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 $\boldsymbol{Z}$.
Nous pouvons alors définir des expressions à partir de variables et de constantes.
L'ensemble des expressions, dénoté par Exp, est défini inductivement de la manière suivante :

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

Expressions arithmétiques

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

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

Sémantique des expressions arithmétiques

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

Sémantique des expressions arithmétiques

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

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

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

$$\uparrow$$

$$x \quad x_0$$

$$y \quad y_0$$

$$z \quad z_0$$

Un exemple

Exemple :

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

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

$$\begin{array}{lll}
 & & \uparrow \\
x & x_0 & x_0 \\
y & y_0 & y_0 \\
z & z_0 & 0
\end{array}$$

Un exemple

Exemple :

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

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

$$\begin{array}{lll} & & \uparrow \\ x & x_0 & 2 * x_0 \\ y & y_0 & y_0 \\ z & z_0 & 0 \end{array}$$

Un exemple

Exemple :

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

Un exemple

Exemple :

$$
\begin{array}{llll}
x,y,z \ : \ \boldsymbol{Z}; & & & \\
 & z := 0; \quad \text{while } z < y \text{ do } x := x * 2; & z := z + 1 \text{ od} & \\
 & & \uparrow & \\
x & x_0 & 4 * x_0 & \\
y & y_0 & y_0 & \\
z & z_0 & 1 &
\end{array}
$$

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

Un exemple

Exemple :

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

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

$$\uparrow$$

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

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

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

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

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.

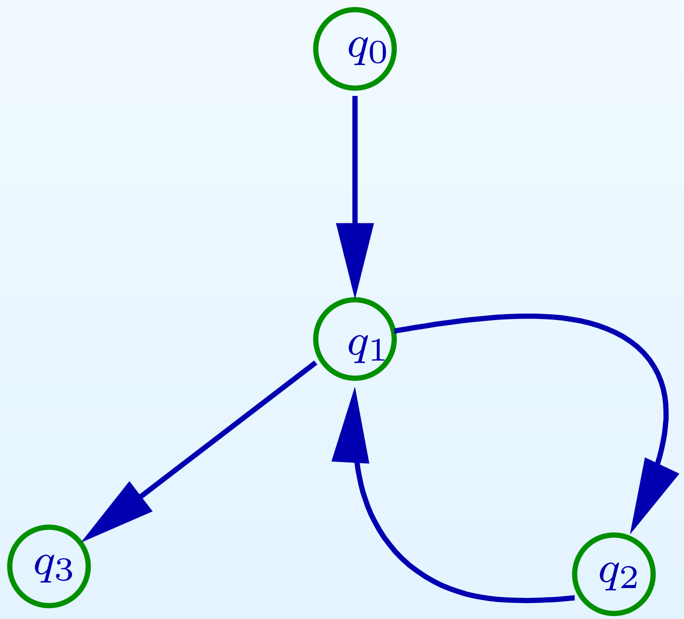
Le programme sous forme d'automate étendu

Le flux de contrôl

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

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

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



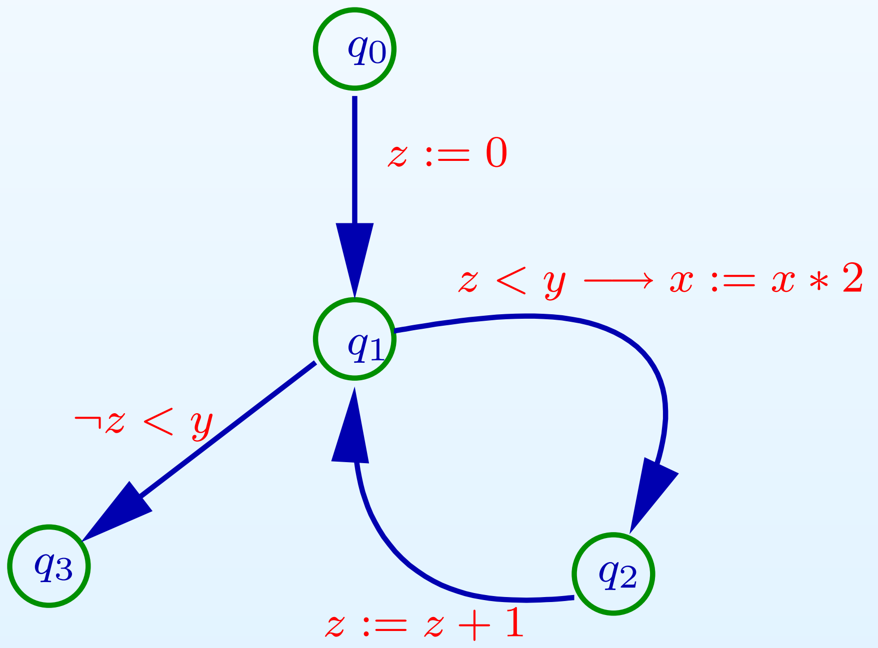
Le programme sous forme d'automate étendu

Le flux de contrôl

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

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

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



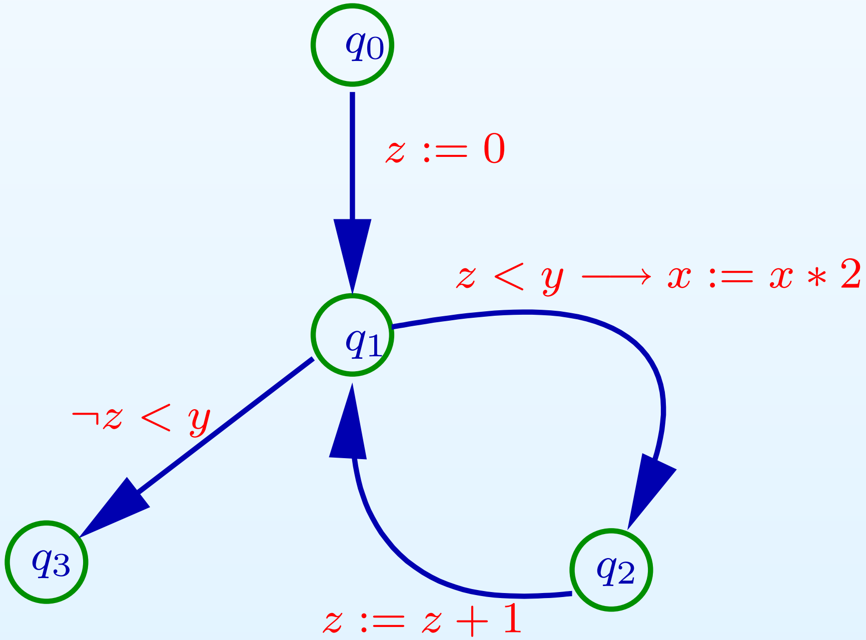
Le programme sous forme d'automate étendu

Le flux de contrôl

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

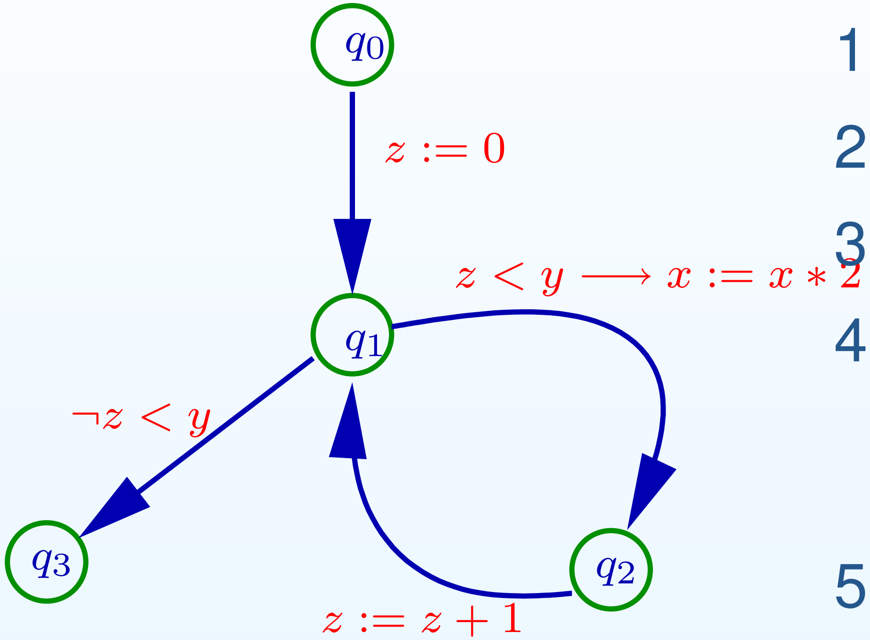
$z := 0;$ while $z < y$ do $x := x * 2;$ $z := z + 1$ od

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



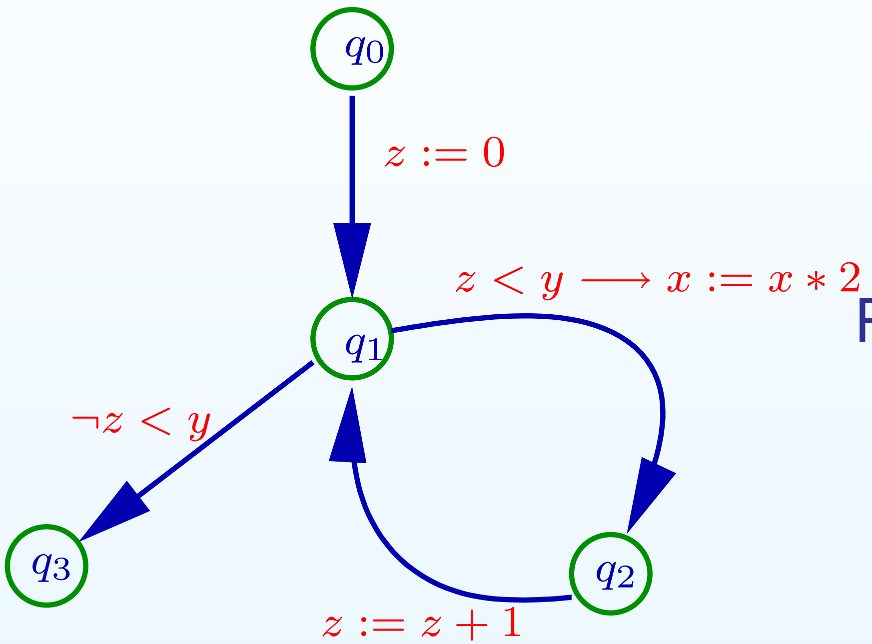
q_0, q_1, q_2, q_3 sont appelés état de contrôle. $(q_0, z := 0, q_1)$ est une transition.

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 $(q, g \to x := e, q')$ où g est un prédicat appelé garde. Uniquement des variables déclarées apparaissent dans la garde g et dans l'affectation $x := e$.
- $Q_t \subseteq Q$ est l'ensemble des états de contrôle terminaux (finals).

Etats et configuration

- Les valeurs des variables à chaque instant de l'exécution sont données par un état. Il faut distinguer entre état et état de contrôle. Un état $\sigma : \mathcal{X} \longrightarrow \boldsymbol{Z}$ 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

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

od

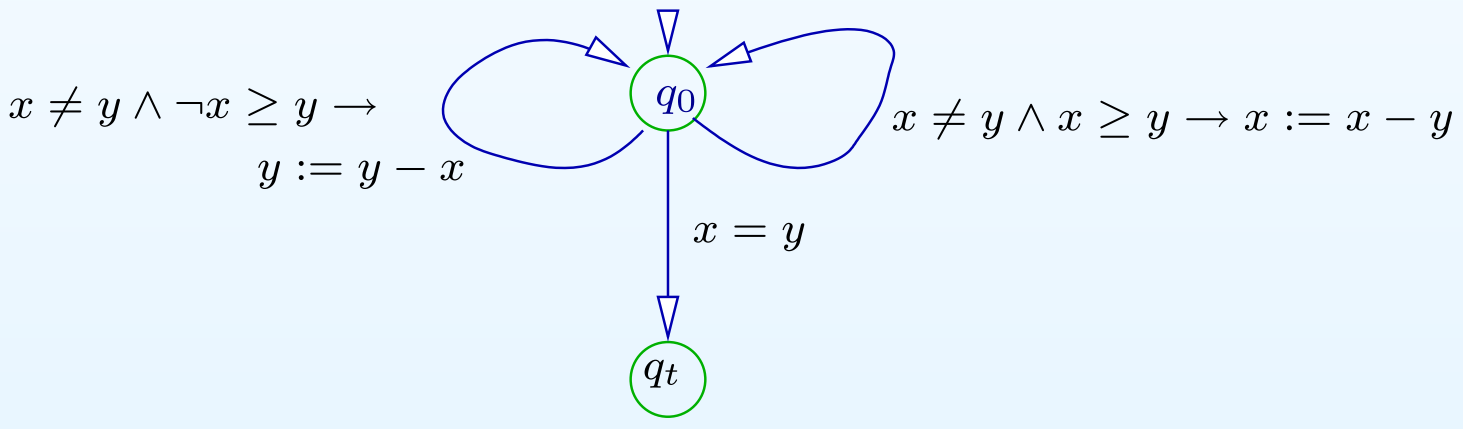
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

