La classe de complexité NP et les problèmes NPC

Marin Costes

Octobre 2020

Objectifs de la théorie de la complexité

Complexité d'algorithmes

Algorithme de Tri	Complexité au Pire
Tri par Sélection	$O(n^2)$
Tri par Insertion	$O(n^2)$
Tri par Propagation	$O(n^2)$
Tri par ABR	$O(n^2)$
Tri Rapide	$O(n^2)$ $O(n\log_2(n))$
Tri par Fusion	$O(n\log_2(n))$
Tri par TAS	$O(n\log_2(n))$

Objectifs de la théorie de la complexité

Complexité d'algorithmes

Algorithme de Tri	Complexité au Pire
Tri par Sélection	$O(n^2)$
Tri par Insertion	$O(n^2)$
Tri par Propagation	$O(n^2)$
Tri par ABR	$O(n^2)$
Tri Rapide	$O(n^2)$ $O(n\log_2(n))$
Tri par Fusion	$O(n\log_2(n))$
Tri par TAS	$O(n\log_2(n))$

Classifier la complexité de grands problèmes:

Problèmes de recherche (on renvoie un résultat):

- Trier une liste
- Calculer une fonction
- Factoriser un entier n
- ….

Objectifs de la théorie de la complexité

Complexité d'algorithmes

Algorithme de Tri	Complexité au Pire
Tri par Sélection	$O(n^2)$
Tri par Insertion	$O(n^2)$
Tri par Propagation	$O(n^2)$
Tri par ABR	$O(n^2)$
Tri Rapide	$O(n^2)$ $O(n\log_2(n))$
Tri par Fusion	$O(n\log_2(n))$
Tri par TAS	$O(n\log_2(n))$

Classifier la complexité de grands problèmes:

Problèmes de recherche (on renvoie un résultat):

- Trier une liste
- Calculer une fonction
- Factoriser un entier n
- ….

Problèmes de décision (réponse par oui ou non):

- Est-ce que n est premier?
- Est-ce que la liste L contient l'élément x?
- Problème du voyageur de commerce
- Problème de satisfiabilité

Objectifs de la théorie de la complexité

Classifier la complexité de grands problèmes:

Complexité d'algorithmes

Algorithme de Tri	Complexité au Pire
Tri par Sélection	$O(n^2)$
Tri par Insertion	$O(n^2)$
Tri par Propagation	$O(n^2)$
Tri par ABR	$O(n^2)$
Tri Rapide	$O(n^2)$ $O(n\log_2(n))$
Tri par Fusion	$O(n\log_2(n))$
Tri par TAS	$O(n\log_2(n))$

Problèmes de recherche (on renvoie un résultat):

- Trier une liste
- Calculer une fonction
- Factoriser un entier n
- ….

Problèmes de décision (réponse par oui ou non):

- Est-ce que n est premier?
- Est-ce que la liste L contient l'élément x?
- Problème du voyageur de commerce
- Problème de satisfiabilité

Langage : Définition

Clôture par concaténation

Pour tout alphabet $\mathbf{\Sigma}$ on note $\mathbf{\Sigma}^*$ la clôture par concaténation de $\mathbf{\Sigma}$, à laquelle on ajoute le mot vide $\boldsymbol{\varepsilon}$.

Langage : Définition

Clôture par concaténation

Pour tout alphabet $\boldsymbol{\Sigma}$ on note $\boldsymbol{\Sigma}^*$ la clôture par concaténation de $\boldsymbol{\Sigma}$, à laquelle on ajoute le mot vide $\boldsymbol{\varepsilon}$.

Exemple:

- Si $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\boldsymbol{\Sigma}^* = \{\boldsymbol{\varepsilon}, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11}, \mathbf{000}, \dots\}$

Langage : Définition

Clôture par concaténation

Pour tout alphabet $\boldsymbol{\Sigma}$ on note $\boldsymbol{\Sigma}^*$ la clôture par concaténation de $\boldsymbol{\Sigma}$, à laquelle on ajoute le mot vide $\boldsymbol{\varepsilon}$.

Exemple:

- Si $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\boldsymbol{\Sigma}^* = \{\boldsymbol{\varepsilon}, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11}, \mathbf{000}, \dots\}$

Mots

Un mot w sur l'alphabet $\boldsymbol{\Sigma}$ est un élément de $\boldsymbol{\Sigma}^*$.

Langage : Définition

Clôture par concaténation

Pour tout alphabet $\mathbf{\Sigma}$ on note $\mathbf{\Sigma}^*$ la clôture par concaténation de $\mathbf{\Sigma}$, à laquelle on ajoute le mot vide $\boldsymbol{\varepsilon}$.

Exemple:

- Si $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\mathbf{\Sigma}^* = \{\boldsymbol{\varepsilon}, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11}, \mathbf{000}, \dots\}$

Mots

Un mot w sur l'alphabet $\mathbf{\Sigma}$ est un élément de $\mathbf{\Sigma}^*$.

Exemple:

- Si $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\mathrm{w} = \mathbf{00011001}$ est un mot

Langage : Définition

Langage

Un langage L sur un alphabet $\mathbf{\Sigma}$ est une sous partie de $\mathbf{\Sigma}^*$, i.e. :

$$L \subseteq \mathbf{\Sigma}^*$$

Langage : Définition

Langage

Un langage L sur un alphabet $\mathbf{\Sigma}$ est une sous partie de $\mathbf{\Sigma}^*$, i.e. :

$$L \subseteq \mathbf{\Sigma}^*$$

Exemples:

- $\mathbf{\Sigma}^*$ est un langage

Langage : Définition

Langage

Un langage L sur un alphabet $\mathbf{\Sigma}$ est une sous partie de $\mathbf{\Sigma}^*$, i.e. :

$$L \subseteq \mathbf{\Sigma}^*$$

Exemples:

- $\mathbf{\Sigma}^*$ est un langage
- Si $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\mathrm{L} = \{\mathbf{00}, \mathbf{111001}, \mathbf{1}\}$ est un langage

Langage : Définition

Langage

Un langage L sur un alphabet $\mathbf{\Sigma}$ est une sous partie de $\mathbf{\Sigma}^*$, i.e. :

$$L \subseteq \mathbf{\Sigma}^*$$

Exemples:

- $\mathbf{\Sigma}^*$ est un langage
- Si $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\mathrm{L} = \{\mathbf{00}, \mathbf{111001}, \mathbf{1}\}$ est un langage
- Si $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, $\mathrm{L} = \{\mathbf{01}\}^*$ est un langage

Problème de décision

Exemple: Problème de primalité

Problème de décision

Exemple: Problème de primalité

Donnée:

Un entier n

Question:

Est-ce que n est premier?

Problème de décision

Exemple: Problème de primalité

Donnée:
Un entier n

Question:
Est-ce que n est premier?

Donnée:
Un mot ω dans $\mathbf{\Sigma}^* = \{0,1\}^*$

Problème de décision

Exemple: Problème de primalité

Donnée:
Un entier n

Question:
Est-ce que n est premier?

Donnée:
Un mot ω dans $\mathbf{\Sigma}^* = \{0,1\}^*$

Question:
Est-ce que ω est dans
$L_p = \{10,11,101,111,1011, \ldots\}$?

Problème de décision

Exemple: Problème de primalité

Donnée:
Un entier n

Question:
Est-ce que n est premier?

Donnée:
Un mot ω dans $\mathbf{\Sigma}^* = \{0,1\}^*$

Question:
Est-ce que ω est dans
$L_p = \{10,11,101,111,1011, \dots\}$?

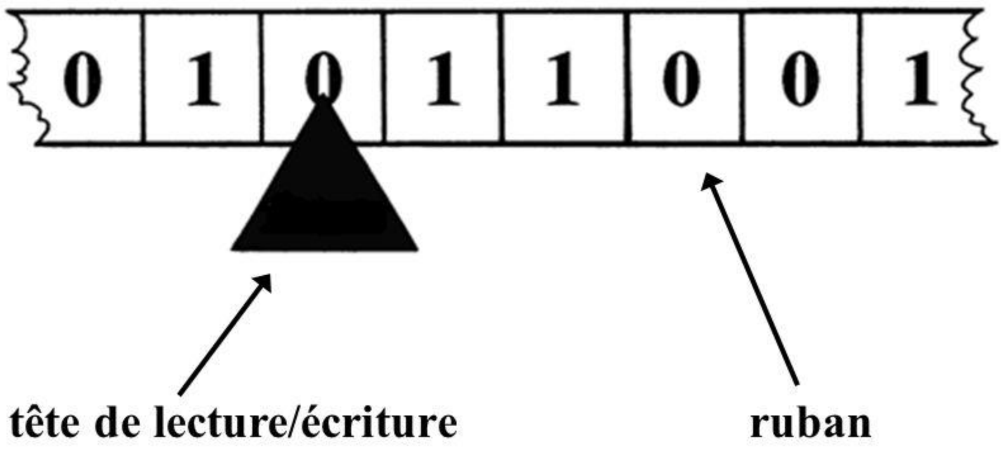
Résoudre un problème de décision équivaut à reconnaitre un langage!

Machine de Turing : Définition

Ensemble fini d'états

Contrôle

Schéma d'une MT



Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} = \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.

Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} \;= \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).

Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} = \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.

Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} \ = \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.
- $\boldsymbol{q_0}$ est **l'état initial**.

Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} \ = \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.
- $\boldsymbol{q_0}$ est **l'état initial**.
- $\boldsymbol{F}$ est l'ensemble des **états acceptants** .

Machine de Turing : Définition

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} = \{q_0; q_1; \dots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.
- $\boldsymbol{q_0}$ est **l'état initial**.
- $\boldsymbol{F}$ est l'ensemble des **états acceptants** .
- $\boldsymbol{\delta} : (\Gamma \times Q) \rightarrow (\Gamma \times \{\rightarrow, \leftarrow, \downarrow\} \times Q)$ est la **fonction de transition**.

Langage d'une MT

Mots acceptés

Un mot ω est accepté par une MT M si le calcul de M sur ω termine sur un état acceptant $\boldsymbol{q_f} \in \boldsymbol{F}$.

Langage d'une MT

Mots acceptés

Un mot ω est accepté par une MT M si le calcul de M sur ω termine sur un état acceptant $\boldsymbol{q_f} \in \boldsymbol{F}$.

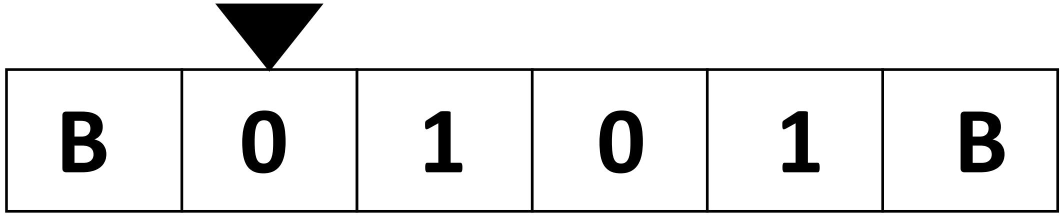
Langage reconnu

On appelle langage reconnu par M , l'ensemble $\mathrm{L}(M)$ des mots acceptés par M.

Exemple: reconnaitre $\{\mathbf{01}\}^*$

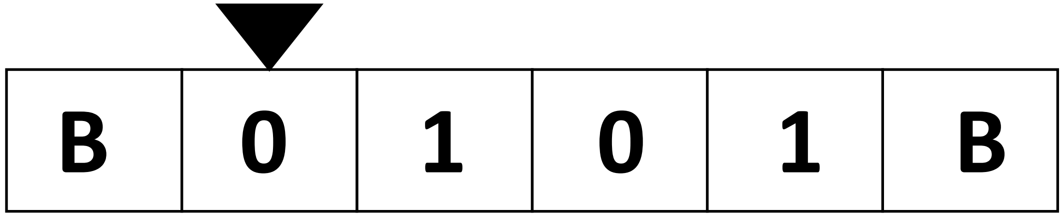
Etat / Lettre lue	q_0	q_1
0		
1		
B		

On cherche une MT en mode « lecture » qui reconnait $\{\mathbf{0}, \mathbf{1}\}^*$, en effectuant que des mouvement de tête vers la droite.



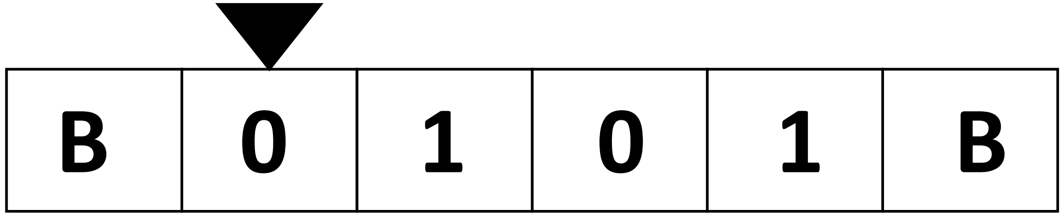
Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	$(0, \rightarrow, q_1)$	Reject
1		
B		



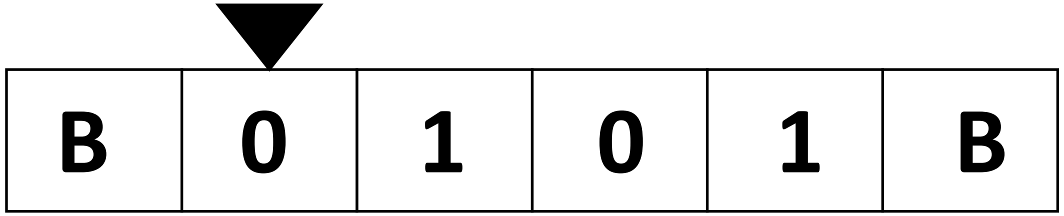
Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1		
B		



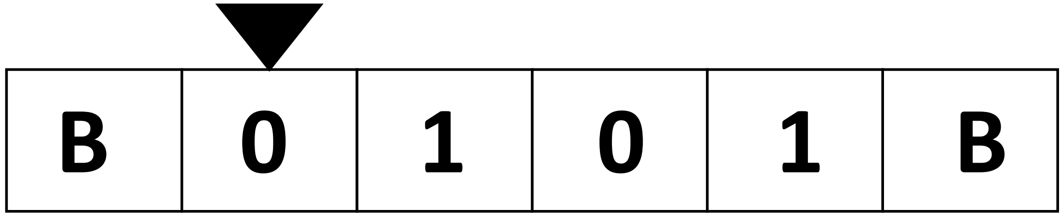
Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B		



Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject



Exemple: reconnaitre $\{01\}^*$

Lettre lue \ Etat	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:

q_0

B	0	1	0	1	B

Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:

q_1

B	0	1	0	1	B

Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:
q_0

B	0	1	0	1	B

Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:
q_1

B	0	1	0	1	B

Exemple: reconnaitre $\{01\}^*$

Lettre lue \ Etat	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:

q_0

B	0	1	0	1	B

Exemple: reconnaitre $\{\mathbf{01}\}^*$

Etat / Lettre lue	q_0	q_1
0	q_1	Reject
1	Reject	q_0
B	q_f	Reject

Etat:
q_f

B	0	1	0	1	B

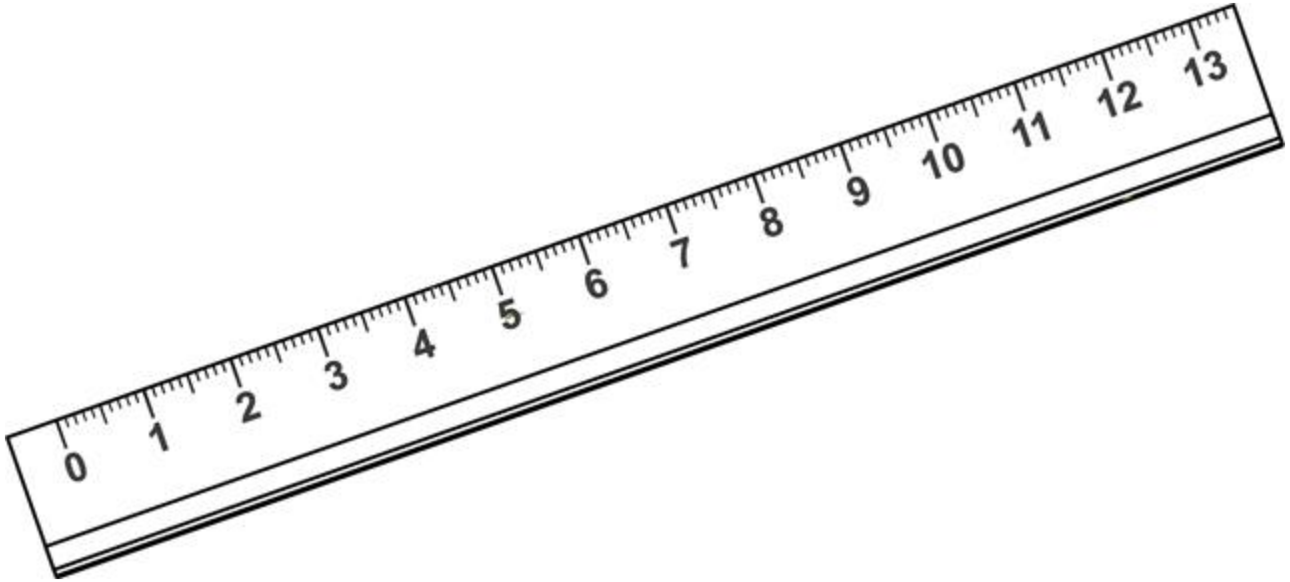
Complexité

Complexité

Complexité temporelle

Complexité

Complexité temporelle



Complexité spatiale

Complexité temporelle

Complexité temporelle

Une MT de complexité f(n), est une MT qui effectue un **nombre de transitions borné par f(n)** sur **toute entrée** de taille n.

Complexité temporelle

Une MT de complexité f(n), est une MT qui effectue un **nombre de transitions borné par f(n)** sur **toute entrée** de taille n.

Un langage de complexité f(n), est un langage reconnu par une machine de complexité f(n).

Complexité temporelle

Une MT de complexité *f(n)*, est une MT qui effectue un **nombre de transitions borné par *f(n)*** sur **toute entrée** de taille *n*.

Un langage de complexité *f(n)*, est un langage reconnu par une machine de complexité *f(n)*.

Sur l'exemple précédent :

M est une MT de complexité n.

$\{\mathbf{01}\}^*$ est un langage de complexité n.

MT non déterministe

Une machine de Turing est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} \ = \{q_0; q_1; \ldots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.
- $\boldsymbol{q_0}$ est **l'état initial**.
- $\boldsymbol{F}$ est l'ensemble des **états acceptants** .
- $\boldsymbol{\delta} : (\Gamma \times Q) \rightarrow (\Gamma \times \{\rightarrow, \leftarrow, \downarrow\} \times Q)$ est la **fonction de transition**.

MT non déterministe

Une MT non déterministe est un hexuplet **M =** $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ où:

- $\boldsymbol{Q} = \{q_0; q_1; \ldots; q_k\}$ est l'ensemble fini des **états** de la machine.
- $\boldsymbol{B}$ est un symbole spécial associé à une **case vide** (toutes les cases sauf un nombre fini contiennent B).
- $\boldsymbol{\Sigma}$ est l'ensemble des symboles constituant **l'alphabet**. Il ne contient pas B. On note $\Gamma = \Sigma \cup B$ l'alphabet de travail.
- $\boldsymbol{q_0}$ est **l'état initial**.
- $\boldsymbol{F}$ est l'ensemble des **états acceptants** .
- $\boldsymbol{\delta} \subseteq (\Gamma \times Q) \times (\Gamma \times \{\rightarrow, \leftarrow, \downarrow\} \times Q)$ est la **relation de transition.**

MT non déterministe

Mots acceptés

Un mot ω est accepté par une MT M si le calcul de M sur ω termine sur un état acceptant $\boldsymbol{q_f} \in \boldsymbol{F}$.

.

MT non déterministe

Mots acceptés

Un mot ω est accepté par une MT non déterministe M si il existe un calcul de M sur ω terminant sur un état acceptant $\boldsymbol{q_f} \in \boldsymbol{F}$.

MT non déterministe

Mots acceptés

Un mot ω est accepté par une MT non déterministe M si il existe un calcul de M sur ω terminant sur un état acceptant $\boldsymbol{q_f} \in \boldsymbol{F}$.

Complexité temporelle

M est de complexité $f(n)$ ssi pour tout ω de taille *n* tous les calculs possibles de *M* sur ω se résolvent en moins de *f(n)* étapes

Exemple : Langage des palindromes pairs

$$L = \{\omega\omega^{I} \mid \omega \in \{0,1\}^* \}$$
$$L = \{\varepsilon, 11, 00, 1111, 0110, \dots \}$$

Exemple : Langage des palindromes pairs

$L = \{\omega\omega^I \mid \omega \in \{0,1\}^* \}$
$L = \{\varepsilon, 11,00,1111,0110, \dots\}$

On veut que la première bande soit parcourue une seule fois en mode « lecture »

Exemple : Langage des palindromes pairs

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

$$L = \{\omega\omega^{I} \mid \omega \in \{0,1\}^{*} \}$$
$$L = \{\varepsilon, 11{,}00{,}1111{,}0110, \ldots\}$$

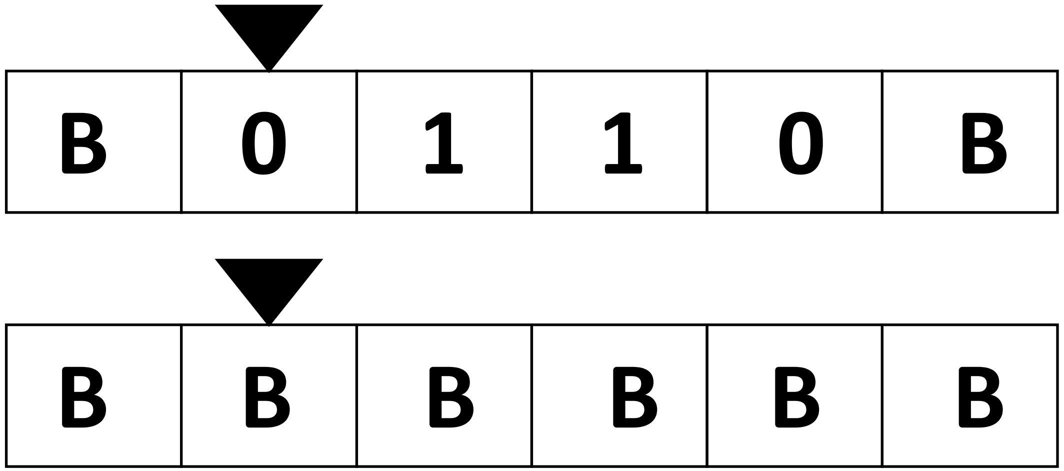
On veut que la première bande soit parcourue une seule fois en mode « lecture »

Exemple : calcul non-acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

Etat:

$q_{\rightarrow}$

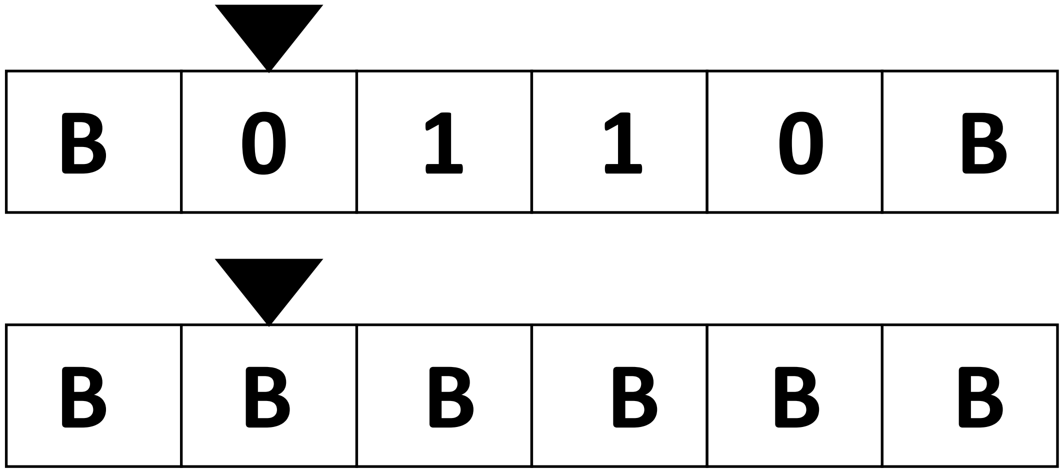


Exemple : calcul non-acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

Etat:

$q_{\rightarrow}$

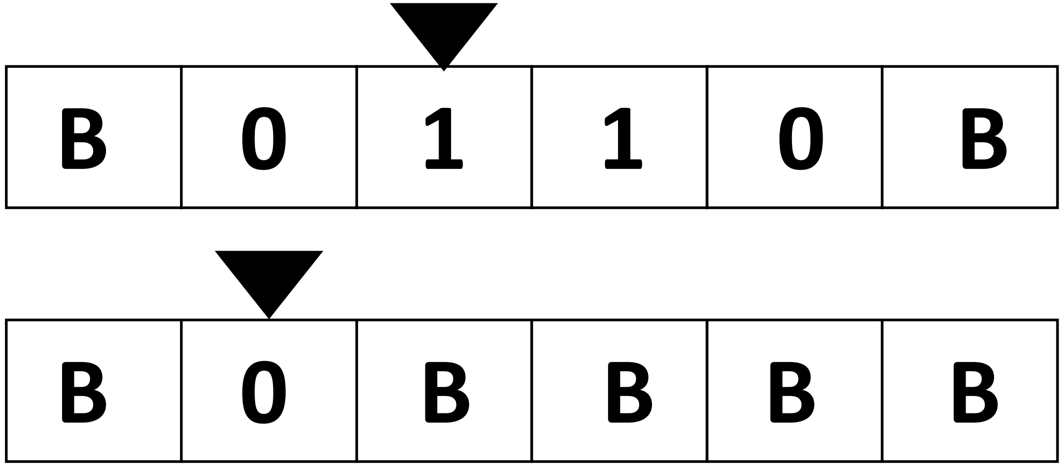


Exemple : calcul non-acceptant

Etat:

$q_{\leftarrow}$

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

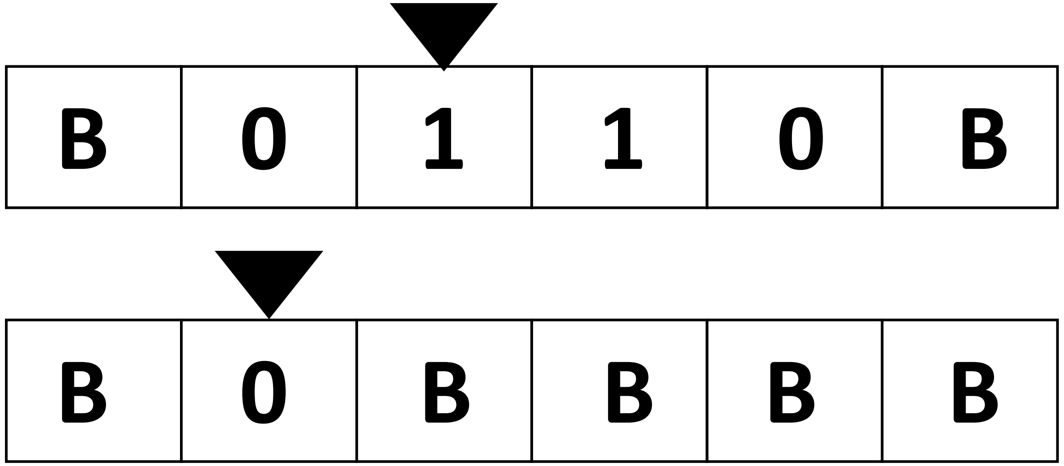


Exemple : calcul non-acceptant

Etat:

