MCAL
Modèles de calcul et Validation d'algorithmes

2008/09

Intervenant:

- Cristian Ene e-mail: Cristian.Ene@imag.fr

Page du cours: http://www-verimag.imag.fr/˜ene/mcal/

Objectifs

Objectifs :

- Apprendre à analyser formellement un problème algorithmique
- Construire une solution algorithmique quand possible
- Apprendre à analyser les algorithmes (correction, terminaison, coût)

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.

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 $\mathbb{Z}$.
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 $\mathbb{Z}$ 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 depend de la valeur de x.

Sémantique des expressions arithmétiques

Quelle valeur dénote l'expression $0 + x$?
Ceci depend 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 apppelée 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 définisson l'application $[\![\ ]\!] : Exp \longrightarrow (\Sigma \longrightarrow \boldsymbol{Z})$:

- Cas de base :
  - pour une variable x, on a $[\![x]\!]\sigma = \sigma(x)$
  - pour une 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 $\sigma : \mathcal{X} \longrightarrow \boldsymbol{Z}$ un état et $n \in \boldsymbol{Z}$.
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 different de x
- $\sigma[\sigma(x)/x] = \sigma$
- $[\![e]\!]\sigma[n/x] = [\![e]\!]\sigma$, si x n'apparait 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
- $\exists y \cdot x = 2 * y$: depend 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 $\sigma \models P$.
- Un prédicat P est insatisfiable, si pour tout état σ on a $\sigma \not\models P$.

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 d’état et corréction partielle 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 :

$$
\begin{array}{llll}
x, y, z \ : \ \boldsymbol{Z}; & & & \\
& z := 0; & \text{while } z < y \text{ do } x := x * 2; \quad z := z + 1 \text{ od} & \\
& & \uparrow & \\
x & x_0 & x_0 & \\
y & y_0 & y_0 & \\
z & z_0 & 0 &
\end{array}
$$

Un exemple

Exemple :

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

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

$$\uparrow$$

x	x_0	$2 * x_0$
y	y_0	y_0
z	z_0	0

Un exemple

Exemple :

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

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

$$\uparrow$$

x	x_0	$2 * x_0$
y	y_0	y_0
z	z_0	1

Un exemple

Exemple :

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

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

$$\uparrow$$

x	x_0	$4 * x_0$
y	y_0	y_0
z	z_0	1

Un exemple

Exemple :

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

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

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

Un exemple

Exemple :

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

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

$$\uparrow$$

x	x_0	$4 * x_0$
y	y_0	y_0
z	z_0	2

Le programme sous forme d'automate étendu

Le flux de contrôl

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

$z := 0;$ 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 contrôl

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

$z := 0;$ 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.

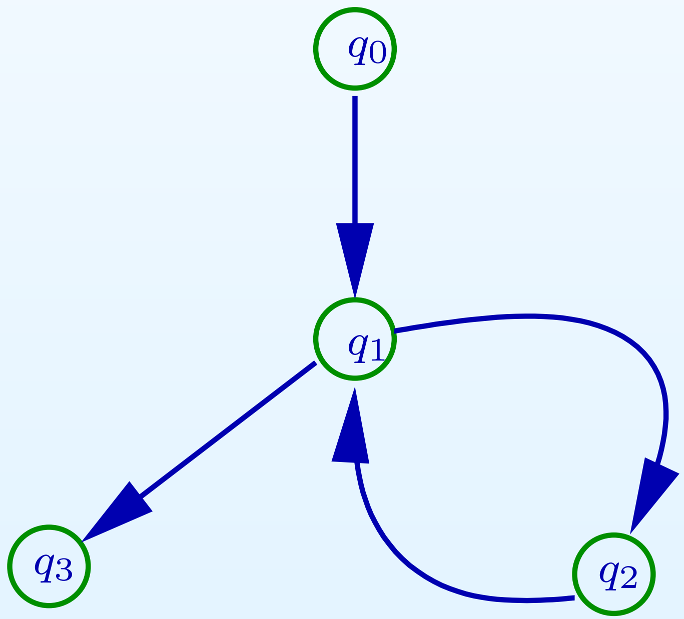
Le programme sous forme d'automate étendu

Le flux de contrôl

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

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

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



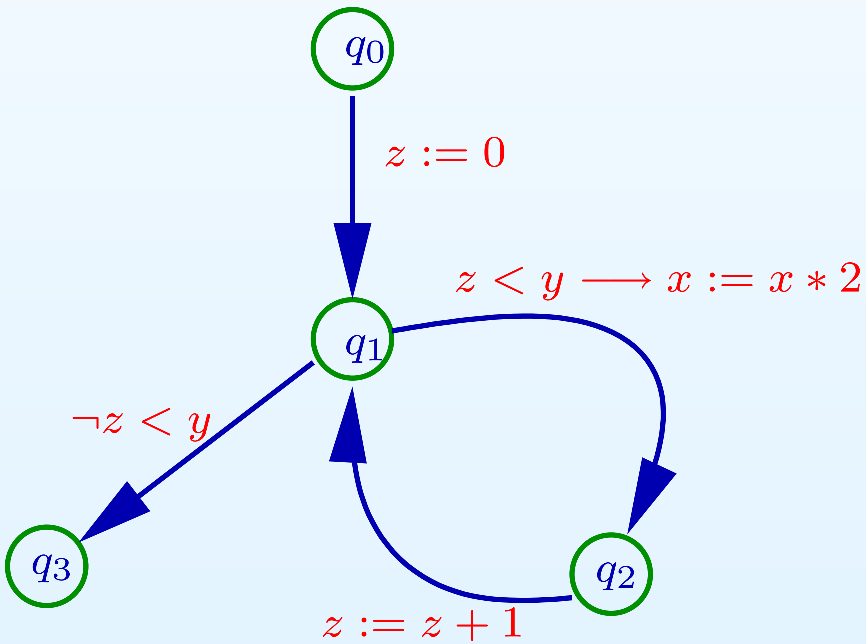
Le programme sous forme d'automate étendu

Le flux de contrôl

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

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

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



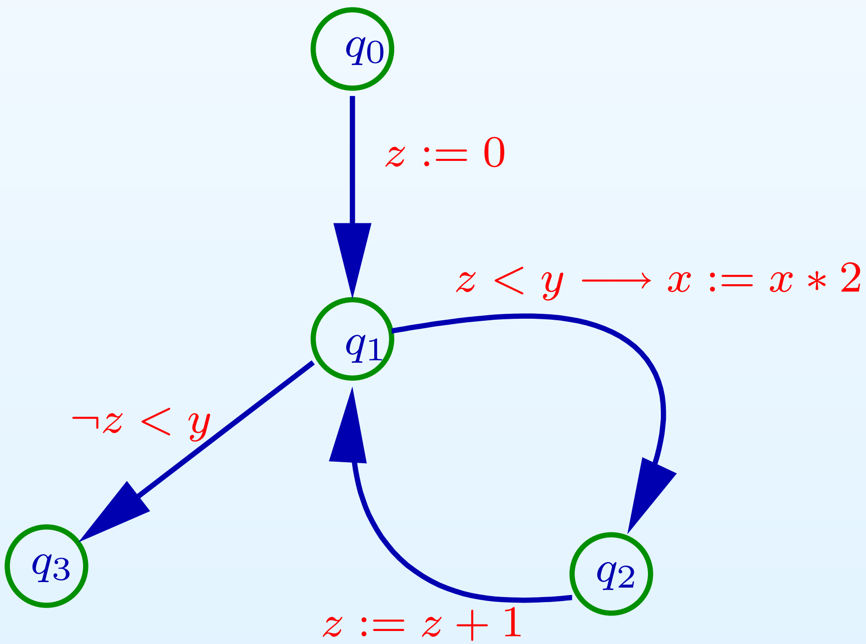
Le programme sous forme d'automate étendu

Le flux de contrôl

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

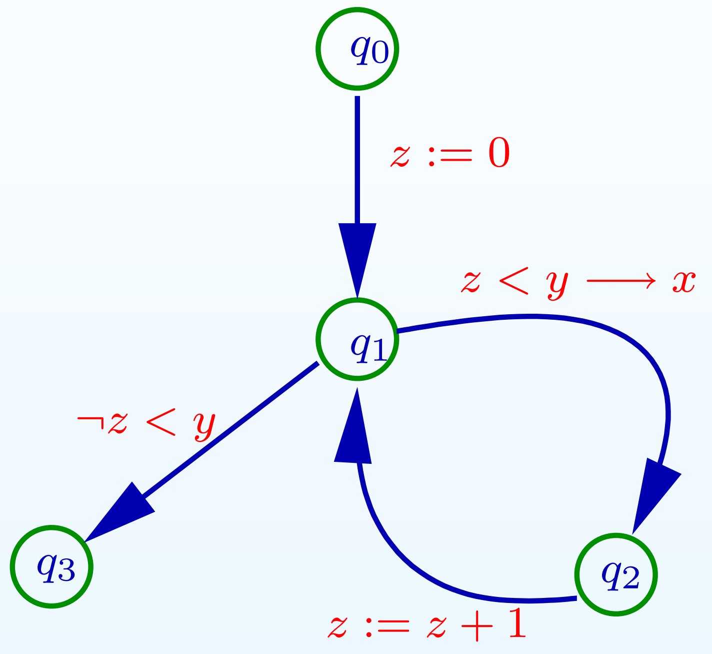
$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

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



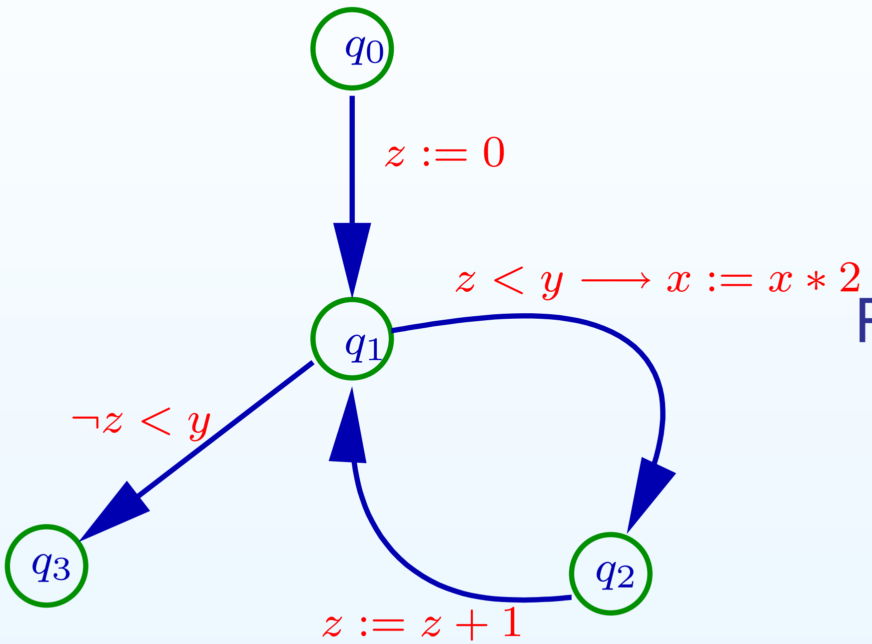
q_0, q_1, q_2, q_3 sont appelés état de contrôle. $(q_0, z := 0, q_1)$ est une transition.

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 $(q_1, z < y \rightarrow x := x * 2, q_2)$ $z < y$ 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'a pas d'affectation donnée explicitement. C'est la même chose qu'avoir l'affectation $x := x$ (l'identité).

Déclaration de variables

Une déclaration est de la forme $x : T$ avec $x \in \mathcal{X}$ et T un ensemble de valeurs comme $\boldsymbol{Z}$, l'ensemble $\boldsymbol{Z}^*$ des listes finies sur $\boldsymbol{Z}$, etc.... . Pour l'instant pour simplifier les choses, nous supposons que toutes les variables sont de type $\boldsymbol{Z}$.
On dit que deux déclarations $x : T$ et $y : T'$ sont disjointes, si x et y sont differentes.
Dans notre exemple nous avons les déclarations :

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

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 disjointes deux-à-deux.
- Q est un ensemble fini d'états de contrôle.
- $Q_0 \subseteq Q$ est l'ensemble des états de contrôle de départ. On dit aussi états de contrôle initials.
- $\mathcal{T}$ est un ensemble fini de transitions de la forme $(q, g \rightarrow x := e, q')$ 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 $\sigma : \mathcal{X} \longrightarrow \boldsymbol{Z}$ est donc une application qui associe à chaque variable une valeur dans $\boldsymbol{Z}$.
- 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 :

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

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

$$
\begin{array}{llll}
x, y : \mathbb{Z} & & & \\
\text{while} \;\; x \neq y & \text{do if } x \geq y & \text{then } x := x - y & \\
 & & \text{else } y := y - x \text{ fi} & \\
 & \text{od} & &
\end{array}
$$

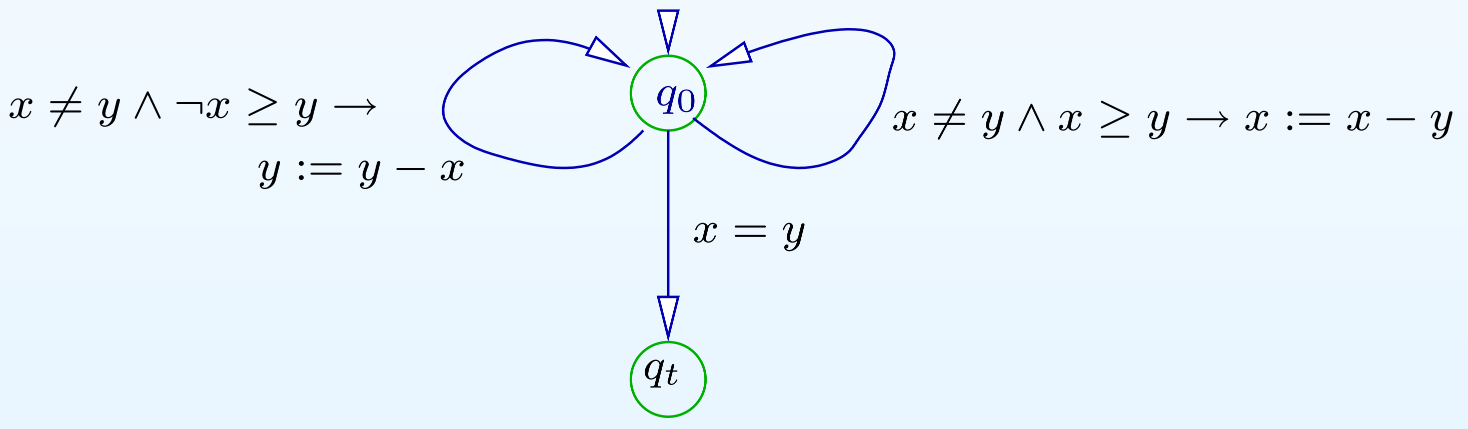
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

od



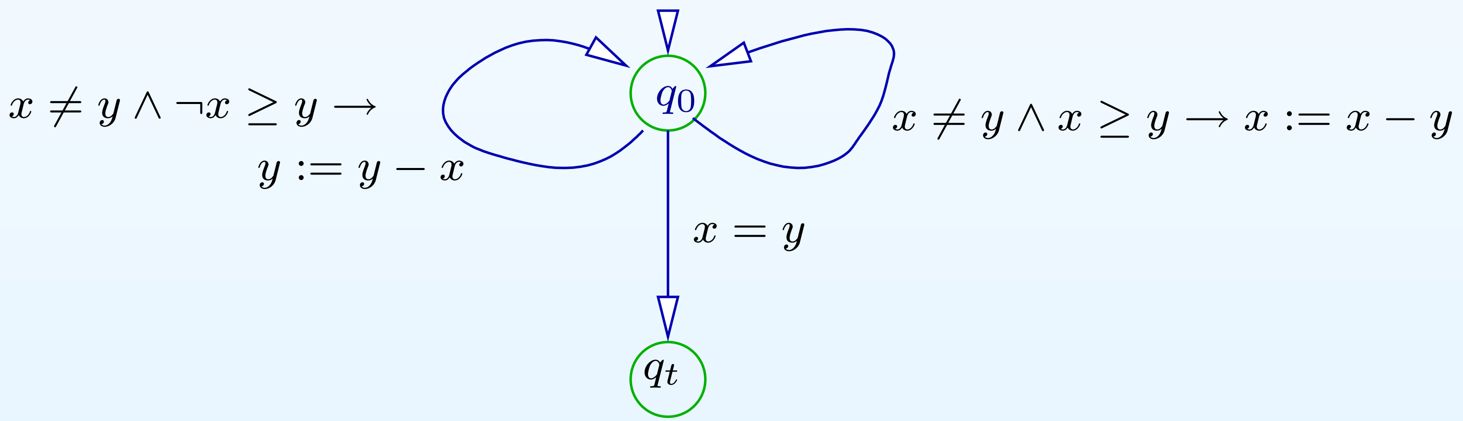
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

od



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

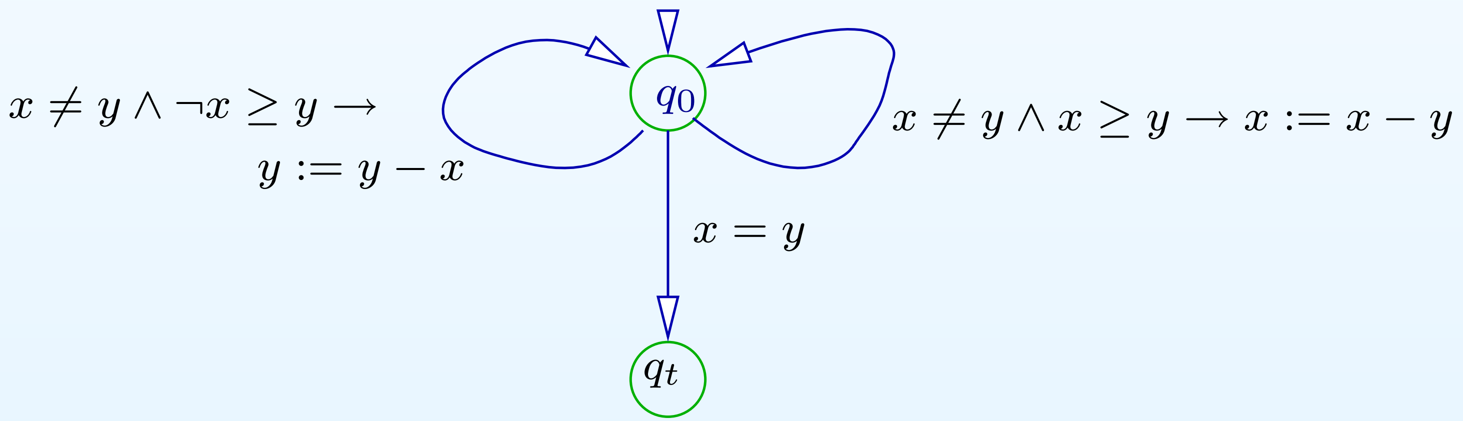
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

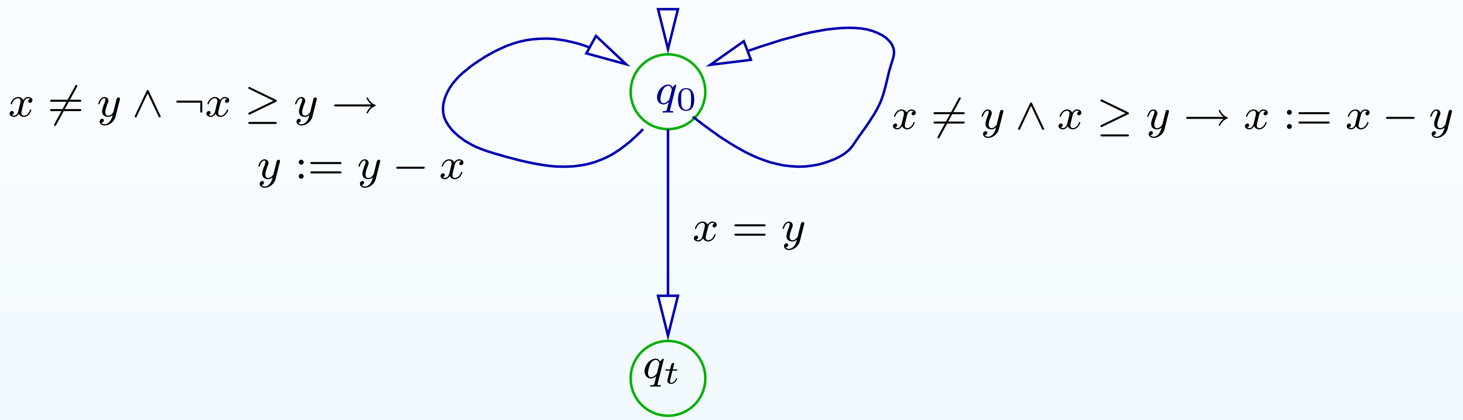
od



Pre-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



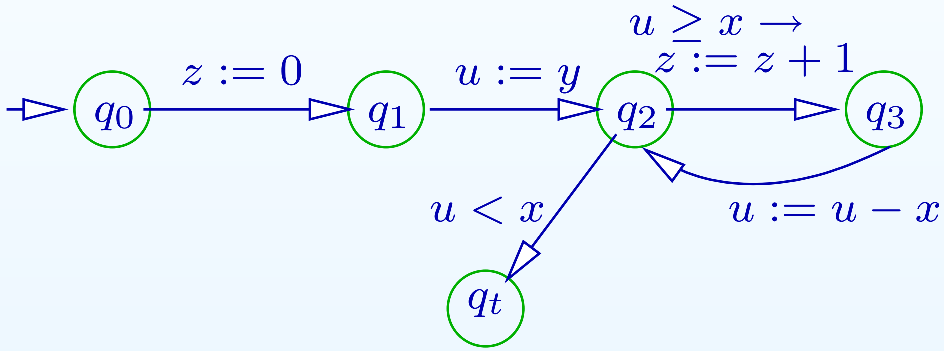
Développer (au tableau) les traces d'exècution avec une configuration initiale qui satisfait :

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

Exemples d'automates étendus-2

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

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

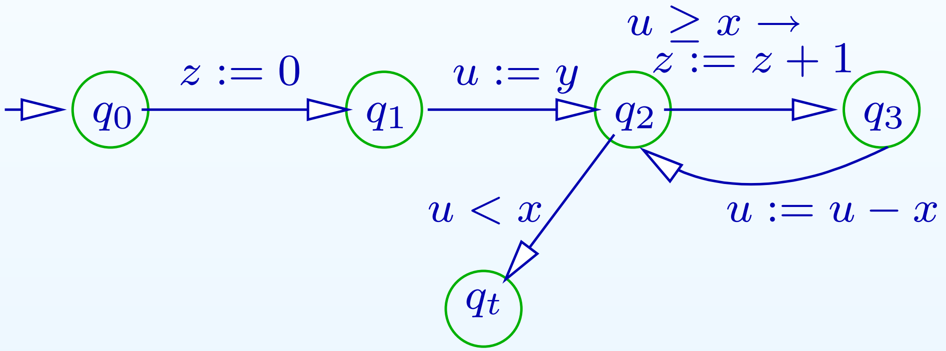


Pre-condition : $x > 0 \wedge y \geq 0$.

Exemples d'automates étendus-2

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

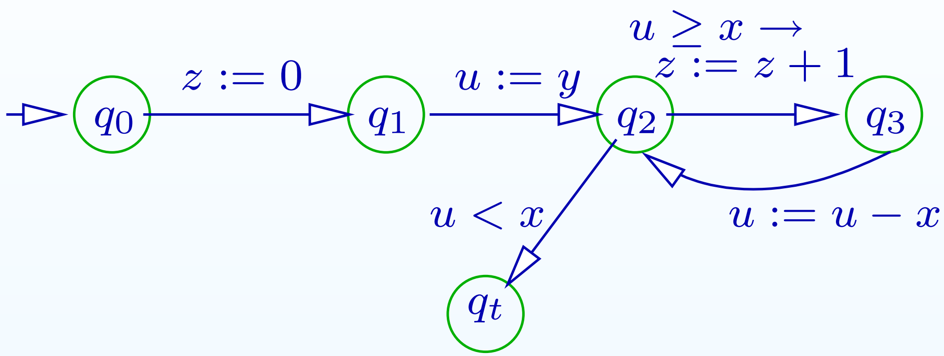
$z := 0; u := y;$ while $u \geq x$ do $z := z + 1; u := u - x$ od



Pre-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'exècution 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 et σ_0 un état (initial) des variables.
Une *trace finie* de A et σ_0,est une séquence de configurations :

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

$n \in I\!N$ 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 $\mathrm{Tr}_f(A, \sigma_0)$.

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 et σ_0 est dénoté $\mathrm{Tr}_{fm}(A, \sigma_0)$.

L'ensemble de toutes les traces finies terminales de A et σ_0 est dénoté $\mathrm{Tr}_{ft}(A, \sigma_0)$.

Traces d'exècution infinies

Une *trace infinie* de A et σ_0 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 tout $i \in I\!N$ on a

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

L'ensemble de toutes les traces infinies de A et σ_0 est dénoté

Tr$_{inf}(A, \sigma_0)$.

Relation entrée-sortie induite par 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 terminale $(q_0, \sigma_0) \cdots (q_n, \sigma_n)$ de A telle que :

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

Au tableau : déterminer les relations induites par les automates vus dans les exemples.