od



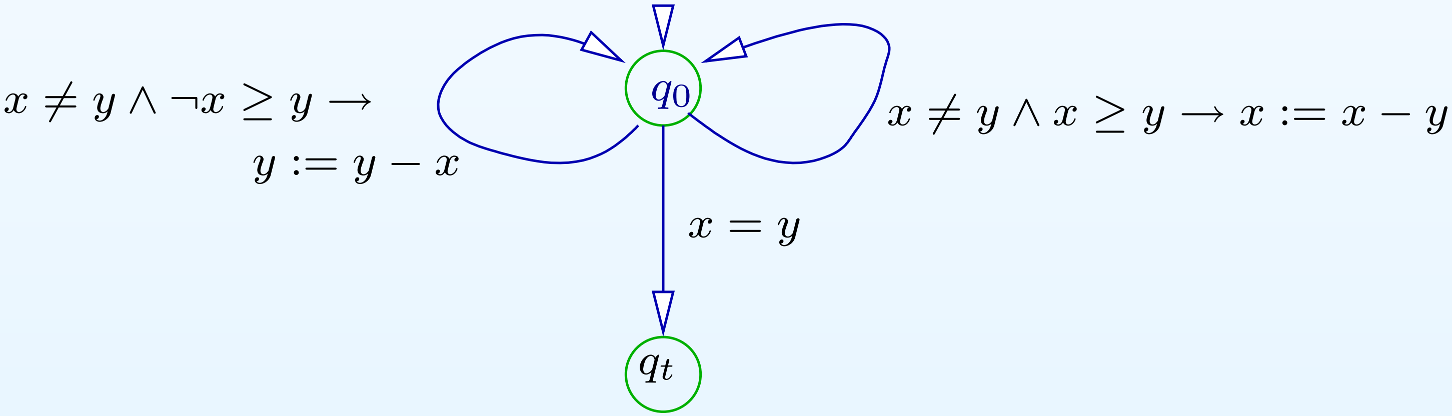
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

od



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

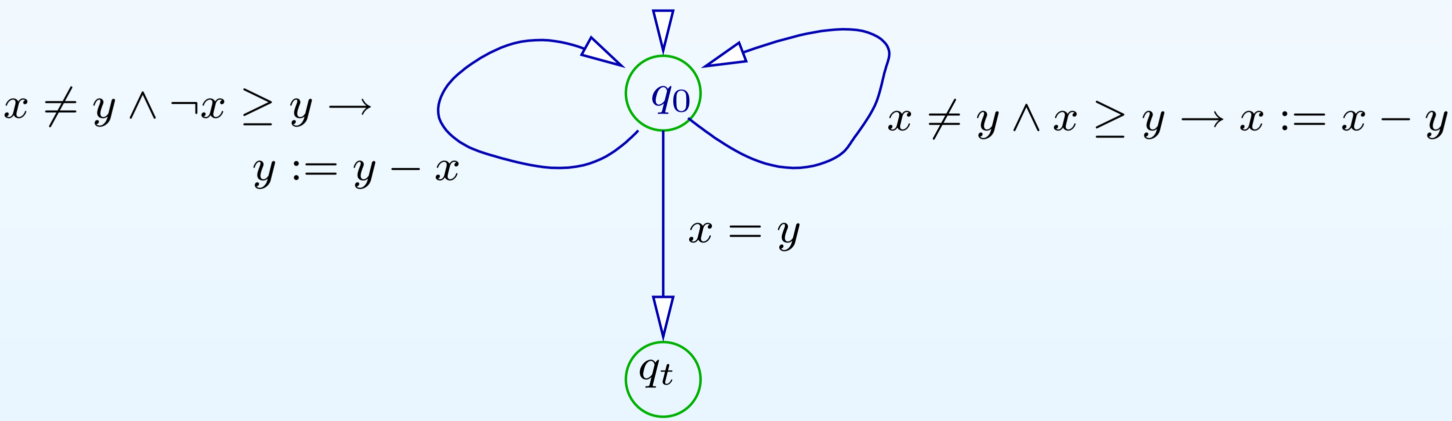
Exemples d'automates étendus-1

$x, y : \boldsymbol{Z}$

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

else $y := y - x$ fi

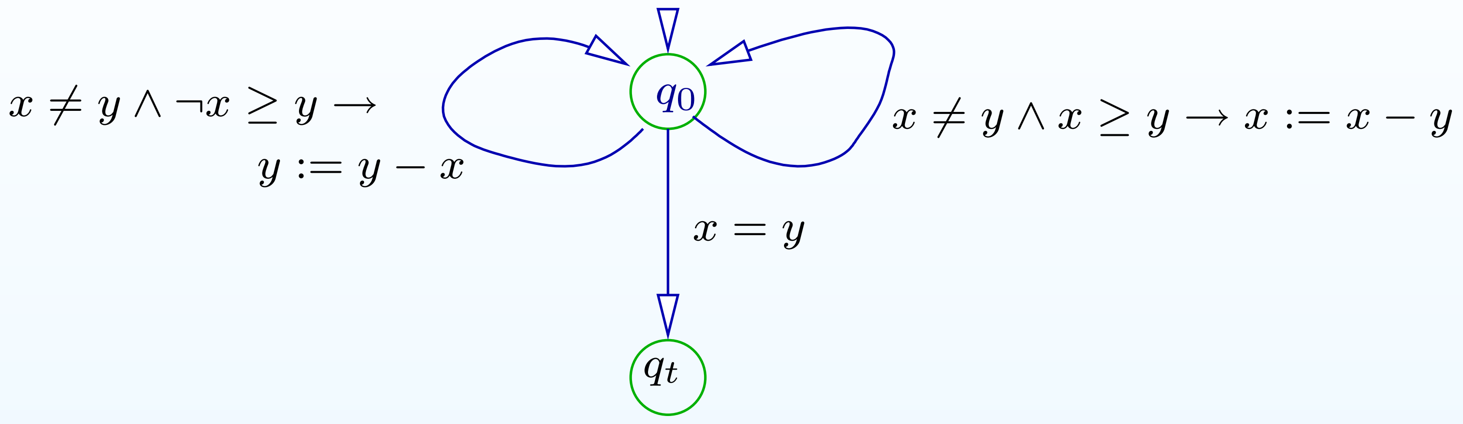
od



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



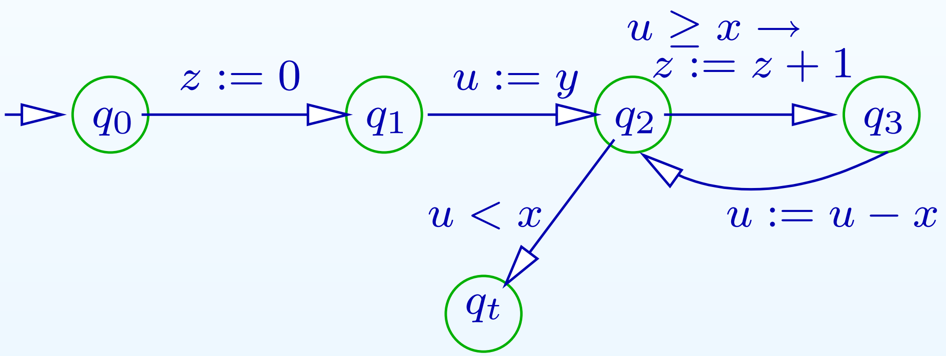
Développer (au tableau) les traces d’exècution avec une configuration initiale qui satisfait :

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

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

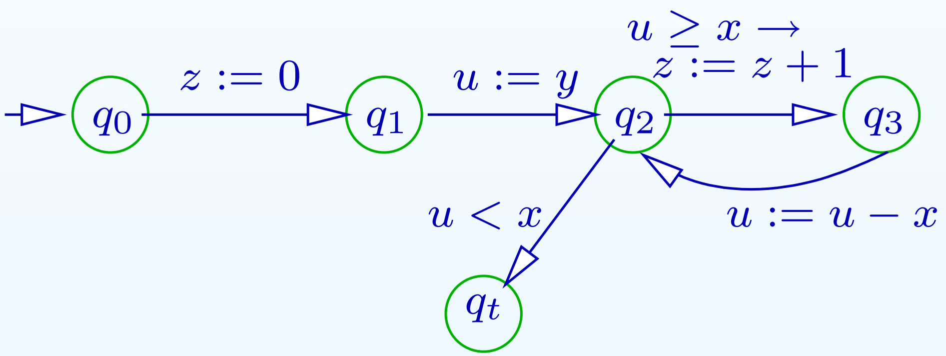


Pre-condition : $x > 0 \wedge y \geq 0$.

Exemples d'automates étendus-2

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

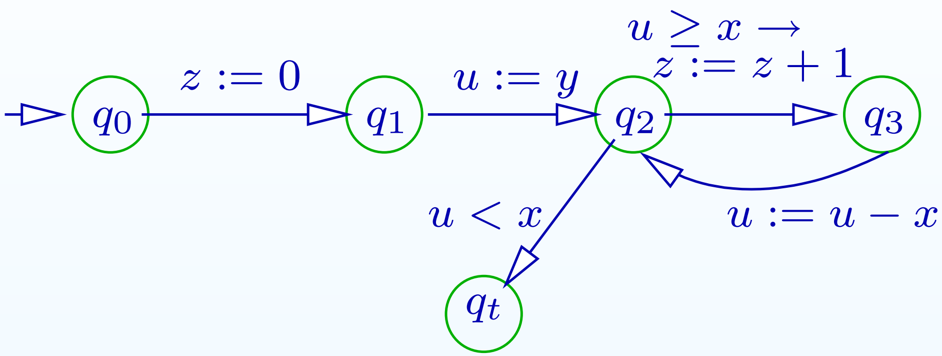
$z := 0; u := y;$ while $u \geq x$ do $z := z + 1; u := u - x$ od



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

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

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

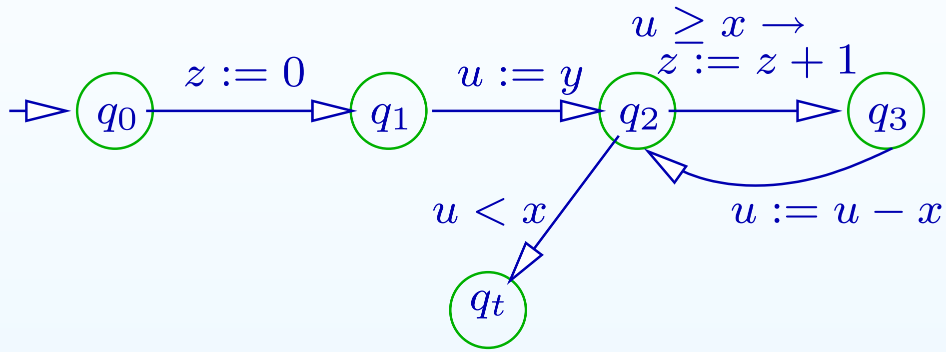
$$\sigma'[\sigma(x)/x_0, \cdots, \sigma(y)/y_0] \models Q$$

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).

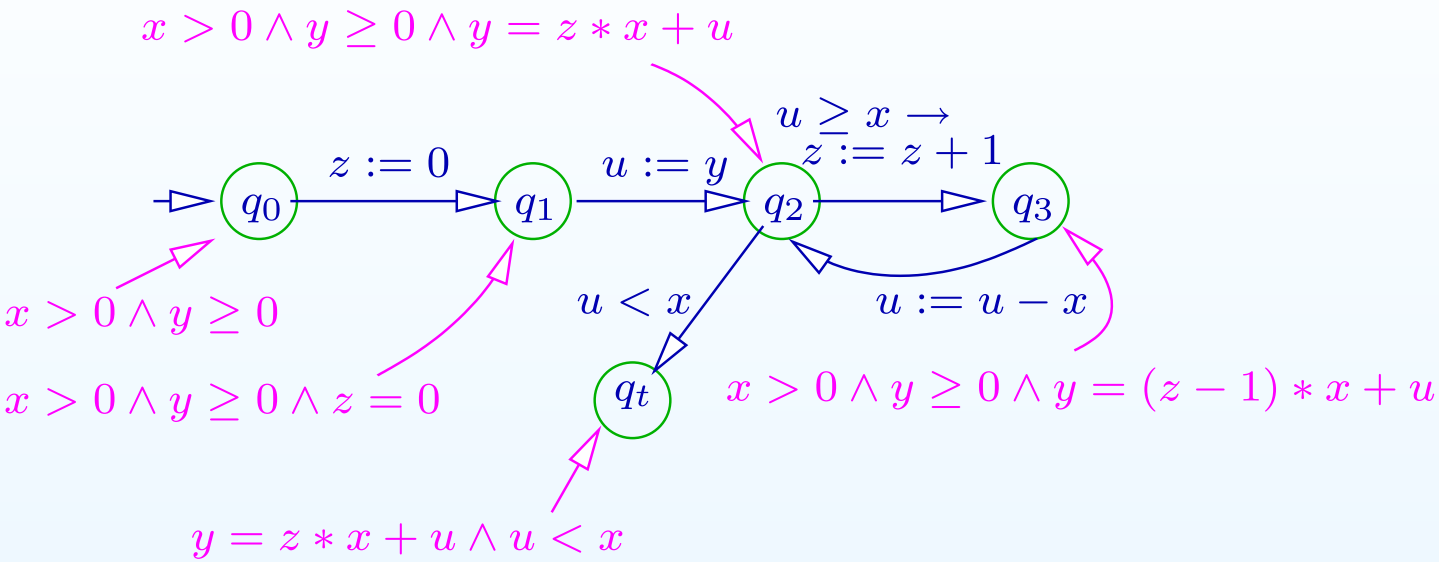
Méthode de vérification

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.