$q_{\leftarrow}$

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**

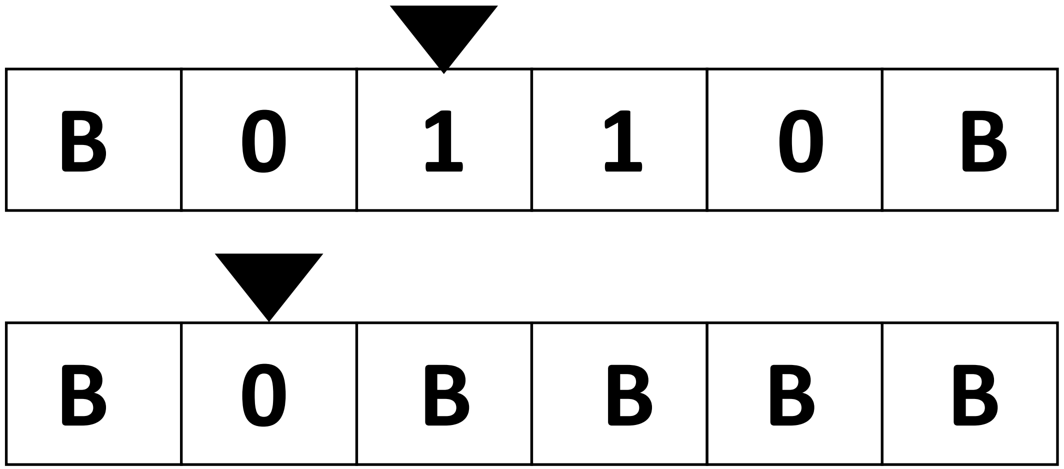


Exemple : calcul non-acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) OU (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**

Etat:

REJECT

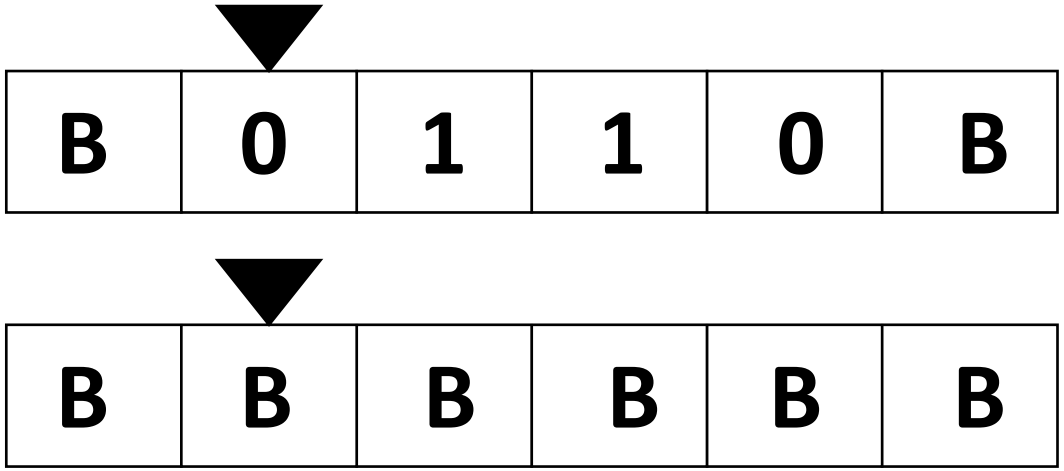


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) OU (x, $\downarrow$, $q_{\leftarrow}$)	REJECT
(x,x)	REJECT	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	REJECT	REJECT

Etat:

$q_{\rightarrow}$

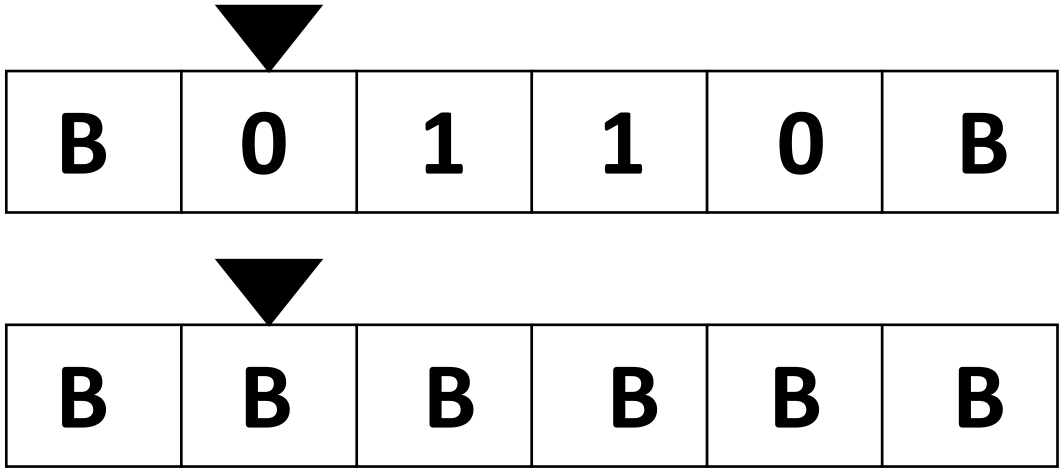


Exemple : calcul acceptant

Etat:

$q_{\rightarrow}$

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	**(x, →, $q_{\rightarrow}$)** OU (x, ↓ , $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, ←, $q_{\leftarrow}$)
(B,B)	(x, →, $q_{\leftarrow}$)	(B, ←, q_f)
autre	**REJECT**	**REJECT**

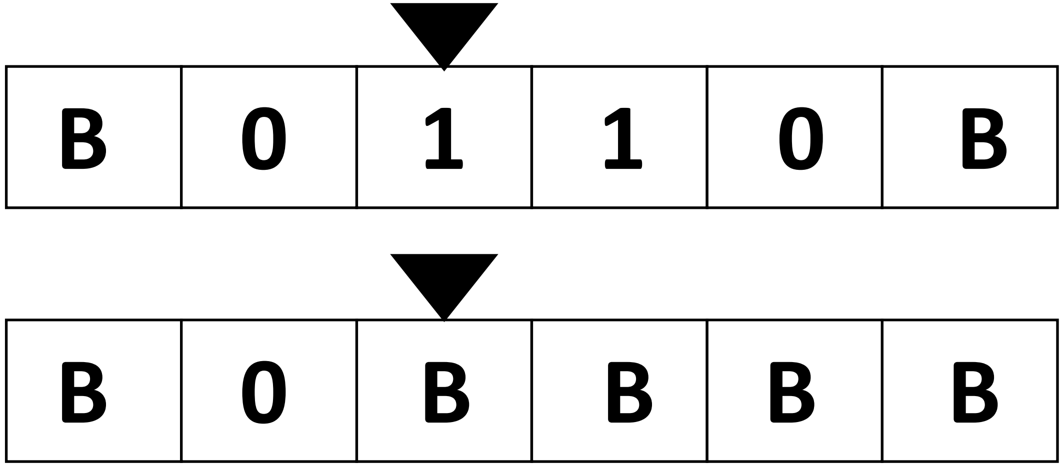


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

Etat:

$q_{\rightarrow}$

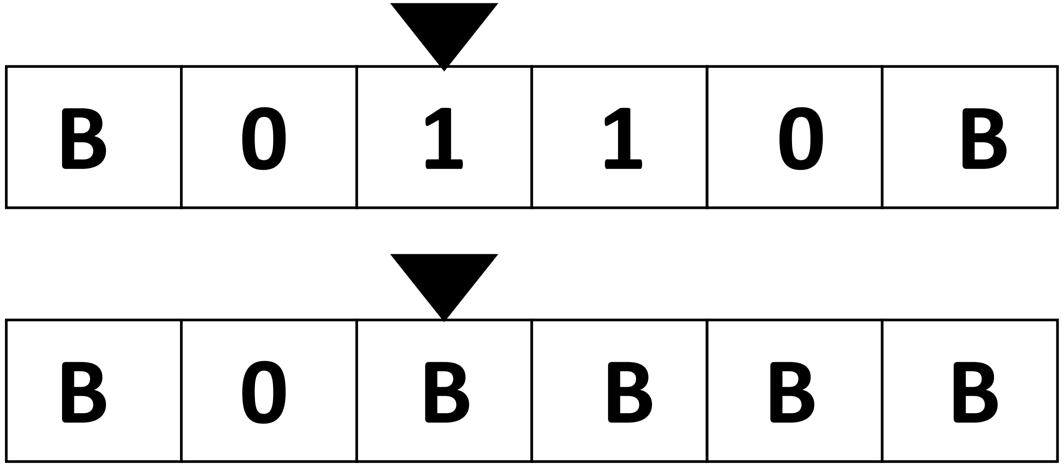


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	$(x, \rightarrow, q_{\rightarrow})$ **OU** $(x, \downarrow, q_{\leftarrow})$	**REJECT**
(x,x)	**REJECT**	$(x, \leftarrow, q_{\leftarrow})$
(B,B)	$(x, \rightarrow, q_{\leftarrow})$	$(B, \leftarrow, q_f)$
autre	**REJECT**	**REJECT**

Etat:

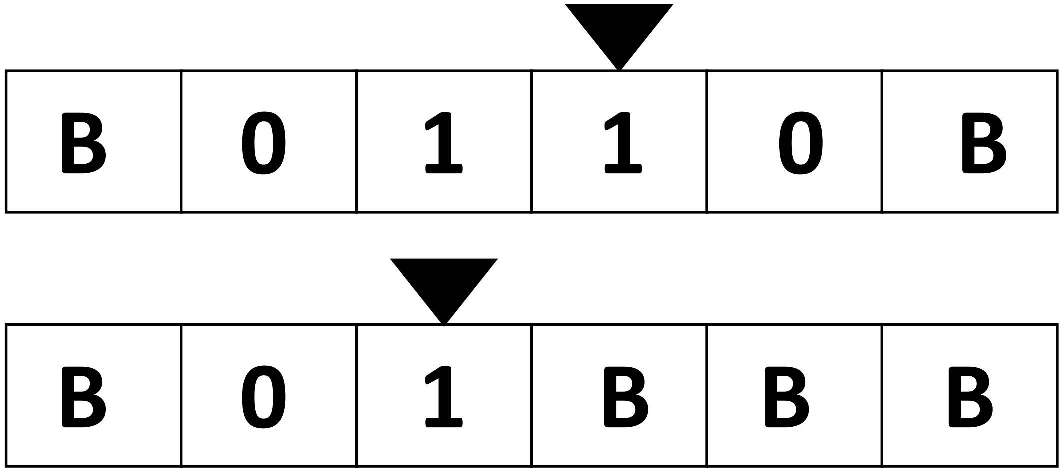
$q_{\rightarrow}$



Exemple : calcul acceptant

Etat:
$q_{\leftarrow}$

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_{f})
autre	**REJECT**	**REJECT**

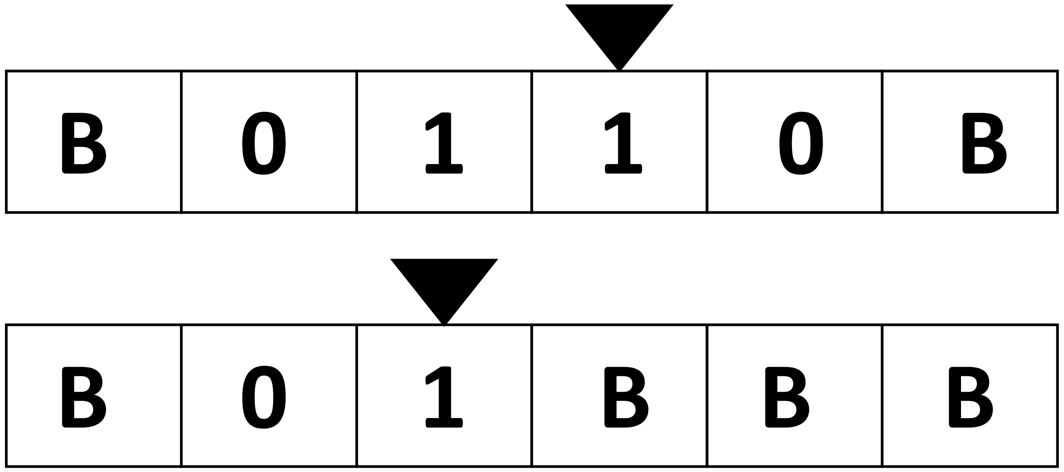


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) OU (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	**(x, $\leftarrow$, $q_{\leftarrow}$)**
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**

Etat:

$q_{\leftarrow}$

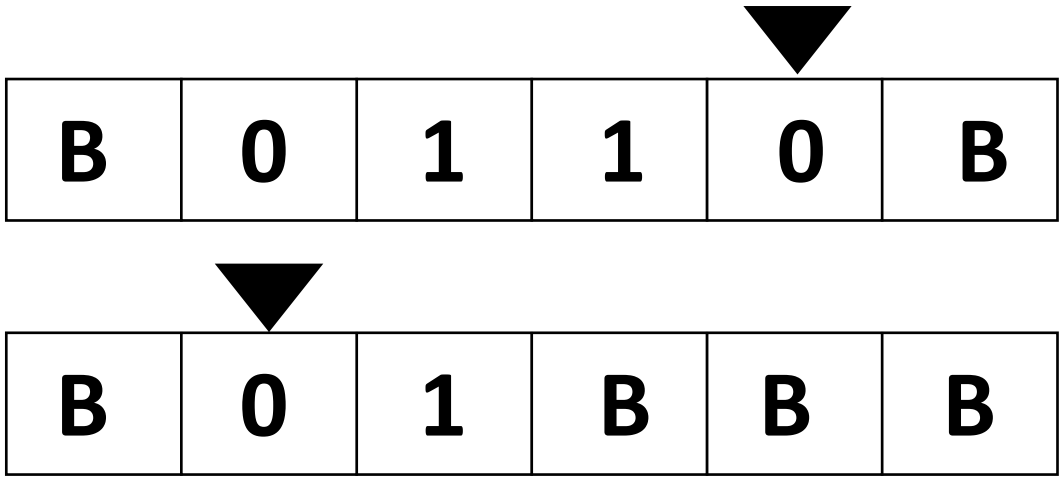


Exemple : calcul acceptant

Etat Lettres lues	$q_\rightarrow$	$q_\leftarrow$
(x,B)	(x, $\rightarrow$, $q_\rightarrow$) **OU** (x, $\downarrow$, $q_\leftarrow$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_\leftarrow$)
(B,B)	(x, $\rightarrow$, $q_\leftarrow$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**

Etat:

$q_\leftarrow$

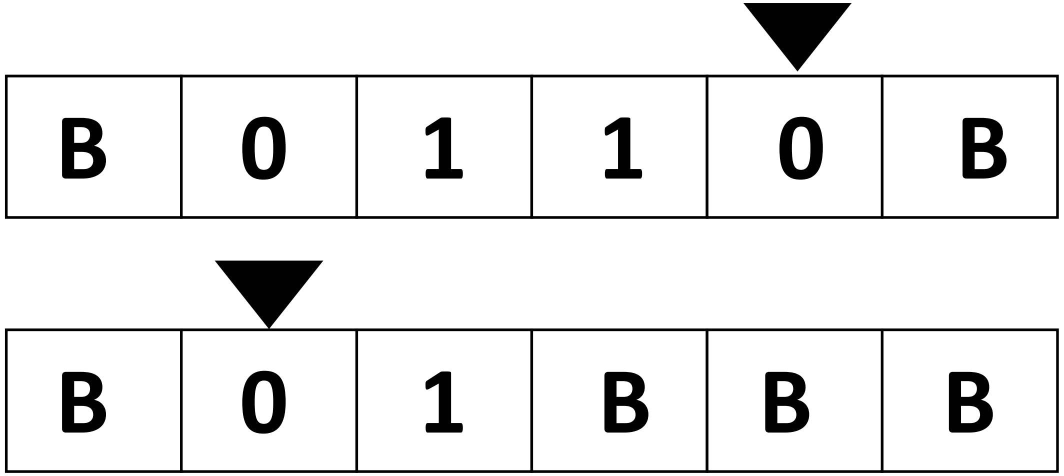


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) OU (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	**(x, $\leftarrow$, $q_{\leftarrow}$)**
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**

Etat:

$q_{\leftarrow}$

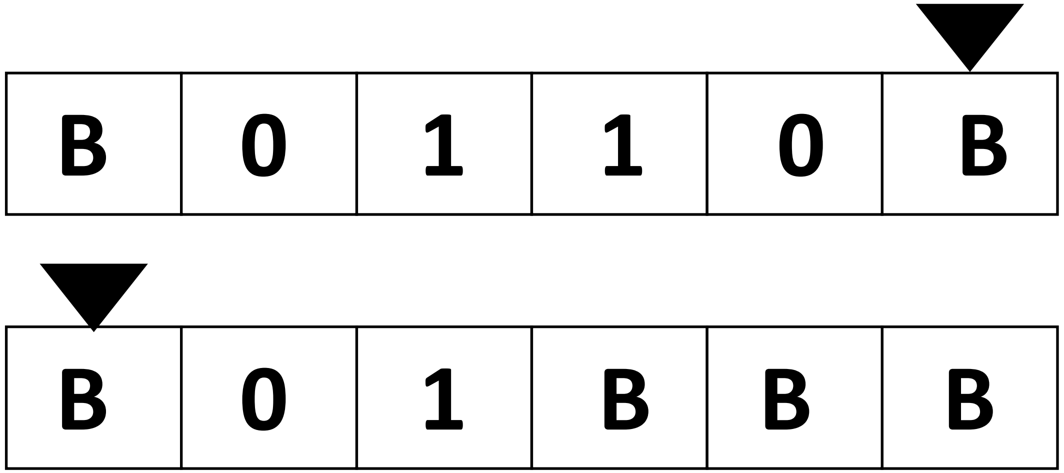


Exemple : calcul acceptant

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	$(x, \rightarrow, q_{\rightarrow})$ **OU** $(x, \downarrow, q_{\leftarrow})$	**REJECT**
(x,x)	**REJECT**	$(x, \leftarrow, q_{\leftarrow})$
(B,B)	$(x, \rightarrow, q_{\leftarrow})$	$(B, \leftarrow, q_f)$
autre	**REJECT**	**REJECT**

Etat:

$q_{\leftarrow}$

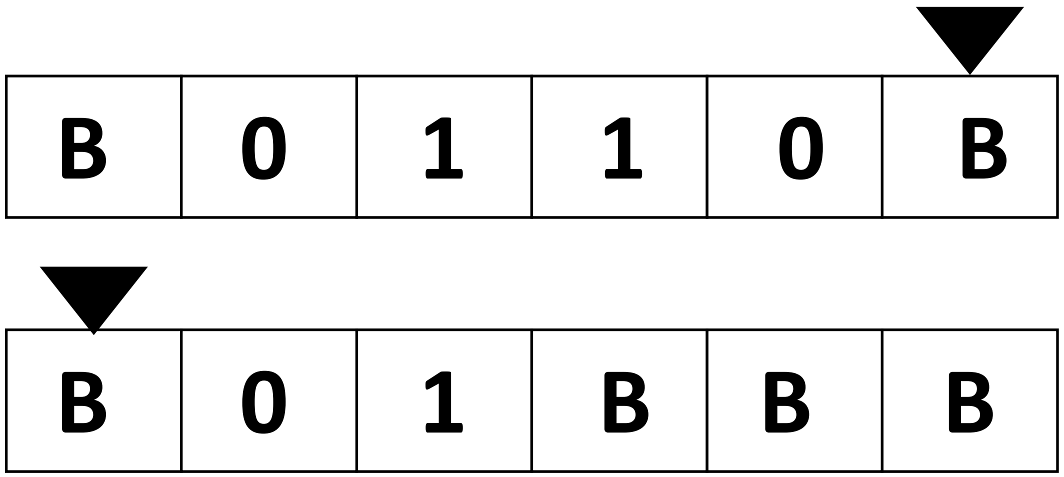


Exemple : calcul acceptant

Etat Lettres lues	$q_\rightarrow$	$q_\leftarrow$
(x,B)	$(x, \rightarrow, q_\rightarrow)$ **OU** $(x, \downarrow, q_\leftarrow)$	**REJECT**
(x,x)	**REJECT**	$(x, \leftarrow, q_\leftarrow)$
(B,B)	$(x, \rightarrow, q_\leftarrow)$	$(B, \leftarrow, q_f)$
autre	**REJECT**	**REJECT**

Etat:

$q_\leftarrow$

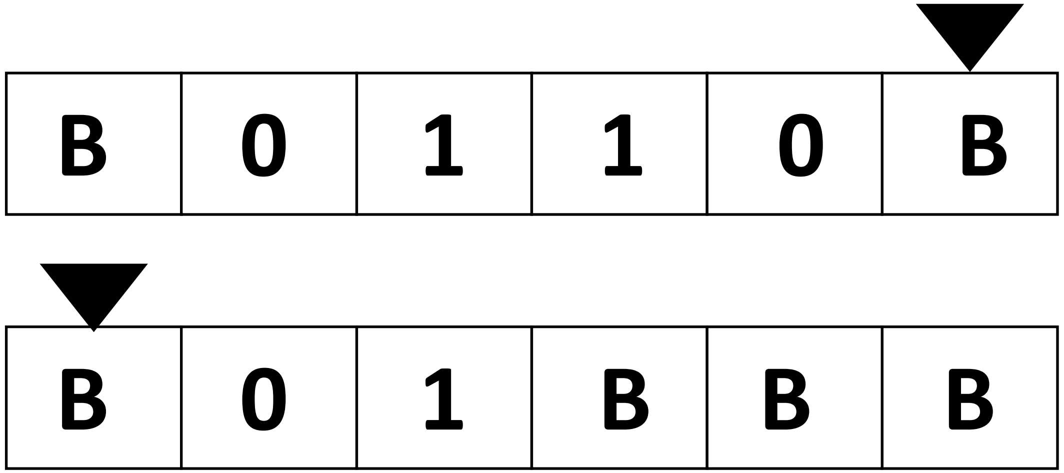


Exemple : calcul acceptant

Etat:

q_f

Etat / Lettres lues	$q_{\rightarrow}$	$q_{\leftarrow}$
(x,B)	(x, $\rightarrow$, $q_{\rightarrow}$) **OU** (x, $\downarrow$, $q_{\leftarrow}$)	**REJECT**
(x,x)	**REJECT**	(x, $\leftarrow$, $q_{\leftarrow}$)
(B,B)	(x, $\rightarrow$, $q_{\leftarrow}$)	(B, $\leftarrow$, q_f)
autre	**REJECT**	**REJECT**