Spécification de propriétés

Nous allons considerer des pairs de prédicats (P, Q) comme spécifications de propriétés d'automates étendus.
Dans la pair (P, Q), P est appelé *pre-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 designer 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 correcte* par rapport à la spécification (P, Q) ssi pour tout $\sigma, \sigma' \in \Sigma$,
Si

$$\sigma \models P \text{ et } (\sigma, \sigma') \in R(A)$$

alors

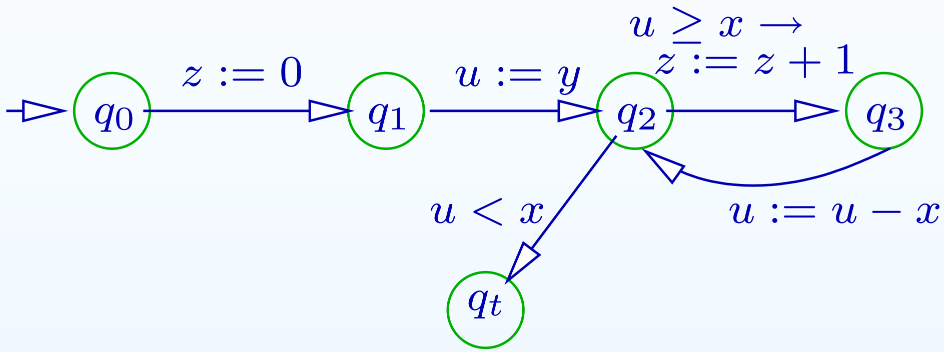
$$\sigma'[\sigma(x)/x_0, \cdots, \sigma(y)/y_0] \models Q$$

Au tableau : vérifier informellement pour tous les exemples vus qu'ils sont correctes par rapport aux spécifications données.

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

Exemple d'automates étendus annotés

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).

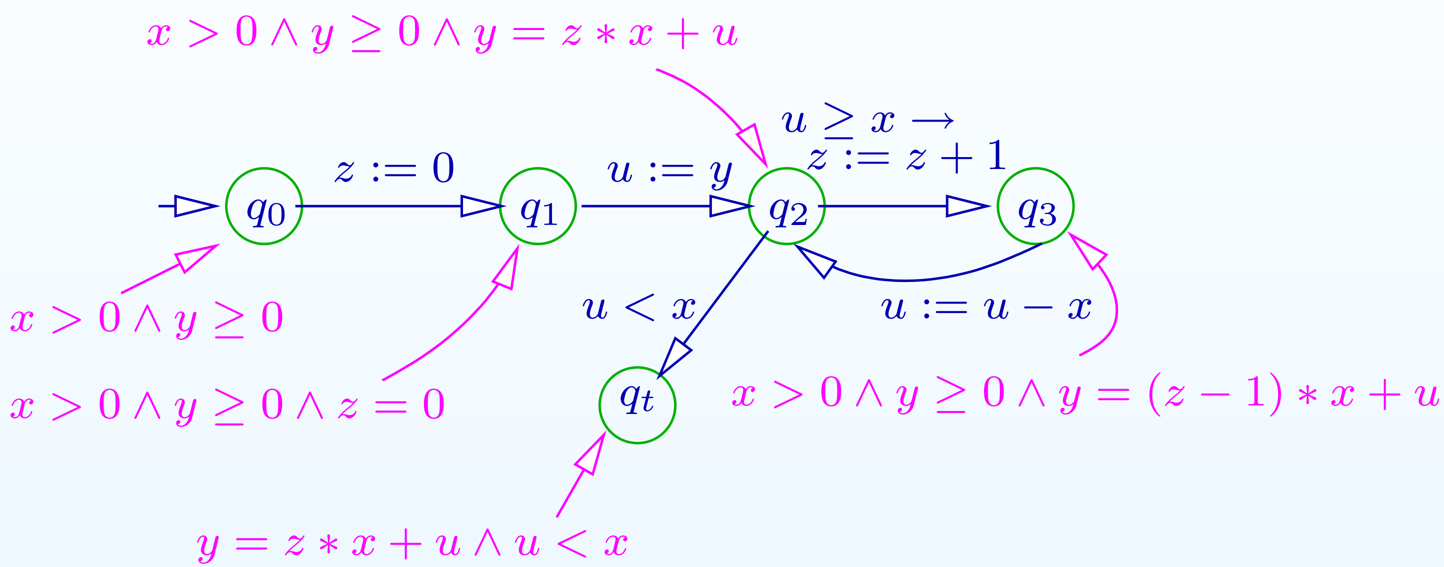
Méthode de vérification

1. On annote chaque état de controle q par un prédicat qu'on va appeler P_q.
2. On vérifie que, chaque fois où dans une exécution on est dans q, les valeurs des variables satisfont P_q.
3. On vérifie que pour chaque état de controle initial q, la pré-condition implique P_q.
4. On vérifie que pour chaque état de controle terminal q, la P_q implique la postcondition.

Méthode de vérification

1. On annote chaque état de controle q par un prédicat qu'on va appeler P_q.
2. 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.
3. On vérifie que pour chaque état de controle initial q, la pré-condition implique P_q.
4. On vérifie que pour chaque état de controle terminal q, la P_q implique la postcondition.

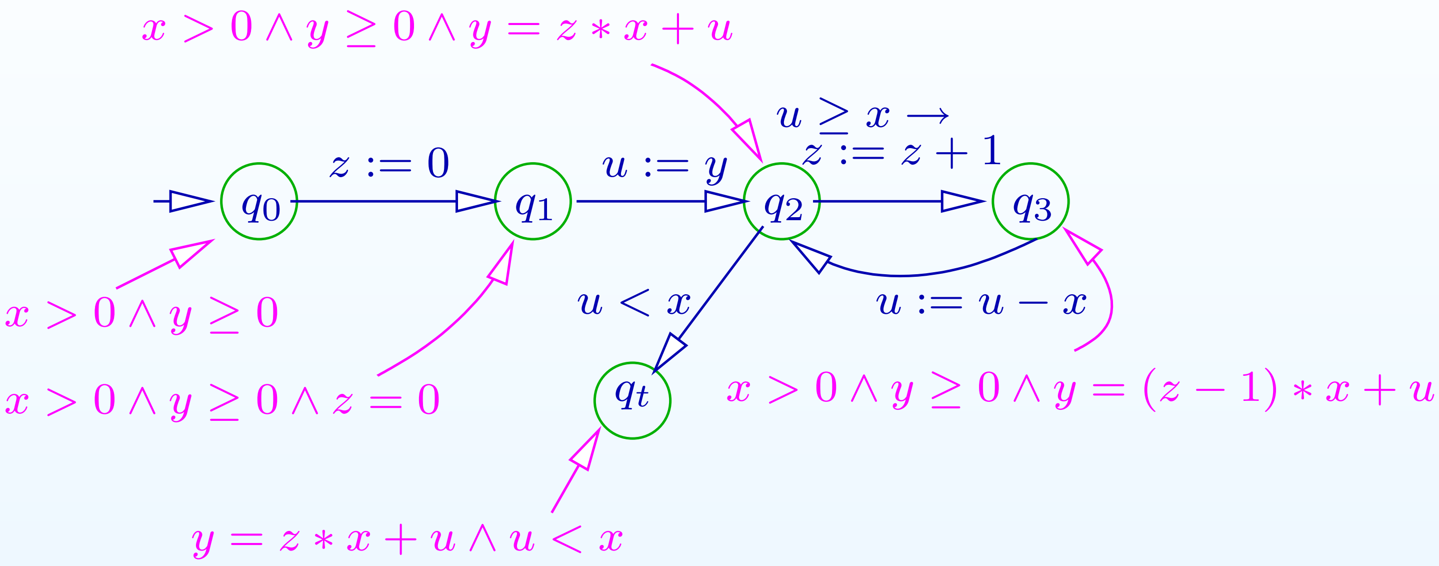
L'automate Div annoté



Nous allons vérifier au tableau les conditions

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

L'automate Div annoté



Mais avant essayons de voir comment on peut montrer la première condition de manière efficace.

Inductivité d'un automate étendu

Pour vérifier la première condition, il suffit de vérifier :

Pour toute transition $(q, g \to x := e, q')$
Pour tout état $\sigma \in \Sigma$,
si $\sigma \models g \wedge P_q$ alors $\sigma[[\![e]\!]\sigma/x] \models P_{q'}$.

Cette condition nous l'appelerons l'inductivité de l'automate annoté.

Inductivité d'un automate étendu

Pour vérifier la première condition, il suffit de vérifier :

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

Cette condition nous l'appelerons l'inductivité de l'automate annoté. 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 siffisante. Montrer qu'elle n'est pas nécessaire.

Automates étendus annotés - définition

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

Theorem 0.1 *Théorème : (sans démonstration)*
La méthode de vérification de Floyd est correcte 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.

Automates étendus, invariants de transition et términaison de programmes

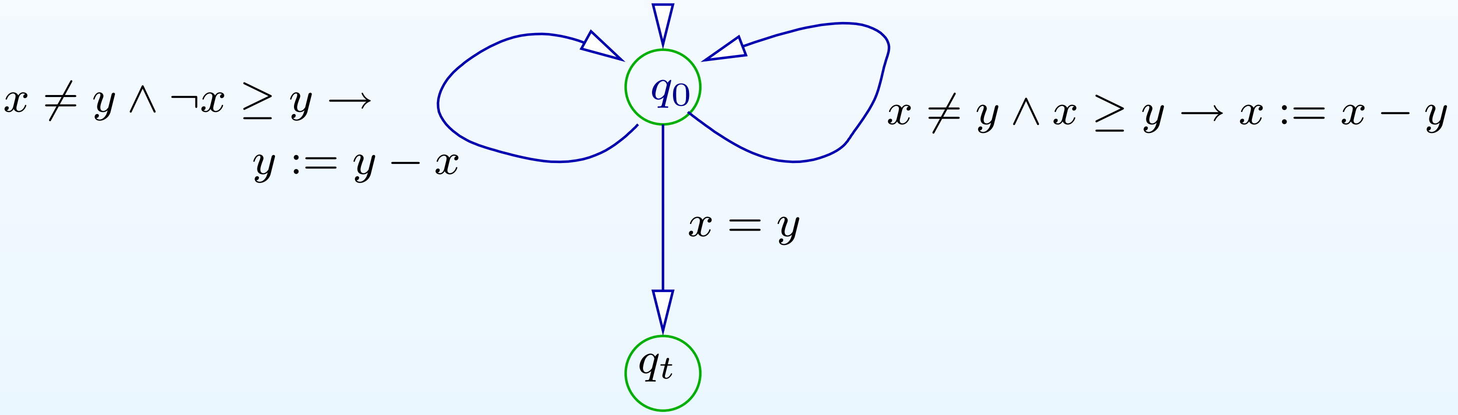
Terminaison d'automates

Un automate étendu $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ avec l'état initial des variables σ_0 *termine*, si il n'admet pas de trace infinie partant de (q_0, σ_0) , c.a.d. $\mathrm{Tr}_{inf}(A, \sigma_0) = \emptyset$.
Comment montrer qu'un automate termine?

Terminaison d'automates

Un automate étendu $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ avec l'état initial des variables σ_0 *termine*, si il n'admet pas de trace infinie partant de (q_0, σ_0) , c.a.d. $\text{Tr}_{inf}(A, \sigma_0) = \emptyset$.

Comment montrer qu'un automate termine?



Propriétés des relations

Pour toutes les relations dans les examples on suppose:
$R_i \subseteq \{1,2,3\} \times \{1,2,3\}$

- $R \subseteq A \times A$ est réflexive, si $\forall x \in A \cdot (x,x) \in R$.
  Exemples
  - $R_1 \stackrel{def}{=} \{(1,1),(1,2),(2,2),(3,3)\}$ réflexive
  - $R_2 \stackrel{def}{=} \{(1,1),(2,1),(3,3)\}$ pas réflexive

Propriétés des relations

Pour toutes les relations dans les examples on suppose:
$R_i \subseteq \{1,2,3\} \times \{1,2,3\}$

- $R \subseteq A \times A$ est réflexive, si $\forall x \in A \cdot (x,x) \in R$.
  Exemples
  - $R_1 \stackrel{def}{=} \{(1,1),(1,2),(2,2),(3,3)\}$ réflexive
  - $R_2 \stackrel{def}{=} \{(1,1),(2,1),(3,3)\}$ pas réflexive
- $R \subseteq A \times A$ est transitive, si
  $\forall x,y,z \in A \cdot (x,y) \in R \wedge (y,z) \in R \implies (x,z) \in R$
  Exemples
  - $R_3 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(3,2)\}$ transitive
  - $R_4 \stackrel{def}{=} \{(1,2),(2,3),(2,2)\}$ pas transitive

Propriétés des relations(II)

- $R \subseteq A \times A$ est symétrique, si
  $\forall x, y \in A \cdot (x, y) \in R \implies (y, x) \in R$
  Exemples
  - $R_5 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(1,3)\}$ symétrique
  - $R_3 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(3,2)\}$ pas symétrique

Propriétés des relations(II)

- $R \subseteq A \times A$ est symétrique, si
  $\forall x, y \in A \cdot (x, y) \in R \implies (y, x) \in R$
  Exemples
  - $R_5 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(1,3)\}$ symétrique
  - $R_3 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(3,2)\}$ pas symétrique
- $R \subseteq A \times A$ est anti-symétrique, si
  $\forall x, y \in A \cdot (x, y) \in R \wedge (y, x) \in R \implies y = x$
  Exemples
  - $R_7 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(3,3),(3,1),(1,3)\}$ anti-symétrique
  - $R_5 \stackrel{def}{=} \{(1,2),(2,1),(1,1),(2,2),(3,1),(1,3)\}$ pas anti-symétrique

Composition de relations

Soient $R \subseteq A \times B$ et $R' \subseteq B \times C$ deux relations.

- $R \circ R' = \{(a, c) \in A \times C \mid \exists b \in B \cdot (a, b) \in R \wedge (b, c) \in R'\}$.

Propriétés :

Composition de relations

Soient $R \subseteq A \times B$ et $R' \subseteq B \times C$ deux relations.

- $R \circ R' = \{(a, c) \in A \times C \mid \exists b \in B \cdot (a, b) \in R \wedge (b, c) \in R'\}$.

Propriétés :

1. La composition des relations est associative.

Composition de relations

Soient $R \subseteq A \times B$ et $R' \subseteq B \times C$ deux relations.

- $R \circ R' = \{(a,c) \in A \times C \mid \exists b \in B \cdot (a,b) \in R \wedge (b,c) \in R'\}$.

Propriétés :

1. La composition des relations est associative.
2. Elle est monotone.

Composition de relations

Soient $R \subseteq A \times B$ et $R' \subseteq B \times C$ deux relations.

- $R \circ R' = \{(a, c) \in A \times C \mid \exists b \in B \cdot (a, b) \in R \wedge (b, c) \in R'\}$.

Propriétés :

1. La composition des relations est associative.
2. Elle est monotone.

L'inverse d'une relation R est la relation
$R^{-1} = \{(b, a) \in B \times A \mid \ (a, b) \in R\}$. Propriété : $(R^{-1})^{-1} = R$.

Composition de relations

Soient $R \subseteq A \times B$ et $R' \subseteq B \times C$ deux relations.

- $R \circ R' = \{(a, c) \in A \times C \mid \exists b \in B \cdot (a, b) \in R \wedge (b, c) \in R'\}$.

Propriétés :

1. La composition des relations est associative.
2. Elle est monotone.

L'inverse d'une relation R est la relation $R^{-1} = \{(b, a) \in B \times A \mid \ (a, b) \in R\}$. Propriété : $(R^{-1})^{-1} = R$.

Si $R \subseteq A \times A$ et $B \subseteq A$, alors la restriction de R a B, noteé $R|_B$ est la relation $R|_B \stackrel{def}{=} \{(x, y) | (x, y) \in R, x \in B, y \in B\}$

Fermetures des relations

Fermer une relation par une propriété revient à *compléter* la relation pour qu'elle vérifie cette propriété.
Soit $R \subseteq A \times A$ une relation.

Fermetures des relations

Fermer une relation par une propriété revient à *compléter* la relation pour qu'elle vérifie cette propriété.
Soit $R \subseteq A \times A$ une relation.

1. La fermeture réflexive de R est la *plus petite* relation $Q \subseteq A \times B$ qui contient R et qui est reflexive :

$$(\forall x, y \in A \cdot xRy \implies xQy) \wedge (\forall x \in A \cdot \cdot xQx)$$

Fermetures des relations

Fermer une relation par une propriété revient à *compléter* la relation pour qu'elle vérifie cette propriété.
Soit $R \subseteq A \times A$ une relation.

1. La fermeture réflexive de R est la *plus petite* relation $Q \subseteq A \times B$ qui contient R et qui est reflexive :

$$(\forall x, y \in A \cdot xRy \implies xQy) \wedge (\forall x \in A \cdot \cdot xQx)$$

2. La fermeture transitive de R, notée R^+ est la *plus petite* relation $Q \subseteq A \times B$ qui est transitive et qui contient R:

$$(\forall x, y \in A \cdot xRy \implies xQy) \wedge (\forall x, y, z \in A \cdot xQy \wedge yQz \implies xQz)$$

Fermetures des relations

Fermer une relation par une propriété revient à *compléter* la relation pour qu'elle vérifie cette propriété.
Soit $R \subseteq A \times A$ une relation.

1. La fermeture réflexive de R est la *plus petite* relation $Q \subseteq A \times B$ qui contient R et qui est reflexive :

$$(\forall x, y \in A \cdot xRy \implies xQy) \wedge (\forall x \in A \cdot \cdot xQx)$$

2. La fermeture transitive de R, notée R^+ est la *plus petite* relation $Q \subseteq A \times B$ qui est transitive et qui contient R:

$$(\forall x, y \in A \cdot xRy \implies xQy) \wedge (\forall x, y, z \in A \cdot xQy \wedge yQz \implies xQz)$$

3. La fermeture réflexive-transitive est notée R^*. C'est la *plus petite* relation qui contient R et qui est reflexive et transitive.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(I\!N), \subset)$ et $(\mathcal{P}(I\!N), \supset)$ ne sont pas bien-fondées.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(I\!N), \subset)$ et $(\mathcal{P}(I\!N), \supset)$ ne sont pas bien-fondées.
- $(\mathcal{P}_{fin}(I\!N), \supset)$ est bien-fondée.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(I\!N), \subset)$ et $(\mathcal{P}(I\!N), \supset)$ ne sont pas bien-fondées.
- $(\mathcal{P}_{fin}(I\!N), \supset)$ est bien-fondée.
- $(\mathcal{P}_{fin}(I\!N), \subset)$ n'est pas bien-fondée.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(I\!N), \subset)$ et $(\mathcal{P}(I\!N), \supset)$ ne sont pas bien-fondées.
- $(\mathcal{P}_{fin}(I\!N), \supset)$ est bien-fondée.
- $(\mathcal{P}_{fin}(I\!N), \subset)$ n'est pas bien-fondée.
- $(\boldsymbol{Z}, <)$ et $(\boldsymbol{Z}, >)$ ne sont pas bien-fondées.

Relation bien-fondée

Une relation $R \subseteq A \times A$ est dite *bien-fondée* s'il n'y a pas de séquence infinie $a_0, a_1, a_2 \cdots a_n \cdots$ d'éléments de A (pas nécesairement distincts) telle que $\forall i \in I\!N,\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(I\!N, >)$ est bien-fondée.
- $(I\!N, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(I\!N), \subset)$ et $(\mathcal{P}(I\!N), \supset)$ ne sont pas bien-fondées.
- $(\mathcal{P}_{fin}(I\!N), \supset)$ est bien-fondée.
- $(\mathcal{P}_{fin}(I\!N), \subset)$ n'est pas bien-fondée.
- $(\boldsymbol{Z}, <)$ et $(\boldsymbol{Z}, >)$ ne sont pas bien-fondées.
- $(I\!R, <)$ et $(I\!R, >)$ ne sont pas bien-fondées.

Invariant de transition

Soit $A = (D, Q, \{q_0\}, \mathcal{T}, Q_t)$ un automate étendu, et σ_0 un état (initial) des variables.

Invariant de transition

Soit $A = (D, Q, \{q_0\}, \mathcal{T}, Q_t)$ un automate étendu, et σ_0 un état (initial) des variables.

- L'ensemble de *configurations accesibles* dans A à partir de (q_0, σ_0), noté Acc(A, σ_0), est l'ensemble de configurations (q, σ), tel qu'il existe une trace finie qui méne à (q, σ), $(q_0, \sigma_0) \to \cdots \to (q, \sigma)$, formellement,

$$((q_0, \sigma_0), (q, \sigma)) \in \to^*$$

Invariant de transition

Soit $A = (D, Q, \{q_0\}, \mathcal{T}, Q_t)$ un automate étendu, et σ_0 un état (initial) des variables.

- L'ensemble de *configurations accesibles* dans A à partir de (q_0, σ_0), noté Acc(A, σ_0), est l'ensemble de configurations (q, σ), tel qu'il existe une trace finie qui méne à (q, σ), $(q_0, \sigma_0) \to \cdots \to (q, \sigma)$, formellement,

$$((q_0, \sigma_0), (q, \sigma)) \in \to^*$$

- Une relation $T \subseteq Conf \times Conf$ est *un invariant de transition* pour A et σ_0, si elle contient la fermeture transitive de $\to$ restreinte aux état accesibles de (q_0, σ_0), formellement, Condition (inv):

$$\to^+ |_{\mathsf{Acc}(A,\sigma_0)} \subseteq T$$

Une condition nécessaire et suffisante

Theorem 0.2 *Un automate étendu* $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ *avec l'état initial des variables* σ_0 termine*, ssi il existe un invariant de transition* T *bien-fondé pour* A *et* σ_0*.*

Une condition nécessaire et suffisante

Theorem 0.3 *Un automate étendu $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ avec l'état initial des variables σ_0* termine*, ssi il existe un invariant de transition T bien-fondé pour A et σ_0.*

En pratique le résultat est difficile à appliquer car pour vérifier la condition (inv), il faut retrouver et manipuler Acc(A, σ_0) et $\rightarrow^+$!

Terminaison d'automates - outils

Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et σ_0 l'état initial des variables. Alors une ralation $T \subseteq Conf \times Conf$ est *inductive* pour A et σ_0, si elle est transitive et contient la relation de transition $\rightarrow$ restreinte aux état accesibles de (q_0, σ_0), formellement,

Condition (ind):

$$T \textit{ transitive et } \quad \rightarrow |_{\mathsf{Acc}_{(A,\sigma_0)}} \subseteq T$$

Terminaison d'automates - outils

Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et σ_0 l'état initial des variables. Alors une ralation $T \subseteq Conf \times Conf$ est *inductive* pour A et σ_0, si elle est transitive et contient la relation de transition $\rightarrow$ restreinte aux état accesibles de (q_0, σ_0), formellement,

Condition (ind):

$$T \textit{ transitive et } \quad \rightarrow|_{\text{Acc}_{(A,\sigma_0)}} \subseteq T$$

Corollaire 0.1 *Une relation inductive pour A et σ_0 est un invariant de transition pour A et σ_0.*

Terminaison d'automates - outils(II)

Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et $(P_q)_{q \in Q}$ une annotation inductive de A. Pour vérifier (ind) il suffit de vérifier:

Condition (ind_s):

-

Terminaison d'automates - outils(II)

Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et $(P_q)_{q \in Q}$ une annotation inductive de A. Pour vérifier (ind) il suffit de vérifier:

Condition (ind_s):

- $T \circ T \subseteq T$,

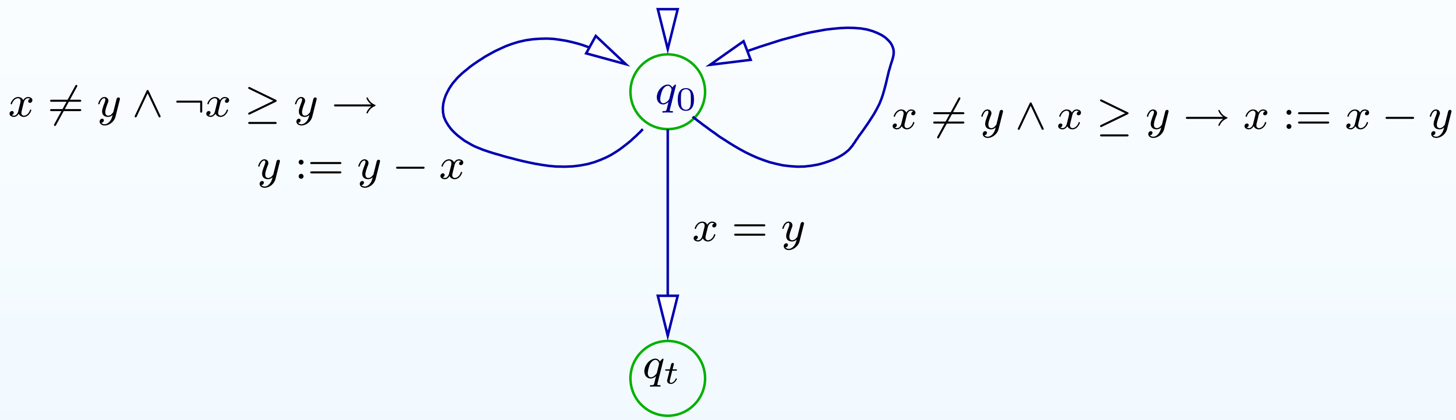
Terminaison d'automates - outils(II)

Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et $(P_q)_{q \in Q}$ une annotation inductive de A. Pour vérifier (ind) il suffit de vérifier:

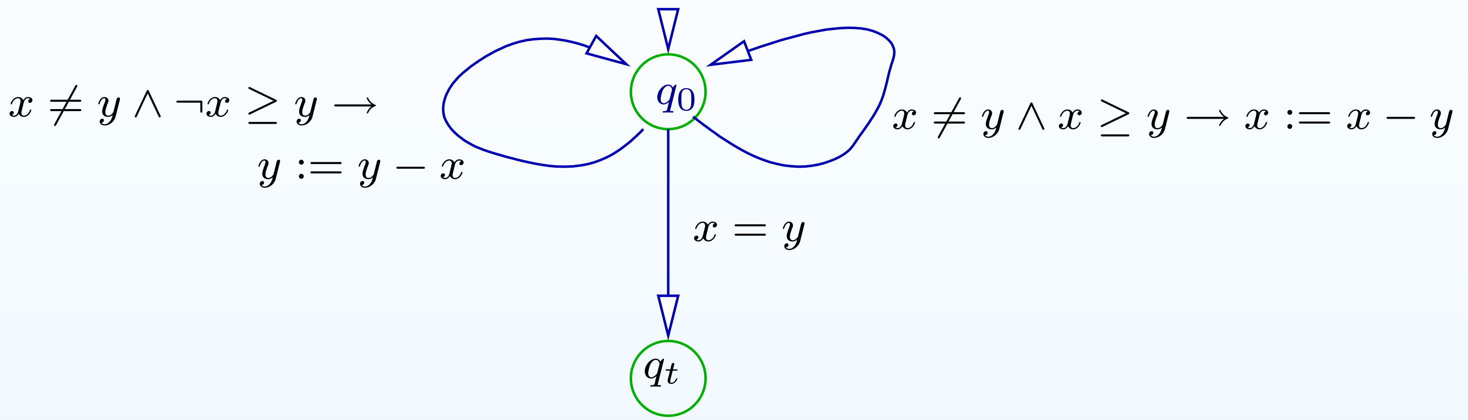
Condition (ind_s):

- $T \circ T \subseteq T$,
- Pour toute transition $(q, g \rightarrow x := e, q')$ et tout état $\sigma \in \Sigma$, si $\sigma \models g \wedge P_q$ alors $((q, \sigma), (q', \sigma[\llbracket e \rrbracket \sigma / x])) \in T$.