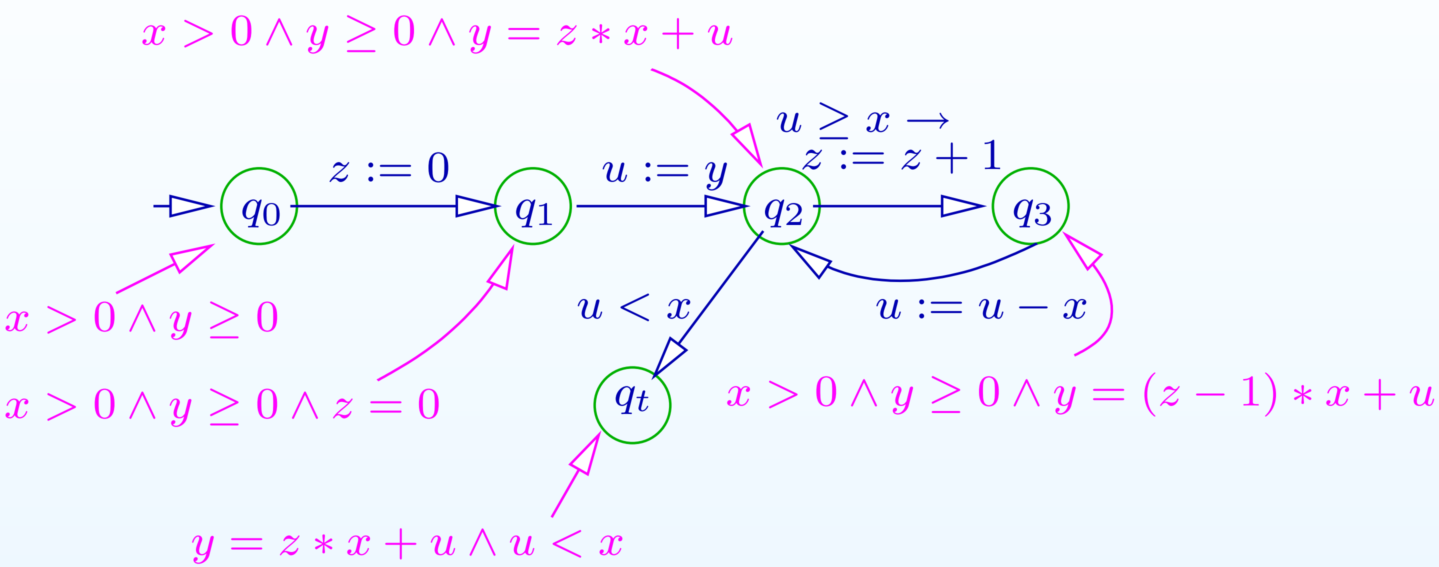
L'automate Div annoté



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 $(q, g \rightarrow x := e, q')$
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 $\sigma[\llbracket e \rrbracket\sigma/x] \models P_{q'}$.

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.

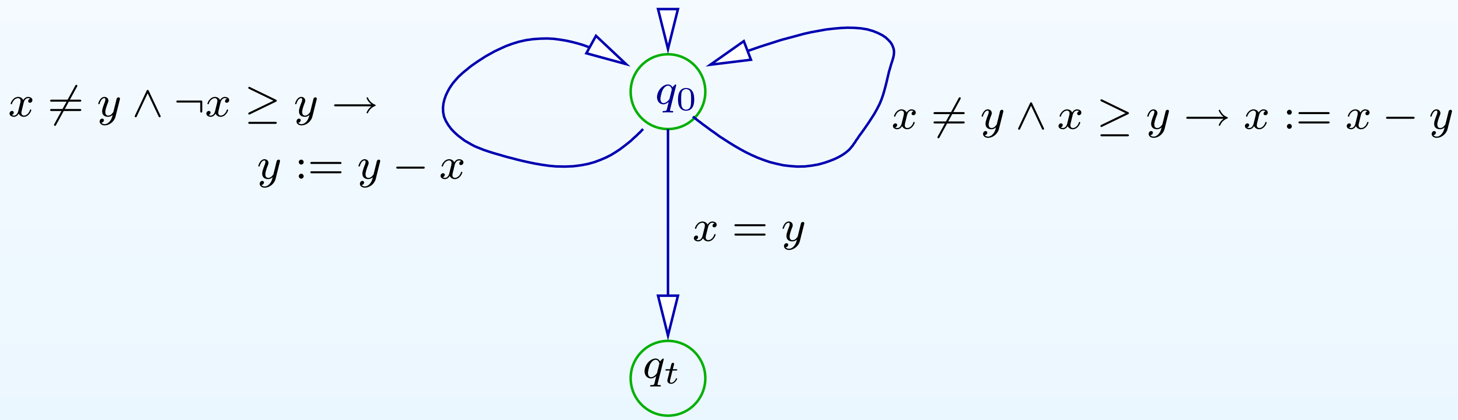
Automates étendus, invariants de transition et términaison de programmes

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 $\forall i \in \mathbb{N},\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(\mathbb{N}, >)$ 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 $\forall i \in \mathbb{N},\ (a_i, a_{i+1}) \in A$.
Exemples:

- $(\mathbb{N}, >)$ est bien-fondée.
- $(\mathbb{N}, <)$ n'est pas bien-fondée.
- $(\mathcal{P}(\mathbb{N}), \subset)$ et $(\mathcal{P}(\mathbb{N}), \supset)$ ne sont pas bien-fondées.
- $(\mathcal{P}_{fin}(\mathbb{N}), \supset)$ est bien-fondée.
- $(\mathcal{P}_{fin}(\mathbb{N}), \subset)$ n'est pas bien-fondée.
- $(\mathbb{Z}, <)$ et $(\mathbb{Z}, >)$ ne sont pas bien-fondées.
- $(\mathbb{R}, <)$ et $(\mathbb{R}, >)$ 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 $\text{Acc}(A, \sigma_0)$, est l'ensemble de configurations (q, σ), tel qu'il existe une trace finie qui méne à (q, σ), $(q_0, \sigma_0) \longrightarrow \cdots \longrightarrow (q, \sigma)$, formellement,

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

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 $\mathsf{Acc}(A, \sigma_0)$, 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 contient la relation de transition $\rightarrow$ et est fermée à la composition avec $\rightarrow$, restreinte aux état accesibles de (q_0, σ_0), formellement,
Condition (ind):

$$(\rightarrow |_{\mathsf{Acc}_{(A,\sigma_0)}} \cup (T|_{\mathsf{Acc}_{(A,\sigma_0)}} \circ \rightarrow |_{\mathsf{Acc}_{(A,\sigma_0)}})) \subseteq T$$

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 contient la relation de transition $\rightarrow$ et est fermée à la composition avec $\rightarrow$, restreinte aux état accesibles de (q_0, σ_0), formellement,
Condition (ind):

$$(\rightarrow |_{\mathsf{Acc}_{(A,\sigma_0)}} \cup (T|_{\mathsf{Acc}_{(A,\sigma_0)}} \circ \rightarrow |_{\mathsf{Acc}_{(A,\sigma_0)}})) \subseteq T$$

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. Si $\sigma_0 \models P_{q_0}$, alors pour vérifier (ind) il suffit de vérifier:

Condition (ind_s):
Pour toute transition $(q, g \to x := e, q')$
Pour tout état $\sigma \in \Sigma$,
si $\sigma \models g \wedge P_q$ alors

-

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. Si $\sigma_0 \models P_{q_0}$, alors pour vérifier (ind) il suffit de vérifier:

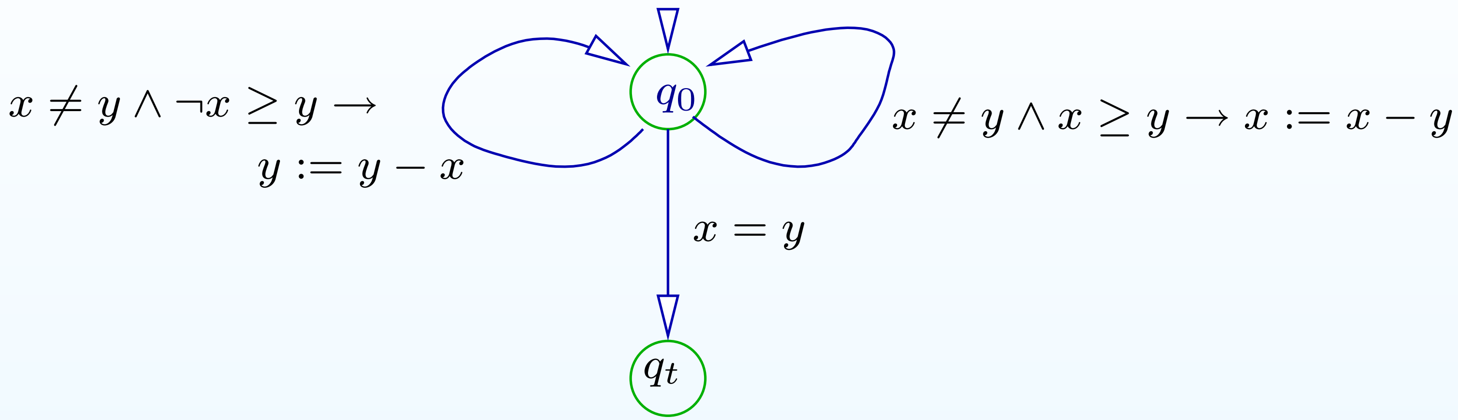
- Condition (ind_s):
  Pour toute transition $(q, g \rightarrow x := e, q')$
  Pour tout état $\sigma \in \Sigma$,
  si $\sigma \models g \wedge P_q$ alors
  - $((q, \sigma), (q', \sigma[\llbracket e \rrbracket \sigma / x])) \in T$.

Terminaison d'automates - outils(II)

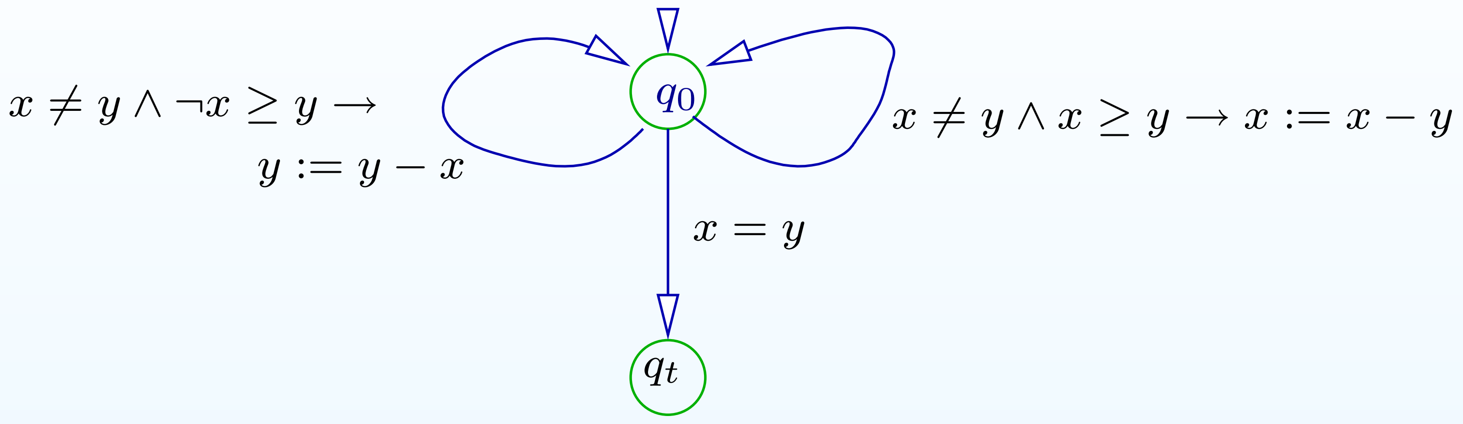
Soit $A = (D, Q, Q_0, \mathcal{T}, Q_t)$ un automate étendu et $(P_q)_{q \in Q}$ une annotation inductive de A. Si $\sigma_0 \models P_{q_0}$, alors pour vérifier (ind) il suffit de vérifier:

- Condition (ind_s):
  Pour toute transition $(q, g \rightarrow x := e, q')$
  Pour tout état $\sigma \in \Sigma$,
  si $\sigma \models g \wedge P_q$ alors
  - $((q, \sigma), (q', \sigma[\llbracket e \rrbracket \sigma / x])) \in T$.
  - Pour tout pair $((r, \sigma_r), (q, \sigma)) \in T$, si $\sigma_r \models P_r$ alors $((r, \sigma_r), (q', \sigma[\llbracket e \rrbracket \sigma / x])) \in T$.

