_0 \in Q_0$
2. pour rout $i \in I\!N$ on a

$$(q_i, \sigma_i) \longrightarrow (q_{i+1}, \sigma_{i+1})$$

L'ensemble de toutes les traces infinies de A est dénoté $\text{Tr}_{inf}(A)$.

Relation entrée-sortie d'un automate

Soit A un automate étendu.
Alors A réalise la relation $R(A)$ entre états définie de la manière suivante :
$(\sigma, \sigma') \in R(A)$ ssi il existe une trace <u>terminale</u> $(q_0, \sigma_0) \cdots (q_n, \sigma_n)$ de A telle que :

- $\sigma_0 = \sigma$ et $\sigma_n = \sigma'$.

Au tableau : déterminer les relations des automates vus dans les exemples.

Spécification de propriétés

Nous allons considérer des paires de prédicats (P, Q) comme spécifications de propriétés d'automates étendus.
Dans la paire (P, Q), P est appelé *pré-condition* et Q est appelé *post-condition* (cf. INF 231).

Convention : Nous réservons les variables avec 0 comme indice, $x_0, y_0, \cdots$, pour désigner les valeurs initiales de $x, y \cdots$. Nous interdisons donc l'utilisation de ces variables dans les automates étendus. Ces variables sont souvent appelées *variables logiques*.

Correction partielle

un automate étendu A est *partiellement correct* par rapport à la spécification (P, Q) ssi pour tout $\sigma, \sigma' \in \Sigma$, on a :
Si

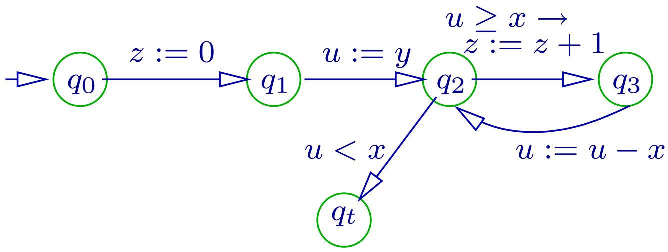
1. $\sigma'(x_0) = \sigma(x)$, pour toute variable $x \in \mathcal{X}$,
2. $\sigma \models P$ et $(\sigma, \sigma') \in R(A)$

alors $\sigma' \models Q$. Au tableau : vérifier informellement pour tous les exemples vus qu'ils sont corrects par rapport aux spécifications données.

Comment faire pour montrer qu'un automate est partiellement correct par rapport à une spécification?

Exemple d'automate étendu annoté

Rappel l'automate Div :



P la pré-condition : $x > 0 \wedge y \geq 0$.
Q la post-condition $y = z * x + u \wedge u < x$.

On veut montrer que Div satisfait la spécification (P, Q).

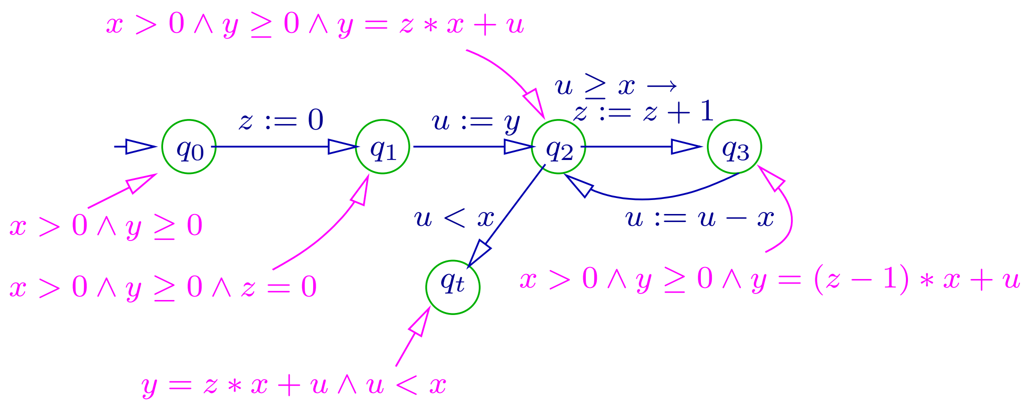
Méthode de vérification

1. On annote chaque état de contrôle q par un prédicat qu'on va appeler P_q. Ces prédicats sont à inventer.
2. On verifie les conditions suivantes:
   (a) On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q.
   (b) On vérifie que pour chaque état de contrôle initial q, la pré-condition implique P_q.
   (c) On vérifie que pour chaque état de contrôle terminal q, P_q implique la post-condition.