Exemples terminaison d'automates étendus-1

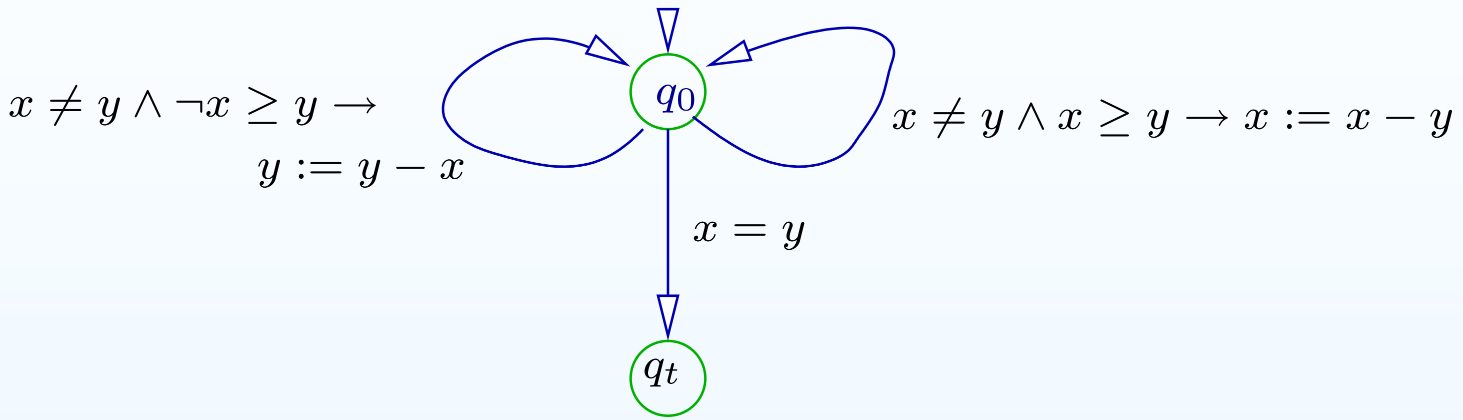


Exemples terminaison d’automates étendus-1



Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) > 0$.

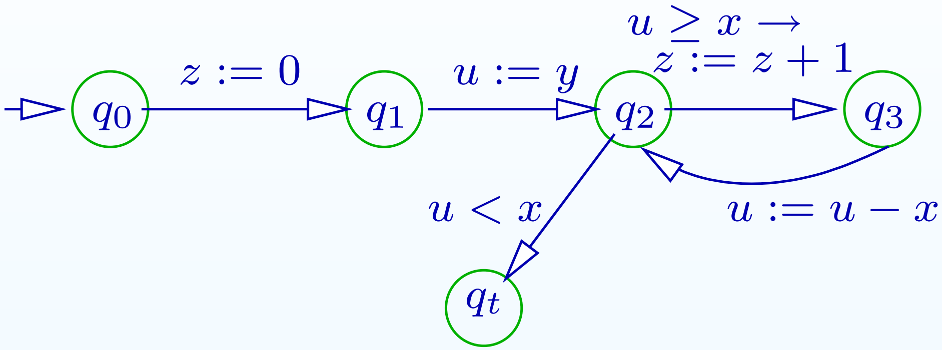
Exemples terminaison d'automates étendus-1



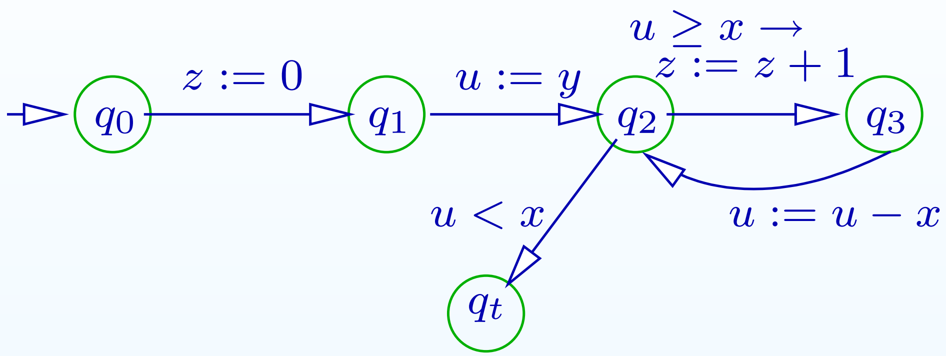
Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) > 0$.
Invariant :

$$\begin{aligned} T = \quad & \{((q_0, \sigma), (q_0, \sigma')) \mid |\sigma'(x) + \sigma'(y)| < |\sigma(x) + \sigma(y)|\} \cup \\ & \{((q_0, \sigma), (q_1, \sigma'))\} \end{aligned}$$

Exemples terminaison d'automates étendus-2

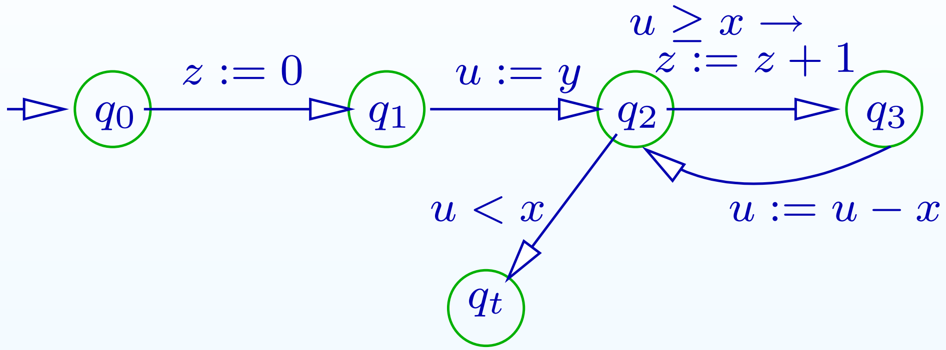


Exemples terminaison d'automates étendus-2



Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) \geq 0$.

Exemples terminaison d'automates étendus-2



Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) \geq 0$.
Invariant :

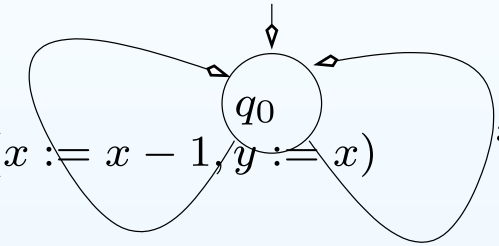
$$
\begin{aligned}
T = \quad & \{((q_0,\sigma),(q,\sigma'))|q \in \{q_1,q_2,q_3,q_t\}\}\cup \\
& \{((q_1,\sigma),(q,\sigma'))|q \in \{q_2,q_3,q_t\}\}\cup \\
& \{((q_2,\sigma),(q_t,\sigma'))\} \cup \{((q_3,\sigma),(q_t,\sigma'))\}\cup \\
& \{((q_3,\sigma),(q_2,\sigma')) \mid \sigma'(u)-\sigma'(x) < \sigma(u)-\sigma(x) \wedge \sigma(u)-\sigma(x) \geq 0\}\cup \\
& \{((q_2,\sigma),(q_2,\sigma')) \mid \sigma'(u)-\sigma'(x) < \sigma(u)-\sigma(x) \wedge \sigma(u)-\sigma(x) \geq 0\}\cup \\
& \{((q_3,\sigma),(q_3,\sigma')) \mid \sigma'(u)-\sigma'(x) < \sigma(u)-\sigma(x) \wedge \sigma(u)-\sigma(x) \geq 0\}\cup \\
& \{((q_2,\sigma),(q_3,\sigma')) \mid \sigma'(u)-\sigma'(x) = \sigma(u)-\sigma(x) \wedge \sigma(u)-\sigma(x) \geq 0\}.
\end{aligned}
$$

Exemples terminaison d’automates étendus-3

Dans la pratique, quand il y a plusieurs boucles imbriqués, trouver un invariant de transitions T bien-fondé peut s’avérer difficile...

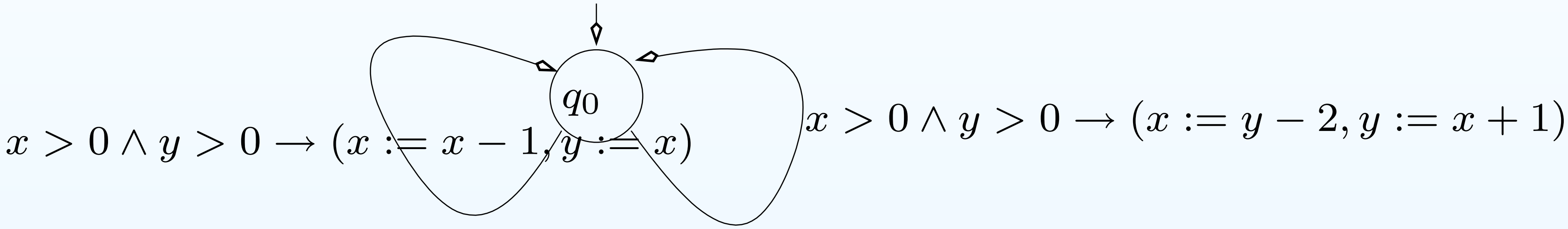
Exemples terminaison d'automates étendus-3

Dans la pratique, quand il y a plusieurs boucles imbriqués, trouver un invariant de transitions T bien-fondé peut s'avérer difficile...



Exemples terminaison d'automates étendus-3

Dans la pratique, quand il y a plusieurs boucles imbriqués, trouver un invariant de transitions T bien-fondé peut s'avérer difficile...



T invariant de transitions bien-fondé?

Terminaison d'automates - outils(III)

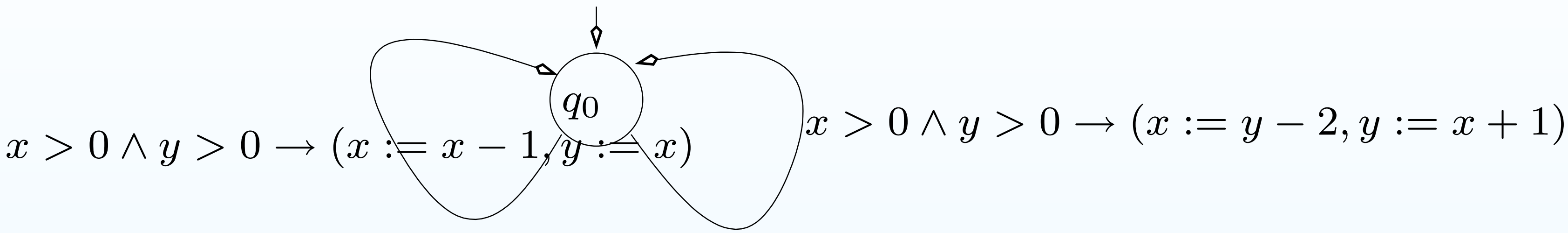
Une relation $R \subseteq A \times A$ est dite *union-bien-fondée* s'il existe un nombre fini de relations bien-fondées $T_1, \ldots, T_n$, telles que $T = T_1 \cup \ldots \cup T_n$.

Terminaison d'automates - outils(III)

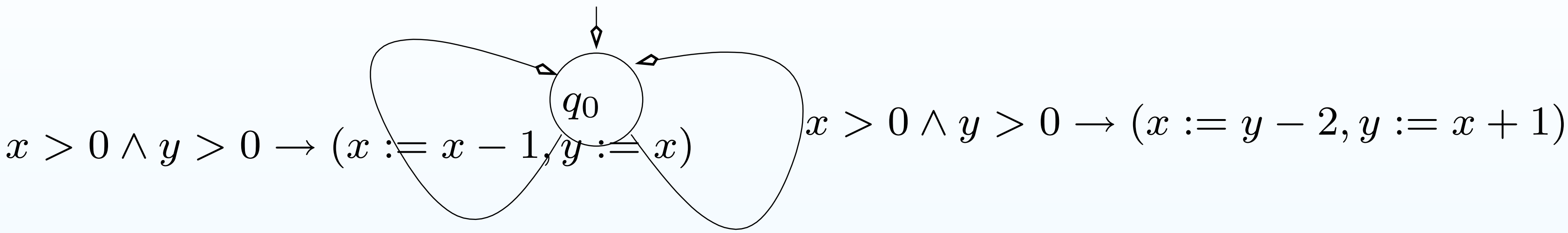
Une relation $R \subseteq A \times A$ est dite *union-bien-fondée* s'il existe un nombre fini de relations bien-fondées $T_1, \ldots, T_n$, telles que $T = T_1 \cup \ldots \cup T_n$.

Theorem 0.4 *Un automate étendu $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ avec l'état initial des variables σ_0* termine*, ssi il existe un invariant de transition T union-bien-fondé pour A et σ_0.*

Exemples terminaison d’automates étendus-4



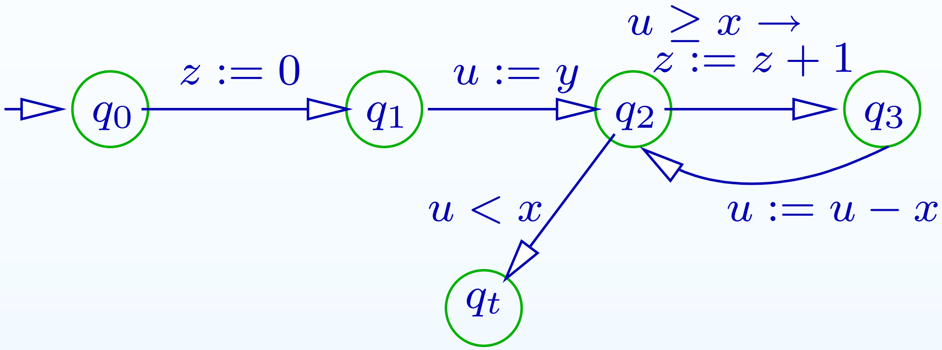
Exemples terminaison d'automates étendus-4



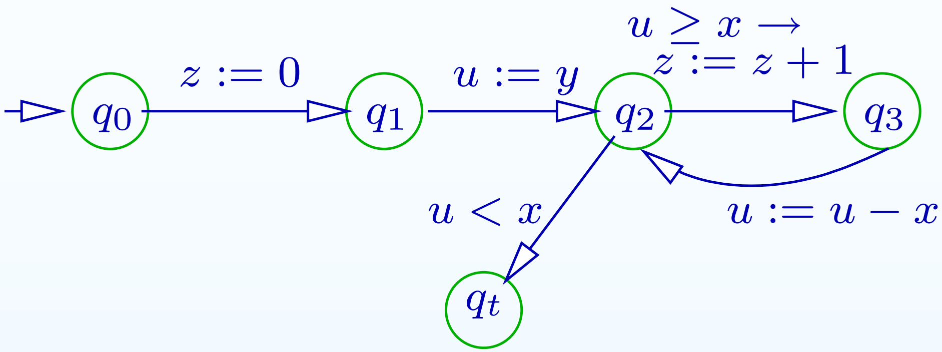
Invariant :

$$
\begin{array}{ll}
T = & T_1 \cup T_2 \cup T_3 \cup T_4 \text{ où} \\
T_1 = & \{((q_0, \sigma), (q_0, \sigma')) \mid Pos(\sigma, x, y) \wedge \sigma'(x) < \sigma(x) \wedge \sigma'(y) \leq \sigma(x)\} \\
T_2 = & \{((q_0, \sigma), (q_0, \sigma')) \mid Pos(\sigma, x, y) \wedge \sigma'(x) < \sigma(y) - 1 \wedge \sigma'(y) \leq \sigma(x) + \\
T_3 = & \{((q_0, \sigma), (q_0, \sigma')) \mid Pos(\sigma, x, y) \wedge \sigma'(x) < \sigma(y) - 1 \wedge \sigma'(y) < \sigma(y)\} \\
T_4 = & \{((q_0, \sigma), (q_0, \sigma')) \mid Pos(\sigma, x, y) \wedge \sigma'(x) < \sigma(x) \wedge \sigma'(y) < \sigma(y)\} \\
\text{où} & Pos(\sigma, x, y) = \sigma(x) > 0 \wedge \sigma(y) > 0
\end{array}
$$

Exemples terminaison d'automates étendus-5

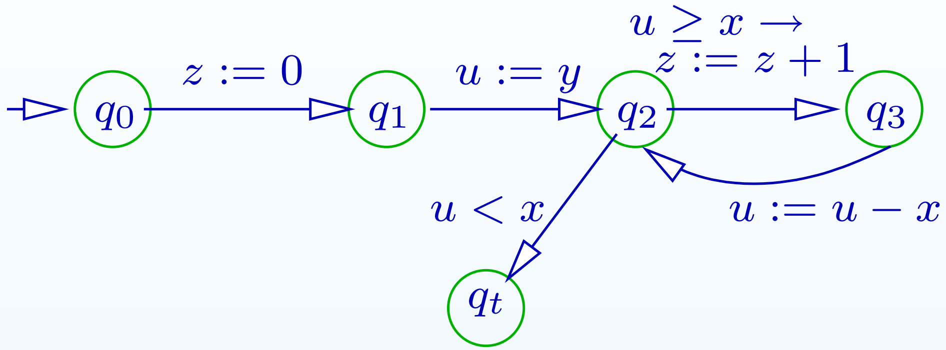


Exemples terminaison d'automates étendus-5



Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) \geq 0$.

Exemples terminaison d'automates étendus-5



Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) \geq 0$.
Invariant :

$$
\begin{array}{ll}
T = & T' \cup \bigcup_{i \neq j, i=0, j=0}^{i=3, j=3} T_{ij} \text{ où} \\
T_{ij} = & \{((q_i, \sigma), (q_j, \sigma')) \mid i \neq j\} \\
T' & \{((q, \sigma), (q, \sigma')) \mid \sigma'(u) - \sigma'(x) < \sigma(u) - \sigma(x) \wedge \sigma(u) - \sigma(x) \geq 0\}.
\end{array}
$$

Correction totalle

Un automate étendu A est *totallement correcte* par rapport à la spécification (P, Q) si A est partiellement correcte par rapport à la spécification (P, Q), et pour tout état initial des variables $\sigma_0 \models P$, A finit toujours son éxecution dans un état final, c.a.d. $\mathsf{Tr}_{inf}(A, \sigma_0) = \emptyset$ et $\mathsf{Tr}_{fm}(A, \sigma_0) = \mathsf{Tr}_{ft}(A, \sigma_0)$.

Correction totalle d’automates - outils

Pour vérifier qu’un automate étendu A est correct totallement par rapport à une spécification (P, Q), il suffit de trouver une relation T et munir A d’une annotation $(P_q)_{q \in Q}$ telle que

-
-
-

Correction totalle d’automates - outils

Pour vérifier qu’un automate étendu A est correct totallement par rapport à une spécification (P, Q), il suffit de trouver une relation T et munir A d’une annotation $(P_q)_{q \in Q}$ telle que

- on obtient un automate étendu annoté correcte par rapport à (P, Q)
-
-

Correction totalle d’automates - outils

Pour vérifier qu’un automate étendu A est correct totallement par rapport à une spécification (P, Q), il suffit de trouver une relation T et munir A d’une annotation $(P_q)_{q \in Q}$ telle que

- on obtient un automate étendu annoté correcte par rapport à (P, Q)
- T est union-bien-fondée
-

Correction totalle d'automates - outils

Pour vérifier qu'un automate étendu A est correct totallement par rapport à une spécification (P, Q), il suffit de trouver une relation T et munir A d'une annotation $(P_q)_{q \in Q}$ telle que

- on obtient un automate étendu annoté correcte par rapport à (P, Q)
- T est union-bien-fondée
- la condition (ind_s) est vérifié

Correction totalle d'automates - outils

Pour vérifier qu'un automate étendu A est correct totallement par rapport à une spécification (P, Q), il suffit de trouver une relation T et munir A d'une annotation $(P_q)_{q \in Q}$ telle que

- on obtient un automate étendu annoté correcte par rapport à (P, Q)
- T est union-bien-fondée
- la condition (ind_s) est vérifié
- Pour chaque état $q \notin Q_t$, si $(q, g_i \to x_i := e_i, q'_i)$, avec $i = 1, \ldots, m$ sont tous les transitions partant de q, alors $P_q \Rightarrow g_1 \vee \ldots \vee g_m$

Alphabets, symboles, mots (1)

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

Exemples

- $\Sigma_1 = \{a, b, c, \ldots, z\}$
- $\Sigma_2 = \{0, 1\}$
- $\Sigma_3 = \{+, -, *, \%, \$\}$

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "·". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.

$\square$

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.

$\square$

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à a_1a_2.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.

$\square$

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.
- L'ensemble de tous les mots sur Σ est noté Σ^*.

$\square$

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.
- L'ensemble de tous les mots sur Σ est noté Σ^*.

$\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à a_1a_2.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.
- L'ensemble de tous les mots sur Σ est noté Σ^*.

$\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- ϵ est le mot de longueur 0.

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.
- L'ensemble de tous les mots sur Σ est noté Σ^*.

$\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- 0 et 1 sont les mots de longueur 1.

Alphabets, symboles, mots (2)

Definition

- Un *mot* sur l'alphabet Σ est une chaine de symboles dans Σ. Formellement, un *mot sur* Σ est un élément du monoide libre engendre par Σ et l'opération associative de concaténation "$\cdot$". $a_1 \cdot a_2$ sera raccourci à $a_1 a_2$.
- Le *mot vide* est l'élément neutre du monoide; noté ϵ.
- On note par $|u|$ la longueur du mot u.
- L'ensemble de tous les mots sur Σ est noté Σ^*.

$\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- 00, 01, 10 et 11 sont les mots de longueur 2.

Alphabets, symboles, mots (3)

Ordres partiels

- u préfixe de v si $\exists u',\ v = uu'$
- u suffixe de v si $\exists u',\ v = u'u$
- u facteur de v si $\exists u', u'',\ v = u'uu''$
- u sous-mot de v si $v = v_0u_1v_1 \ldots u_nv_n$ avec $u_i, v_i \in \Sigma^*$ et $u = u_1 \ldots u_n$

Theorem 0.5 **(Higman)** *L'ordre sous-mot est un bon ordre, i.e. (de toute suite infinie on peut extraire une sous-suite infinie croissante)*
(ou toute ensemble de mots a un nombre fini d'éléments minimaux)

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ est un langage sur Σ. Il est appelé le *langage vide*.

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ est un langage sur Σ. Il est appelé le *langage vide*.
- Σ^* est un langage sur sur Σ. Il est appelé le *langage universel*.

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ 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.

Alphabets, symboles, mots (4)

Definition Un *langage sur* Σ est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ 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.
- $\{0, 11, 001\}$ est aussi un langage sur Σ.

Alphabets, symboles, mots (4)

Definition Un *langage sur Σ* est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). □

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ 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.
- $\{0, 11, 001\}$ est aussi un langage sur Σ.
- L'ensemble des mots qui contiennent un nombre impair de 0 est un langage sur Σ.

Alphabets, symboles, mots (4)

Definition Un *langage sur Σ* est un ensemble de mots sur l'alphabet Σ (et donc un sous-ensemble de Σ^*). $\square$

Considérons l'alphabet $\Sigma = \{0, 1\}$. Alors:

- $\emptyset$ 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.
- $\{0, 11, 001\}$ 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.

Opérations sur les langages

- Ensemblistes: union, intersection, complément, différence...
- Concaténation de Langages
  On peut étendre la concaténation des mots aux langages de la manière suivante:

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

  - $L \cdot \emptyset = \emptyset$
  - $L \cdot \Sigma^* = \Sigma^*$ si et seulement si $\epsilon \in L$
  - $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$
  - Si $L_1 = \{01\}$ et $L_2 = \{111, 0, 10\}$ alors $L_1 \cdot L_2 = \{01111, 010, 0110\}$.

Puissance

La puissance d'un mot w, notée w^n où $n \geq 0$ est définie par:

- $w^0 = \epsilon$
- $w^{n+1} = w^n \cdot w$

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

- $L^0 = \{\epsilon\}$
- $L^{n+1} = L^n \cdot L$

La fermeture itérative (ou la fermeture de Kleene):

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \bigcup_{i \geq 0} L^i$$

Auomates d'états finis

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 le 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).