Arbre de résolution

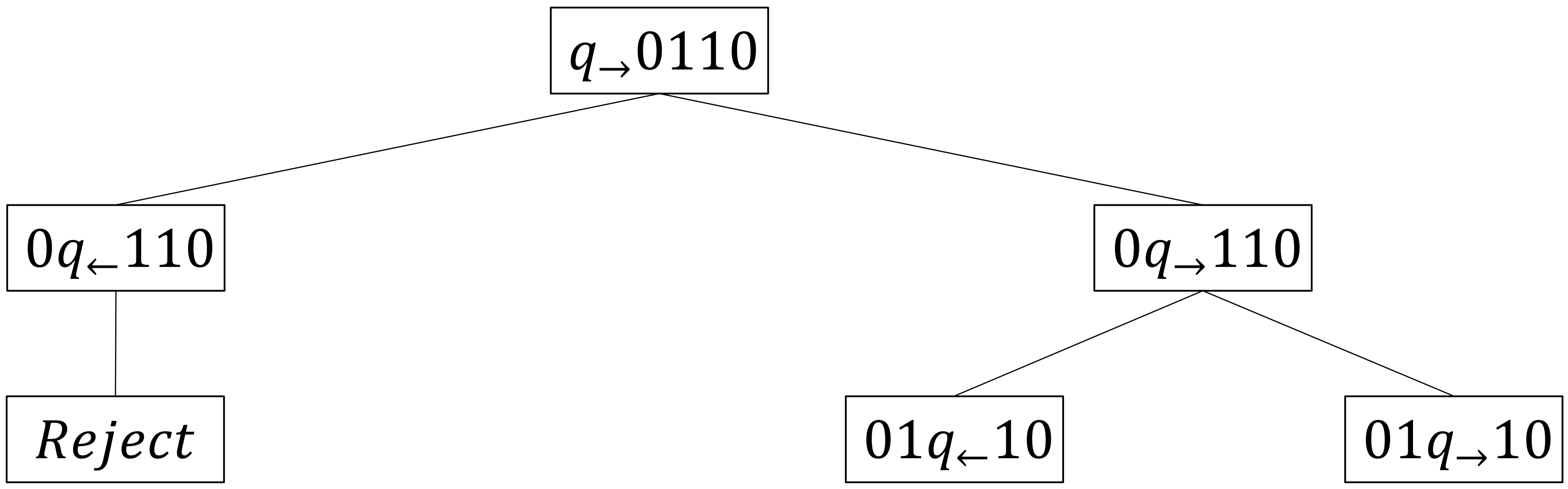
$\boxed{q_{\rightarrow}0110}$

Arbre de résolution

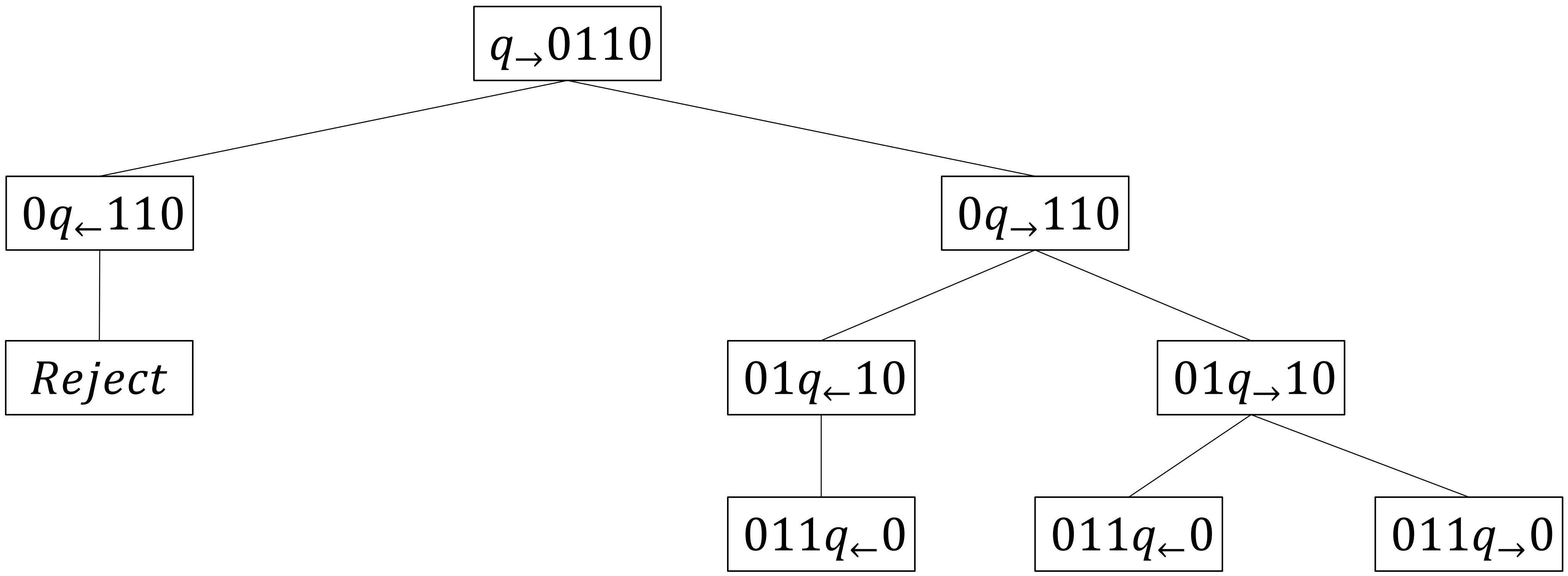
$$\boxed{q_{\rightarrow}0110}$$

$$\boxed{0q_{\leftarrow}110} \qquad \boxed{0q_{\rightarrow}110}$$

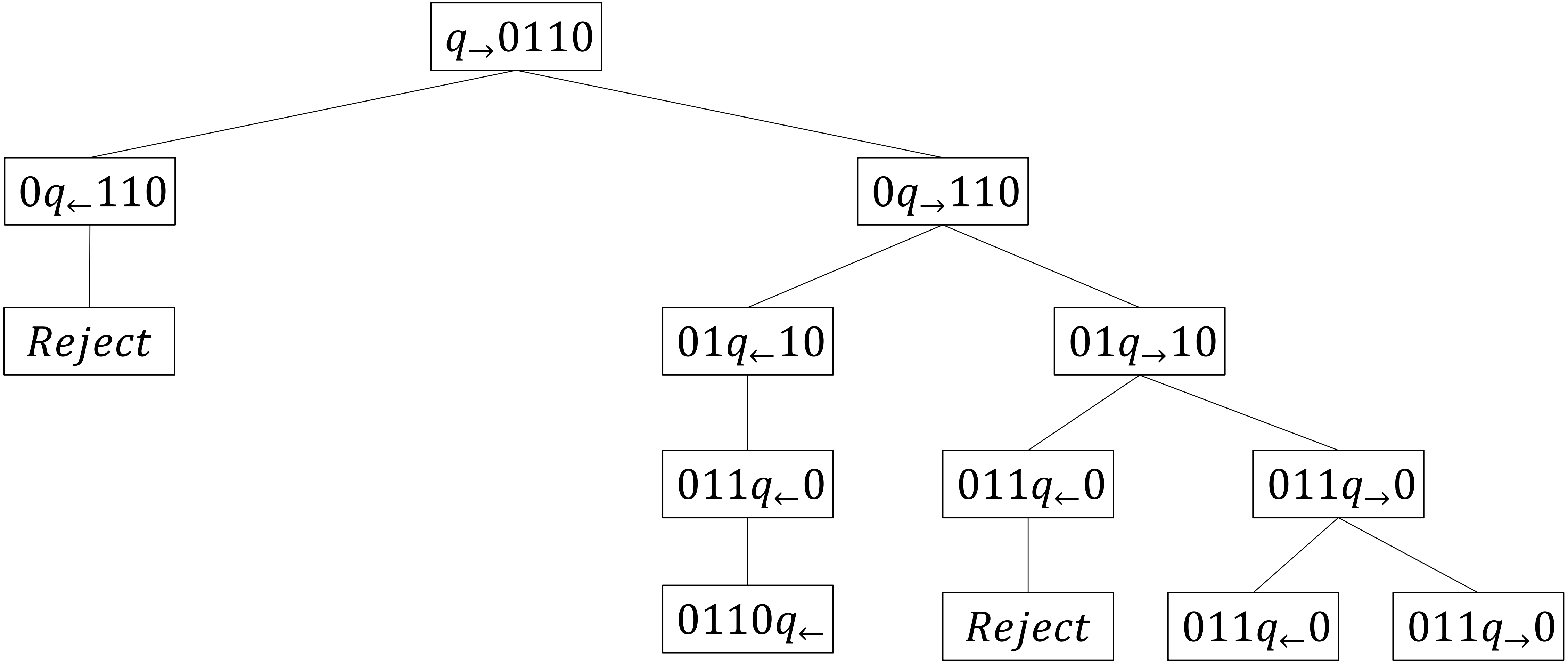
Arbre de résolution



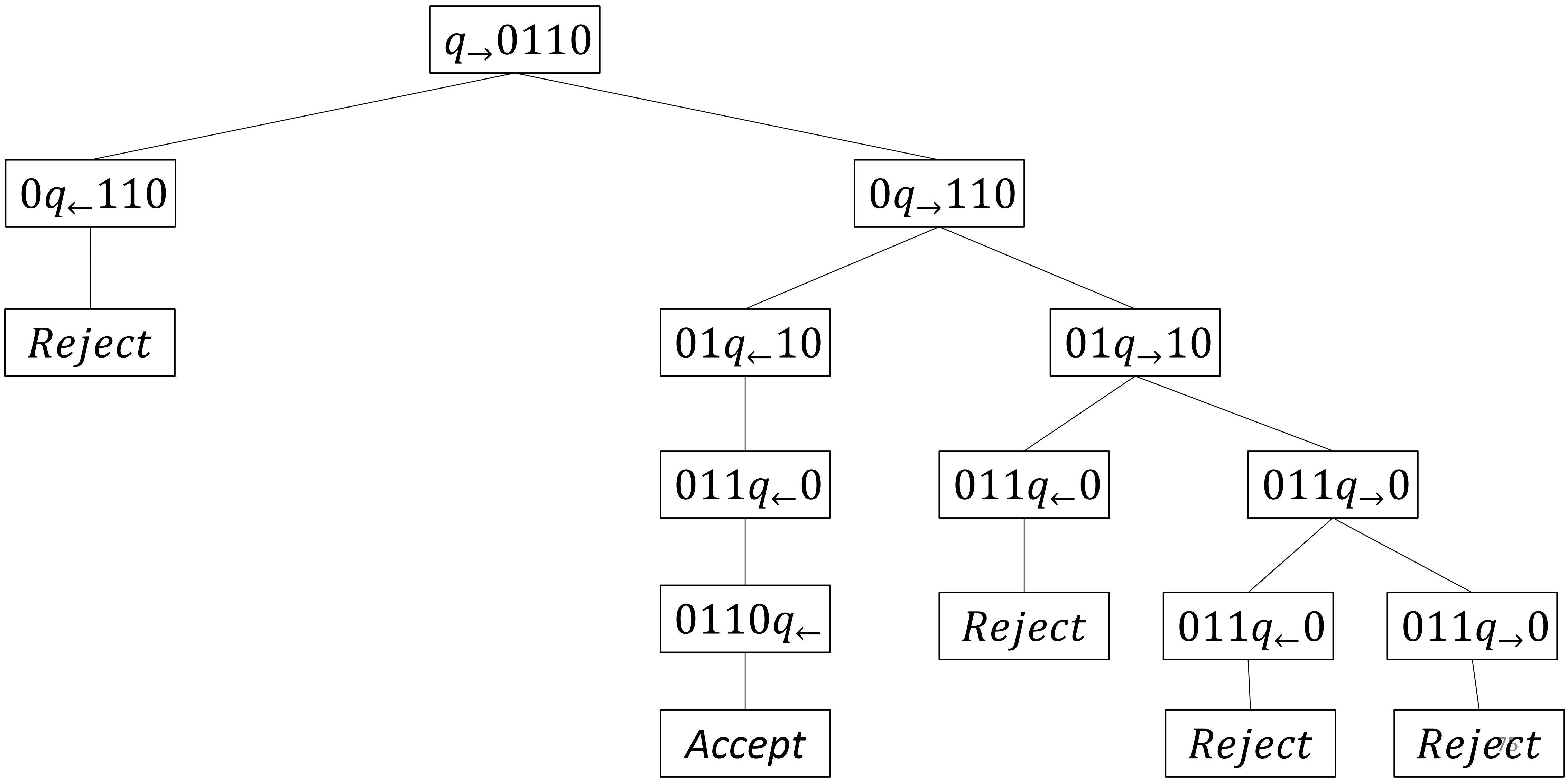
Arbre de résolution



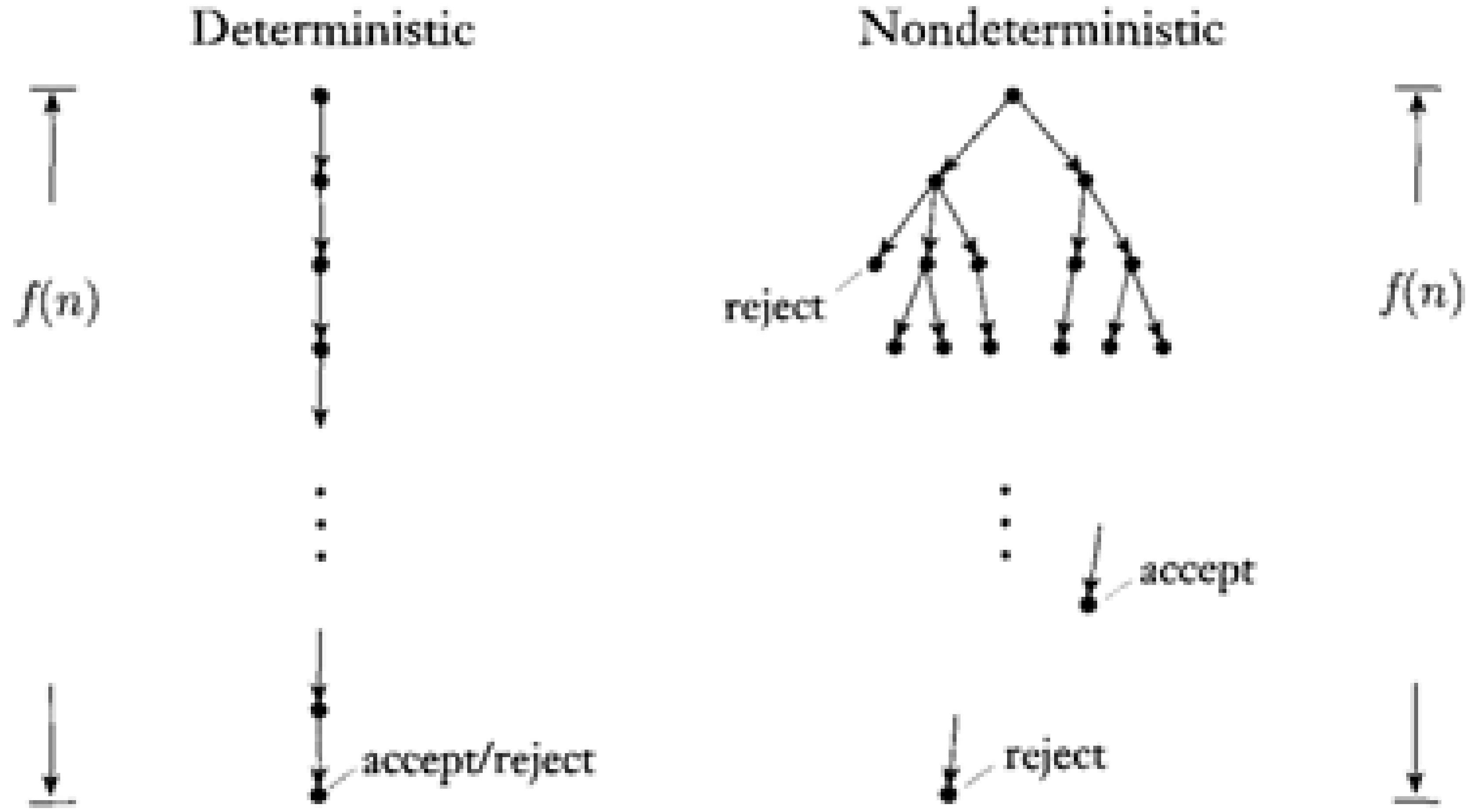
Arbre de résolution



Arbre de résolution



Arbre de résolution



Définition d’une classe de complexité

DTEMPS(f(n)) = famille des langages acceptés par une MT déterministe de complexité f(n)

NTEMPS(f(n)) = famille des langages acceptés par une MT non déterministe de complexité f(n)

Deux classes canoniques: P et NP

Deux classes canoniques: P et NP

$$P = \bigcup_{i \in N} DTEMPS(n^i)$$

Exemples:

- Problème de primalité
- Problème d'optimisation linéaire

Deux classes canoniques: P et NP

$$P = \bigcup_{i \in N} DTEMPS(n^i)$$

Exemples:

- Problème de primalité
- Problème d'optimisation linéaire

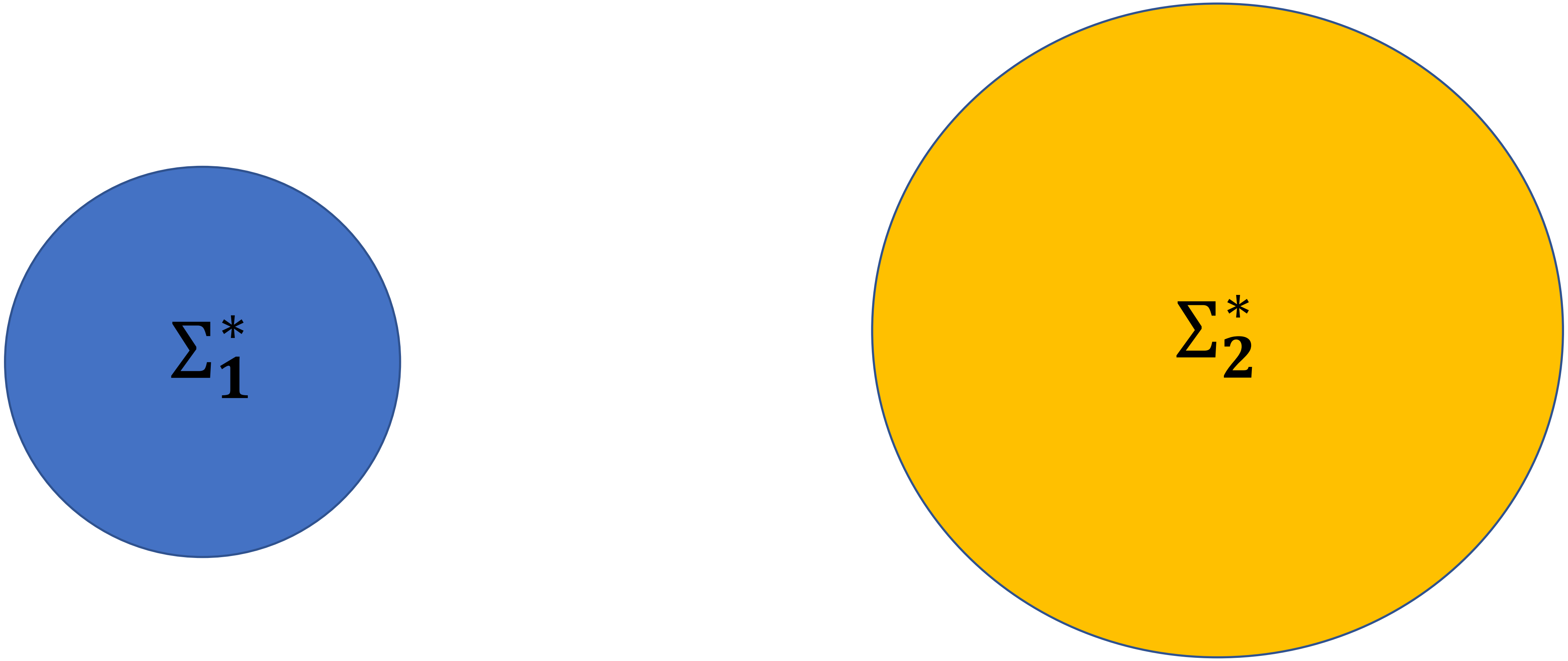
$$NP = \bigcup_{i \in N} NTEMPS(n^i)$$

Exemples:

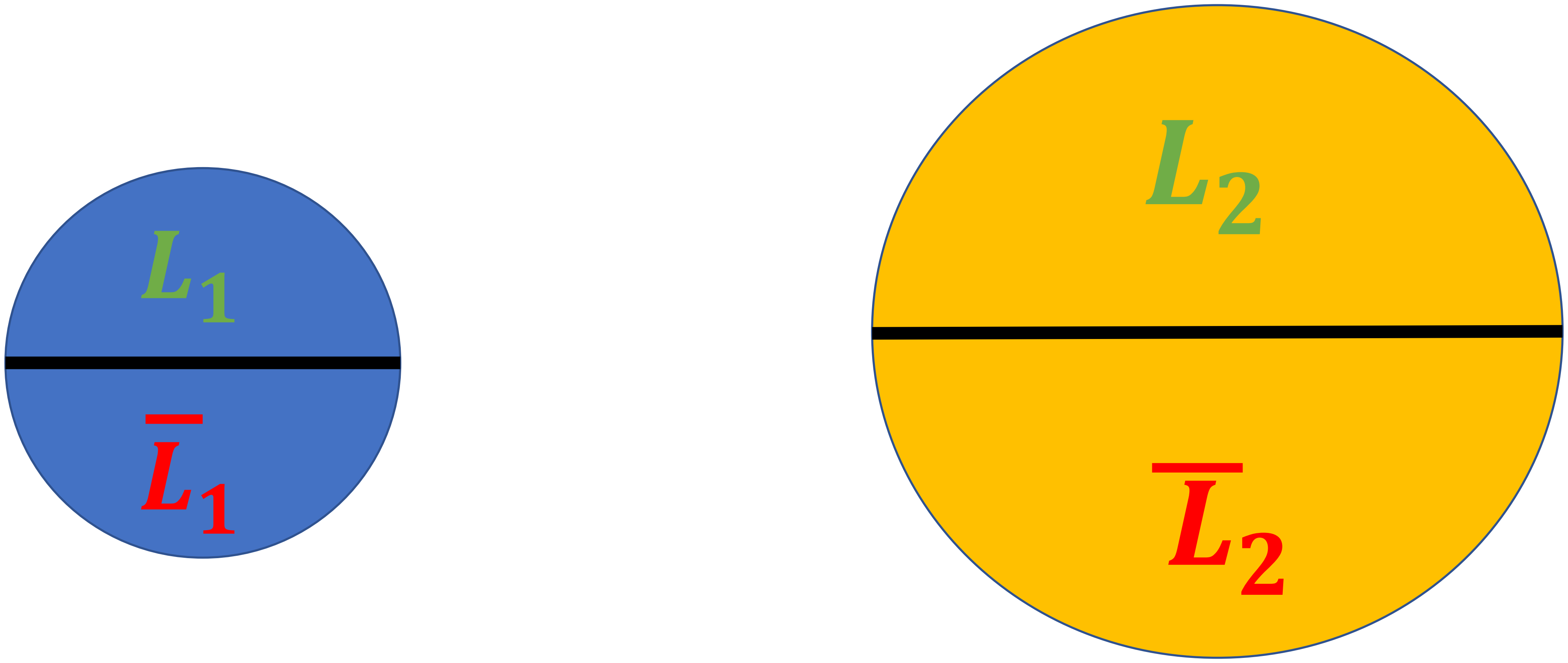
- Problème de factorisation
- Problème du logarithme discret

Réduction polynomiale

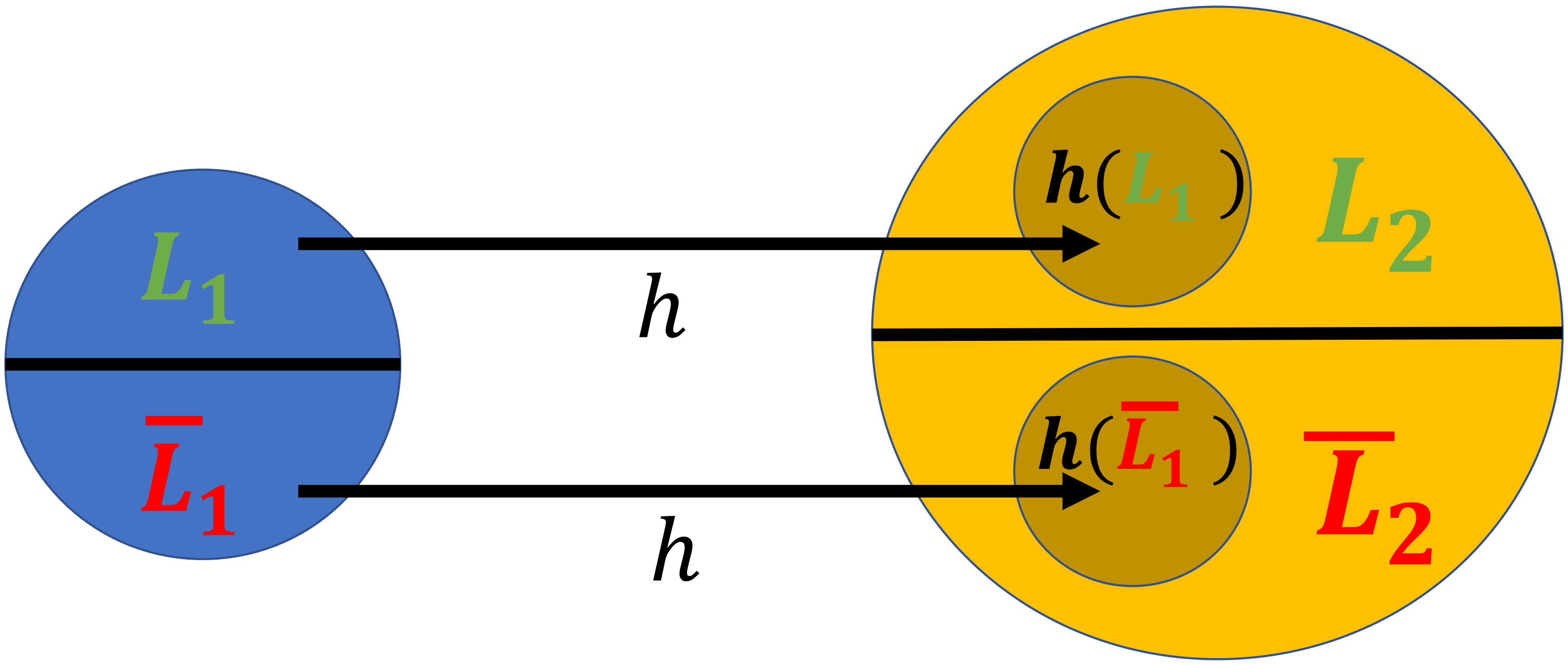
Réduction polynomiale



Réduction polynomiale



Réduction polynomiale



Réduction polynomiale

L_1 et L_2 sont deux langages

Réduction polynomiale

L_1 et L_2 sont deux langages

Une réduction de $\boldsymbol{L_1}$ **vers** $\boldsymbol{L_2}$ est une fonction h telle que:

$\boldsymbol{x \in L_1}$ si et seulement si $\boldsymbol{h(x) \in L_2}$

On dit alors que $\boldsymbol{L_1 \leq L_2}$

Réduction polynomiale

L_1 et L_2 sont deux langages

Une réduction de $\boldsymbol{L_1}$ **vers** $\boldsymbol{L_2}$ est une fonction h telle que:
$\boldsymbol{x} \in \boldsymbol{L_1}$ si et seulement si $\boldsymbol{h(x)} \in \boldsymbol{L_2}$

On dit alors que $\boldsymbol{L_1 \leq L_2}$

Si on peut calculer h en temps polynomial, savoir reconnaitre L_2 permet de reconnaitre L_1 avec la même complexité!

Problèmes NP complets

Problèmes NP complets

Definition:

Un problème est dit NP-complet si et seulement si le langage L correspondant satisfait les conditions suivantes:

1) L est dans NP

Problèmes NP complets

Définition:

Un problème est dit NP-complet si et seulement si le langage L correspondant satisfait les conditions suivantes:

1) L est dans NP

2) L est NP-difficile, i.e. :

$$\forall L' \in NP, L' \leq_P L$$

Problèmes NP complets

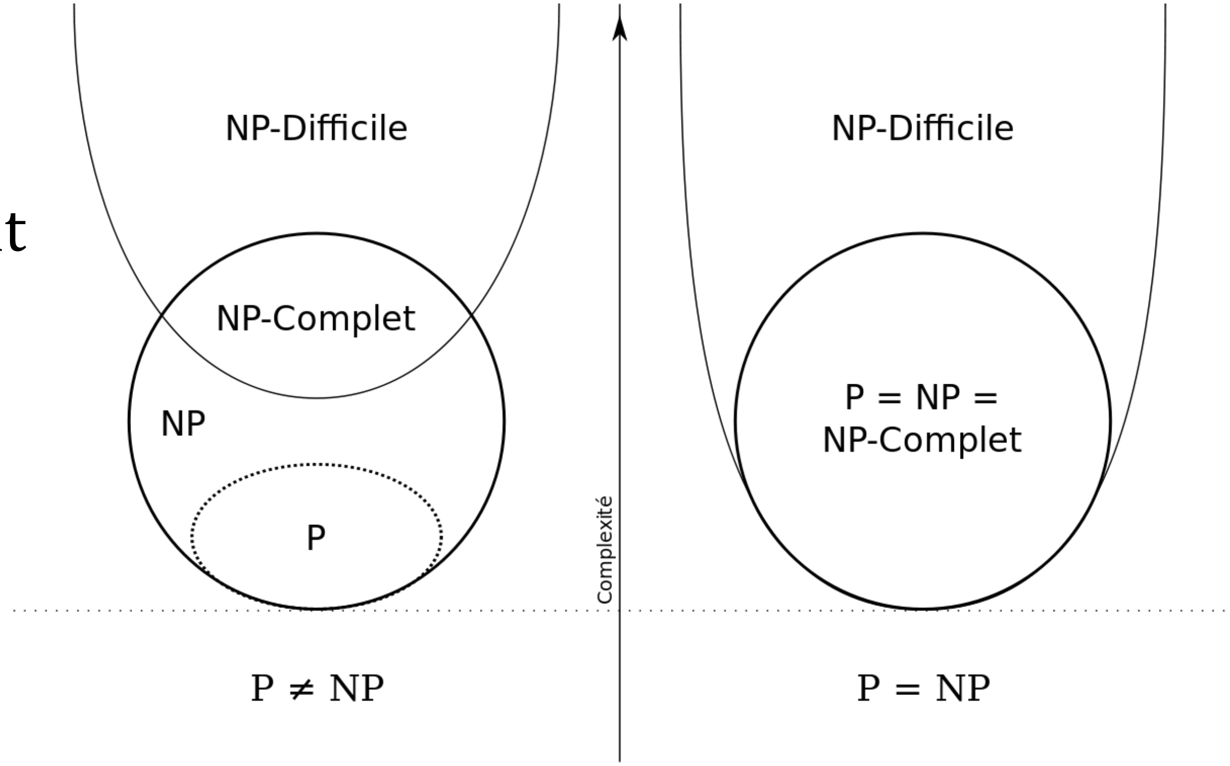
Definition:

Un problème est dit NP-complet si et seulement si le langage L correspondant satisfait les conditions suivantes:

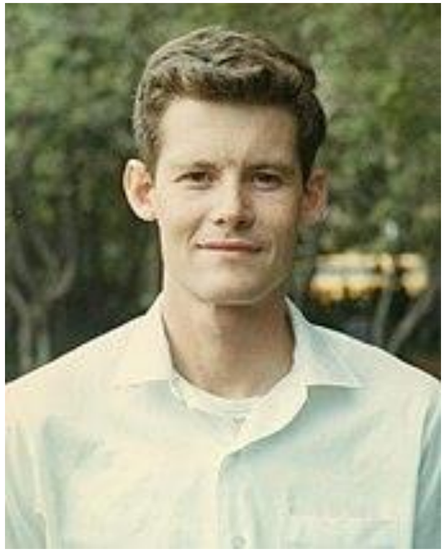
1) L est dans NP

2) L est NP-difficile, i.e. :

$$\forall L' \in NP, L' \leq_P L$$



Le théorème de Cook (1971)

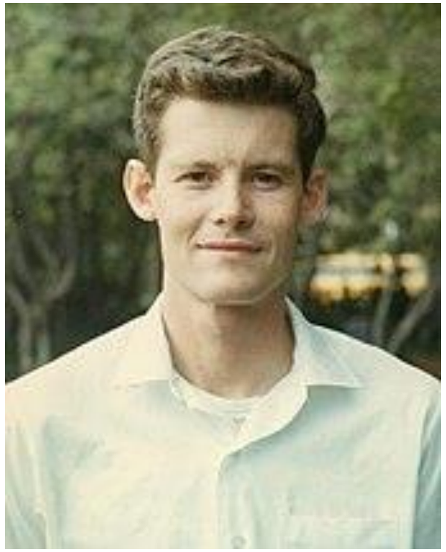


Stephen Cook en 1968

Enoncé:

Le problème de satisfiabilité SAT est NP complet.

Le théorème de Cook (1971)



Stephen Cook en 1968

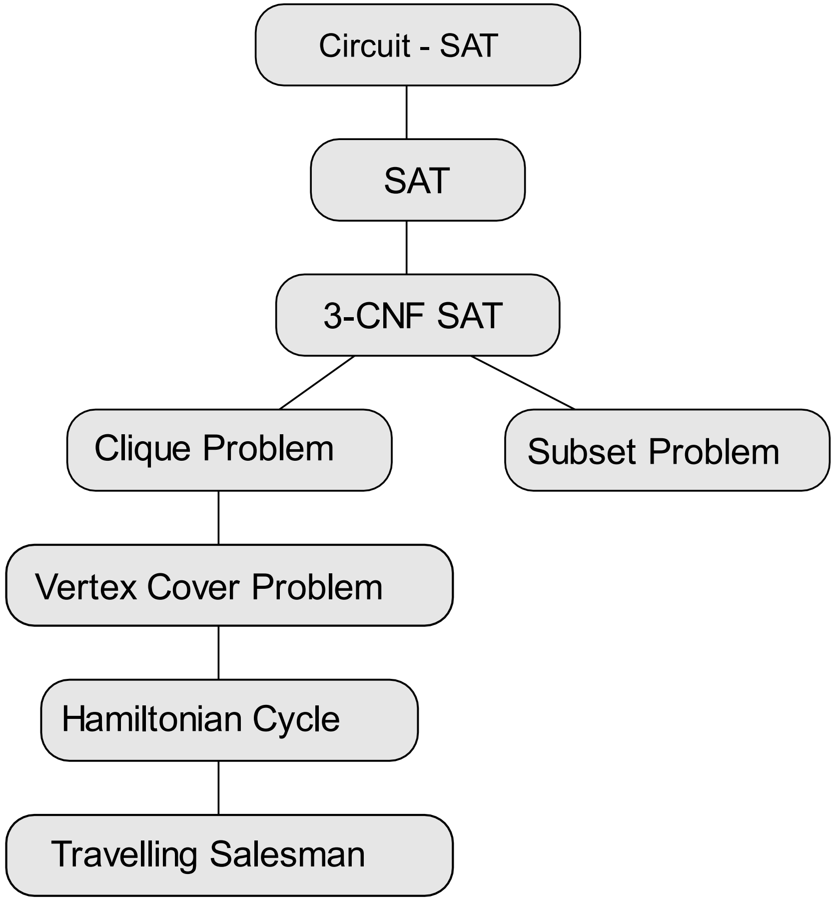
Enoncé:

Le problème de satisfiabilité SAT est NP complet.