Méthode de vérification

1. On annote chaque état de contrôle q par un prédicat qu'on va appeler P_q. Ces prédicats sont à inventer.
2. On verifie les conditions suivantes:
   (a) On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q. Nous verrons comment montrer ceci sans considérer toutes les exécutions une par une.
   (b) On vérifie que pour chaque état de contrôle initial q, la pré-condition implique P_q.
   (c) On vérifie que pour chaque état de contrôle terminal q, P_q implique la post-condition.

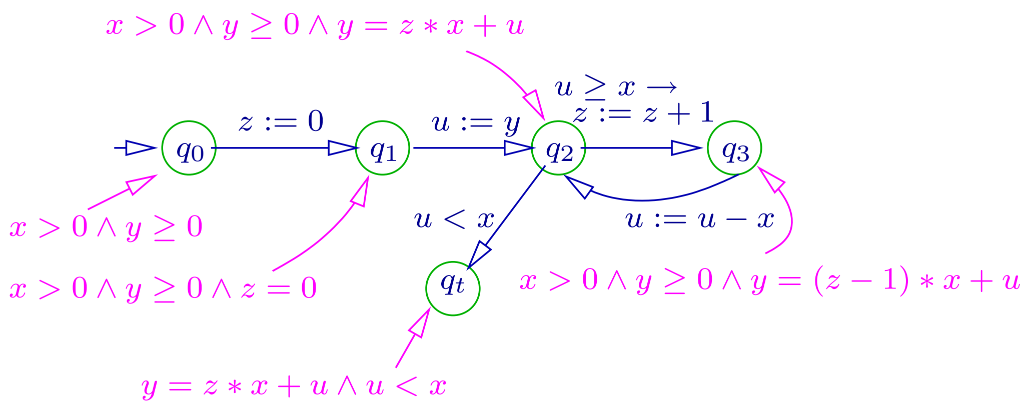
L'automate Div annoté



Nous allons vérifier au tableau les conditions

1. Chaque fois que dans une exécution on est dans q, les valeurs des variables satisfont P_q.
2. Pour chaque état de contrôle initial q, la pré-condition implique P_q.
3. Pour chaque état de contrôle terminal q, la P_q implique la post-condition.

L'automate Div annoté



Mais avant essayons de voir comment on peut montrer la condition (a) :

> On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q

de manière efficace.

Automate étendu inductif

Pour vérifier la condition : On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q il suffit de vérifier :

> Pour toute transition $(q, g \rightarrow x := e, q')$
> Pour tout états $\sigma \in \Sigma$,
> si $\sigma \models g \wedge P_q$ alors $\sigma[\llbracket e \rrbracket / x] \models P_{q'}$.

Quand cette condition est satisfaire nous disons que l'automate annoté est inductif.

Automate étendu inductif

Pour vérifier la condition : On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q il suffit de vérifier :

> Pour toute transition $(q, g \rightarrow x := e, q')$
> Pour tout états $\sigma \in \Sigma$,
> si $\sigma \models g \wedge P_q$ alors $\sigma[\llbracket e \rrbracket / x] \models P_{q'}$.

Quand cette condition est satisfaire nous disons que l'automate annoté est inductif. Exercice : Montrer que cette condition implique la condition :

> Chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q.

C'est donc une condition suffisante. Montrer qu'elle n'est pas nécessaire.

Automates étendus annotés - définition

- Un *automate étendu annoté* est un automate étendu muni d'une fonction qui associe à chaque état de contrôle q un prédicat P_q.
- Un automate étendu annoté est *correct* par rapport à la spécification (P, Q), si les conditions suivantes sont satisfaites :
  1. Il est inductif.
  2. Pour tout état de contrôle initial q, la formule $P \Rightarrow P_q$ est valide.
  3. Pour tout état de contrôle terminal q, la formule $P_q \Rightarrow Q$ est valide.

Méthode de vérification de Floyd

Pour vérifier qu'un automate étendu A satisfait une spécification (P, Q), il suffit de munir A d'une annotation telle qu'on obtient un automate étendu annoté correct par rapport à (P, Q).
Théorème : (sans démonstration)

- La méthode de vérification de Floyd est correct c.a.d. si on peut annoter un automate étendu A correctement par rapport à (P, Q), alors A est correcte par rapport à (P, Q).

Pour terminer appliquons la méthode de Floyd aux exemples vus dans le cours.