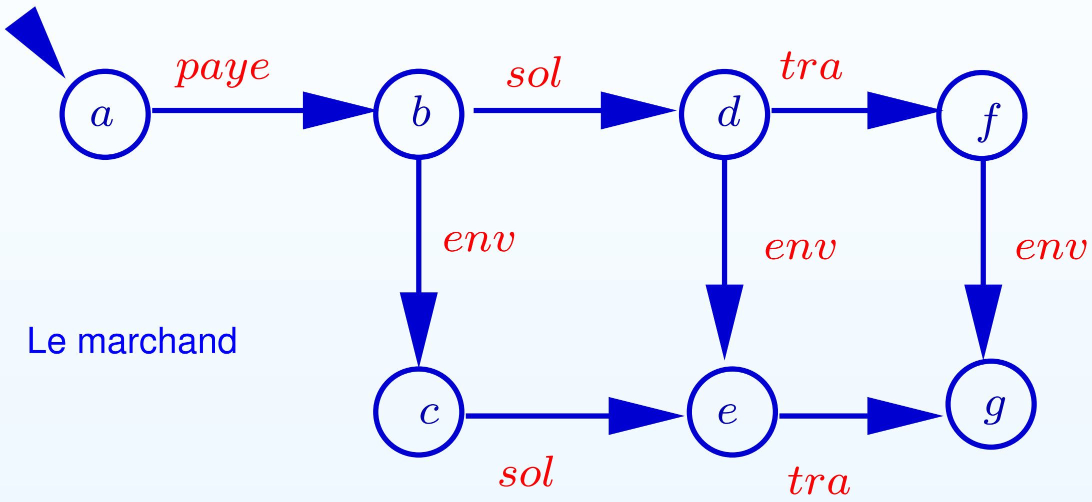
Hypothèses sur le comportement des participants

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.

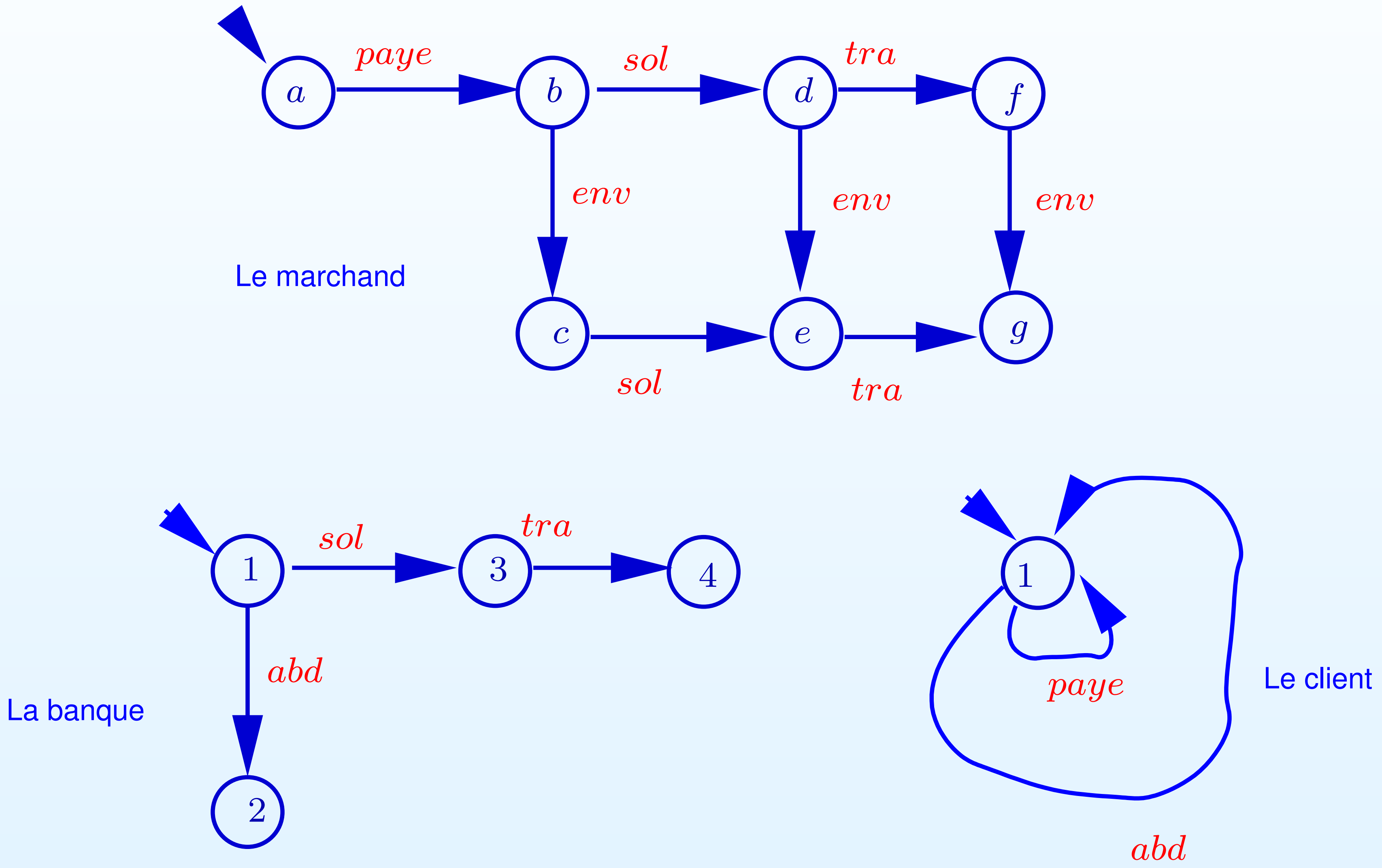
On veut donc modéliser le comportement des trois participants et voir s'il y a un moyen pour le client de recevoir la marchandise sans payer.

Modélisation des participants



Le marchand

Modélisation des participants



Vérification du modèle

Approche:

- On calcule un automate fini 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: Donnez un automate fini qui modélise la composition des trois participants.

Automates finis et applications

Etude des automates finis pour savoir:

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

Éxemples d'automates finis

- Systèmes réactifs
- Protocoles dans les réseaux
- Analyseurs lexicaux

Automates déterministes

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

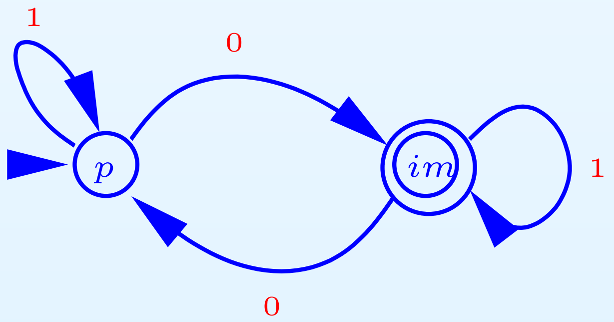
- Q est un ensemble fini d'*états*.
- Σ est l'alphabet de l'automate.
- $q_0 \in Q$ est l'*état initial*.
- La fonction $\delta : Q \times \Sigma \to Q$ est la *fonction de transition*.
- $F \subseteq Q$ est l'ensemble des *états accepteurs*.

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

Exemple : Nombre pair de 0

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

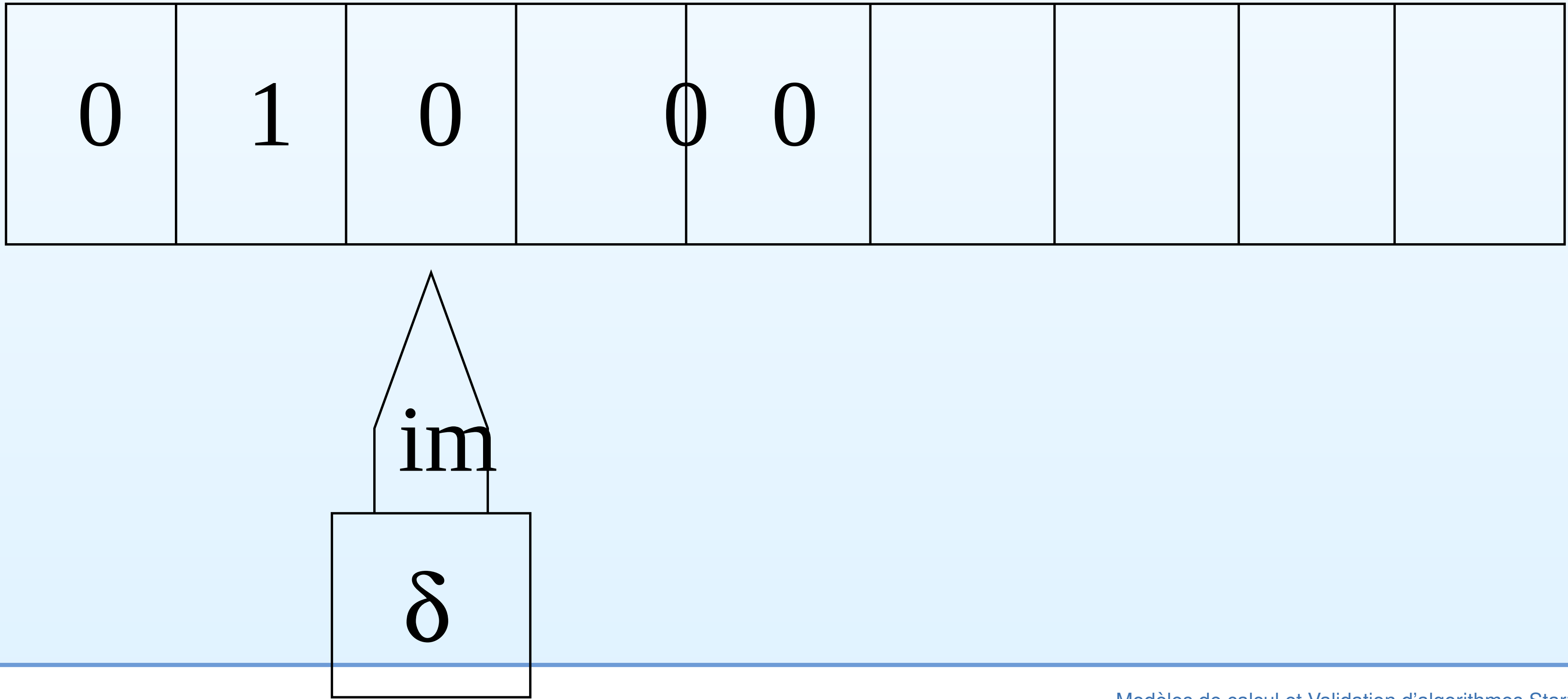
- $Q = \{p, im\}$
- $\Sigma = \{0, 1\}$
- p est l'état initial
- im est l'état accepteur
- les fleches donnent la fonction de transition



□

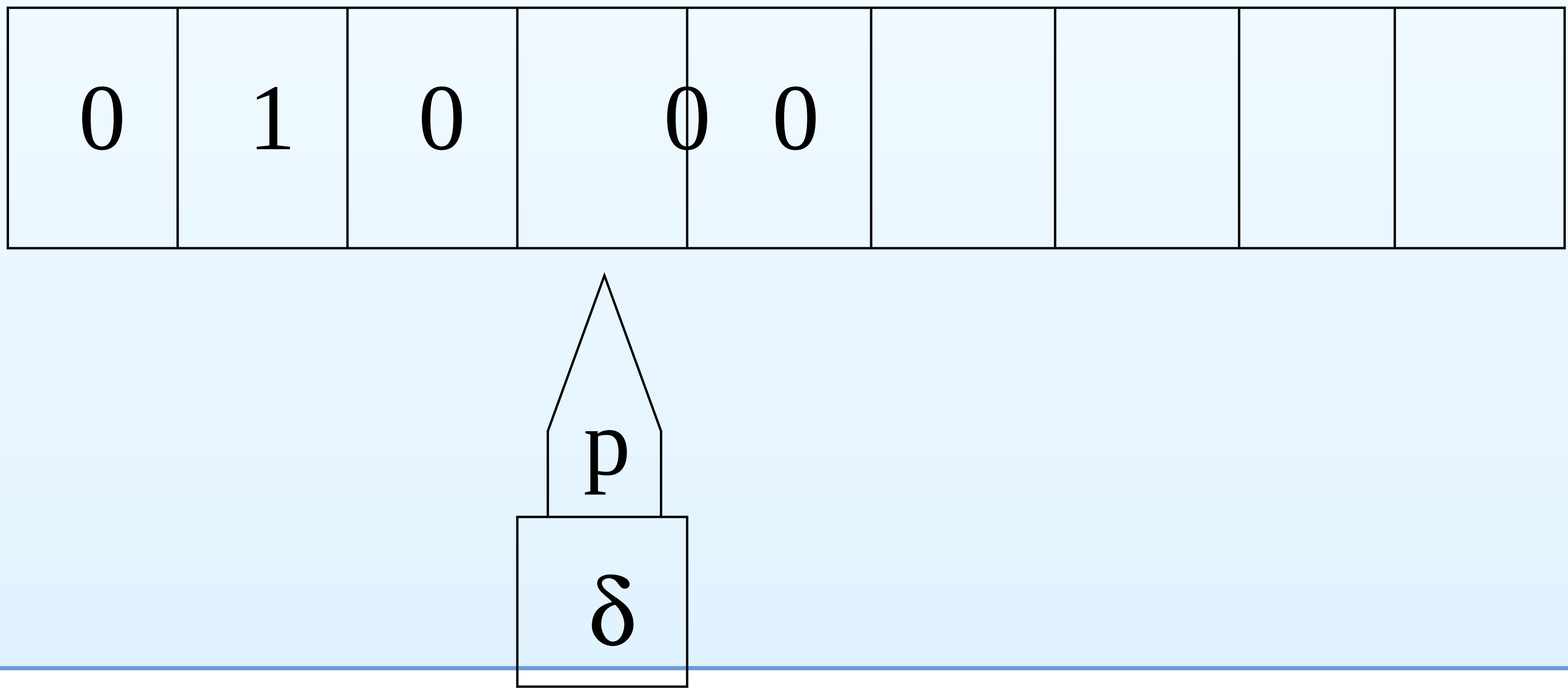
Autre reprsentation graphique

- La bande de lecture n'est pas modifiable (read-only) et la tête de lecture avance d'une case vers la droite à chaque transition
- L'état im évolue vers l'état $p = \delta(q, 0)$ si le caractère 0 est sous la tête de lecture.



Autre reprsentation graphique

- La bande de lecture n'est pas modifiable (read-only) et la tête de lecture avance d'une case vers la droite à chaque transition
- L'état im évolue vers l'état $p = \delta(q, 0)$ si le caractère 0 est sous la tête de lecture.



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:

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

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

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

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

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. $\square$

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 $\{u \in \Sigma^* \mid u \text{ *est accepté par* } A\}$.
- 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.

$\square$

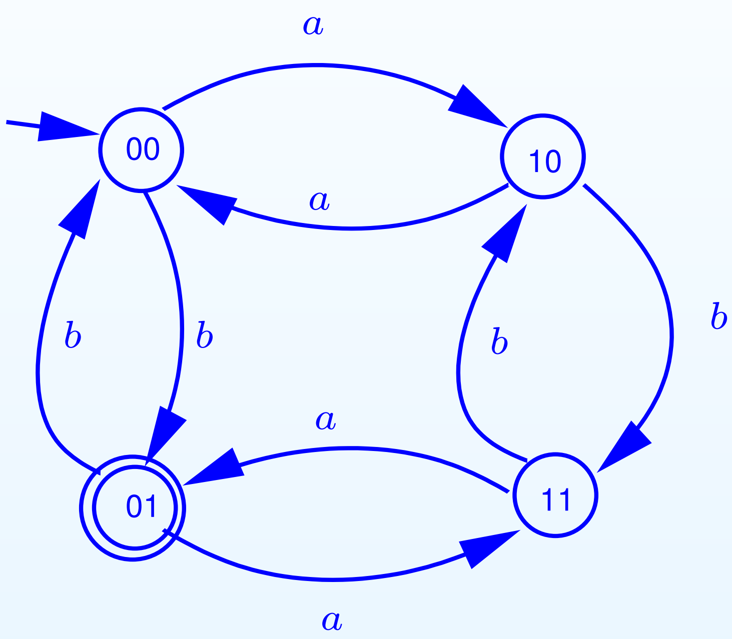
Exemples

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

- L'ensemble des mots dans lesquel b ne precède 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?

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 reconnait $\Sigma^* \setminus L(A)$?

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 reconnait $\Sigma^* \setminus L(A)$?

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 telque $C(\delta) : Q \times \Sigma \to Q$ 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

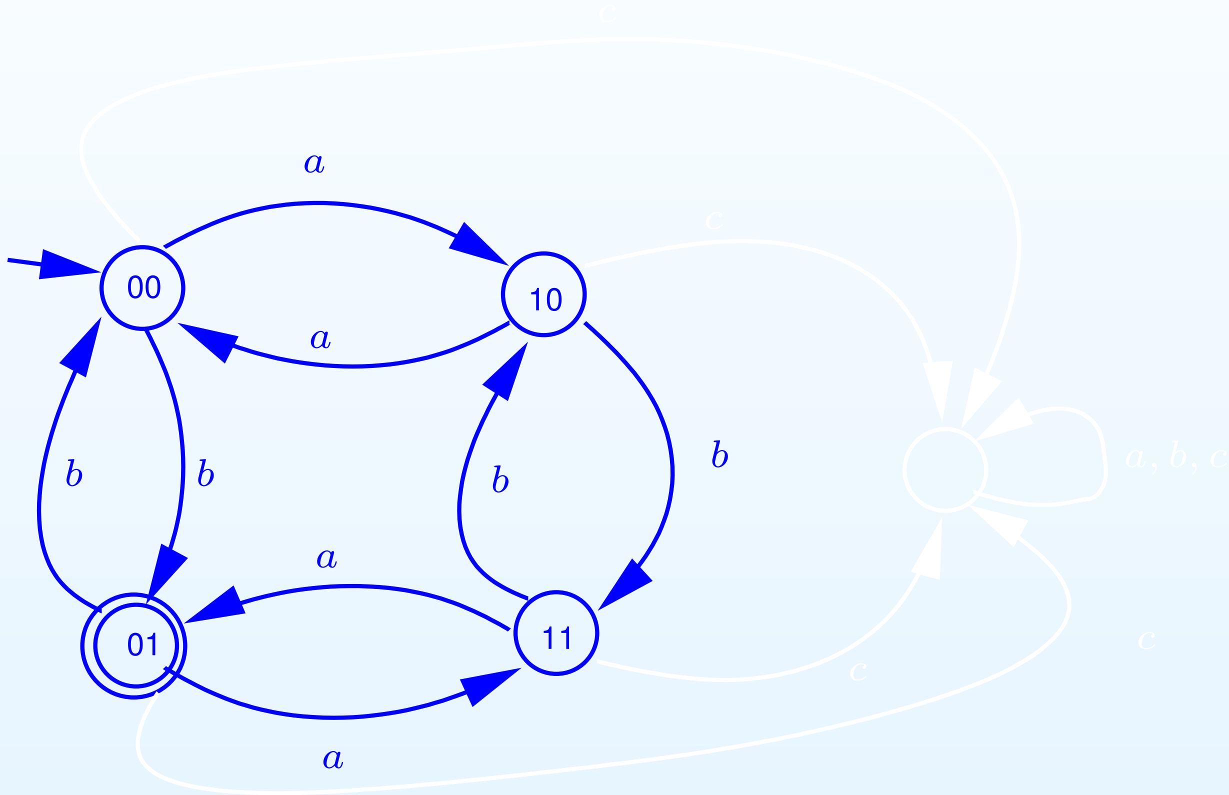
Donné un ADEF $A = (Q, \Sigma, q_0, \delta, F)$. Pour construire un automate qui reconnait $\Sigma^* \setminus L(A)$, on suit les pâs suivant:

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

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

Procédure de complémetation

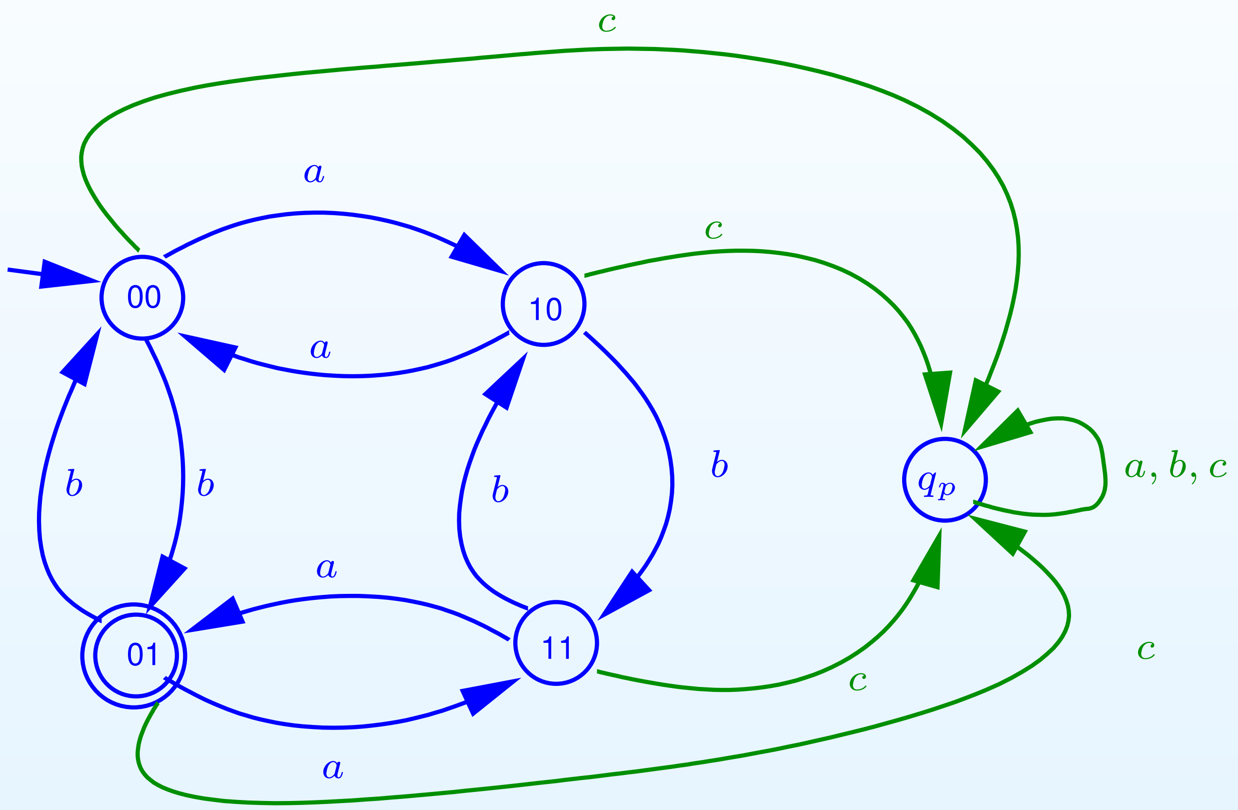
Exemple $\Sigma = \{a, b, c\}$.



□

Procédure de complémetation

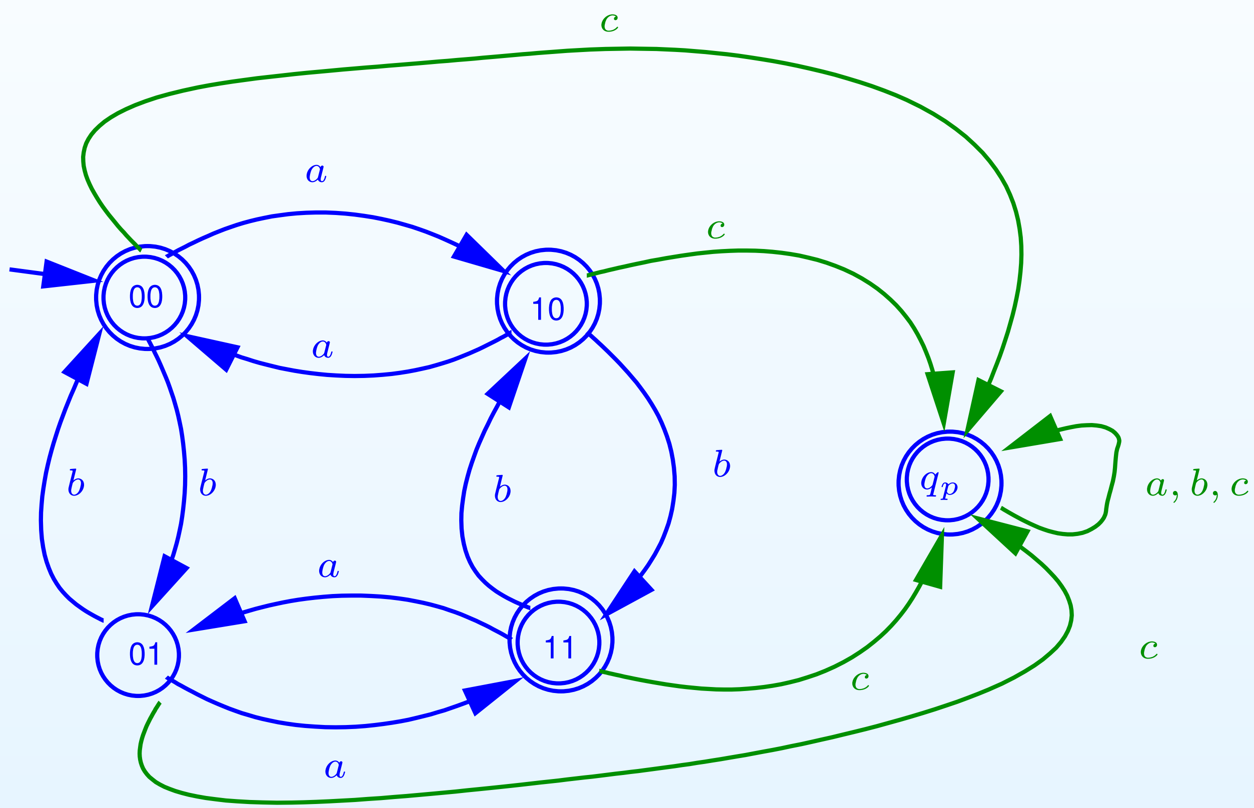
Exemple $\Sigma = \{a, b, c\}$.