Exemples terminaison d’automates étendus-1

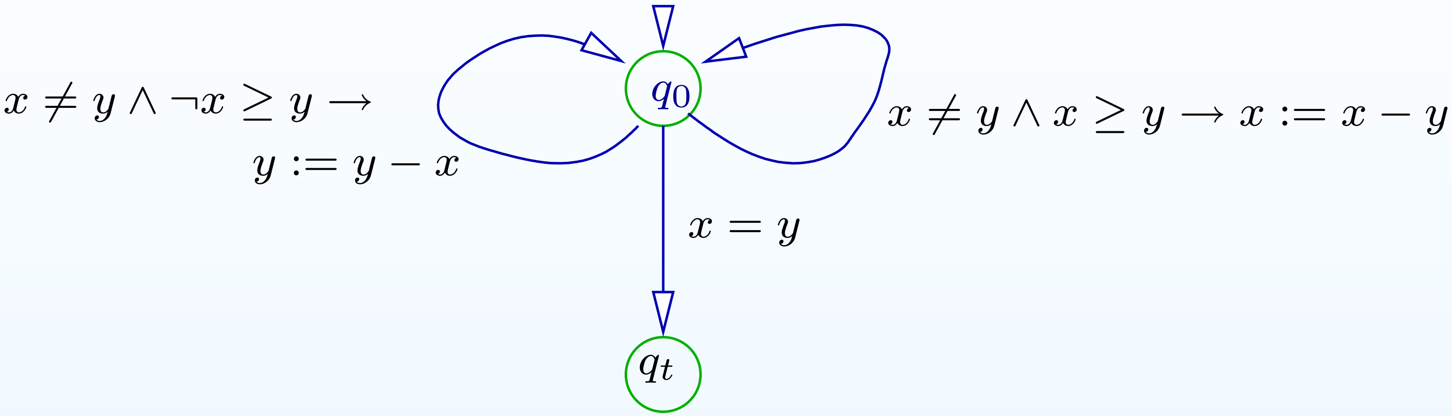


Exemples terminaison d'automates étendus-1



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

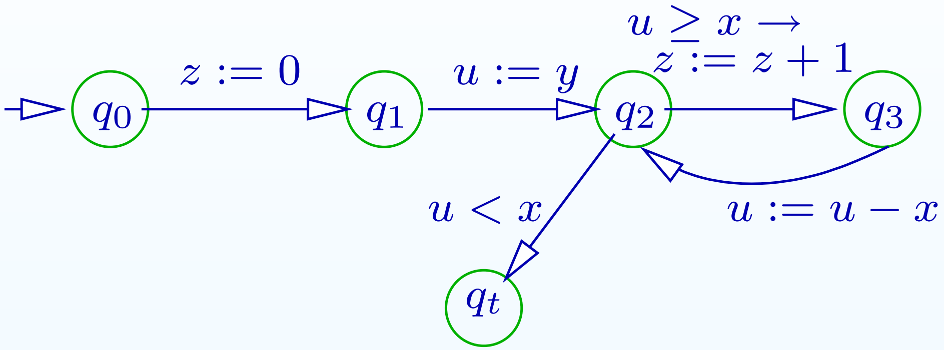
Exemples terminaison d’automates étendus-1



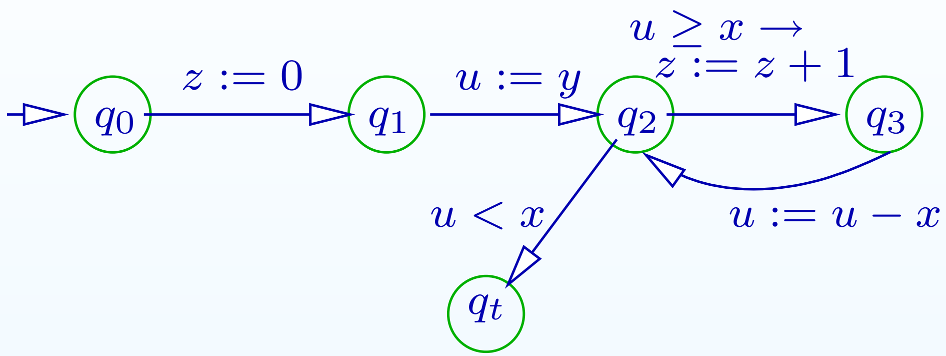
Etat initial des variables : $\sigma_0(x) > 0 \wedge \sigma_0(y) > 0$.
Invariant :

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

Exemples terminaison d'automates étendus-2

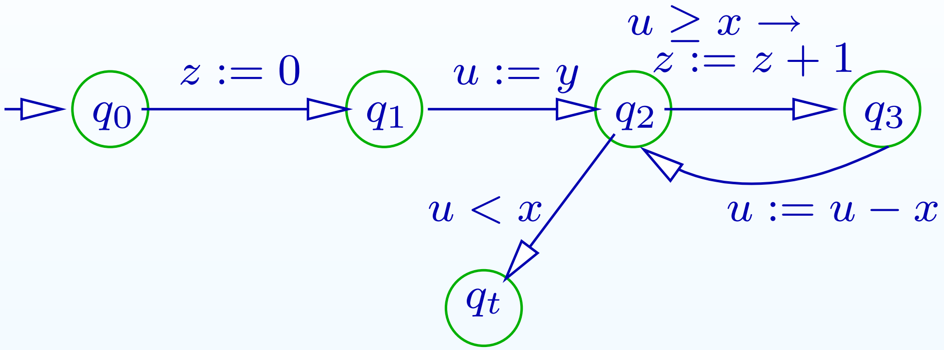


Exemples terminaison d'automates étendus-2



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 :

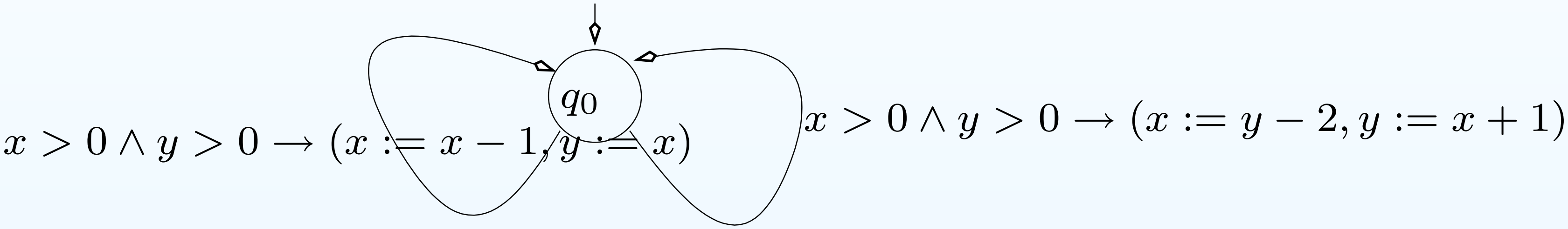
$$
\begin{aligned}
T = \quad & \{((q_0, \sigma), (q, \sigma'))|q \in \{q_1, q_2, q_3, q_t\}\} \cup \\
& \{((q_1, \sigma), (q, \sigma'))|q \in \{q_2, q_3, q_t\}\} \cup \\
& \{((q_2, \sigma), (q_t, \sigma'))\} \cup \{((q_3, \sigma), (q_t, \sigma'))\} \cup \\
& \{((q_3, \sigma), (q_2, \sigma')) \mid \sigma'(u)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

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...

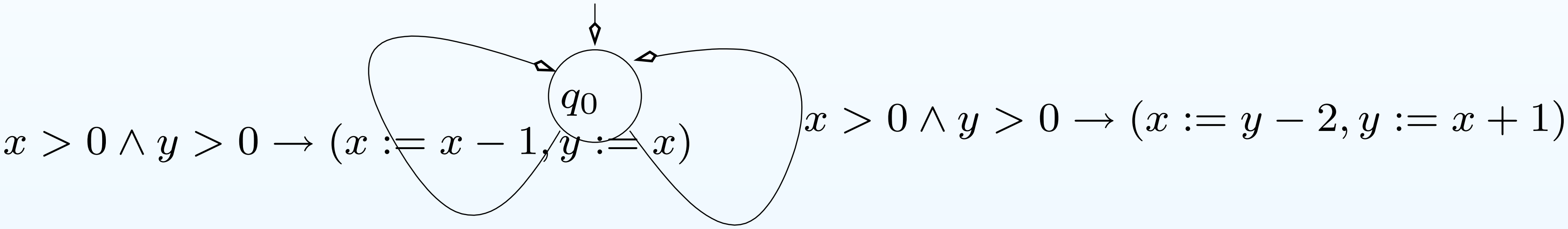
Exemples terminaison d'automates étendus-3

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)

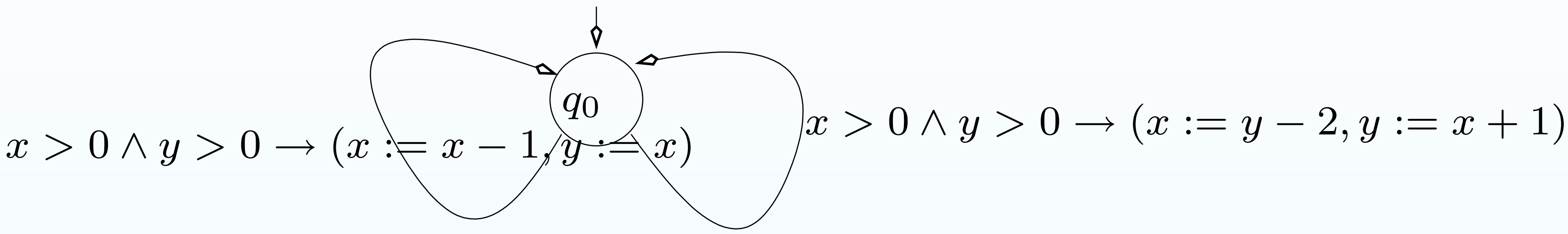
Une relation $R \subseteq A \times A$ est dite *union-bien-fondée* s'il existe un nombre fini de relations bien-fondées $T_1, \ldots, T_n$, telles que $T = T_1 \cup \ldots \cup T_n$.

Terminaison d’automates - outils(III)

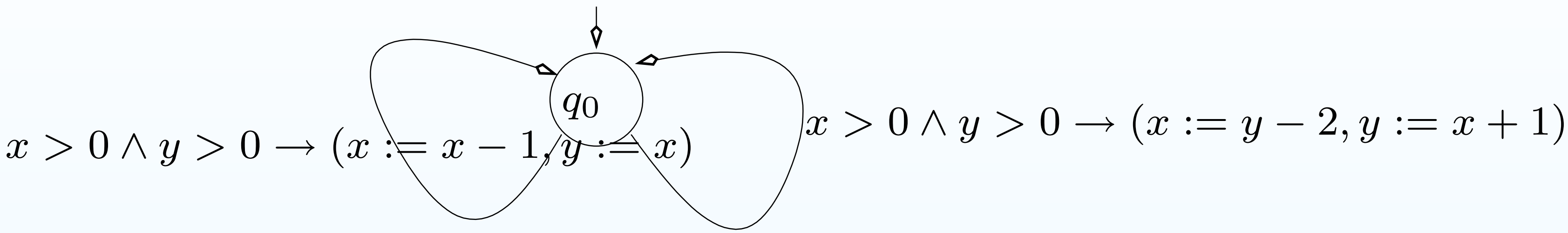
Une relation $R \subseteq A \times A$ est dite *union-bien-fondée* s’il existe un nombre fini de relations bien-fondées $T_1, \ldots, T_n$, telles que $T = T_1 \cup \ldots \cup T_n$.