Conséquences:

Si on trouve un algorithme polynomial qui résout SAT alors P=NP.

Permet d'exhiber plusieurs autres problèmes NPC



SAT le problème de satisfiabilité

Variable propositionnelle

Une variable propositionnelle est une variable qui peut prendre deux valeurs, 0 (False) ou 1 (True).

SAT le problème de satisfiabilité

Variable propositionnelle

Une variable propositionnelle est une variable qui peut prendre deux valeurs, 0 (False) ou 1 (True).

Littéral

Soit $U=\{u_1, u_2, ... , u_n\}$ un ensemble de variables propositionnelle.

Un littéral sur *U* est une variable prépositionnelle ou sa négation.

Exemple de littéral: $\neg u_2$

SAT le problème de satisfiabilité

Clause

Une clause c_1 est une disjonction de littéraux. On dit qu'une affectation de valeur de vérité $f: U \rightarrow \{0,1\}$ satisfait c_1 si un des littéraux de c_1 est évalué à 1 par f.

Exemple de clause: $\neg u_7 \vee \neg u_2 \vee u_3$

SAT le problème de satisfiabilité

Clause

Une clause c_1 est une disjonction de littéraux. On dit qu'une affectation de valeur de vérité $f: U \rightarrow \{0,1\}$ satisfait c_1 si un des littéraux de c_1 est évalué à 1 par f.

Exemple de clause: $\neg u_7 \vee \neg u_2 \vee u_3$

Ensemble de clauses

Une affectation de valeur de vérité f satisfait un ensemble de clauses C si f satisfait chacune des clauses de C.

SAT le problème de satisfiabilité

Données:

$U = \{u_1, u_2, \ldots., u_n\}$ un ensemble de variables propositionnelles.

$C = \{c_1, c_2, \ldots., c_k\}$ un ensemble de clauses.

SAT le problème de satisfiabilité

Données:

$U = \{u_1, u_2, \dots, u_n\}$ un ensemble de variables propositionnelles.

$C = \{c_1, c_2, \dots, c_k\}$ un ensemble de clauses.

Question:

Est-ce que C est satisfaisable?

SAT le problème de satisfiabilité

Données:

$U = \{u_1, u_2, \ldots. , u_n\}$ un ensemble de variables propositionnelles.

$C = \{c_1, c_2, \ldots. , c_k\}$ un ensemble de clauses.

Question:

Est-ce que C est satisfaisable?

Remarque: En notant $|C|$ le nombre de clauses et $|U|$ le nombre de vp, la taille de l'instance est en O($|C|$ $|U|$).

Une instance de SAT

Données:

$$U = \{p, q, r\}$$

$$C = \{p \vee \neg q \;,\; \mathrm{r} \vee q \;,\; \neg p \vee \neg r\}$$

Une instance de SAT

Données:

$$U = \{p, q, r\}$$

$$C = \{p \vee \neg q \ , \ \mathrm{r} \vee q \ , \ \neg p \ \vee \neg r\}$$

Réponse:

C est satisfaisable, par exemple avec l'affectation suivante:

$$p \mapsto 0$$

$$q \mapsto 0$$

$$r \mapsto 1$$

Universalité des ensembles de clause

Proposition:

Toute formule logique peut être reformulée en un ensemble de clauses.

Universalité des ensembles de clause

Proposition:

Toute formule logique peut être reformulée en un ensemble de clauses.

Exemples:

$$p \Rightarrow q$$

Universalité des ensembles de clause

Proposition:

Toute formule logique peut être reformulée en un ensemble de clauses.

Exemples:

$$p \Rightarrow q$$

$$\{\neg p \vee q\}$$

Universalité des ensembles de clause

Proposition:

Toute formule logique peut être reformulée en un ensemble de clauses.

Exemples:

$$p \land q$$

Universalité des ensembles de clause

Proposition:

Toute formule logique peut être reformulée en un ensemble de clauses.

Exemples:

$$p \land q$$

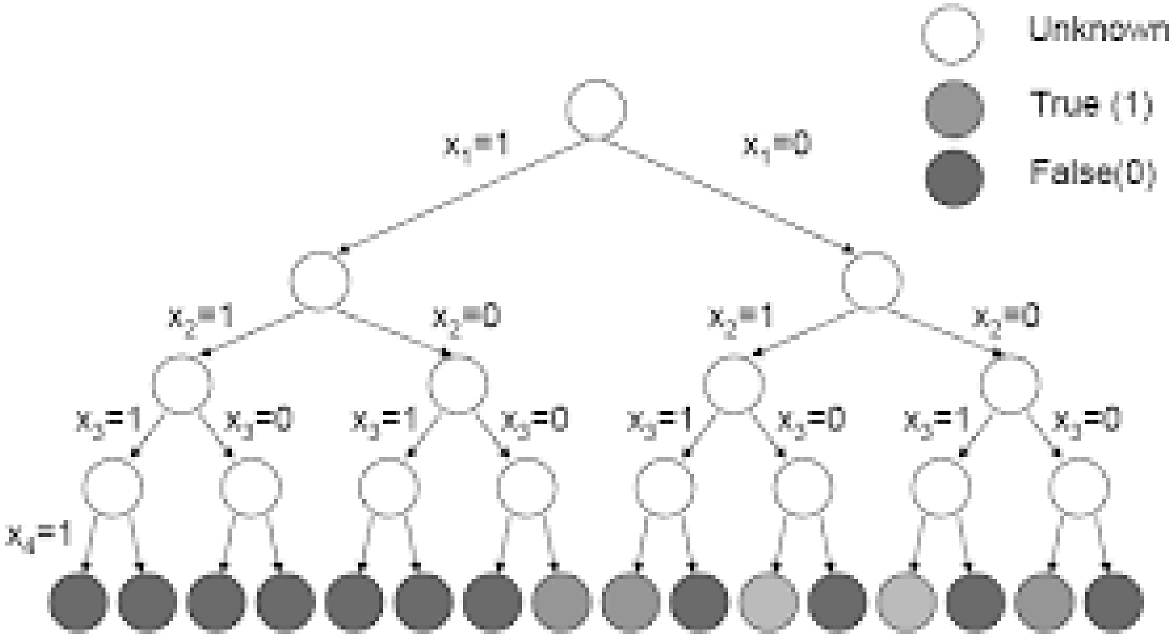
$$\{p, q\}$$

SAT est dans NP

SAT est dans NP

Première étape:

On parcourt *U* et on associe à chaque variable une valeur de vérité de manière non déterministe. On stocke cette application sur un deuxième ruban.



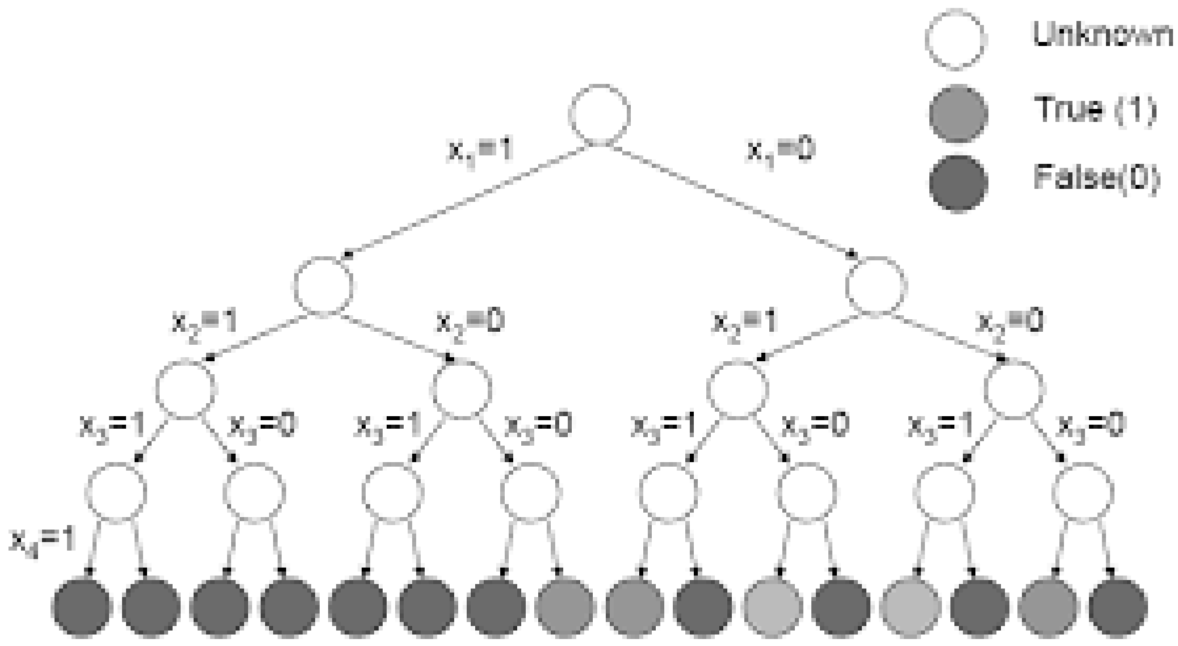
SAT est dans NP

Première étape:

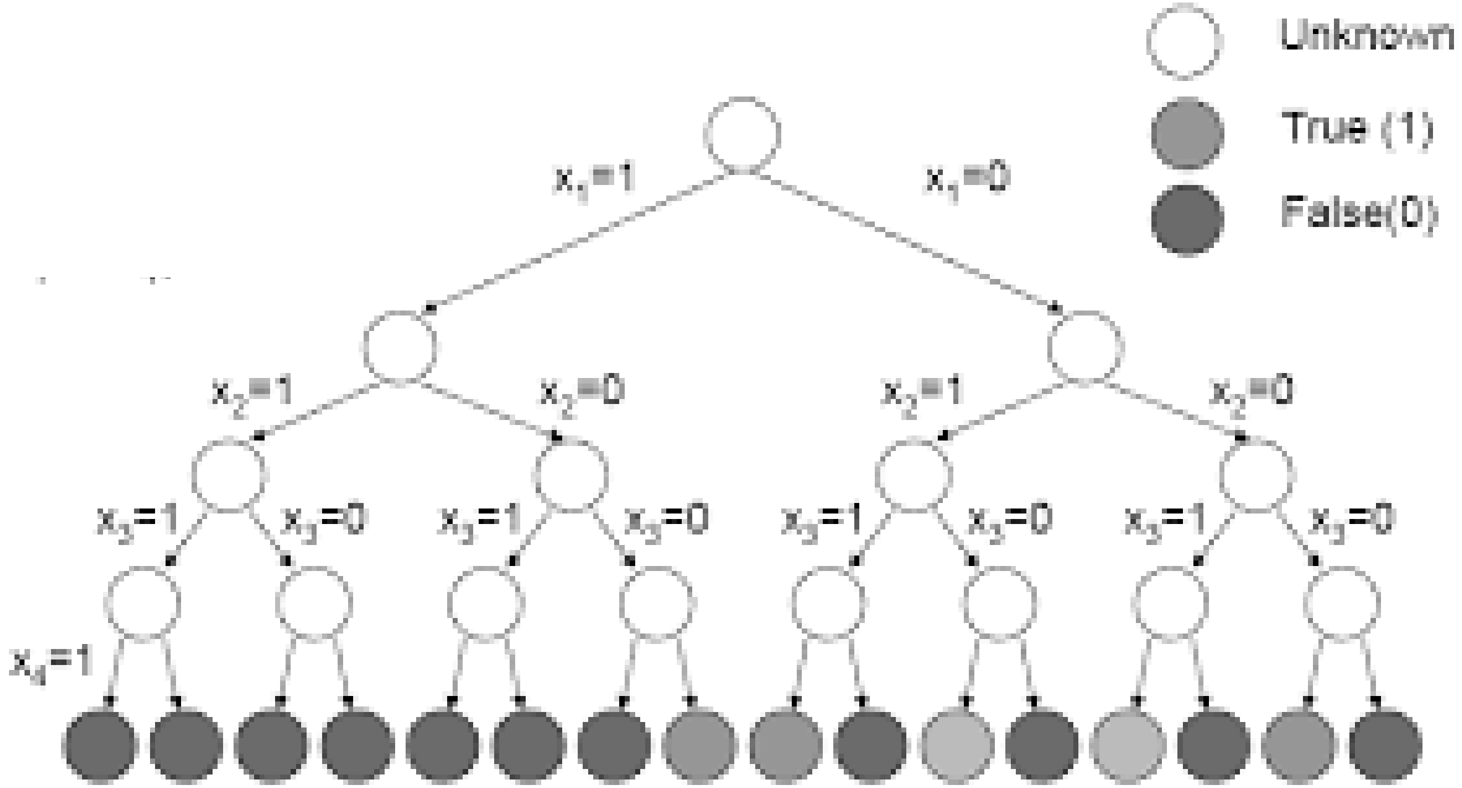
On parcourt *U* et on associe à chaque variable une valeur de vérité de manière non déterministe. On stocke cette application sur un deuxième ruban.