□

Procédure de complémetation

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



$\square$

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:

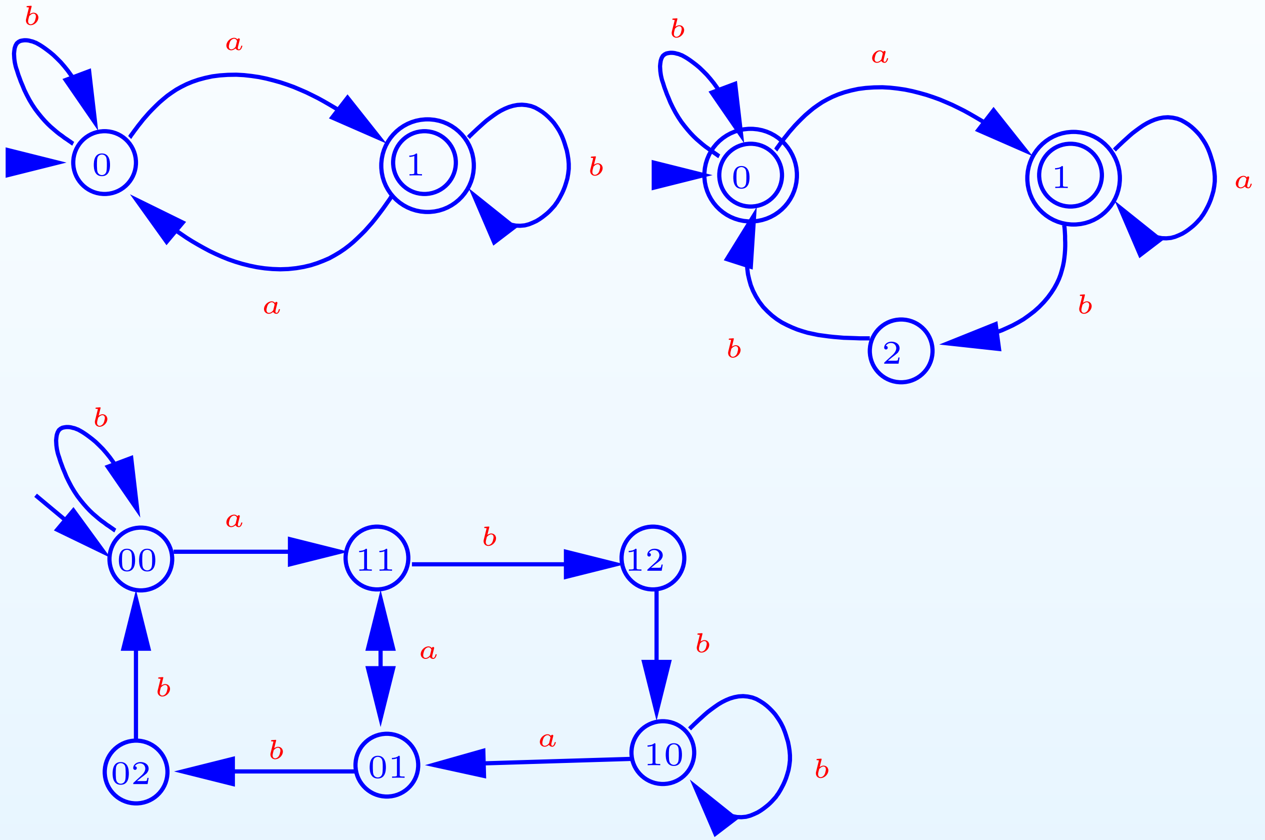
- $Q = Q^A \times Q^B$.
- $q_0 = (q_0^A, q_0^B)$
- $\delta : (Q^A \times Q^B) \times \Sigma \to (Q^A \times Q^B)$ est telle que

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

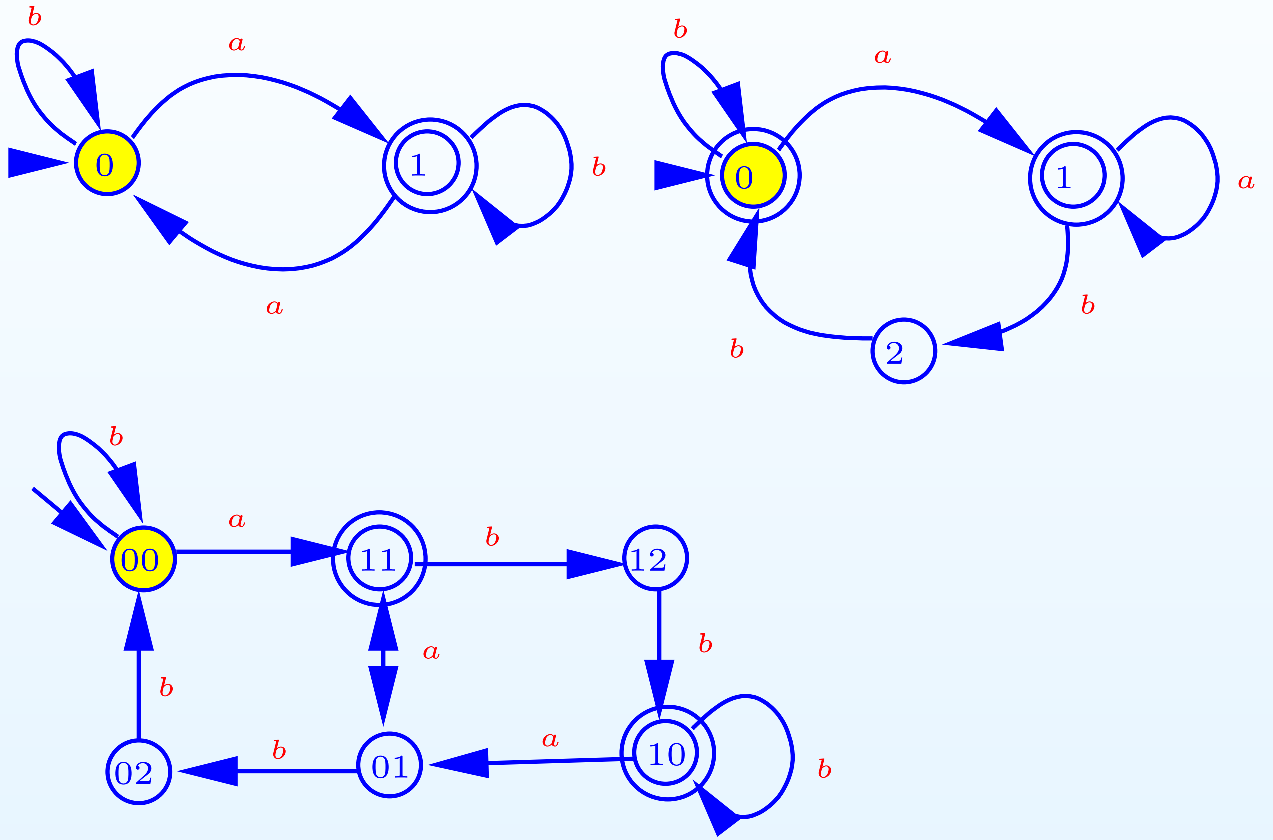
- $F = F^A \times F^B$.

$\square$

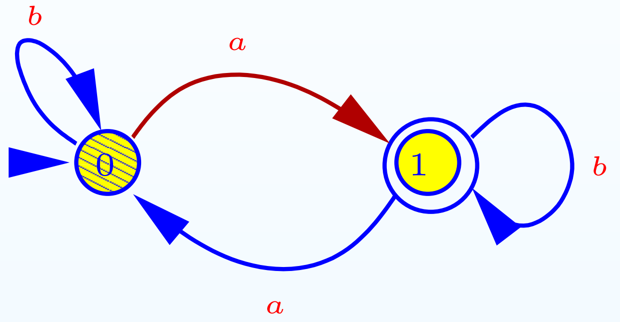
Exemple de produit d'automates

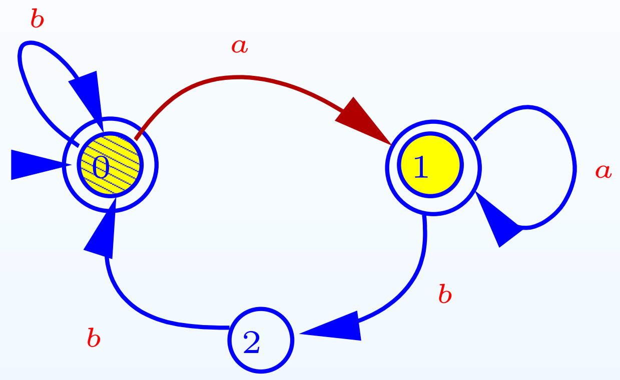


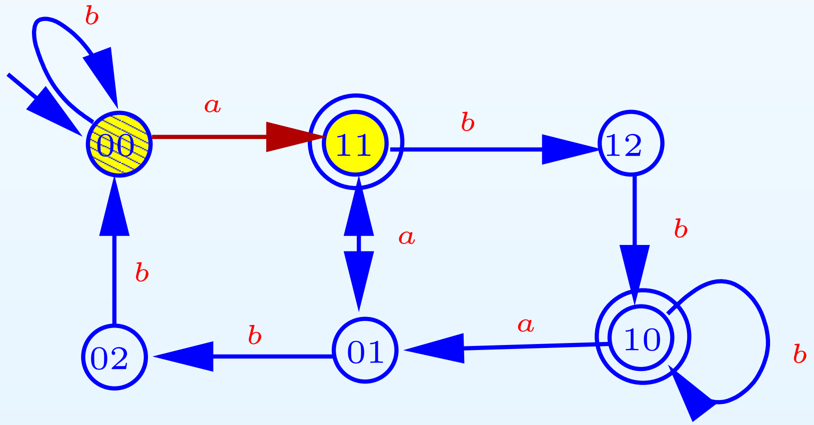
Exemple de produit d'automates



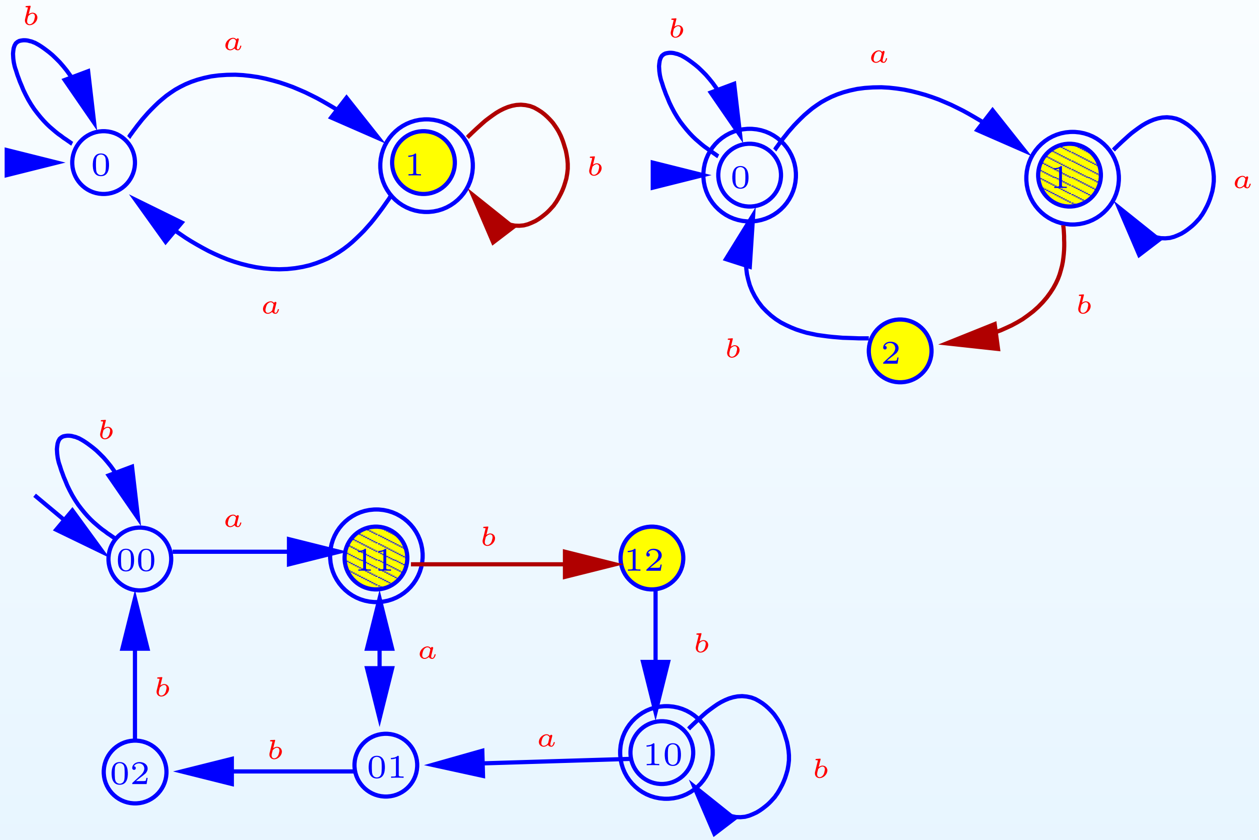
Exemple de produit d'automates



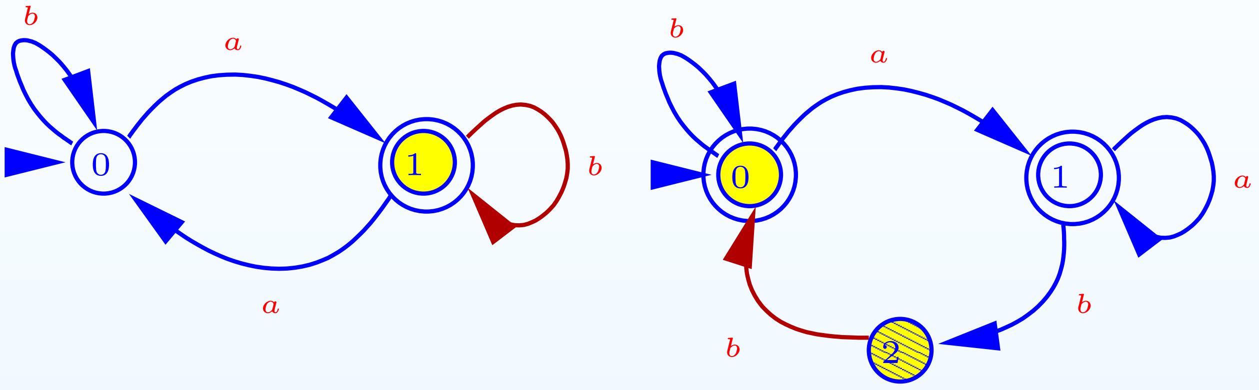


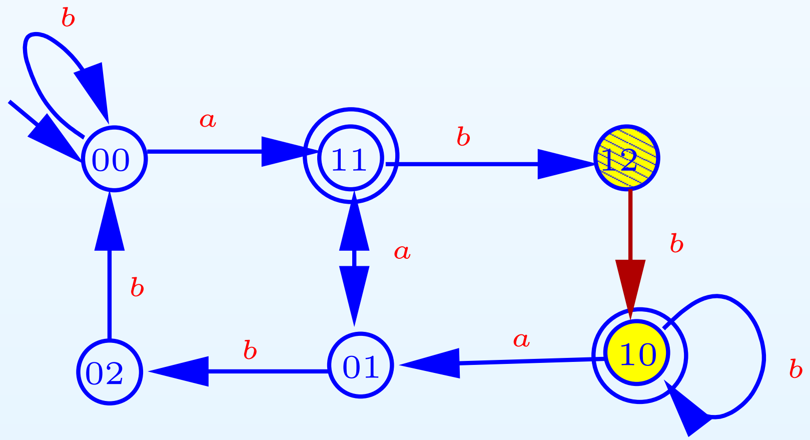


Exemple de produit d'automates

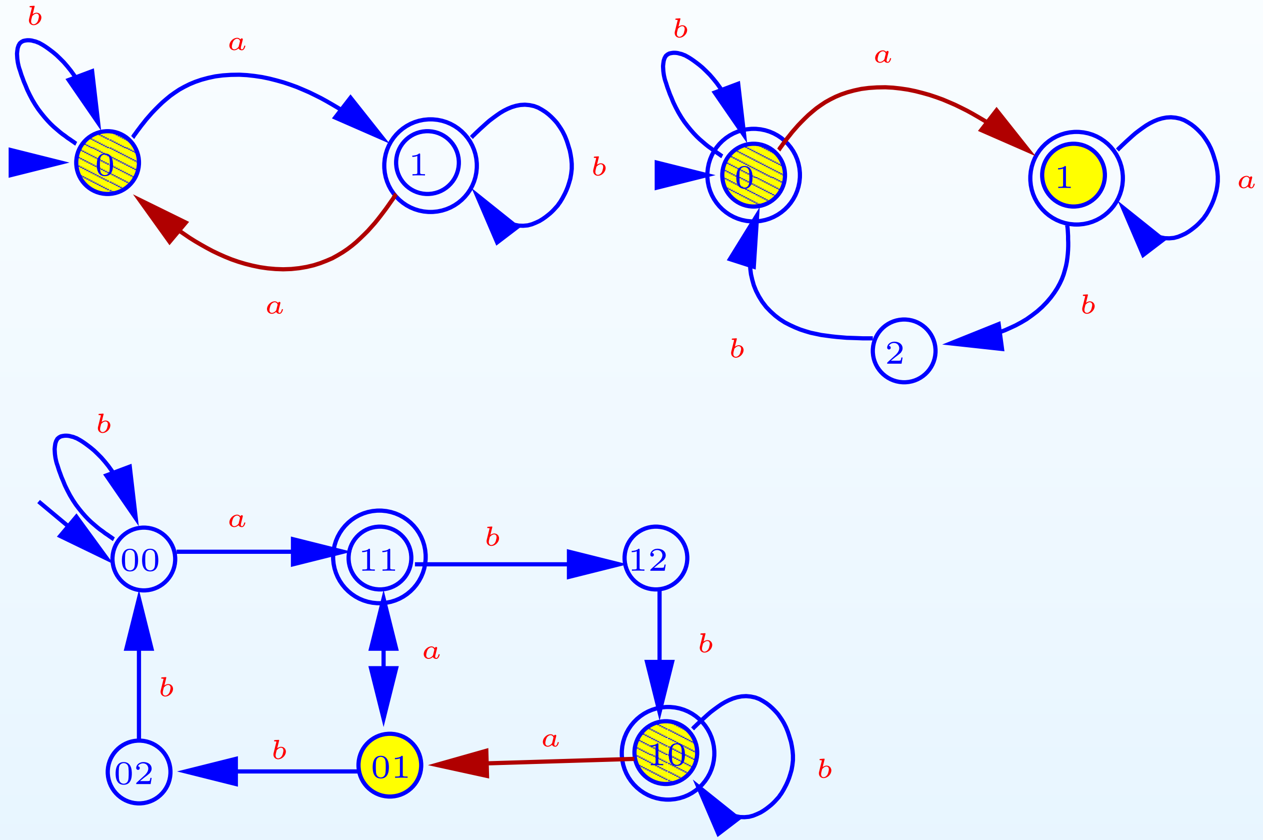


Exemple de produit d'automates





Exemple de produit d'automates



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. $L(A) \cap L(B) \subseteq L(A \times B$.

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:

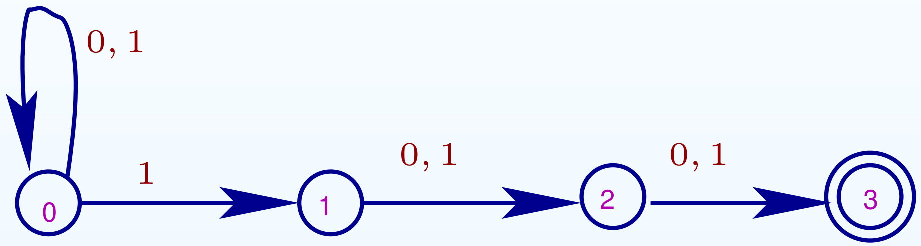
- Il est souvent plus facile de trouver un automate non-déterministe qui reconnait un langage L qu'un automate déterministe.
- Pour certains langages, on peut trouver un automate non-déterministe qui les reconnait et qui est plus petit que tout automate déterministe qui les reconnait.
- Mais, comme on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

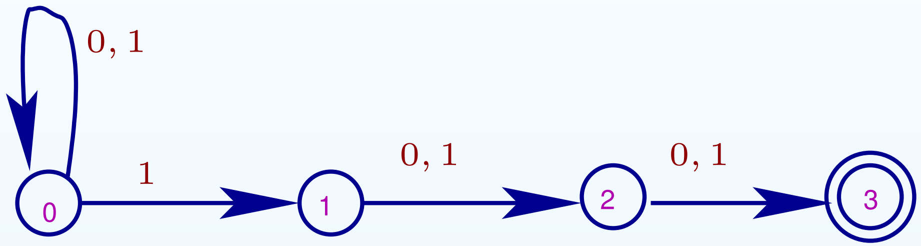
Exemple

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

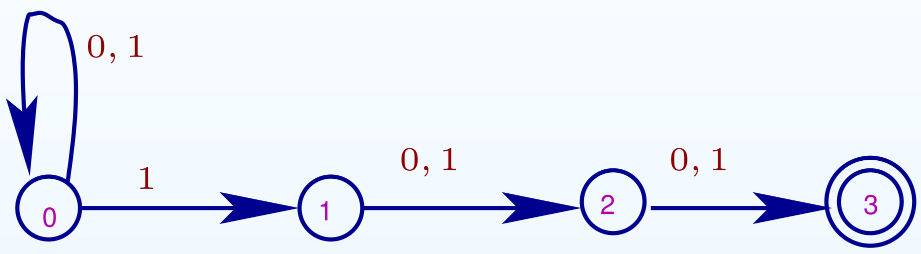
Soit $\Sigma = \{0, 1\}$. Soit L_3 le langage constitué des mots de longueur ≥ 3 et dont le 3ème symbole de droite est 1.



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 $\xrightarrow{*}_\Delta$ la fermeture réflexive et transition de $\longrightarrow_\Delta$.

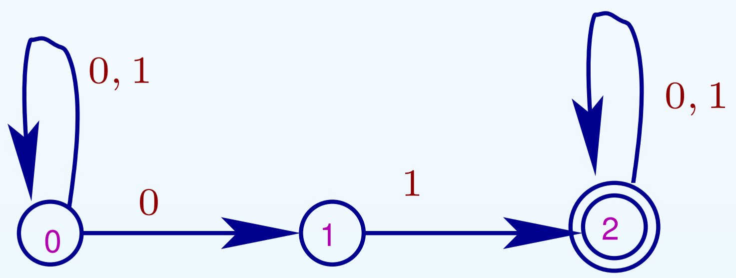
Langage reconnu par un ANDEF

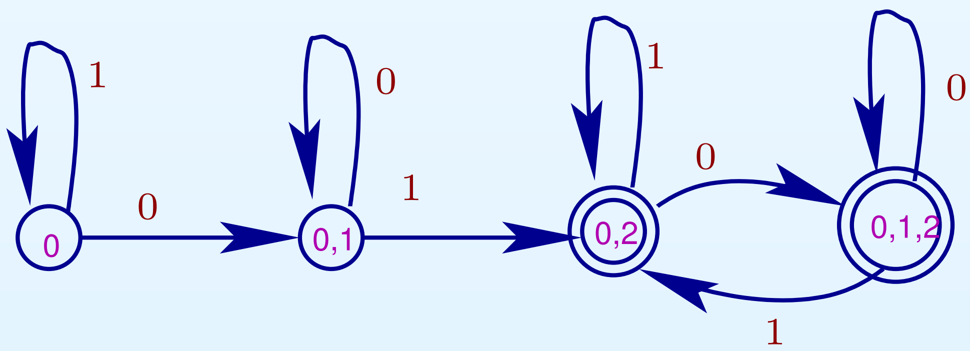
Soit $A = (Q, \Sigma, q_0, \Delta, F)$.

- 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 d'etats finis défini 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 $X \cap F \neq \emptyset$.

□

Nous avons le résultat suivant:

Correction de la procédure de déterminisation.