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



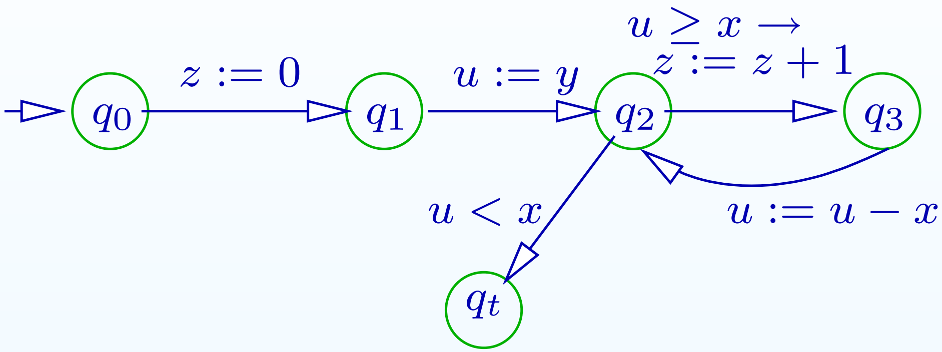
Exemples terminaison d'automates étendus-4



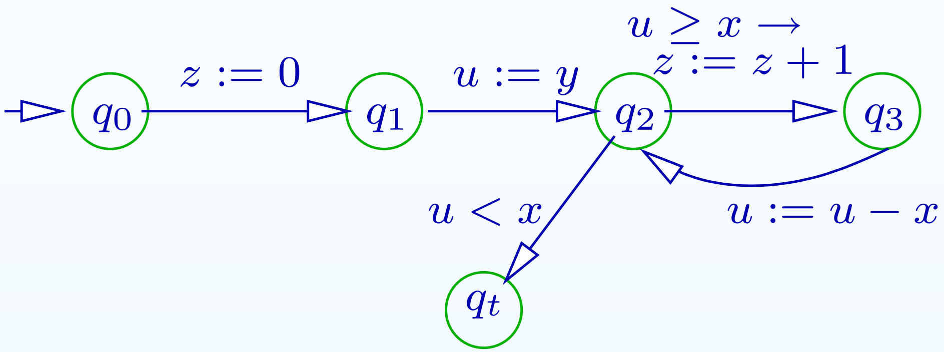
Invariant :

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

Exemples terminaison d'automates étendus-5

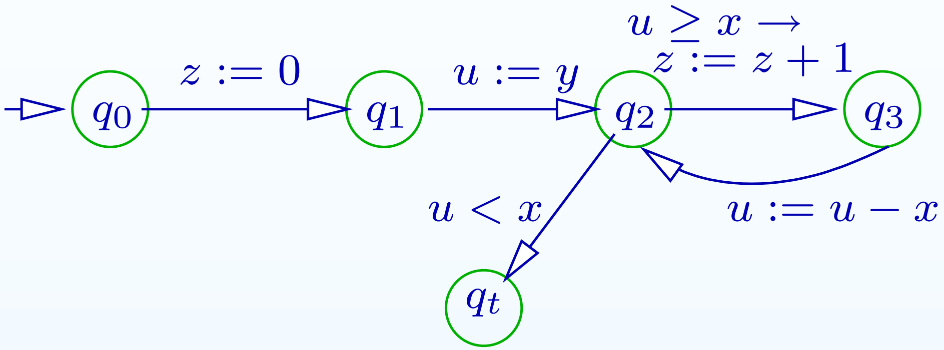


Exemples terminaison d'automates étendus-5



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 totale

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 $P_q \Rightarrow g_1 \vee \ldots \vee g_m$

Modèles de calcul

Motivation

Comprendre les limites de l'informatique.

- Que veut dire qu'une fonction soit calculable ou qu'un problème soit soluble par des algorithmes.
- Existe-il des problèmes insolubles par des algorithmes.
- Peut-on avoir des réponses à ces questions indépendantes :
  - du langage de programmation
  - de l'ordinateur sur lequel les programmes sont exécutés

Plusieurs modèles de calcul

- Mémoire finie : Automates d'états finis, expressions règulière.
- Mémoire finie + pile : Automates à pile
- Mémoire infinie :
  - Machines de Turing (Alan Turing)
  - Systèmes de Post (Emil Post)
  - Fonctions μ-récursives (Kurt Gödel, Jacques Herbrand)
  - λ-Calcul (Alonzo Church, Stephen C. Kleene)
  - Logique des combinateurs (Moses Schönfinkel, Haskell B. Curry)

Problèmes et Langages

- Quels problèmes sont solubles par un programme exécuté sur un ordinateur?

Il faut préciser :

- la notion de problème
  - Fonction de $\mathbb{N}^k \longrightarrow \mathbb{N}$ ou
  - Langage :
    - Alphabet : les symboles pour décrire les instances du problème
    - Le langage qui décrit les instances positives
- la notion de programme exécuté sur un ordinateur : Machine de Turing

Exemples de Problèmes

- - **Entrée :** le codage binaire $\hat{n}$ d'un entier naturel $n \in I\!N$
  - **Sortie :** n est pair

  Formalisation
  - $\Sigma = \{0, 1\}$
  - $P = \{u \in \Sigma^* \mid \exists n \in I\!N \cdot \hat{n} = u \wedge n \equiv_2 0\}$.
- Il existe un programme avec une mémoire fini pour résoudre ce problème : P est un langage régulier.

Exemples de Problèmes

- - **Entrée :** le codage binaire $\hat{n}$ d'un entier naturel $n \in I\!N$
  - **Sortie :** n est un nombre premier

  Formalisation
  - $\Sigma = \{0, 1\}$
  - $P = \{u \in \Sigma^* \mid \exists n \in I\!N \cdot \hat{n} = u \wedge n$ premier$\}$.
  - Il n'existe pas de programme avec une mémoire fini pour résoudre ce problème : P n'est pas régulier.
  - Par contre, il existe un programme avec une mémoire infinie pour résoudre ce problème.

Exemples de Problèmes

- ◦ **Entrée :** une chaine de caractères $\hat{\pi}$ représentant un C-programme π
  ◦ **Sortie :** π s'arrête sur l'entrée 0

  Formalisation

  ◦ Σ l'alphabet du langage C
  ◦ $P = \{\pi \in \Sigma^* \mid \exists \pi \cdot$
  π est un programme C et l'exécution de π sur 0 s'arrête$\}$.
  ◦ Il n'existe pas de programme même avec une mémoire infinie pour résoudre ce problème.

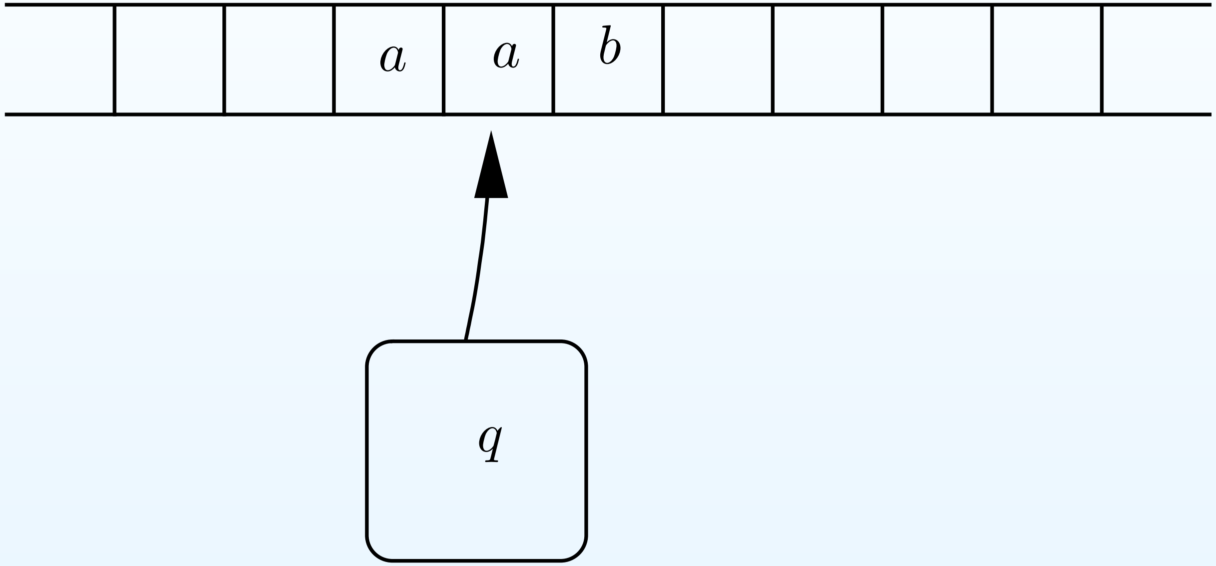
Machines de Turing

Alan Turing inventa cette machine abstraite en 1936 pour définir la notion de "fonction calculable".

- toute tâche exécutée par une machine de Turing MT peut l'être sur un ordinateur ... et vice-versa !
- peut simuler n'importe quel automate fini, automates à pile et même n'importe quel programme exécutable sur un ordinateur
- il existe une MT universelle capable de simuler toutes les autres MT : programme comme donnée

Machines de Turing

- Une MT est composée d'une unité de contrôle, d'un ruban infini divisé en cases et d'une tête de lecture/ écriture.



- en fonction de ce qu'elle lit sur la case courante et de son état courant (conformément à sa relation de transition δ), elle
  1. écrit un symbole sur sa case courante
  2. déplace la tête de lecture/ écriture d'une case (à droite, à gauche)
  3. change d'état.

Machines de Turing

Une *Machine de Turing* M est donnée par

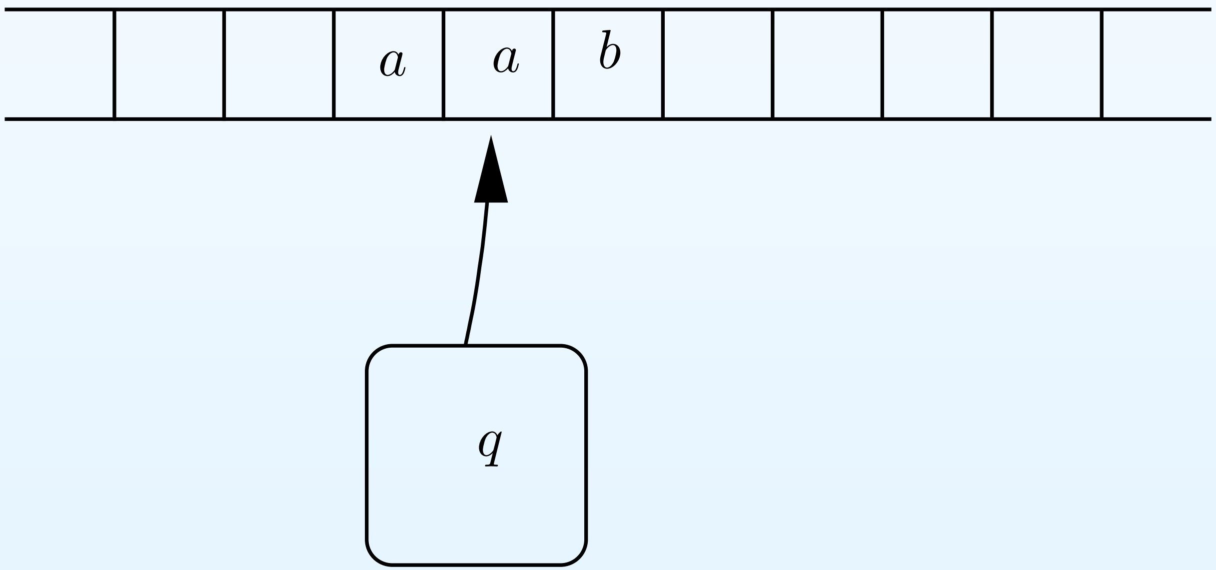
$$(Q, \Sigma, \Gamma, \delta, q_0, F) \text{ où :}$$

- Q est un ensemble fini d'états.
- Σ est l'alphabet (fini) d'entrée
- Γ est l'alphabet (fini) du ruban, tel que $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{G, D, I\}$ est la fonction de transition.
- q_0 est l'état initial.
- $F \subseteq Q$ est l'ensemble des états accepteurs.
- On suppose qu'aucune transition n'est définie dans les états accépteurs.
- On considère un symbole $B \in \Gamma \setminus \Sigma$ particulier dit blanc.

Configuration

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

- q est l'état courant.
- u est le mot qui est à gauche de la tête de lecture/ écriture.
- v est le mot qui est à droite de la tête de lecture/ écriture.



est représentée par (a, q, ab)

Relation de Transition

La machine M réalise une transition d'une configuration (u, q, v) vers une configuration (u', q', v'), noté

$$(u, q, v) \rightarrow (u', q', v'),$$

si une des conditions suivantes est satisfaite :

- $v = av''$, $\delta(q, a) = (q', b, G)$ et $u = u'c$, $v' = cbv''$
- $v = av''$, $\delta(q, a) = (q', b, d)$ et $u' = ub$, $v' = v''$
- $\delta(q, a) = (q', b, I)$ et $u' = u$, $v' = bv''$

Langage reconnu

- Une configuration initiale est de la forme (ϵ, q_0, u).
- Une configuration finale est de la forme (u, q, v) avec $q \in F$.
- Un mot u est accepté par M, s'il existe $q \in F$ avec :

$$(\epsilon, q_0, u) \to^* (u', q, v').$$

- Le langage reconnu par M est

$$L(M) = \{u \mid \exists q \in F \cdot (\epsilon, q_0, u) \to^* (u', q, v')\}.$$

Exemple

Le langage $L = \{a^i b^i \mid i \geq 0\}$ est reconnu par la Machine de Turing définie par $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$.
- $\Sigma = \{a, b\}$.
- $\Gamma = \{a, b, X, Y, \square\}$.
- $$\begin{array}{lll} \delta(q_0,\ a) = (q_1, X, D) & \delta(q_0,\ b) = (q_2, Y, G) & \delta(q_1,\ a) \ = \ (q_1, a, D) \\ \delta(q_1,\ b) = (q_2, Y, G) & \delta(q_1,\ Y) = (q_1, Y, D) & \delta(q_2,\ a) \ = \ (g_2, a, G) \\ \delta(q_2,\ X) = (q_0, X, D) & \delta(q_2,\ Y) = (q_2, Y, G) & \delta(q_3,\ Y) = (q_3, Y, D) \\ \delta(q_3,\ B) = (q_4, B, D) & & \end{array}$$
- q_0 est l'état initial.
- $F = \{q_4\}$