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

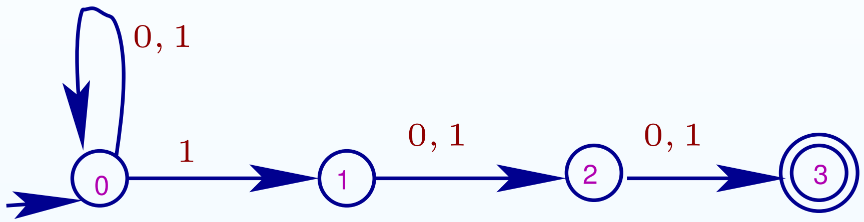
Preuve

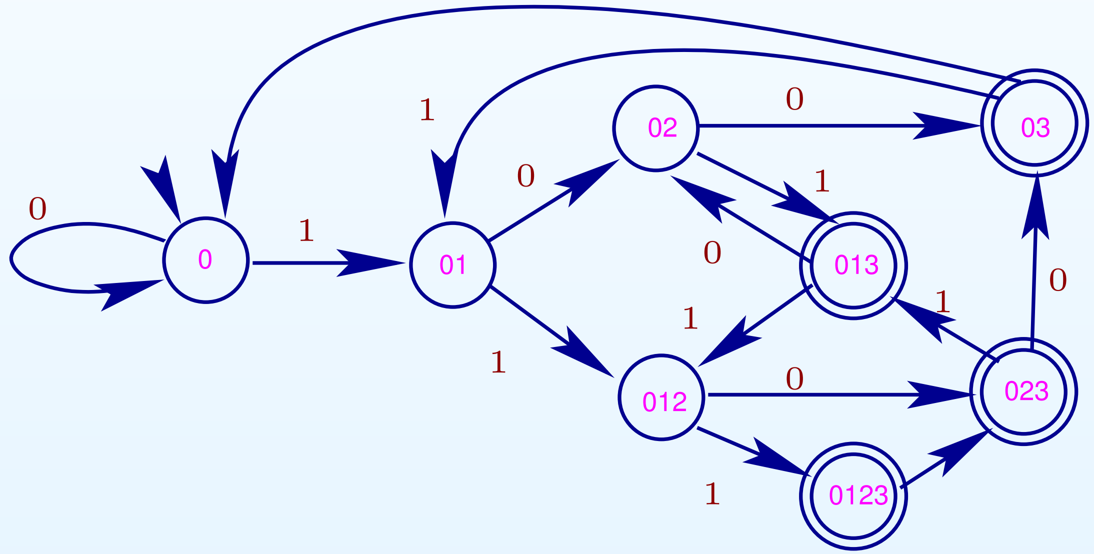
- On peut montrer $L(A) \subseteq L(\text{Det}(A))$ en montrant $A \sqsubseteq_{\in} \text{Det}(A)$.
- Pour montrer $L(\text{Det}(A)) \subseteq L(A)$, on montre:

  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:

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

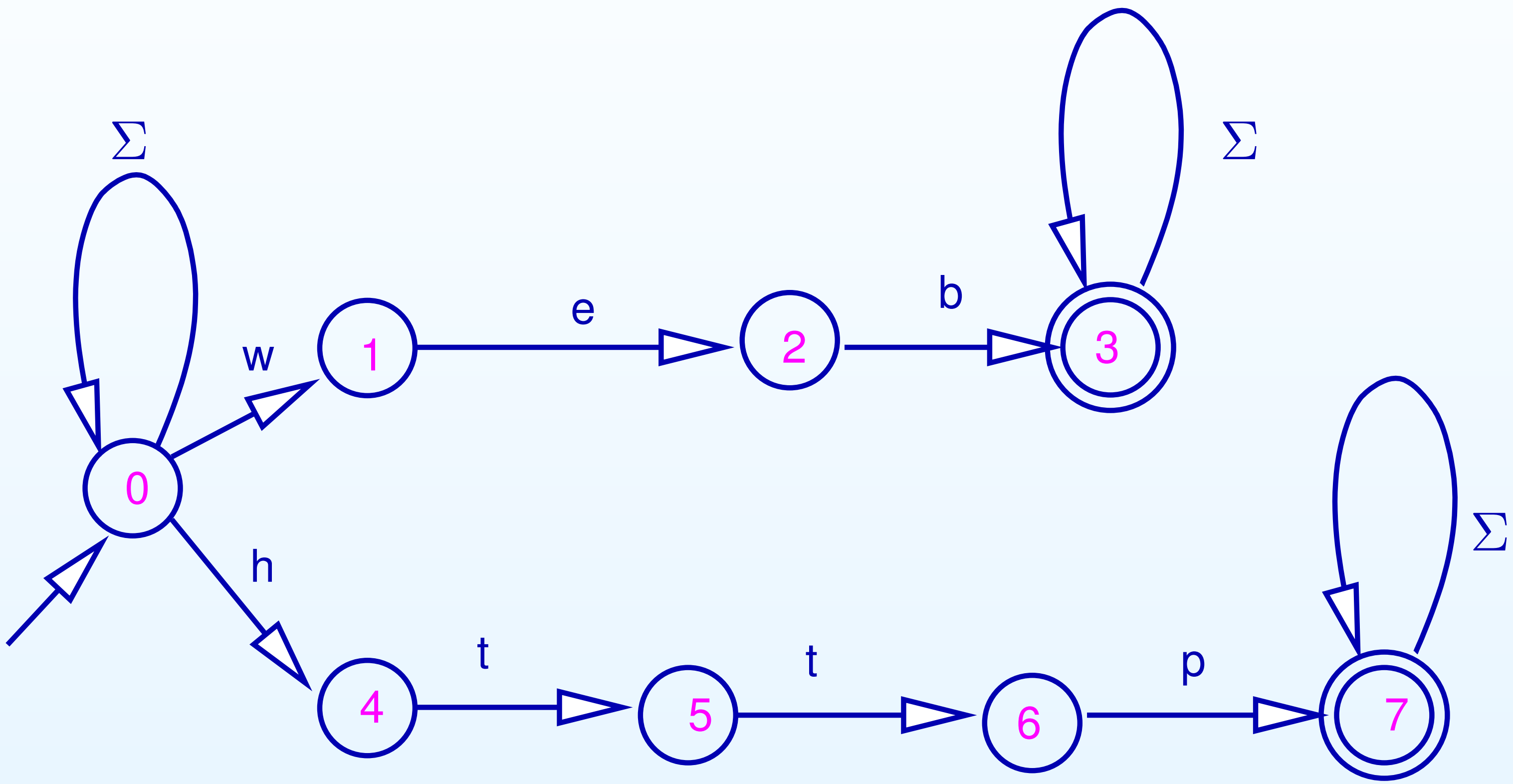
Lemme Aucun automate déterministe avec moins de 2^k états ne reconnait L_k. $\square$

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 differents 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 symetrie 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. *Le fermeture de Kleene de L*, dénoté L^*, est le plus petit langage tel que:

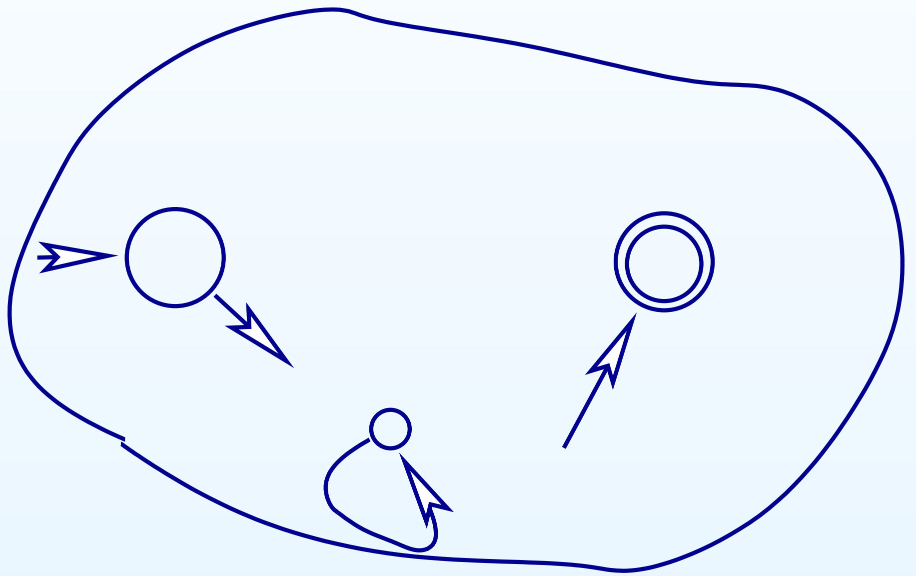
- $\epsilon \in L^*$ et
- pout tout $u \in L^*$ et $u' \in L$, on a $u \cdot u' \in L^*$.

$\square$

On peut montrer qu'on a: $u \in L^*$ ssi $u = \epsilon$ ou $\exists n \in \mathbb{N} \exists u_0, \cdots, u_n \in L \cdot u = u_1 \cdots u_n$. On veut montrer que si L est EF alors aussi L^*.

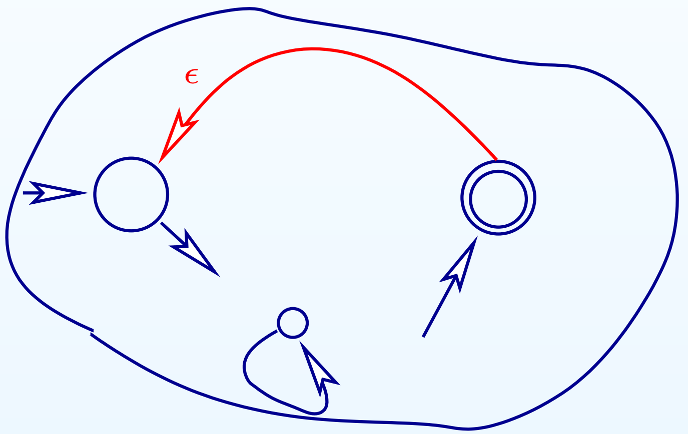
Fermeture par l'opérateur de Kleene

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



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*.

□

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

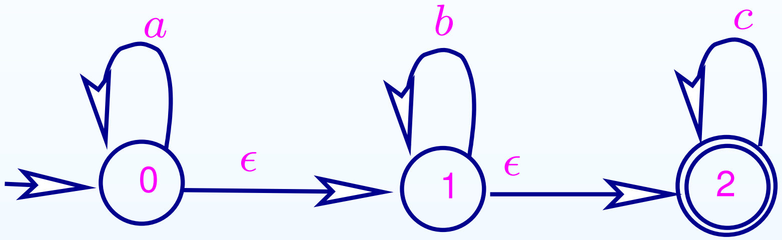
$$[(q, a, q') \in \Delta \text{ et } u' = u] \text{ ou } [a \cdot u = u' \text{ et } (q, \epsilon, q') \in \Delta].$$

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

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

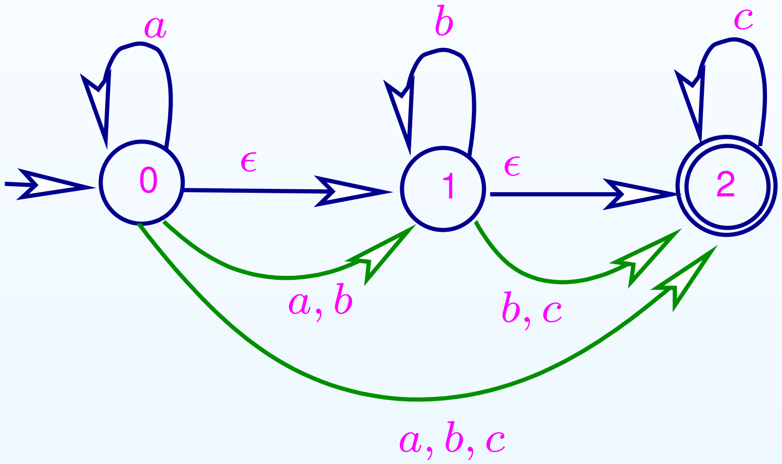
Exemple d'automate avec ϵ-transition

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



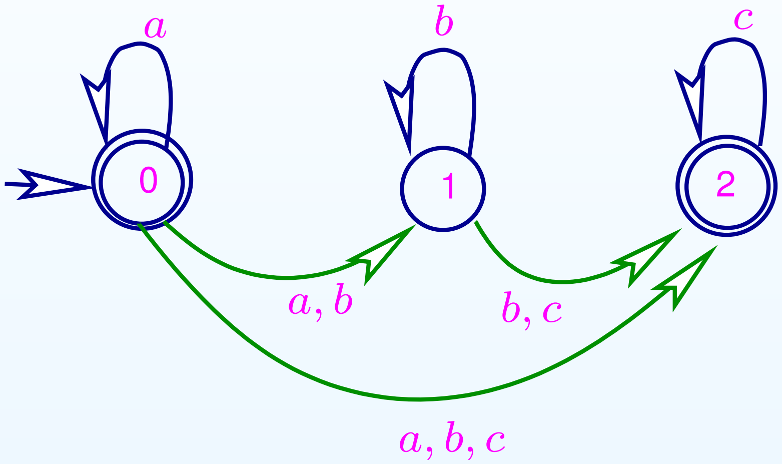
Exemple d'automate avec ϵ-transition

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. $q \xrightarrow{*}{}^{\epsilon} q_1$
2. $(q_1, a, q_2) \in \Delta$
3. $q_2 \xrightarrow{*}{}^{\epsilon} q'$.

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

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

Automate minimal

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 étend δ à des mots:
  $\delta^*(q, \epsilon) = q \quad \delta^*(q, au) = \delta^*(\delta(q, a), u)$
- Pour chaque paire $p, q \in Q$, on définit

$$discr(p, q) \stackrel{def}{=} \{u \in \Sigma^* | \quad \begin{array}{l} (\delta^*(p, u) \in F \wedge \delta^*(q, u) \notin F) \\ \vee(\delta^*(p, u) \notin F \wedge \delta^*(q, u) \in F)\} \end{array}$$

$\square$

Recherche de chaînes discriminantes

Pour chaque paire $p, q \in Q$:

si $(p \in F \wedge q \notin F) \vee (q \in F \wedge p \notin F)$

alors $discr(p, q) = \epsilon$

sinon $discr(p, q) = \emptyset$

Répéter :

Pour chaque paire $p, q \in Q$ et chaque action $a \in \Sigma$:

si $discr(p, q) = \emptyset$ et $discr(\delta(p, a), \delta(q, a)) \neq \emptyset$

alors $discr(p, q) = a \cdot discr(\delta(p, a), \delta(q, a))$

jusqu'à ce que la table $discr$ ne soit plus modifiable .

Remarque: si $|Q| = n$, alors l'itération s'éxecute au plus n^2 fois.

Équivalence des états indiscriminables

On définit une relation d'équivalence $\approx$ sur Q:

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

Remarque:$p \approx q$ ssi $discr(p, q) = \emptyset$.

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

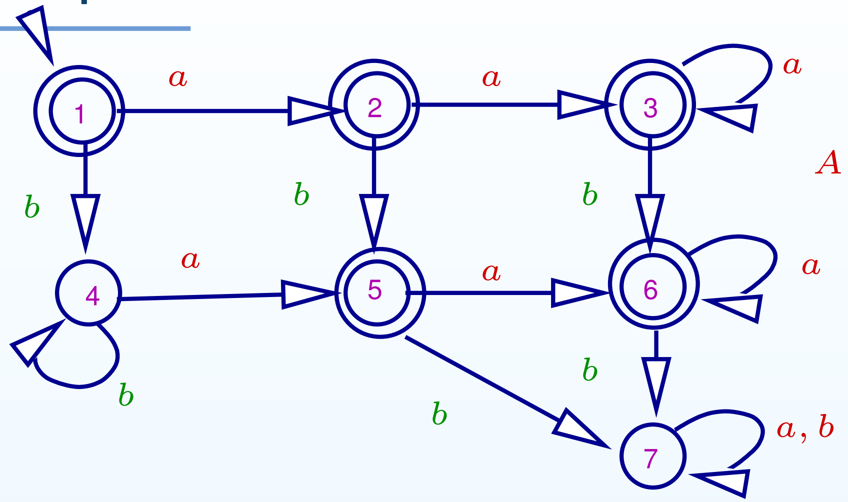
Théorème Soit $A = (Q, \Sigma, \delta, q_0, F)$ un ADEF complet dont tous les états sont accessibles.
Soit $A_{/\approx} = (Q_{/\approx}, \Sigma, [q_0], \delta_{/\approx}, F_{/\approx})$ où:

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

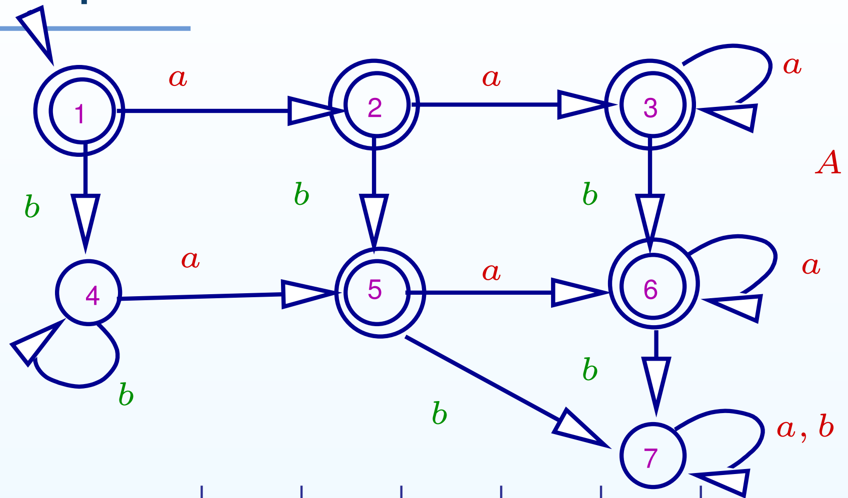
Alors,

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 $A_{/\approx}$.

Exemple

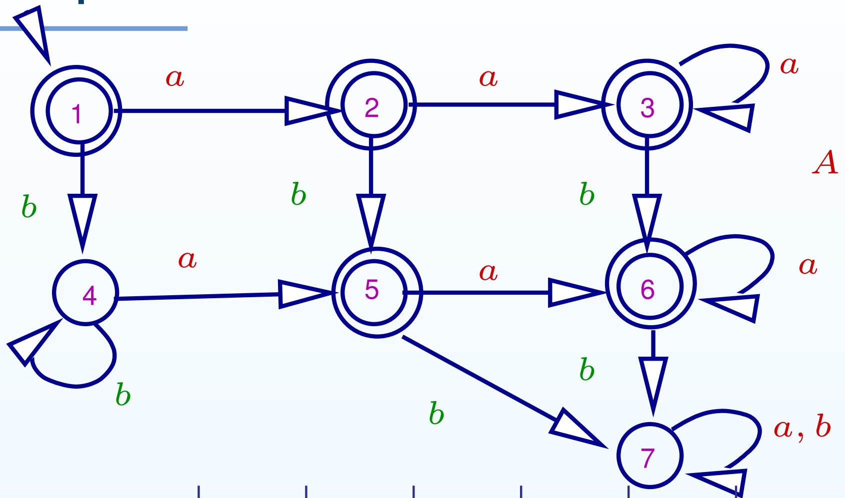


Exemple



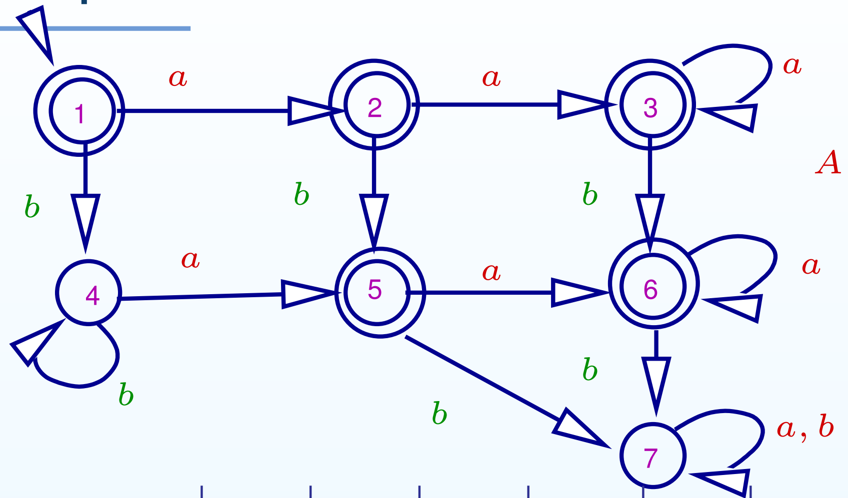
discr	1	2	3	4	5	6
7						
6						x
5					x	x
4				x	x	x
3			x	x	x	x
2		x	x	x	x	x

Exemple



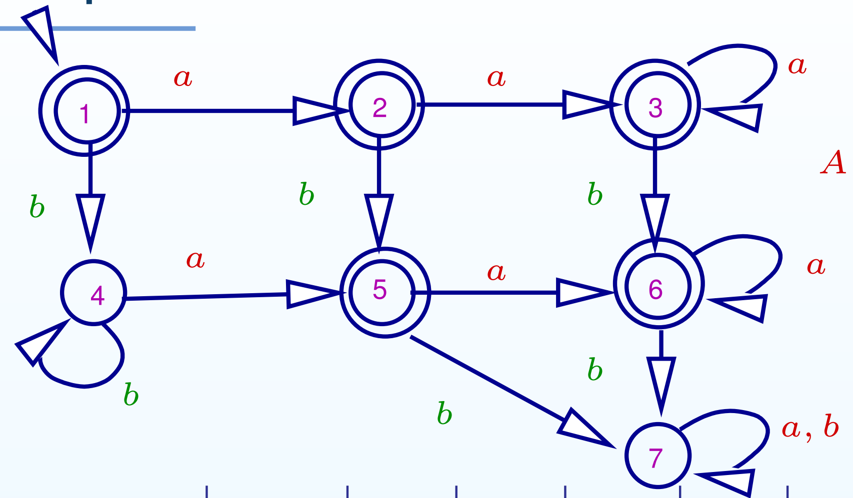
discr	1	2	3	4	5	6
7	ϵ	ϵ	ϵ		ϵ	ϵ
6				ϵ		x
5				ϵ	x	x
4	ϵ	ϵ	ϵ	x	x	x
3			x	x	x	x
2		x	x	x	x	x

Exemple



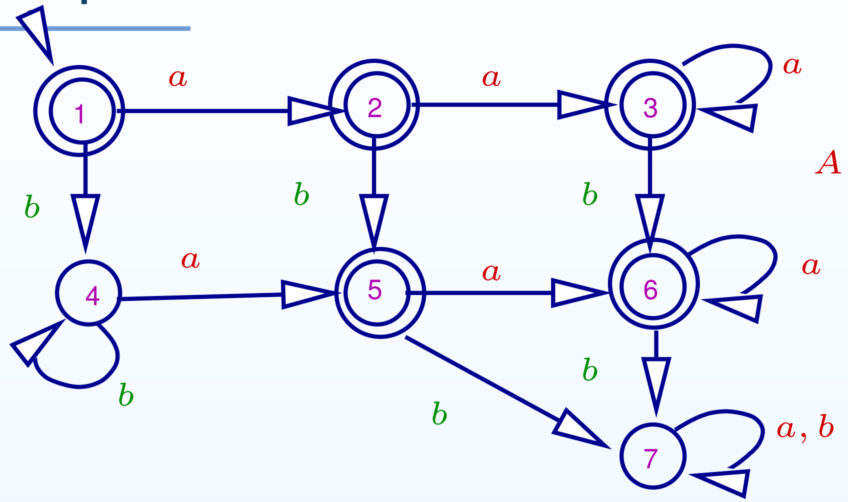
discr	1	2	3	4	5	6
7	ϵ	ϵ	ϵ	a	ϵ	ϵ
6		b	b	ϵ		x
5		b	b	ϵ	x	x
4	ϵ	ϵ	ϵ	x	x	x
3	b		x	x	x	x
2	b	x	x	x	x	x

Exemple

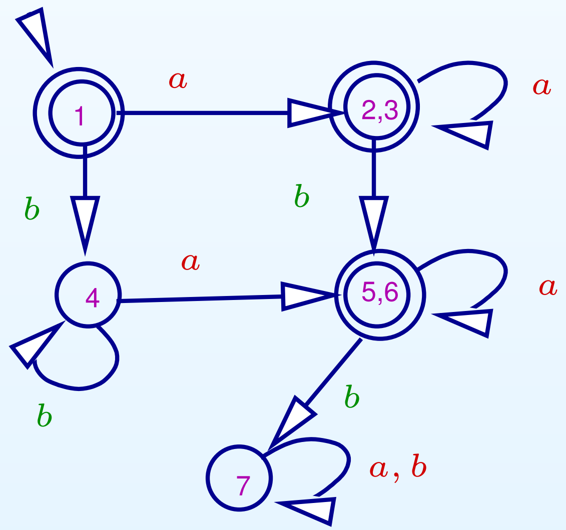


discr	1	2	3	4	5	6
7	ϵ	ϵ	ϵ	a	ϵ	ϵ
6	ab	b	b	ϵ		x
5	ab	b	b	ϵ	x	x
4	ϵ	ϵ	ϵ	x	x	x
3	b		x	x	x	x
2	b	x	x	x	x	x

Exemple

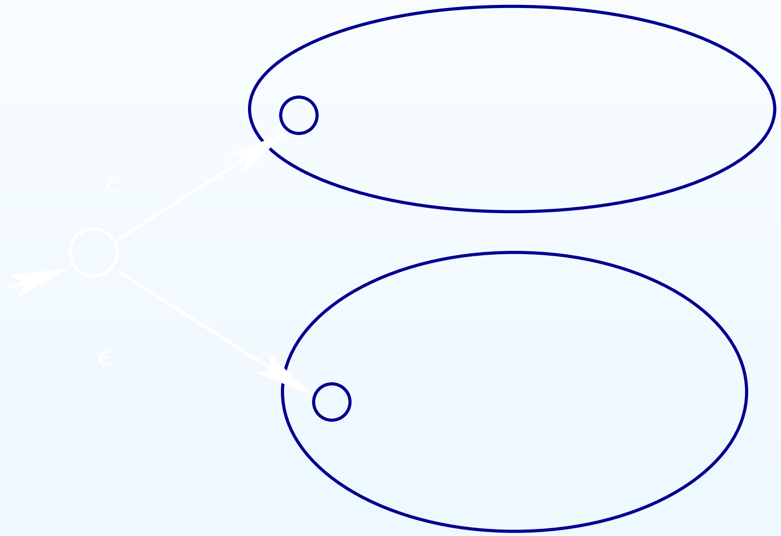


discr	1	2	3	4	5	6
7	ϵ	ϵ	ϵ	a	ϵ	ϵ
6	ab	b	b	ϵ		x
5	ab	b	b	ϵ	x	x
4	ϵ	ϵ	ϵ	x	x	x
3	b		x	x	x	x
2	b	x	x	x	x	x