Exemple (2)

Le langage $\{\, a^n b^n c^n \ :\ n \geq 1 \,\}$ est reconnu par la Machine de Turing définie par $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ où :
$Q = \{q_0,\, q_1,\, q_2,\, q_3,\, q_4,\, q_5,\, q_6\}$, $\Sigma = \{a,\, b,\, c\}$,
$\Gamma = \{a,\, b,\, c,\, X,\, Y,\, Z,\, \Box\}$, $F = \{q_6\}$, et δ est définit par

$$
\begin{array}{lll}
\delta(q_0,\, a) = (q_1, X, D) & \delta(q_1,\, a) = (q_1, a, D) & \delta(q_1,\, y) = (q_1, Y, D) \\
\delta(q_1,\, b) = (q_2, Y, D) & \delta(q_2,\, b) = (q_2, b, D) & \delta(q_2,\, z) = (q_2, Z, D) \\
\delta(q_2,\, c) = (q_3, Z, G) & \delta(q_3,\, z) = (q_3, Z, G) & \delta(q_3,\, b) = (q_3, b, G) \\
\delta(q_3,\, y) = (q_3, Y, G) & \delta(q_3,\, a) = (q_3, a, G) & \delta(q_3,\, x) = (q_0, X, D) \\
\delta(q_0,\, y) = (q_4, Y, D) & \delta(q_4,\, y) = (q_4, Y, D) & \delta(q_4,\, z) = (q_5, Z, D) \\
\delta(q_5,\, z) = (q_5, Z, D) & \delta(q_5,\, \Box) = (q_6, \Box, D) &
\end{array}
$$

Excution

Une exécution de la machine, qui permet d'accepter le mot *aabbcc,*:

$q_0aabbcc \longrightarrow xq_1abbcc \longrightarrow xaq_1bbcc \longrightarrow xayq_2bcc \longrightarrow xaybq_2cc \longrightarrow$
$xayq_3bzc \longrightarrow$
$xaq_3ybzc \longrightarrow xq_3aybzc \longrightarrow q_3xaybzc \longrightarrow xq_0aybzc \longrightarrow xxq_1ybzc \longrightarrow$
$xxyq_1bxc \longrightarrow$
$xxyyq_2zc \longrightarrow xxyyzq_2c \longrightarrow xxyyq_3zz \longrightarrow xxyq_3yzz \longrightarrow xxq_3yyzz \longrightarrow$
$xq_3xyyzz \longrightarrow$
$xxq_0yyzz \longrightarrow xxyq_4yzz \longrightarrow xxyyq_4zz \longrightarrow xxyyzq_5z \longrightarrow$
$xxyyzzq_5\Box \longrightarrow xxyyzz\Box q_6\Box.$

Les sorties possibles d'une machine de Turing

Soit M une machine de Turing.
On dit que M **s'arrête pour l'entrée** $u \in \Sigma^*$, s'il existe une configuration (v, q, w) telle que $(\epsilon, q_0, u) \rightarrow^* (v, q, aw)$ et $\delta(q, a)$ n'est pas définit.
C.a.d. M atteint une configuration où aucune transition n'est possible.
Pour une entrée u, M peut :

1. s'arrêter dans un état accepteur
2. s'arrêter dans un état qui n'est pas accepteur
3. ne jamais s'arrêter.

Langage rcursif

Definition Un langage $L \subseteq \Sigma^*$ est *récursif*, s'il existe une machine de Turing M telle que :

1. $L(M) = L$ et
2. M s'arrête pour tout mot dans Σ^*.

$\square$

On dit que L est *décidé* par M.

Langage rcursivement numrable

Definition   Un langage $L \subseteq \Sigma^*$ est *récursivement énumérable (r.e.)*, s'il existe une machine de Turing M telle que $L(M) = L$.
$\square$

Autrement dit, L est reconnaissable par une machine de Turing.

Liens entre récursifs et récursivement énumérables

Théorème

- Un langage L est décidable ssi son complément L^c est décidable.
- Un langage L est décidable ssi L et son complément L^c sont récursivement énumérable.

Liens entre récursifs et récursivement énumérables

Théorème

- Un langage L est décidable ssi son complément L^c est décidable.
- Un langage L est décidable ssi L et son complément L^c sont récursivement énumérable.

Preuve

1. Si L est décidé par une MT M alors L^c est décidé par la MT M' obtenue à partir de M en inversant les états accépteurs et non-accépteurs.

□

Liens entre récursifs et récursivement énumérables

Théorème

- Un langage L est décidable ssi son complément L^c est décidable.
- Un langage L est décidable ssi L et son complément L^c sont récursivement énumérable.

Preuve

1. Si L est décidé par une MT M alors L^c est décidé par la MT M' obtenue à partir de M en inversant les états accépteurs et non-accépteurs.
2. Si L est décidé par une MT M alors L est reconnu par M et L^c est reconnu par M'.

$\square$

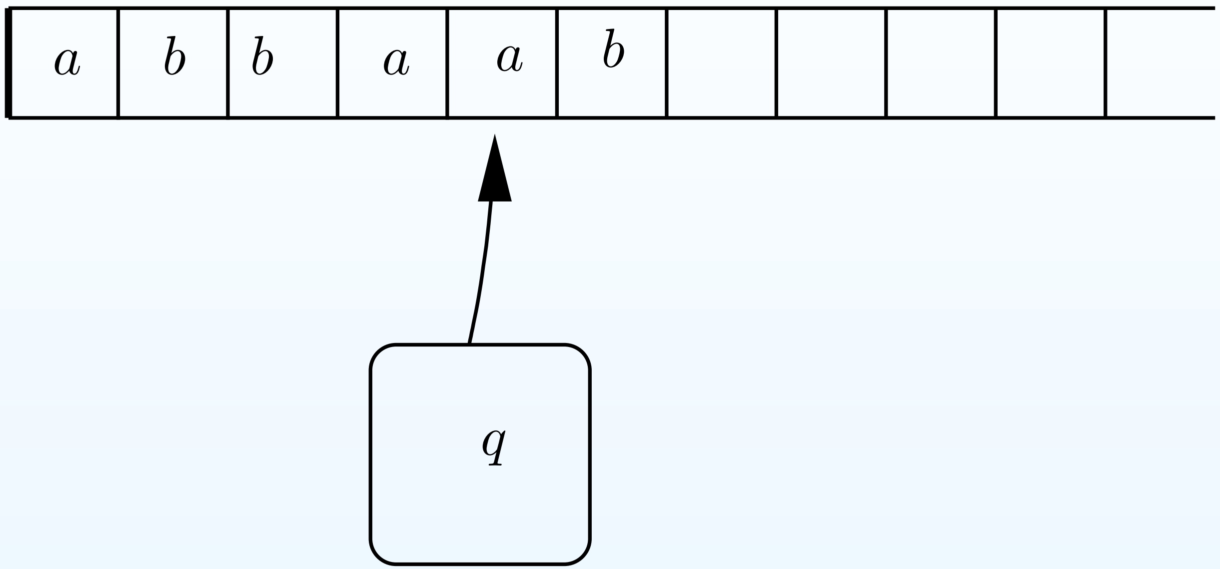
Machine de Turing à plusieurs rubans

- Une MT à k rubans, avec $k \geq 1$, est composée d'une unité de contrôle, de k rubans infinis divisé en cases, de k têtes de lecture/ écriture.
  Chaque rubans a sa propre tête de lecture.
- en fonction de ce qu'elle lit sur les k cases courantes et de son état courant (conformément à sa relation de transition δ), elle
  1. écrit un symbole sur la case courante de chaque ruban
  2. déplace chacune des têtes de lecture/ écriture d'une case (à droite, à gauche)
  3. change d'état.

Equivalence entre Machines à 1-ruban et plusieurs rubans

Théorème Tout langage reconnu (déciidié) par une machine de Turing à plusieurs rubans est récursivement énumérable (décidable).

Machine de Turing à semi-ruban



Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

Théorème Tout langage récursivement énumérable (décidable) est reconnaissable (décidable) par une machine de Turing à semi-ruban.

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n S v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n S v_1 v_m

u_1 u_n v_1 v_m

Equivalence entre Machine de Turing et Machine de Turing à semi-ruban

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n S v_1 v_m

u_1 u_n v_1 v_m

u_1 u_n v_1 v_m

MT et Automates à 2 piles

Théorème Tout langage récursivement énumérable (décidable) est reconnaissable (décidable) par un automates à 2 piles et inversement.

Modèles équivalents

- La définition des machines de Turing peut être modifiée sans changer leur pouvoir de reconnaissance :
  1. il peut y avoir plusieurs rubans
  2. le(s) ruban(s) peuvent être semi-infinis
  3. la tête de lecture/écriture peut ou ne peut pas rester stationnaire
  4. on peut écrire ou non le symbole blanc
  5. la fonction de transition peut être déterministe ou pas
- à vouloir améliorer les MT, on retombe toujours sur des machines reconnaissant la même classe de langages: elles semblent englober toute idée de procédure effective

Machine à compteurs

- Un ruban en lécture seulement sur lequel se trouve le mot d'entrée.
- Des variables $x_1, \cdots, x_n$ qui prennent des valeurs dans $\mathbb{N}$.
- Les instructions: d est une direction dans $\{G, D, I\}$.
  - $\ell : x := x + 1$ $\{$ **lire** a, $d\}$ **goto** ℓ'
  - $\ell : x := x - 1$ $\{$ **lire** $a\, d\}$ **goto** ℓ'
  - **if** $x = 0$ **then goto** ℓ' **else goto** ℓ''.
  - Un label **fin**
  - Configuration initiale donnée par un label ℓ_0, le pointeur sur mot d'entrée et les variables initialisées à 0.
  - On accèpte le mot d'entrée si on atteint le label **fin**.

MT et Machine à compteurs

Théorème

1. Tout langage récursivement énumérable (décidable) est reconnaissable (décidable) par une machine à 2 compteurs, et inversement.
2. Un langage reconnu (décidé) par une machine à n compteurs est reconnu (décidé) une machine à 2 compteurs.

Preuve

1. Une machine à 2 piles avec $\Gamma = \{0, 1\}$ peut être simulée par une machine à 2 compteurs : empiler 0 = 2x; empiler 1 = 2x+1; dépiler 0 = $\frac{x}{2}$; dépiler 1 = $\frac{x-1}{2}$.
2. Gödelisation.

□

Thèse de Church-Turing

- Thèse de Church-Turing: Les fonctions calculables par une procédure effective le sont par une machine de Turing.
- adopter cette thèse, c'est choisir une modélisation du concept de procédure effective; il n'y a pas de démonstration, ce n'est pas un théorème
- la théorie de la calculabilité se fonde sur cette thèse; elle permet aussi de montrer l'existence de fonctions non calculables, par un simple argument de diagonalisation

La non-décidabilité

- Existence de problèmes indécidable.
- Technique de la réduction pour montrer l'indécidabilité.

Correspondance entre problème et langage

Un problème est représenté par l'ensemble de ces instance positives.

Definition

- La classe de décidabilité R est l'ensemble des langages récursifs.
- La classe de décidabilité RE est l'ensemble des langages récursivement énumérables.
- La classe de décidabilité co-RE est l'ensemble des langages dont le complément est récursivement énumérables.

□

Liens entre les classe

Lemme La classe R est incluse dans la classe RE ($R \subseteq RE$). $\square$

Lemme La classe R est incluse dans la classe co-RE ($R \subseteq$ co-RE). $\square$

Lemme Un langage et son complément sont dans RE ssi il est dans R ($RE \cap$ co-$RE = R$). $\square$

Propriétés de fermeture

Les classes est R, RE et co-RE sont fermées par:

- Intersection.
- Union.
- Concaténation.
- Kleene star (L^*).
- Homomorphisme et inverse homomorphisme.

La classe R est en plus fermée par l'opération du complément.

Lemme Le complément d'un langage dans R est dans R. □

Fonctions calculable

Soit $f : \sigma_1^* \times \cdots \times \Sigma_n^* \rightarrow \Sigma$ une fonction.
On définit

$$L_f = \{u_1\# \cdots \#u_n\#u \mid f(u_1, \cdots, u_n) = u\}$$

sur l'alphabet $\{\#\} \cup \bigcup_{i=1}^m \Sigma_i \cup \Sigma$ où $\# \notin \bigcup_{i=1}^m \Sigma_i \cup \Sigma$.
Definition f est *calculable*, si $L_f \in$ R.
f est *semi-calculable*, si $L_f \in$ RE. $\square$

Exemple

La fonction d'addition $f : I\!N \times I\!N \longrightarrow I\!N$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

Exemple

La fonction d'addition $f : I\!N \times I\!N \longrightarrow I\!N$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.

Exemple

La fonction d'addition $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $\Sigma = \{0, 1\}$.

Exemple

La fonction d'addition $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $\Sigma = \{0, 1\}$.
- $\Gamma = \{0, 1, \Box\}$.