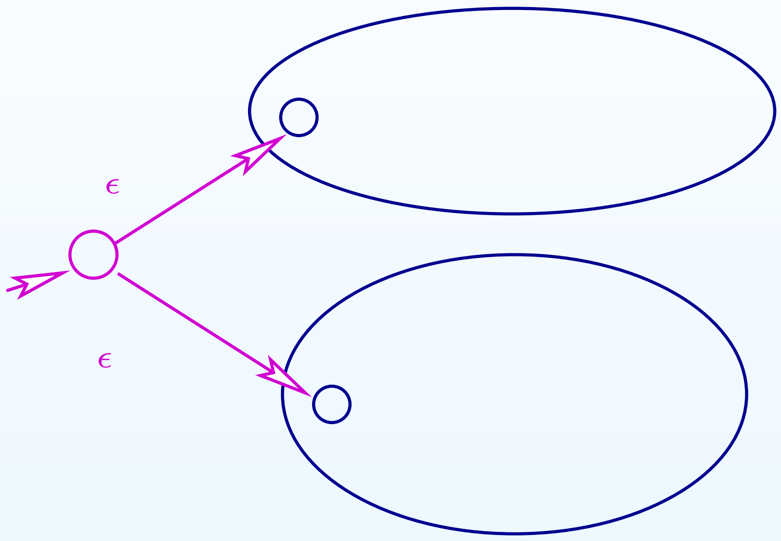
Fermeture de EF par Union et concaténaton

Union



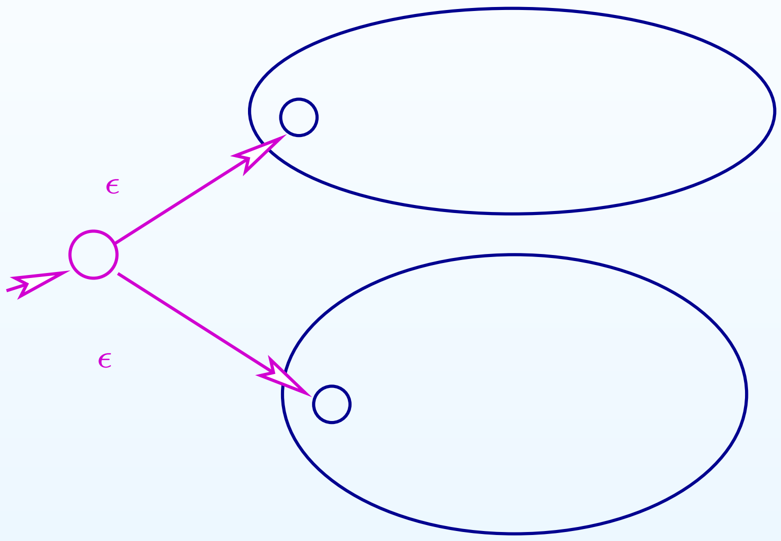
Fermeture de EF par Union et concaténaton

Union

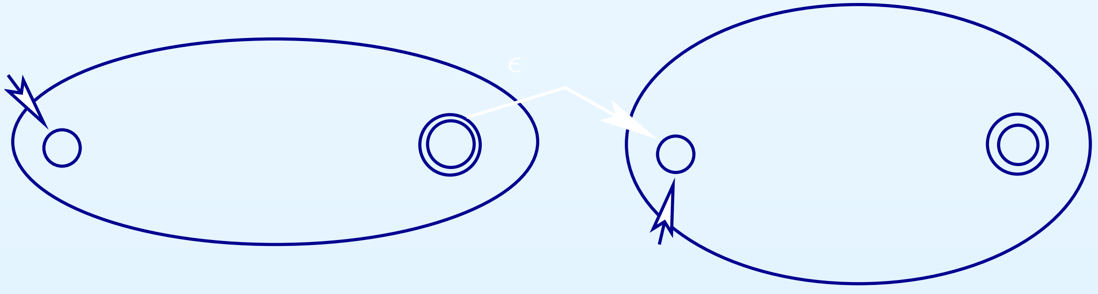


Fermeture de EF par Union et concaténaton

Union

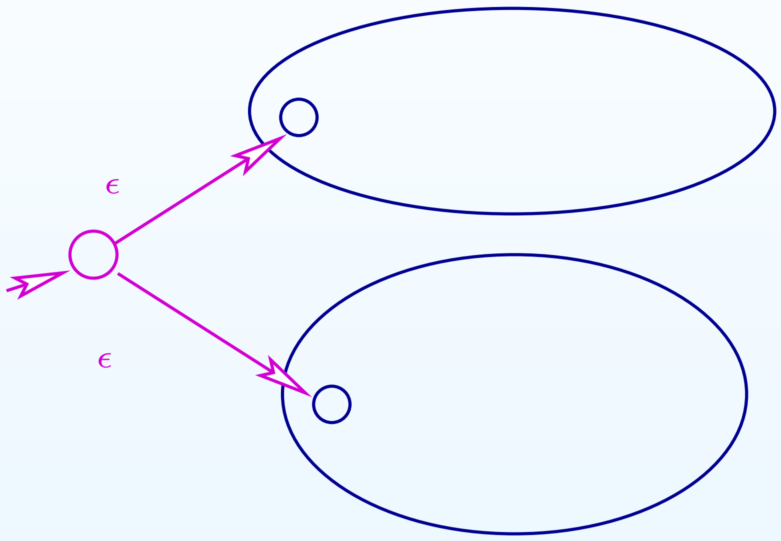


Concaténation

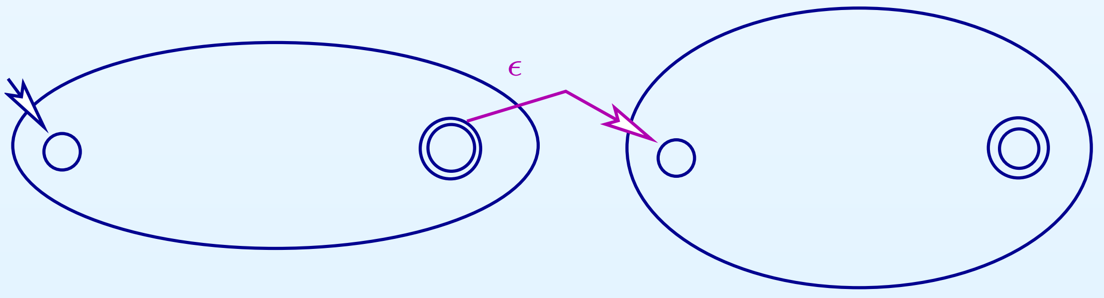


Fermeture de EF par Union et concaténaton

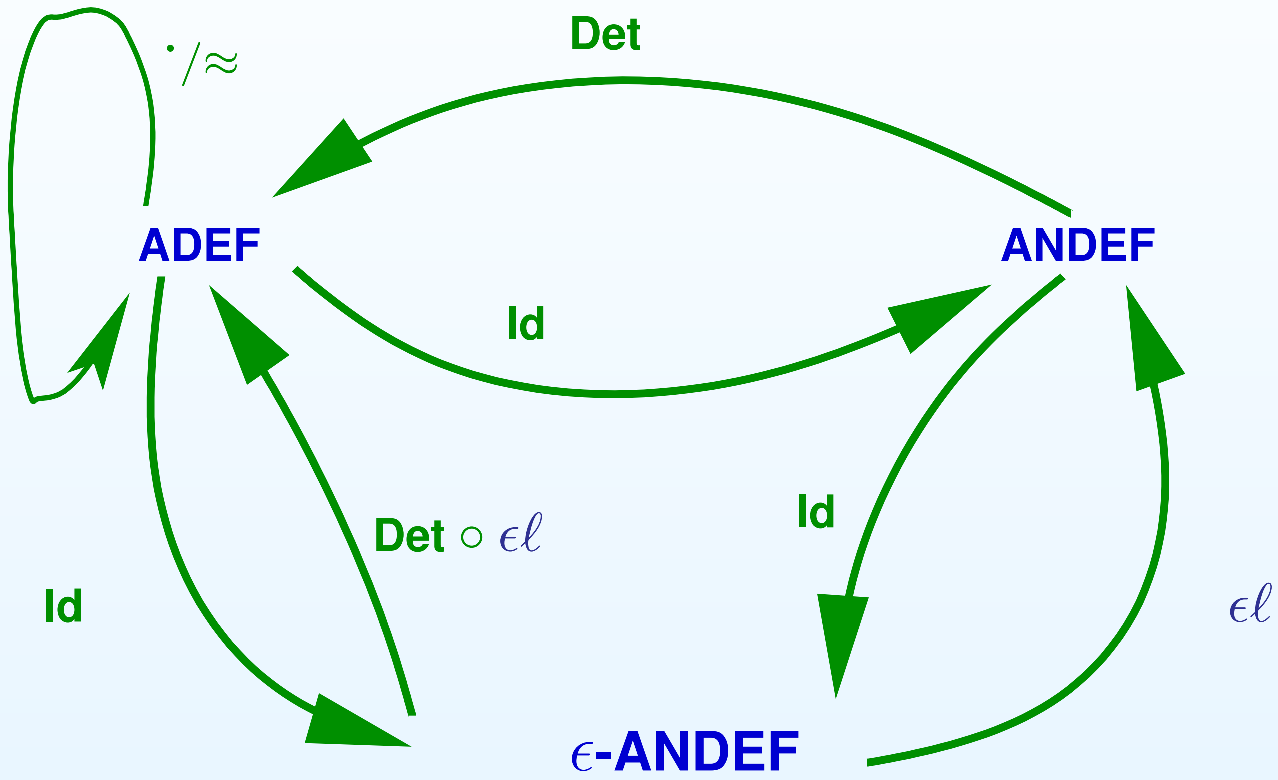
Union



Concaténation



Résumé I



Det
$\cdot/\approx$
ADEF
ANDEF
Id
Id
Det $\circ$ $\epsilon\ell$
Id
$\epsilon\ell$
ϵ-ANDEF

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^*).

Procédure de décision:

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

Langages qui ne sont pas d'états finis

Dénombrabilité

- un ensemble A est dénombrable si ses éléments peuvent être enumérés, c.a.d. si on peut définir une bijection entre A et $I\!N$ ou une partie de $I\!N$

Dénombrabilité

- un ensemble A est dénombrable si ses éléments peuvent être enumérés, c.a.d. si on peut définir une bijection entre A et $I\!N$ ou une partie de $I\!N$
- un alphabet Σ est un ensemble fini de symboles

Dénombrabilité

- un ensemble A est dénombrable si ses éléments peuvent être enumérés, c.a.d. si on peut définir une bijection entre A et $I\!N$ ou une partie de $I\!N$
- un alphabet Σ est un ensemble fini de symboles
- l'ensemble des mots Σ^* sur l'alphabet Σ est dénombrable

Dénombrabilité

- un ensemble A est dénombrable si ses éléments peuvent être enumérés, c.a.d. si on peut définir une bijection entre A et $I\!N$ ou une partie de $I\!N$
- un alphabet Σ est un ensemble fini de symboles
- l'ensemble des mots Σ^* sur l'alphabet Σ est dénombrable
- l'ensemble des langages sur l'alphabet Σ est non-dénombrable

Preuve de non-dénombrabilité

	w_1	w_2	w_3	$\cdots$	w_n	$\cdots$
L_1	$\color{red}{0}$	1	1	$\cdots$	0	$\cdots$
L_2	0	$\color{red}{1}$	0	$\cdots$	0	$\cdots$
L_3	1	1	$\color{red}{1}$	$\cdots$	1	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
L_n	0	1	1	$\cdots$	$\color{red}{0}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Preuve de non-dénombrabilité

	w_1	w_2	w_3	$\cdots$	w_n	$\cdots$
L_1	0	1	1	$\cdots$	0	$\cdots$
L_2	0	1	0	$\cdots$	0	$\cdots$
L_3	1	1	1	$\cdots$	1	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
L_n	0	1	1	$\cdots$	0	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

- soit $L \stackrel{def}{=} \{w_i \in \Sigma^* \mid w_i \notin L_i\}$, c.a.d. $w_i \in L$ ssi $w_i \notin L_i$

Preuve de non-dénombrabilité

	w_1	w_2	w_3	$\cdots$	w_n	$\cdots$
L_1	0	1	1	$\cdots$	0	$\cdots$
L_2	0	1	0	$\cdots$	0	$\cdots$
L_3	1	1	1	$\cdots$	1	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
L_n	0	1	1	$\cdots$	0	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

- soit $L \stackrel{def}{=} \{w_i \in \Sigma^* \mid w_i \notin L_i\}$, c.a.d. $w_i \in L$ ssi $w_i \notin L_i$
- si L serait parmi les L_i, alors il existe un j tel que $L = L_j$. Donc $w_j \in L_j$ ssi $w_j \in L$ ssi $w_j \notin L_j$... absurde. Donc L n'est pas parmi les L_i ...

Preuve de non-dénombrabilité

	w_1	w_2	w_3	$\cdots$	w_n	$\cdots$
L_1	0	1	1	$\cdots$	0	$\cdots$
L_2	0	1	0	$\cdots$	0	$\cdots$
L_3	1	1	1	$\cdots$	1	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
L_n	0	1	1	$\cdots$	0	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

- soit $L \stackrel{def}{=} \{w_i \in \Sigma^* \mid w_i \notin L_i\}$, c.a.d. $w_i \in L$ ssi $w_i \notin L_i$
- si L serait parmi les L_i, alors il existe un j tel que $L = L_j$. Donc $w_j \in L_j$ ssi $w_j \in L$ ssi $w_j \notin L_j$... absurde. Donc L n'est pas parmi les L_i ...
- donc l'ensemble des langages sur l'alphabet Σ est non-dénombrable

Langages qui ne sont pas d'états finis?

- l'ensemble des langages sur l'alphabet fini Σ est non-dénombrable

Langages qui ne sont pas d'états finis?

- l'ensemble des langages sur l'alphabet fini Σ est non-dénombrable
- l'ensemble des mots sur un alphabet fini quelconque est dénombrable

Langages qui ne sont pas d’états finis?

- l’ensemble des langages sur l’alphabet fini Σ est non-dénombrable
- l’ensemble des mots sur un alphabet fini quelconque est dénombrable
- l’ensemble des langages d’états finis sur l’alphabet Σ est dénombrable: un automate peut être encodé par un simple mot sur un alphabet fini qui serait l’union de Σ avec quelques symboles utilitaires (2 symboles permettant d’encoder les états, la virgule, les acolades,...)

Langages qui ne sont pas d'états finis?

- l'ensemble des langages sur l'alphabet fini Σ est non-dénombrable
- l'ensemble des mots sur un alphabet fini quelconque est dénombrable
- l'ensemble des langages d'états finis sur l'alphabet Σ est dénombrable: un automate peut être encodé par un simple mot sur un alphabet fini qui serait l'union de Σ avec quelques symboles utilitaires (2 symboles permettant d'encoder les états, la virgule, les acolades,...)
- donc, il existe des langages qui ne sont pas d'états finis

Langages qui ne sont pas d'états finis

1. Tous les langages finis sont d'états finis.

Langages qui ne sont pas d'états finis

1. Tous les langages finis sont d'états finis.
2. Un langage qui n'est pas d'états finis doit avoir un nombre infini de mots.

Langages qui ne sont pas d'états finis

1. Tous les langages finis sont d'états finis.
2. Un langage qui n'est pas d'états finis doit avoir un nombre infini de mots.
3. Si un langage comporte un nombre infini de mots, il n'y a pas de borne à la taille des mots faisant partie du langage.

Langages qui ne sont pas d'états finis

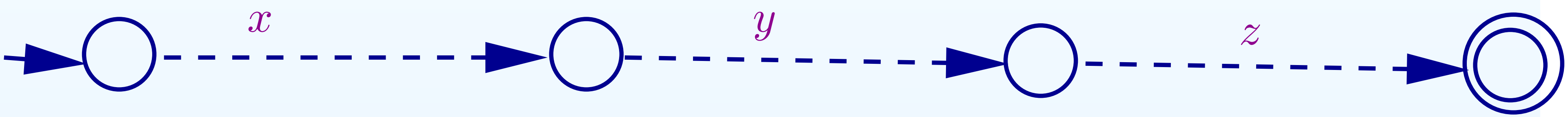
1. Tous les langages finis sont d'états finis.
2. Un langage qui n'est pas d'états finis doit avoir un nombre infini de mots.
3. Si un langage comporte un nombre infini de mots, il n'y a pas de borne à la taille des mots faisant partie du langage.
4. Tout langage d'états finis est accepté par un automate fini comportant un nombre fixé d'états, soit n.

Langages qui ne sont pas d'états finis

1. Considérons un langage infini d'états finis et un automate à n états acceptant ce langage. Pour tout mot de longueur supérieure à n, l'exécution de l'automate sur ce mot doit passer par un même état au moin deux fois, avec une partie non vide du mot séparant ces deux passages.

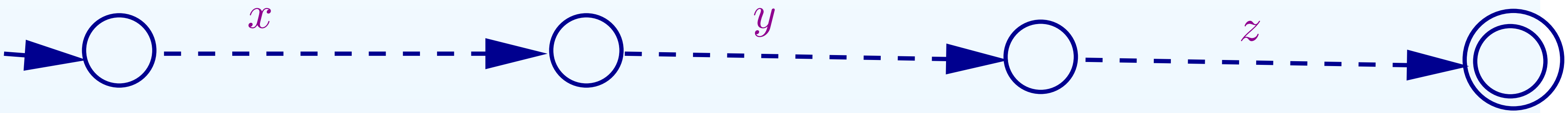
Langages qui ne sont pas d'états finis

1. Considérons un langage infini d'états finis et un automate à n états acceptant ce langage. Pour tout mot de longueur supérieure à n, l'exécution de l'automate sur ce mot doit passer par un même état au moin deux fois, avec une partie non vide du mot séparant ces deux passages.



Langages qui ne sont pas d'états finis

1. Considérons un langage infini d'états finis et un automate à n états acceptant ce langage. Pour tout mot de longueur supérieure à n, l'exécution de l'automate sur ce mot doit passer par un même état au moin deux fois, avec une partie non vide du mot séparant ces deux passages.



2. Donc, tous les mots de la forme xy^*z seront acceptés.

Lemme de l'itération

Comment montrer qu'un langage L n'est pas d'états finis?

Théorème (Lemme de l'itération, Pumping lemma)

Soit L un langage d'états finis. Alors, il existe $n \in \mathbb{N}$ 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 $k \in \mathbb{N}$, $xy^k z \in L$.

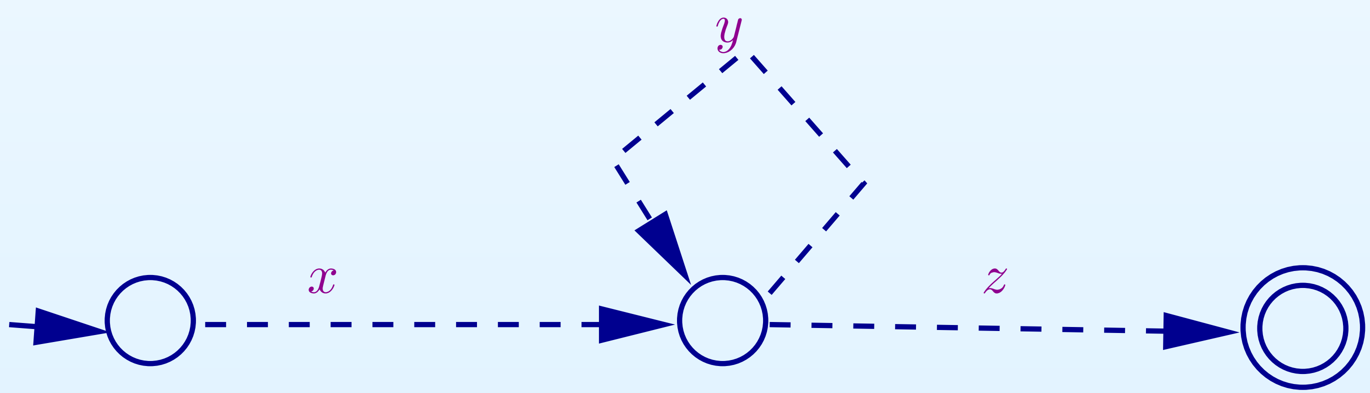
Lemme de l'itération

Comment montrer qu'un langage L n'est pas d'états finis?

Théorème (Lemme de l'itération, Pumping lemma)

Soit L un langage d'états finis. Alors, il existe $n \in \mathbb{N}$ 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 $k \in \mathbb{N}$, $xy^k z \in L$.

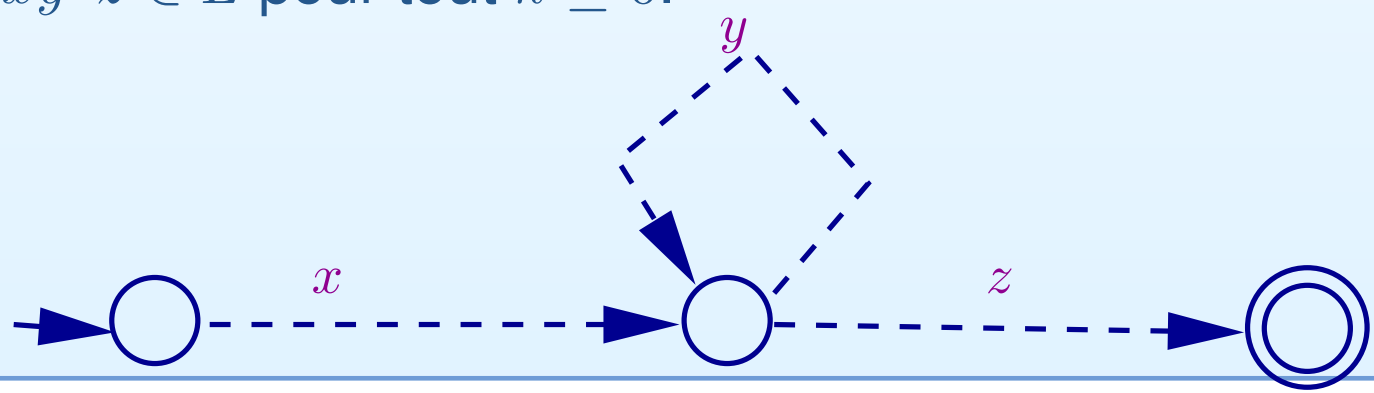


Démonstration du Lemme de l'itération

Soit L un langage d'états finis. 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, $xy^kz \in L$ 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 d'états finis.

Preuve par contradiction

Supposons que L est d'états finis. Alors, on sait par le Lemme de l'itération, 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 d'états finis.

Application du Lemme de l'itération (2)

Soit $L = \{a^{m^2} \mid m \geq 0\}$. L n'est pas d'états finis.

Preuve par contradiction

Supposons que L est d'états finis. Alors, on sait par le Lemme de l'itération, 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 $k \geq 0$.

Soient $w = a^{n^2}$ et $x, y, z \in \Sigma^*$ comme ci-dessus. Alors, comme $|xy| \leq n$ et $y \neq \epsilon$, on a $x = a^p$, $y = a^q$, $z = a^r$ avec $0 \leq p < n$, $0 < q \leq n$ et $0 \leq r$. Soit $w' = xy^2z = a^{p+2q+r}$. Alors, d'un côté on a $w' \in L$ mais aussi $w' \notin L$ car $n^2 < p + 2q + r \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$. Ce qui est une contradiction. Donc L n'est pas d'états finis.

Application du Lemme de l'itération (3)

Soit $L = \{a^m \mid m \text{ premier}\}$. L n'est pas d'états finis.

Preuve par contradiction

Supposons que L est d'états finis. Alors, on sait par le Lemme de l'itération, 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 $k \geq 0$.

Soientt $w = a^n$, $x, y, z \in \Sigma^*$ comme ci-dessus. Alors, comme $|xy| \leq n$ et $y \neq \epsilon$, on a $x = a^p$, $y = a^q$, $z = a^r$ avec $0 \leq p < n$, $0 < q \leq n$ et $0 \leq r$. Soit $w' = xy^{p+2q+r+2}z = a^{p+q(p+2q+r+2)+r}$. Alors, d'un côté on a $w' \in L$ mais aussi $w' \notin L$ car $p+q(p+2q+r+2)+r = p+r+q(p+r)+q(2q+2) = (q+1)(p+r+2q)$, donc contradiction. Donc L n'est pas d'états finis.

Langages qui ne sont pas d'états finis (I)

Un automate d'états finis ne peut reconaitre $L = \{a^i b^i \mid i \geq 0\}$ car il "ne sait pas compter"...il n'a pas de memoire...

Langages qui ne sont pas d’états finis (I)

Un automate d’états finis ne peut reconaitre $L = \{a^i b^i \mid i \geq 0\}$ car il “ne sait pas compter”...il n’a pas de memoire...

idée: un automate d’états finis + un compteur (positif) incrementé ou décrementé par les transitions.
On incrémente le compteur quand on rencontre un a dans le mot, et on le décrémente quand on rencontre un b.

Langages qui ne sont pas d'états finis (I)

Un automate d'états finis ne peut reconaitre $L = \{a^i b^i \mid i \geq 0\}$ car il "ne sait pas compter"...il n'a pas de memoire...

idée: un automate d'états finis + un compteur (positif) incrementé ou décrementé par les transitions.
On incrémente le compteur quand on rencontre un a dans le mot, et on le décrémente quand on rencontre un b.

On a alors les transitions suivantes:
$(q, +1) \in \delta(q, a) \quad (q', -1) \in \delta(q, b) \quad (q', -1) \in \delta(q', b)$

Langages qui ne sont pas d'états finis (I)

Un automate d'états finis ne peut reconaitre $L = \{a^i b^i \mid i \geq 0\}$ car il "ne sait pas compter"...il n'a pas de memoire...

idée: un automate d'états finis + un compteur (positif) incrementé ou décrementé par les transitions.
On incrémente le compteur quand on rencontre un a dans le mot, et on le décrémente quand on rencontre un b.