Exemple

La fonction d'addition $f : I\!N \times I\!N \longrightarrow I\!N$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $\Sigma = \{0, 1\}$.
- $\Gamma = \{0, 1, \Box\}$.
- $$\begin{aligned} \Delta = \ & \{(q_0, 1, D, 1, q_0), (q_0, 0, D, 1, q_1), (q_1, 1, D, 1, q_1), \\ & (q_1, \Box, G, \Box, q_2), (q_2, 1, G, \Box, q_3), (q_3, 1, D, \Box, q_4)\} \end{aligned}$$

Exemple

La fonction d'addition $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $\Sigma = \{0, 1\}$.
- $\Gamma = \{0, 1, \Box\}$.
- $$\begin{aligned} \Delta = \quad & \{(q_0, 1, D, 1, q_0), (q_0, 0, D, 1, q_1), (q_1, 1, D, 1, q_1), \\ & (q_1, \Box, G, \Box, q_2), (q_2, 1, G, \Box, q_3), (q_3, 1, D, \Box, q_4)\} \end{aligned}$$
- q_0 est l'état initial.

Exemple

La fonction d'addition $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, $f(x, y) = x + y$ est calculée par la Machine de Turing défini par $TM = (Q, \Sigma, \Gamma, \Delta, q_0, \square, F)$ où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $\Sigma = \{0, 1\}$.
- $\Gamma = \{0, 1, \square\}$.
- $$\begin{aligned}\Delta = \quad & \{(q_0, 1, D, 1, q_0), (q_0, 0, D, 1, q_1), (q_1, 1, D, 1, q_1), \\ & (q_1, \square, G, \square, q_2), (q_2, 1, G, \square, q_3), (q_3, 1, D, \square, q_4)\}\end{aligned}$$
- q_0 est l'état initial.
- $F = \{q_4\}$

Concaténation d'une fonction et d'une machine

Soit $f : \Sigma_1^* \times \cdots \times \Sigma_n^* \rightarrow \Sigma$ une fonction calculable.
On peut définir la concaténation, $f; M$, de f et d'une machine de Turing M sur Σ^* comme la machine qui:

1. prend en entrée $u_1 \# \cdots \# u_n$
2. utilise la machine de L_f pour trouver u tel que $f(u_1, \cdots, u_n) = u$.
3. Lance M sur u et répond la même chose que M si celle-ci s'arrête.

Si f est totale alors la machine $f; M$ ne s'arrête pas uniquement

si M ne s'arrête pas.

Equipotence

- Soient A et B deux ensembles. A et B sont equipotents ssi il existe une bijection de A vers B.
- On note : $A \approx B$.
- L'ensemble A est appelé *dénombrable* ssi $\mathbb{N} \approx A$.

Par exemple, $\mathbb{N}^2$ est dénombrable. Mais $\mathcal{P}(\mathbb{N})$ ne l'est pas.

Diagonalisation

L'idée de la preuve de Cantor que $\mathcal{P}(\mathbb{N})$ n'est pas diagonsable. Supposons au contraire que $\mathcal{P}(\mathbb{N})$ l'est. Soit $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ une bijection.

	0	1	2	3	4	$\cdots$
$f(0)$	0	1	1	0	0	$\cdots$
$f(1)$	1	1	0	1	1	$\cdots$
$f(2)$	0	0	1	0	0	$\cdots$
$f(3)$	1	1	1	0	0	$\cdots$
$f(4)$	1	1	0	0	0	$\cdots$
$\vdots$						

Soit $D = \{i \in \mathbb{N} \mid i \in f(i)\}$, donc les i avec 1 sur la diagonale. Soit $\bar{D}$ le complément: $\bar{D} = \{i \in \mathbb{N} \mid i \notin f(i)\}$. Comme $D' \in \mathcal{P}(\mathbb{N})$, il existe i_0 avec $f(i_0) = D'$.
Or, $i_0 \in f(i_0)$ ssi $i_0 \in D'$ ssi $i_0 \notin f(i_0)$.

Théorème de Cantor

Théorème:

- $\mathcal{P}(I\!N)$ n'est pas dénombrable.
- $\mathcal{P}(A)$ et A ne sont pas equipotents.

Théorème de Cantor

Théorème:

- $\mathcal{P}(I\!N)$ n'est pas dénombrable.
- $\mathcal{P}(A)$ et A ne sont pas equipotents.

Preuve

- Démontrons que A et $\mathcal{P}(A)$ ne sont pas équipotents.
- Il faut donc montrer qu'il n'y a pas de bijection entre A et $\mathcal{P}(A)$.

$\square$

Théorème de Cantor(II)

Preuve

- Soit $X = \{x \in A \mid x \notin f(x)\}$. Donc $X \subseteq A$ et $X \in \mathcal{P}(A)$.

$\square$

Théorème de Cantor(II)

Preuve

- Soit $X = \{x \in A \mid x \notin f(x)\}$. Donc $X \subseteq A$ et $X \in \mathcal{P}(A)$.
- A cause de la surjection de f, il existe x_0 tel que $f(x_0) = X$

$\square$

Théorème de Cantor(II)

Preuve

- Soit $X = \{x \in A \mid x \notin f(x)\}$. Donc $X \subseteq A$ et $X \in \mathcal{P}(A)$.
- A cause de la surjection de f, il existe x_0 tel que $f(x_0) = X$
- Maintenant deux cas :

$\square$

Théorème de Cantor(II)

Preuve

- Soit $X = \{x \in A \mid x \notin f(x)\}$. Donc $X \subseteq A$ et $X \in \mathcal{P}(A)$.
- A cause de la surjection de f, il existe x_0 tel que $f(x_0) = X$
- Maintenant deux cas :
  1. $x_0 \in X$

$\square$

Théorème de Cantor(II)

Preuve

- Soit $X = \{x \in A \mid x \notin f(x)\}$. Donc $X \subseteq A$ et $X \in \mathcal{P}(A)$.
- A cause de la surjection de f, il existe x_0 tel que $f(x_0) = X$
- Maintenant deux cas :
  1. $x_0 \in X \implies x_0 \notin (f(x_0) = X)$ contradiction.
  2. $x_0 \notin X \implies x_0 \in f(x_0) \implies x_0 \in X$ contradiction.

$\square$

L'ensemble des MT est dénombrable

Soient $\Sigma = \{0,1\}$ et $\Gamma = \{0,1,B\}$.

- L'ensemble des mots sur Σ est dénombrable.
- L'ensemble des machines de Turing avec alphabet d'entrée Σ et alphabet de ruban Γ est dénombrable.

Une machine de Turing peut-être codée comme un mot dans $\Sigma = \{0,1\}$. Dans la suite M dénote ce codage. De la même manière un mot $u \in \Sigma^*$ peut être codé comme un mot dans $\Sigma = \{0,1\}$.

Dans la suite on dénote par M_i la machine d'indice $i \in I\!N$ et par

w_i le môt d'indice $i \in I\!N$.

Langages non- récursivement énumérables?

- l'ensemble des mots sur l'alphabet $\Sigma = \{0, 1\}$ est dénombrable

Langages non- récursivement énumérables?

- l'ensemble des mots sur l'alphabet $\Sigma = \{0, 1\}$ est dénombrable
- l'ensemble des langages sur l'alphabet Σ est non-dénombrable

Langages non- récursivement énumérables?

- l'ensemble des mots sur l'alphabet $\Sigma = \{0, 1\}$ est dénombrable
- l'ensemble des langages sur l'alphabet Σ est non-dénombrable
- L'ensemble des MT est dénombrable

Langages non- récursivement énumérables?

- l'ensemble des mots sur l'alphabet $\Sigma = \{0, 1\}$ est dénombrable
- l'ensemble des langages sur l'alphabet Σ est non-dénombrable
- L'ensemble des MT est dénombrable
- donc, il existe des langages non- récursivement énumérables

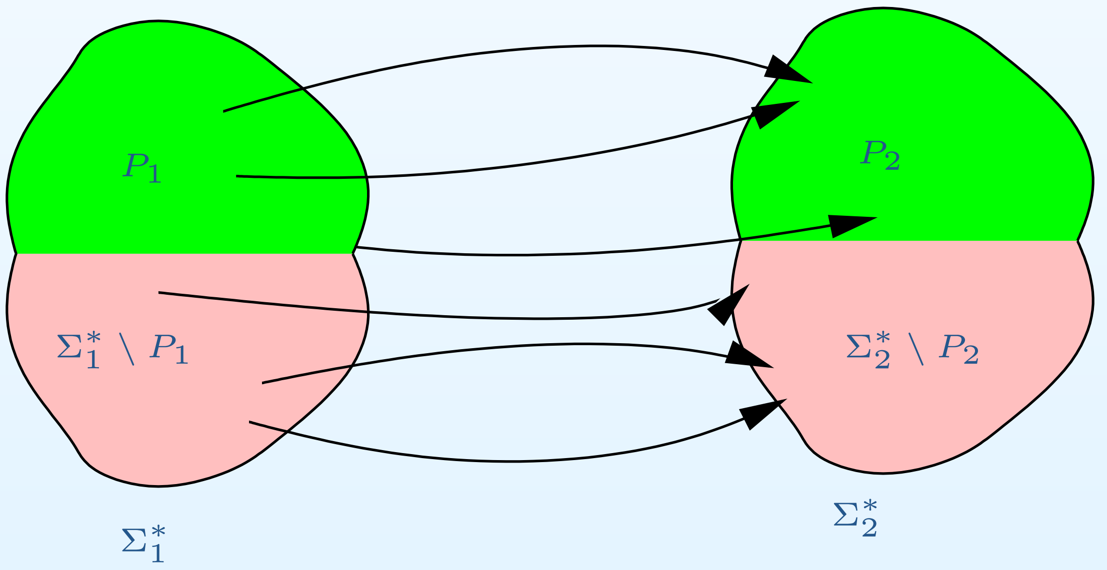
Un problème indécidable

Soit $K = \{w_i \in \Sigma^* \mid w_i \in L(M_i)\}$.
Théorème L'ensemble (problème) K est indécidable, i.e., $K \notin$ R.
Preuve On considère l'ensemble $\bar{K} = \{w_i \mid w_i \notin L(M_i)\}$.
On va montrer que $\bar{K}$ n'est pas dans RE.
Supposons le contraire. Alors, il existe une machine de Turing d'indice i_0 telle que $L(M_{i_0}) = \bar{K}$. Mais alors on a la contradiction suivante : $w_{i_0} \in L(M_{i_0})$ ssi $w_{i_0} \in \bar{K}$ ssi $w_{i_0} \notin L(M_{i_0})$.
Donc $\bar{K}$ n'est pas dans RE. Par conséquent, K n'est pas dans R. $\square$

Technique de la réduction

Soit P_1 et P_2 deux problèmes (langages). Soient Σ_1 et Σ_2 les alphabets de P_1 et P_2.
Réduire P_1 à P_2 consiste à montrer l'existence d'une fonction *totale* et *calculable* de Σ_1^* vers Σ_2^* telle que

$$u \in P_1 \textbf{ ssi } f(u) \in P_2.$$



Reduction

Si on peut réduire P_1 à P_2 alors on peut conclure :

- Si P_1 n'est pas dans R alors P_2 n'est pas dans R. Pareil pour RE.
- Si P_2 est dans R alors $P1$ est dans R.

En effet, supposons que la machine M décide P_2. Alors, la machine $f; M$ décide P_1. Pareil, si $P_2 \in$ RE.

Le langage universel

Le langage universel est :

$$LU = \{M\#w \mid w \in L(M)\}$$

Théorème $LU \in$ RE $\setminus$ R.

Le langage universel

Preuve $LU \notin$ R : soit f la fonction qui à w associe le mot $M_i\#w_i$ tel que i est l'indice de w, c.a.d. $w_i = w$. Alors f est totale et on a $w \in K$ ssi $f(w) \in LU$. En plus $L_f = \{w\#M_i\#w_i \mid f(w) = M_i\#w_i\} \in$ R. En effet, la machine suivante décide L_f:

1. Entrée: u.
2. Si u n'est pas de la forme $u_1\#u_2\#u_3$, ne pas accepter.
3. Sinon, chercher $i \in \mathbb{N}$ tel que :$u_1 = w_i$. Ceci peut être fait de la manière suivante : on commence avec et $i = 0$ et on incrémente i de 1 tant que $w_i \neq u_1$. Le test $w_i \neq u_1$ est clairement décidable.
4. Déterminer M_i.
5. Vérifier $u_2 = M_i \wedge u_3 = u_1$.

$LU \in$ RE : en utilisant la mchinelle universelle. $\square$

Poblème de l'arrêt

Le problème

$$H = \{M\#w \mid M \text{ s'arrête sur } w\}$$

Théorème $H \in$ RE $\setminus$ R.

Preuve $H \notin$ R:

$f(M\#w) = M'\#w$ où M' est la machine suivante :

1. Entrée: u.
2. Exécuter M sur w.
3. Si M s'arrête dans un état non-acceteur, alors entrer dans une boucle infinie.

On a : $M\#w \in LU$ ssi $f(M\#w) \in H$, f est totale et calculable.

$H \in$ RE: en utilisant la machine universelle et en allant dans un état accepteur quand l'éxcution de M termine. $\square$

Théorème de Rice

Soit $\Sigma = \{0, 1\}$.
Une propriété non-triviale sur les machines de Turing est un sous-ensemble $P \subseteq \Sigma^*$ tel que :

1. Pout toutes machines M et N :
   $L(M) = L(N) \Rightarrow [M \in P \Leftrightarrow N \in P]$.
2. Il existe M_0, M_1 : $M_0 \in P$ et $M_1 \notin P$.

Théorème de Rice

Théorème [Rice] Si P est une propriété non-triviale sur le machine de Turing alors $P \notin$ R :

$$P = \{M \mid M \in P\} \notin \mathsf{R}$$

Théorème de Rice : Preuve

La preuve est par réduction de H sur P. Supposons que P satisfait la propriété suivante: pour toute MT M, $L(M) = \emptyset \Rightarrow M \notin P$. Sinon on montre que $\Sigma^* \setminus P \notin$ R.
Soit M_0 une machine telle que $M_0 \in P$. Soit f la fonction telle que :$f(M\#w)$ est une machine de Turing avec

$$L(f(M\#w)) = \begin{cases} L(M_0) & \text{;si } M\#w \in H \\ \emptyset & \text{;sinon} \end{cases}$$

Alors, $M\#w \in H$ ssi $f(M\#w) \in P$ et f est totale. La machine $f(M\#u)$ se comporte de la manière suivante:

1. Sauvegarder l'entrée u dans un deuxième ruban.
2. Exécuter M sur w.
3. Si M s'arrête, alors exécuter M_0 sur u et répondre la même chose.

Exemples

1. $\{M \mid M$ contient 5 états$\}$.
2. $\{M \mid M$ n'a aucun état accepteur $\}$.
3. $\{M \mid L(M) = 0^*1^*\}$.
4. $\{M \mid |L(M)| \equiv_2 0\}$.
5. $\{M \mid L(M)$ est infini$\}$.

Exemples

1. $\{M \mid L(M) \subseteq L_0\}$ pour un langage fixé L_0.
2. $\{M \mid L(M) \supseteq L_0\}$ pour un langage fixé L_0.
3. $\{M \mid L(M) = L_0\}$ pour un langage fixé $L_0 \in$ RE.
4. $\{M \mid L(M) \neq L_0\}$ pour un langage fixé $L_0 \in$ RE.
5. $\{M \mid L(M) = L_0\}$ pour un langage fixé $L_0 \in$ co-RE.
6. $\{M \mid L(M) \neq L_0\}$ pour un langage fixé $L_0 \in$ co-RE.
7. $\{M \mid L(M) = L_0\}$.
8. $\{M \mid \forall u \in L(M)\, |u| \leq k\}$, pour $k \in \mathbb{N}$ (fixé).
9. $\{M\#o^k \mid k \in \mathbb{N} \wedge \ \forall u \in L(M)\, |u| \leq k\}$.

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 $\Delta' = \Delta \cup \{(q_s, \epsilon, Z_0, Z_0 Z_\perp, q_0)\} \cup \{(q, \epsilon, Z_\perp, \epsilon, q_f) \mid q \in Q\}$

$\square$

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

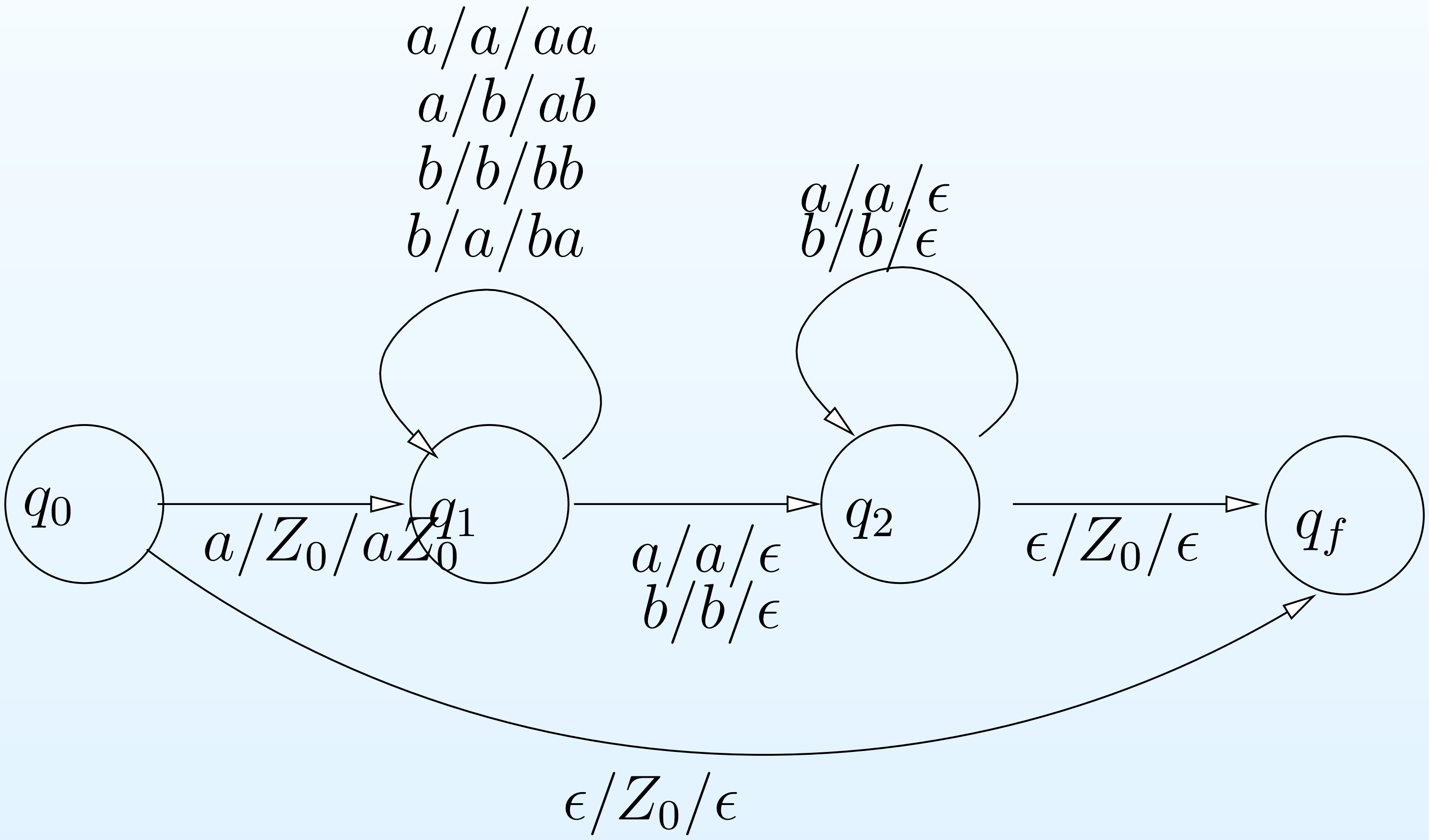
Le langages des palindromes $L = \{w\tilde{w} \mid w \in \{a,b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe.

Langages algébriques non-déterministes

Le langages 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 langages 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 0.6 *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 0.7 *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

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

□

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 0.8 *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

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

On a $|Z_1 Z_2| = 2$ et $|T\gamma| = |Z_1 Z_2 \gamma| - 1$ □

Langages non-algébriques?

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

Langages non-algébriques?

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

Langages non-algébriques?

- 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 algébriques sur l'alphabet Σ est dénombrable: une grammaire peut être encodée par un simple mot sur un alphabet fini qui serait l'union de Σ avec quelques symboles utilitaires (2 symboles permettant d'encoder les non-terminaux, la virgule, la fleche...)

Langages non-algébriques?

- 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 algébriques sur l'alphabet Σ est dénombrable: une grammaire peut être encodée par un simple mot sur un alphabet fini qui serait l'union de Σ avec quelques symboles utilitaires (2 symboles permettant d'encoder les non-terminaux, la virgule, la fleche...)
- donc, il existe des langages non-algébriques

Langages non-algbrique et Lemme de l'itération

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

1. $vy \neq \epsilon$ et $x \neq \epsilon$.
2. $|vxy| \leq n$.
3. pour tout $k \in I\!N$, $uv^kxy^kz \in L$.

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction Supposons que L est algébrique.

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction Alors, par le L.It., il existe $n \geq 0$ tel que pour tout $w \in L$ avec $|w| \geq n$, ils existent $u, v, x, y, z \in \Sigma^*$ avec: $w = uvxyz$, $vy \neq \epsilon$, $x \neq \epsilon$, et $|vxy| \leq n$, $uv^k xy^k z \in L$, pour tout $k \geq 0$.

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction Soit $w = a^n b^n c^n$ (où n est le n du L.It.) et soient $u, v, x, y, z \in \Sigma^*$ comme ci-dessus.

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction Soit $w = a^n b^n c^n$ (où n est le n du L.It.) et soient $u, v, x, y, z \in \Sigma^*$ comme ci-dessus. Alors, comme $|vxy| \leq n$ et $vy \neq \epsilon$ on a que v et y contient au moins un des symboles de $\{a, b, c\}$, mais pas tous les trois. Par exemple supposons que vy contient a mais pas c.

Application du Lemme de l'itération (L.It.)

Soit $L = \{a^i b^i c^i \mid i \geq 0\}$. L n'est pas algébrique.

Preuve par contradiction Soit $w = a^n b^n c^n$ (où n est le n du L.It.) et soient $u, v, x, y, z \in \Sigma^*$ comme ci-dessus. Alors, comme $|vxy| \leq n$ et $vy \neq \epsilon$ on a que v et y contient au moins un des symboles de $\{a, b, c\}$, mais pas tous les trois. Par exemple supposons que vy contient a mais pas c. Soit $w' = uv^2xy^2z$. Alors, w' contient plus de a que de c et donc $w' \notin L$. Mais par le L.It. on a $w' \in L$, ce qui est une contradiction. Donc L n'est pas algébrique.

Propriétés de fermeture

Les langages HC sont fermées par:

- Union.
- Concaténation.
- Kleene star (L^*).

Langages HC et intersection/complmentation

Lemme 0.9 Les langages HC ne sont pas fermés par intersection.

Langages HC et intersection/complmentation

Lemme 0.11 Les langages HC ne sont pas fermés par intersection.

- On a montré que le langage $L = \{a^i b^i c^i \mid i \in I\!\!N\}$ n'est pas hors-contexte.

Langages HC et intersection/complmentation

Lemme 0.13 Les langages HC ne sont pas fermés par intersection.

- On a montré que le langage $L = \{a^ib^ic^i \mid i \in \mathbb{N}\}$ n'est pas hors-contexte.
- Comme $L = \{a^ib^ic^j \mid i, j \in \mathbb{N}\} \cap \{a^ib^jc^j \mid i, j \in \mathbb{N}\}$ et que $\{a^ib^ic^j \mid i, j \in \mathbb{N}\}$ et $\{a^ib^jc^j \mid i, j \in \mathbb{N}\}$ sont hors-contexte nous avons que les langages hors-contexte ne sont pas fermés par intersection.

Langages HC et intersection/complmentation

Lemme 0.15 Les langages HC ne sont pas fermés par intersection.

- On a montré que le langage $L = \{a^i b^i c^i \mid i \in I\!N\}$ n'est pas hors-contexte.
- Comme $L = \{a^i b^i c^j \mid i, j \in I\!N\} \cap \{a^i b^j c^j \mid i, j \in I\!N\}$ et que $\{a^i b^i c^j \mid i, j \in I\!N\}$ et $\{a^i b^j c^j \mid i, j \in I\!N\}$ sont hors-contexte nous avons que les langages hors-contexte ne sont pas fermés par intersection.

Lemme 0.16 Les langages hors-contexte ne sont pas fermés par complémentation.

Langages HC et intersection/complmentation

Lemme 0.17 Les langages HC ne sont pas fermés par intersection.

- On a montré que le langage $L = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ n'est pas hors-contexte.
- Comme $L = \{a^i b^i c^j \mid i, j \in \mathbb{N}\} \cap \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ et que $\{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ et $\{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ sont hors-contexte nous avons que les langages hors-contexte ne sont pas fermés par intersection.

Lemme 0.18 Les langages hors-contexte ne sont pas fermés par complémentation.

- On a montré que cette classe est close par union.

Langages HC et intersection/complmentation

Lemme 0.19 Les langages HC ne sont pas fermés par intersection.

- On a montré que le langage $L = \{a^i b^i c^i \mid i \in I\!N\}$ n'est pas hors-contexte.
- Comme $L = \{a^i b^i c^j \mid i, j \in I\!N\} \cap \{a^i b^j c^j \mid i, j \in I\!N\}$ et que $\{a^i b^i c^j \mid i, j \in I\!N\}$ et $\{a^i b^j c^j \mid i, j \in I\!N\}$ sont hors-contexte nous avons que les langages hors-contexte ne sont pas fermés par intersection.

Lemme 0.20 Les langages hors-contexte ne sont pas fermés par complémentation.

- On a montré que cette classe est close par union.
- Si elle était close par complémentation, elle serait forcément close par intersection, car $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$