On a alors les transitions suivantes:
$(q, +1) \in \delta(q, a) \quad (q', -1) \in \delta(q, b) \quad (q', -1) \in \delta(q', b)$

On accepte le mot si on est dans l'état q' et si le compteur vaut 0.
$(q, aabb, 0) \longrightarrow (q, abb, 1) \longrightarrow (q, bb, 2) \longrightarrow (q', b, 1) \longrightarrow (q', \epsilon, 0)$

Langages qui ne sont pas d'états finis (II)

Mais un compteur peut ne pas être suffisant...
Le langage des palindromes: $L = \{w\tilde{w} \mid w \in \{a,b\}^*\}$.

Langages qui ne sont pas d'états finis (II)

Mais un compteur peut ne pas être suffisant...
Le langage des palindromes: $L = \{w\tilde{w} \mid w \in \{a, b\}^*\}$.
Un compteur = une pile où on ne manipule qu'un seul symbole

- +1 = empiler z
- -1 = depiler z
- accepter si la pile est vide

Langages qui ne sont pas d'états finis (II)

Mais un compteur peut ne pas être suffisant...
Le langage des palindromes: $L = \{w\tilde{w} \mid w \in \{a,b\}^*\}$.
Un compteur = une pile où on ne manipule qu'un seul symbole

- +1 = empiler z
- -1 = depiler z
- accepter si la pile est vide

On va utiliser une pile qui va manipuler un nombre fini de symboles, appelés alphabet de pile.

Automates à pile

Automates à pile

Définition 0.1 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

Automates à pile

Définition 0.2 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*

Automates à pile

Définition 0.3 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*
- Σ *est l'alphabet (fini) d'entrée*

Automates à pile

Définition 0.4 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*
- Σ *est l'alphabet (fini) d'entrée*
- Γ *est l'alphabet (fini) de la pile*

Automates à pile

Définition 0.5 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*
- Σ *est l'alphabet (fini) d'entrée*
- Γ *est l'alphabet (fini) de la pile*
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times \Gamma^* \times Q$ *est la relation de transition.*

Automates à pile

Définition 0.6 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*
- Σ *est l'alphabet (fini) d'entrée*
- Γ *est l'alphabet (fini) de la pile*
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times \Gamma^* \times Q$ *est la relation de transition.*
- q_0 *est l'état initial.*

Automates à pile

Définition 0.7 *Un* automate à pile P *(push-down automaton) (AP) est défini par* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *où :*

- Q *est un ensemble fini d'états.*
- Σ *est l'alphabet (fini) d'entrée*
- Γ *est l'alphabet (fini) de la pile*
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times \Gamma^* \times Q$ *est la relation de transition.*
- q_0 *est l'état initial.*
- $Z_0 \in \Gamma$ *est le symbole initial de la pile.*

Configuration

Une *configuration* de P est un élément $(q, u, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ où

Configuration

Une *configuration* de P est un élément $(q, u, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ où

- q est l'état courant.

Configuration

Une *configuration* de P est un élément $(q, u, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ où

- q est l'état courant.
- u est le mot qui reste à analyser.

Configuration

Une *configuration* de P est un élément $(q, u, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ où

- q est l'état courant.
- u est le mot qui reste à analyser.
- α est le contenu courant de la pile.

Configuration

Une *configuration* de P est un élément $(q, u, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ où

- q est l'état courant.
- u est le mot qui reste à analyser.
- α est le contenu courant de la pile.

Le mot de Γ^* qui represente le contenu de la pile est lu du haut de la pile vers le bas de la pile.

Σ-transition

L'automate P réalise une Σ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

Σ-transition

L'automate P réalise une Σ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, a, Z, \gamma, q') \in \Delta$.

Σ-transition

L'automate P réalise une Σ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, a, Z, \gamma, q') \in \Delta$.
- $u = a \cdot u'$, $\alpha = Z \cdot \beta$.

Σ-transition

L'automate P réalise une Σ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, a, Z, \gamma, q') \in \Delta$.
- $u = a \cdot u'$, $\alpha = Z \cdot \beta$.
- $\alpha' = \gamma \cdot \beta$

ϵ-transition

L'automate P réalise une ϵ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

ϵ-transition

L'automate P réalise une ϵ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, \epsilon, Z, \gamma, q') \in \Delta$.

ϵ-transition

L'automate P réalise une ϵ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, \epsilon, Z, \gamma, q') \in \Delta$.
- $u = u'$, $\alpha = Z \cdot \beta$.

ϵ-transition

L'automate P réalise une ϵ-transition d'une configuration (q, u, α) vers une configuration (q', u', α'), noté $(q, u, \alpha) \vdash_P (q', u', \alpha')$ si:

- il existe une transition $(q, \epsilon, Z, \gamma, q') \in \Delta$.
- $u = u'$, $\alpha = Z \cdot \beta$.
- $\alpha' = \gamma \cdot \beta$

Accéptation par pile vide

- Une configuration initiale est de la forme (q_0, u, Z_0).

Accéptation par pile vide

- Une configuration initiale est de la forme (q_0, u, Z_0).
- Une configuration finale est de la forme (q, ϵ, ϵ).

Accéptation par pile vide

- Une configuration initiale est de la forme (q_0, u, Z_0).
- Une configuration finale est de la forme (q, ϵ, ϵ).
- Un mot u est accépté par P, s'il existe $q \in Q$ avec :

$$(q_0, u, Z_0) \vdash_P^* (q, \epsilon, \epsilon).$$

C'est le critère d'accéptation par pile vide.

Accéptation par pile vide

- Une configuration initiale est de la forme (q_0, u, Z_0).
- Une configuration finale est de la forme (q, ϵ, ϵ).
- Un mot u est accépté par P, s'il existe $q \in Q$ avec :

$$(q_0, u, Z_0) \vdash_P^* (q, \epsilon, \epsilon).$$

  C'est le critère d'accéptation par pile vide.
- Le langage reconnu par P est

$$L(P) = \{u \mid \exists q \in Q \cdot (q_0, u, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}.$$

Accéptation par pile vide

- Une configuration initiale est de la forme (q_0, u, Z_0).
- Une configuration finale est de la forme (q, ϵ, ϵ).
- Un mot u est accépté par P, s'il existe $q \in Q$ avec :

$$(q_0, u, Z_0) \vdash_P^* (q, \epsilon, \epsilon).$$

  C'est le critère d'accéptation par pile vide.
- Le langage reconnu par P est

$$L(P) = \{u \mid \exists q \in Q \cdot (q_0, u, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}.$$

- Les langages acceptés par des automates à pile sont appelés des langages algébriques.

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.
- $\Sigma = \{a, b\}$.

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_a, \epsilon, Z_0, \epsilon, q_a), (q_a, a, Z_0, Z, q_a), \\ & (q_a, a, Z, ZZ, q_a), (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b)\} \end{aligned}$$

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_a, \epsilon, Z_0, \epsilon, q_a), (q_a, a, Z_0, Z, q_a), \\ & (q_a, a, Z, ZZ, q_a), (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b)\} \end{aligned}$$
- q_a est l'état initial.

Exemple (1)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par pile vide par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_a, Z_0)$ où :

- $Q = \{q_a, q_b\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_a, \epsilon, Z_0, \epsilon, q_a), (q_a, a, Z_0, Z, q_a), \\ & (q_a, a, Z, ZZ, q_a), (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b)\} \end{aligned}$$
- q_a est l'état initial.
- $Z_0 \in \Gamma$ est le symbole initial de la pile.

Accéptation par état final

On peut aussi avoir un critère d'accéptation par état final. Dans ce cas un automate à pile est défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F)$ où on donne $F \subseteq Q$ un ensemble d'états finaux (accepteurs).

Accéptation par état final

On peut aussi avoir un critère d'accéptation par état final. Dans ce cas un automate à pile est défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F)$ où on donne $F \subseteq Q$ un ensemble d'états finaux (accepteurs).

- Une configuration finale est de la forme (q, ϵ, γ) avec $q \in F$.

Accéptation par état final

On peut aussi avoir un critère d'accéptation par état final. Dans ce cas un automate à pile est défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F)$ où on donne $F \subseteq Q$ un ensemble d'états finaux (accepteurs).

- Une configuration finale est de la forme (q, ϵ, γ) avec $q \in F$.
- Un mot u est accépté par P, s'il existe $q \in F$ avec :

$$(q_0, u, Z_0) \vdash_P^* (q, \epsilon, \gamma).$$

Accéptation par état final

On peut aussi avoir un critère d'accéptation par état final. Dans ce cas un automate à pile est défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F)$ où on donne $F \subseteq Q$ un ensemble d'états finaux (accepteurs).

- Une configuration finale est de la forme (q, ϵ, γ) avec $q \in F$.
- Un mot u est accépté par P, s'il existe $q \in F$ avec :

$$(q_0, u, Z_0) \vdash_P^* (q, \epsilon, \gamma).$$

- Le langage reconnu par P est

$$L(P) = \{u \mid \exists q \in F \cdot (q_0, u, Z_0) \vdash_P^* (q, \epsilon, \gamma)\}.$$

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.
- $\Sigma = \{a, b\}$.

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_0, \epsilon, Z_0, \epsilon, q_f), (q_0, a, Z_0, ZZ_0, q_a), (q_a, a, Z, ZZ, q_a), \\ & (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b), (q_b, \epsilon, Z_0, \epsilon, q_f)\} \end{aligned}$$

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned}\Delta = \quad & \{(q_0, \epsilon, Z_0, \epsilon, q_f), (q_0, a, Z_0, ZZ_0, q_a), (q_a, a, Z, ZZ, q_a), \\ & (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b), (q_b, \epsilon, Z_0, \epsilon, q_f)\}\end{aligned}$$
- q_0 est l'état initial.

Exemple (2)

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est accépté par état final par l'automate à pile défini par $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, \{q_f\})$ où :

- $Q = \{q_0, q_a, q_b, q_f\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{Z_0, Z\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_0, \epsilon, Z_0, \epsilon, q_f), (q_0, a, Z_0, ZZ_0, q_a), (q_a, a, Z, ZZ, q_a), \\ & (q_a, b, Z, \epsilon, q_b), (q_b, b, Z, \epsilon, q_b), (q_b, \epsilon, Z_0, \epsilon, q_f)\} \end{aligned}$$
- q_0 est l'état initial.
- $Z_0 \in \Gamma$ est le symbole initial de la pile.

Equivalence des accéptations

Lemme 0.1 *Soit L un langage accepté par pile vide par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$. Alors il existe un automate à pile P' tel que le langage $L^F(P')$ accepté par P' par état final et qui vérifie $L = L^F(P')$.*

Preuve

$\square$

Equivalence des accéptations

Lemme 0.2 *Soit* L *un langage accepté par pile vide par un automate* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$*. Alors il existe un automate à pile* P' *tel que le langage* $L^F(P')$ *accepté par* P' *par état final et qui vérifie* $L = L^F(P')$*.*

Preuve Soit $Z_\perp \notin \Gamma$ un nouveau symbole de pile, et soit $q_s \notin Q$ et $q_f \notin Q$ deux nouveaux états.

$\square$

Equivalence des accéptations

Lemme 0.3 *Soit L un langage accepté par pile vide par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$. Alors il existe un automate à pile P' tel que le langage $L^F(P')$ accepté par P' par état final et qui vérifie $L = L^F(P')$.*

Preuve Soit $Z_\perp \notin \Gamma$ un nouveau symbole de pile, et soit $q_s \notin Q$ et $q_f \notin Q$ deux nouveaux états.
On définit $P' = (Q \cup \{q_s, q_f\}, \Sigma, \Gamma \cup \{Z_\perp\}, \Delta', q_s, Z_0, \{q_f\})$,
avec $\Delta' = \Delta \cup \{(q_s, \epsilon, Z_0, Z_0 Z_\perp, q_0)\} \cup \{(q, \epsilon, Z_\perp, \epsilon, q_f) \mid q \in Q\}$

□

Equivalence des accéptations

Lemme 0.4 *Soit L un langage accepté par pile vide par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$. Alors il existe un automate à pile P' tel que le langage $L^F(P')$ accepté par P' par état final et qui vérifie $L = L^F(P')$.*

Preuve Soit $Z_\perp \notin \Gamma$ un nouveau symbole de pile, et soit $q_s \notin Q$ et $q_f \notin Q$ deux nouveaux états.
On définit $P' = (Q \cup \{q_s, q_f\}, \Sigma, \Gamma \cup \{Z_\perp\}, \Delta', q_s, Z_0, \{q_f\})$,
avec $\Delta' = \Delta \cup \{(q_s, \epsilon, Z_0, Z_0 Z_\perp, q_0)\} \cup \{(q, \epsilon, Z_\perp, \epsilon, q_f) \mid q \in Q\}$
Le symbole $Z_\perp$ en haut de la pile dans une execution de P' represente une pile vide dans une execution de P. $\square$

Equivalence des accéptations

Lemme 0.5 *Soit* L *un langage accepté par état final par un automate* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, Q)$*. Alors il existe un automate à pile* P' *tel que le langage* $L(P')$ *accepté par* P' *par pile vide et qui vérifie* $L = L(P')$*.*

Preuve

$\square$

Equivalence des accéptations

Lemme 0.6 *Soit L un langage accepté par état final par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, Q)$. Alors il existe un automate à pile P' tel que le langage $L(P')$ accepté par P' par pile vide et qui vérifie $L = L(P')$.*

Preuve Soit $q_f \notin Q$ un nouveaux état.

$\square$

Equivalence des accéptations

Lemme 0.7 *Soit L un langage accepté par état final par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, Q)$. Alors il existe un automate à pile P' tel que le langage $L(P')$ accepté par P' par pile vide et qui vérifie $L = L(P')$.*

Preuve Soit $q_f \notin Q$ un nouveaux état.
On définit $P' = (Q \cup \{q_f\}, \Sigma, \Gamma, \Delta', q_0, Z_0)$,
avec
$\Delta' = \Delta \cup \{(q, \epsilon, Z, \epsilon, q_f) \mid q \in F, Z \in \Gamma\} \cup \{(q_f, \epsilon, Z, \epsilon, q_f) \mid Z \in \Gamma\}$
$\square$

Equivalence des accéptations

Lemme 0.8 *Soit L un langage accepté par état final par un automate $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0, Q)$. Alors il existe un automate à pile P' tel que le langage $L(P')$ accepté par P' par pile vide et qui vérifie $L = L(P')$.*

Preuve   Soit $q_f \notin Q$ un nouveaux état.
On définit $P' = (Q \cup \{q_f\}, \Sigma, \Gamma, \Delta', q_0, Z_0)$,
avec
$\Delta' = \Delta \cup \{(q, \epsilon, Z, \epsilon, q_f) \mid q \in F, Z \in \Gamma\} \cup \{(q_f, \epsilon, Z, \epsilon, q_f) \mid Z \in \Gamma\}$
Dès qu'on a atteint un état final, on peut vider la pile. □

Automates à pile déterministes

Définition 0.8 *Un automate à pile* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *est* déterministe *si l'ensemble* Δ *des règles des transitions vérifie la propriété suivante: pour toute paires de règles* $(q, a, Z, \gamma_1, q_1) \in \Delta$ *et* $(q, b, Z, \gamma_2, q_2) \in \Delta$

Automates à pile déterministes

Définition 0.9 *Un automate à pile* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *est* déterministe *si l'ensemble* Δ *des règles des transitions vérifie la propriété suivante: pour toute paires de règles* $(q, a, Z, \gamma_1, q_1) \in \Delta$ *et* $(q, b, Z, \gamma_2, q_2) \in \Delta$

- *si* $a = b$ *alors* $\gamma_1 = \gamma_2$ *et* $q_1 = q_2$*.*

Automates à pile déterministes

Définition 0.10 *Un automate à pile* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *est* déterministe *si l'ensemble* Δ *des règles des transitions vérifie la propriété suivante: pour toute paires de règles* $(q, a, Z, \gamma_1, q_1) \in \Delta$ *et* $(q, b, Z, \gamma_2, q_2) \in \Delta$

- *si* $a = b$ *alors* $\gamma_1 = \gamma_2$ *et* $q_1 = q_2$*.*
- *si* $a \in \Sigma$ *alors* $b \in \Sigma$*.*

Automates à pile déterministes

Définition 0.11 *Un automate à pile $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ est* déterministe *si l'ensemble Δ des règles des transitions vérifie la propriété suivante: pour toute paires de règles $(q, a, Z, \gamma_1, q_1) \in \Delta$ et $(q, b, Z, \gamma_2, q_2) \in \Delta$*

- *si $a = b$ alors $\gamma_1 = \gamma_2$ et $q_1 = q_2$.*
- *si $a \in \Sigma$ alors $b \in \Sigma$.*
- *Les langages acceptés par des automates à pile déterministes sont appelés des langages algébriques déterministes.*

Langages algébriques non-déterministes

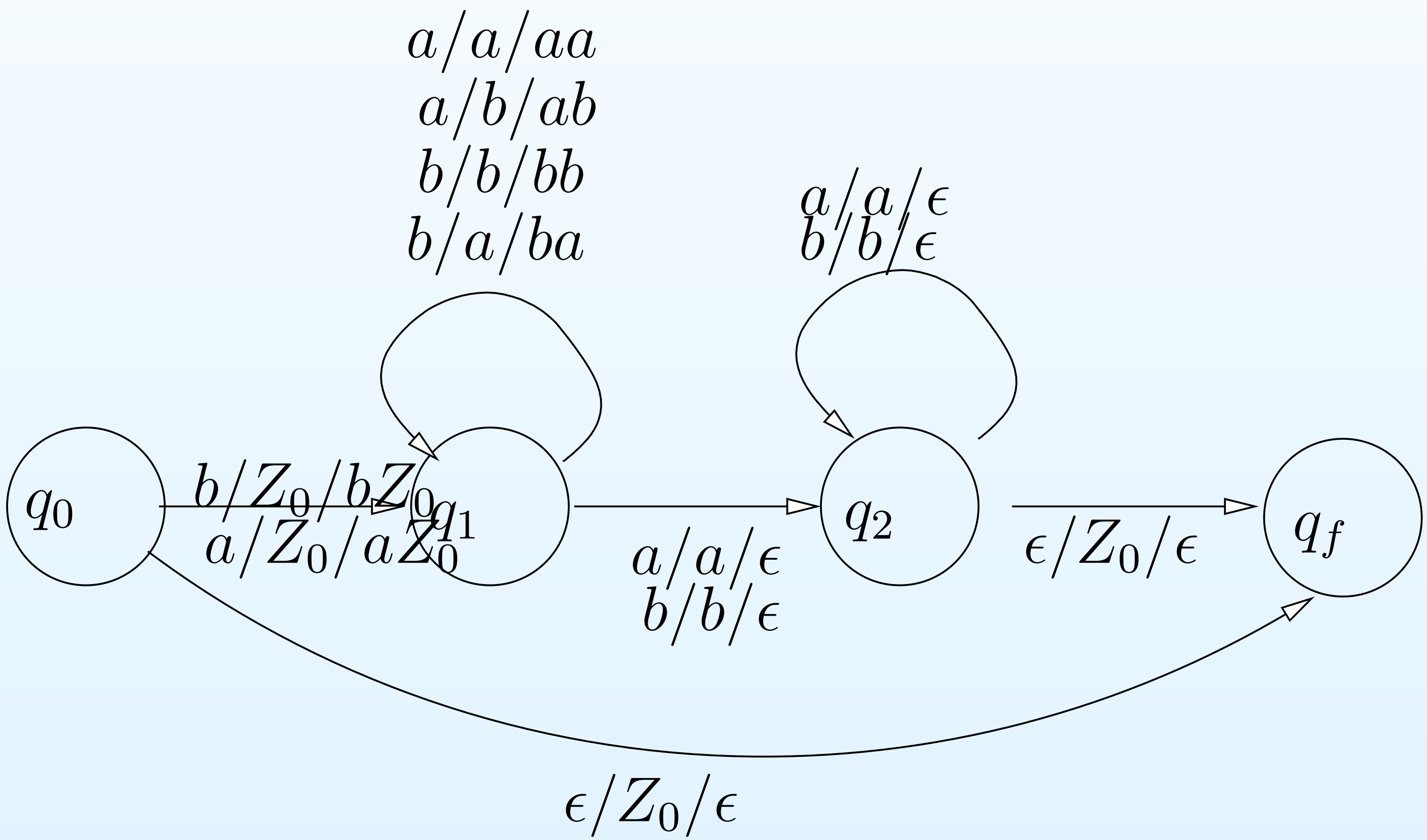
Le langage des palindromes $L = \{w\tilde{w} \mid w \in \{a,b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe.

Langages algébriques non-déterministes

Le langage des palindromes $L = \{w\tilde{w} \mid w \in \{a, b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe.
Mais il est reconnu par l'automate à pile suivant.

Langages algébriques non-déterministes

Le langage des palindromes $L = \{w\tilde{w} \mid w \in \{a,b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe.
Mais il est reconnu par l'automate à pile suivant.



Automates à pile standards

Définition 0.12 *Un automate à pile* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *est* standard *si l'ensemble* Δ *des règles des transitions vérifie la propriété suivante: pour toute règle* $(q, a, Z, \gamma, q') \in \Delta$*, on a* $|\gamma| \leq 2$*.*

Preuve

$\square$

Automates à pile standards

Définition 0.13 *Un automate à pile* $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ *est* standard *si l'ensemble* Δ *des règles des transitions vérifie la propriété suivante: pour toute règle* $(q, a, Z, \gamma, q') \in \Delta$*, on a* $|\gamma| \leq 2$*.*

Theorem 0.7 *A chaque automate à pile* P *on peut associer un automate à pile standard* P' *telle que* $L(P) = L(P')$*.*

Preuve

$\square$

Automates à pile standards

Définition 0.14 *Un automate à pile $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ est* standard *si l'ensemble Δ des règles des transitions vérifie la propriété suivante: pour toute règle $(q, a, Z, \gamma, q') \in \Delta$, on a $|\gamma| \leq 2$.*

Theorem 0.8 *A chaque automate à pile P on peut associer un automate à pile standard P' telle que $L(P) = L(P')$.*

Preuve On itére le processus suivant: pour toute règle $(q, a, Z, Z_1 Z_2 \gamma, q') \in \Delta$ telle que $|\gamma| \geq 1$, on crée un nouvel état q'' et un nouveau symbole de pile T et on remplace dans Δ, la règle $(q, a, Z, Z_1 Z_2 \gamma, q')$ par les règles

$$(q, a, Z, T\gamma, q'') \quad \text{et} \quad (q'', \epsilon, T, Z_1 Z_2, q')$$

$\square$

Automates à pile standards

Définition 0.15 *Un automate à pile $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ est* standard *si l'ensemble Δ des règles des transitions vérifie la propriété suivante: pour toute règle $(q, a, Z, \gamma, q') \in \Delta$, on a $|\gamma| \leq 2$.*

Theorem 0.9 *A chaque automate à pile P on peut associer un automate à pile standard P' telle que $L(P) = L(P')$.*

Preuve On itére le processus suivant: pour toute règle $(q, a, Z, Z_1Z_2\gamma, q') \in \Delta$ telle que $|\gamma| \geq 1$, on crée un nouvel état q'' et un nouveau symbole de pile T et on remplace dans Δ, la règle $(q, a, Z, Z_1Z_2\gamma, q')$ par les règles

$$(q, a, Z, T\gamma, q'') \quad \text{et} \quad (q'', \epsilon, T, Z_1Z_2, q')$$

On a $|Z_1Z_2| = 2$ et $|T\gamma| = |Z_1Z_2\gamma| - 1$ □

Intersection de langages algébriques avec d'états finis

Lemme 0.9 Soit L un langage algébrique et R un langage d'états finis. Alors, $L \cap R$ est algébrique.

Preuve

$\square$

Intersection de langages algébriques avec d'états finis

Lemme 0.10 Soit L un langage algébrique et R un langage d'états finis. Alors, $L \cap R$ est algébrique.

Preuve Soit $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ un automate à pile qui reconnait L et $A = (Q', \Sigma, q_0, \delta, F)$ un automate fini qui reconnait R.

$\square$

Intersection de langages algébriques avec d'états finis

Lemme 0.11 Soit L un langage algébrique et R un langage d'états finis. Alors, $L \cap R$ est algébrique.

Preuve Soit $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ un automate à pile qui reconnait L et $A = (Q', \Sigma, q_0, \delta, F)$ un automate fini qui reconnait R.

On défini l'automate à pile $P' = (Q \times Q', \Sigma, \Gamma, \Delta', q_0, Z_0, Q \times F)$ avec :

- Pour $a \in \Sigma$, on a $((q_1, q_1'), a, Z, \gamma, (q_2, q_2')) \in \Delta'$ ssi $(q_1, a, Z, \gamma, q_2) \in \Delta$ et $\delta(q_1', a) = q_2'$.
- $((q_1, q_1'), \epsilon, Z, \gamma, (q_2, q_2')) \in \Delta'$ ssi $(q_1, \epsilon, Z, \gamma, q_2) \in \Delta$ et $q_1' = q_2'$.

$\square$

Intersection de langages algébriques avec d'états finis

Lemme 0.12 Soit L un langage algébrique et R un langage d'états finis. Alors, $L \cap R$ est algébrique.

Preuve Soit $P = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ un automate à pile qui reconnait L et $A = (Q', \Sigma, q_0, \delta, F)$ un automate fini qui reconnait R.

On défini l'automate à pile $P' = (Q \times Q', \Sigma, \Gamma, \Delta', q_0, Z_0, Q \times F)$ avec :

- Pour $a \in \Sigma$, on a $((q_1, q'_1), a, Z, \gamma, (q_2, q'_2)) \in \Delta'$ ssi $(q_1, a, Z, \gamma, q_2) \in \Delta$ et $\delta(q'_1, a) = q'_2$.
- $((q_1, q'_1), \epsilon, Z, \gamma, (q_2, q'_2)) \in \Delta'$ ssi $(q_1, \epsilon, Z, \gamma, q_2) \in \Delta$ et $q'_1 = q'_2$.

Alors on peut montrer que $L(P') = L(P) \cap L(A)$ $\square$