Deuxième étape:

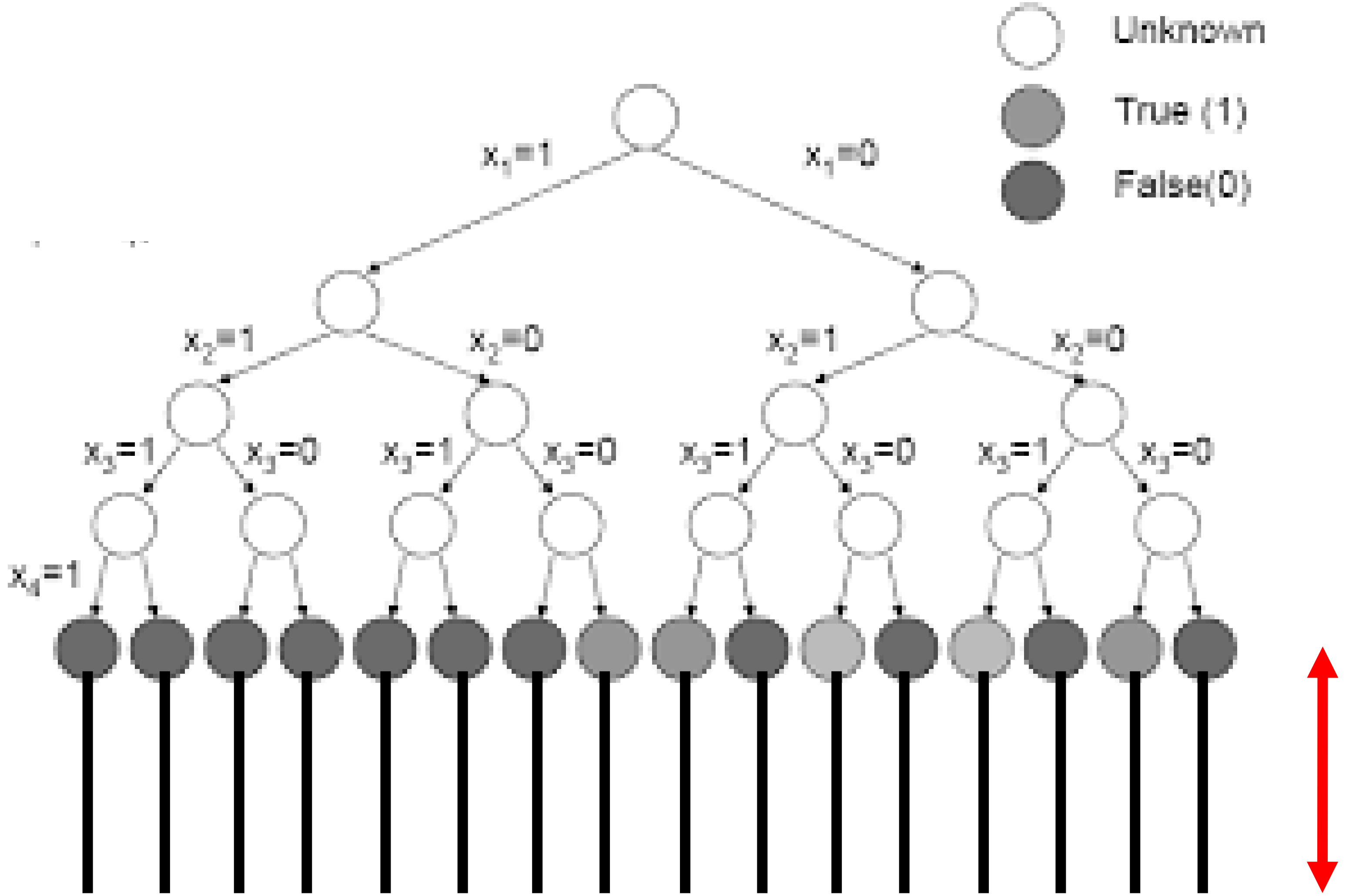
Dans *C* on remplace les variables par leur valeur, puis on vérifie en temps polynomial si toutes les clauses sont satisfaites.



SAT est dans NP

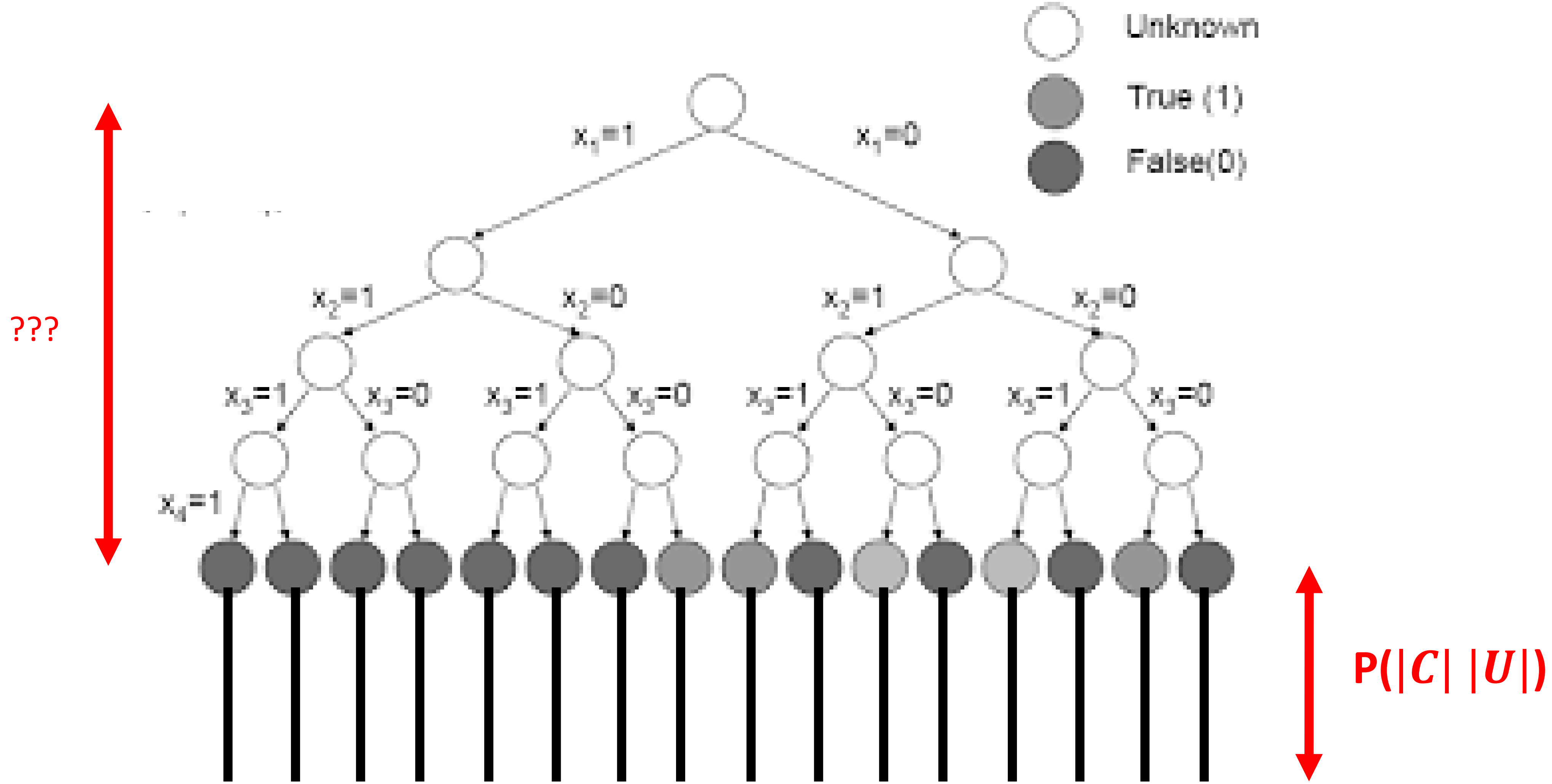


SAT est dans NP

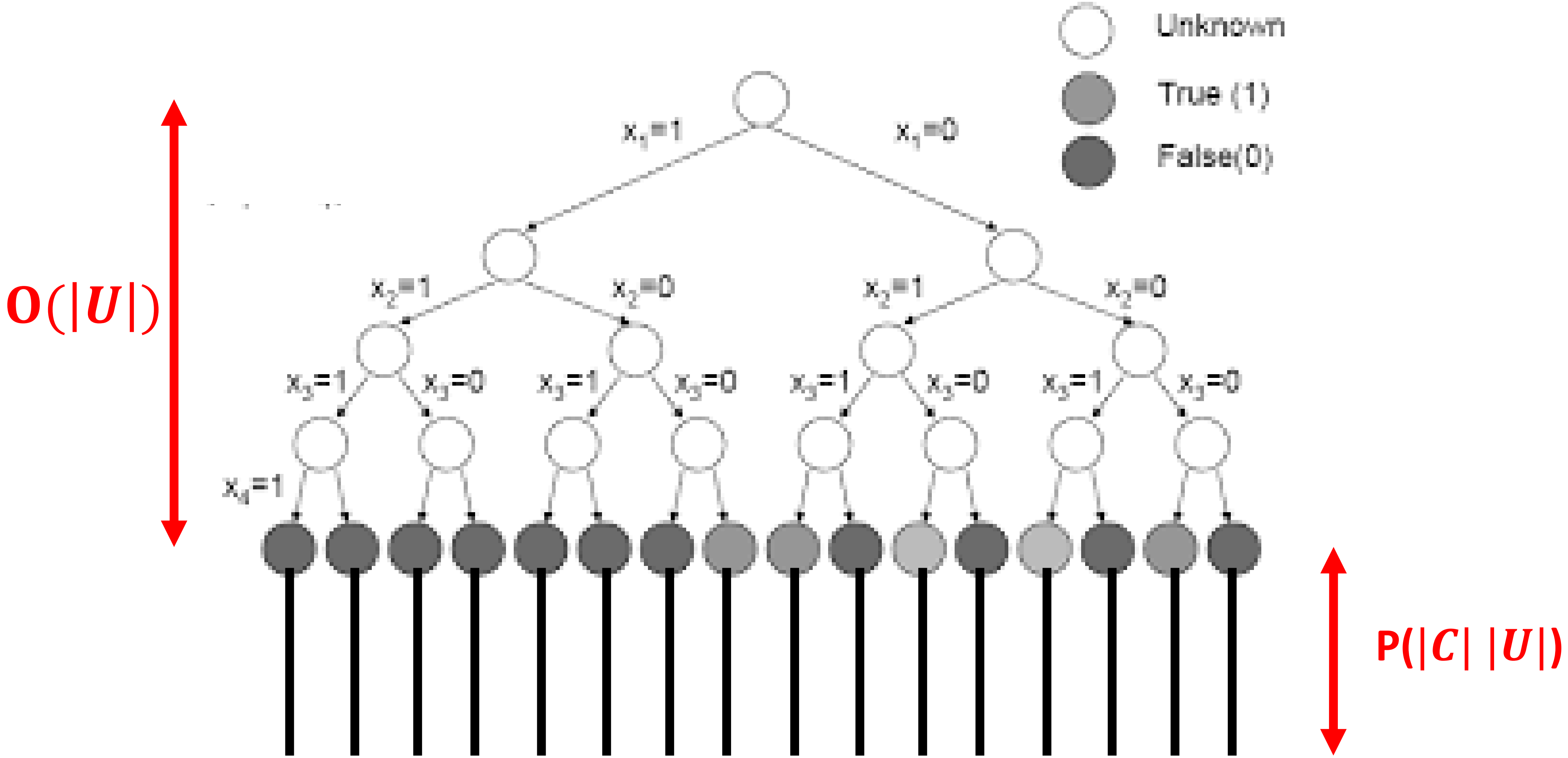


$\mathbf{P}(|C|\ |U|)$

SAT est dans NP



SAT est dans NP



SAT est NP-hard

SAT est NP-hard

Soit L∈NP, on cherche à montrer que $\mathrm{L} \leq_p SAT$

SAT est NP-hard

Soit L∈NP, on cherche à montrer que $\mathrm{L} \leq_p SAT$

Par définition de NP, on dispose de M non déterministe telle que:

1) $\mathrm{L(M) = L}$

2) M calcule en temps borné par n^k

SAT est NP-hard

Soit L∈NP, on cherche à montrer que $\mathrm{L} \leq_p SAT$

Par définition de NP, on dispose de M non déterministe telle que:

1) $\mathrm{L(M) = L}$

2) M calcule en temps borné par n^k

On va construire la fonction de réduction h_M, qui associe à chaque entrée possible sur M, une instance de SAT.

SAT est NP-hard

Soit L∈NP, on cherche à montrer que $\mathrm{L} \leq_p SAT$

Par définition de NP, on dispose de M non déterministe telle que:

1) $\mathrm{L(M) = L}$

2) M calcule en temps borné par n^k

On va construire la fonction de réduction h_M, qui associe à chaque entrée possible sur M, une instance de SAT.

Soit ω une entrée de M de taille n, on procède ainsi:

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état $\boldsymbol{q}$ à l'instant $\boldsymbol{i}$

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état ***q*** à l'instant ***i***

Les $H_{i,c}$ indiquent si la tête est sur la case ***c*** à l'instant ***i***

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état ***q*** à l'instant ***i***

Les $H_{i,c}$ indiquent si la tête est sur la case ***c*** à l'instant ***i***

Les $S_{i,c,s}$ indiquent si la case ***c*** contient le symbole ***s*** à l'instant ***i***

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état ***q*** à l'instant ***i***

Nombre de $\mathrm{Q}_{\mathrm{i,q}}$: $|Q| \times n^k$

Les $H_{i,c}$ indiquent si la tête est sur la case ***c*** à l'instant ***i***

Les $S_{i,c,s}$ indiquent si la case ***c*** contient le symbole ***s*** à l'instant ***i***

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état ***q*** à l'instant ***i***

Nombre de $\mathrm{Q}_{\mathrm{i,q}}$: $|Q| \times n^k$

Les $H_{i,c}$ indiquent si la tête est sur la case ***c*** à l'instant ***i***

Nombre de $H_{i,c}$: $2n^k \times n^k = 2n^{2k}$

Les $S_{i,c,s}$ indiquent si la case ***c*** contient le symbole ***s*** à l'instant ***i***

SAT est NP-hard

Les variables propositionnelles:

Les $Q_{i,q}$ indiquent si M est dans l'état ***q*** à l'instant ***i***

Nombre de $\mathrm{Q_{i,q}}$: $|Q| \times n^k$

Les $H_{i,c}$ indiquent si la tête est sur la case ***c*** à l'instant ***i***

Nombre de $H_{i,c}$: $2n^k \times n^k = 2n^{2k}$

Les $S_{i,c,s}$ indiquent si la case ***c*** contient le symbole ***s*** à l'instant ***i***

Nombre de $S_{i,c,s}$: $2n^k \times |\Sigma| \times n^k = 2|\Sigma|n^{2k}$

SAT est NP-hard

Une première famille de clauses: Unicité de l’état

SAT est NP-hard

Une première famille de clauses: Unicité de l'état

Pour chaque valeur de i entre 0 et n^k on ajoute à C les clauses:

$Q_{i,q} \Rightarrow \neg Q_{i,q'}$ pour tout ***q*** différent de ***q'***

SAT est NP-hard

Une première famille de clauses: Unicité de l'état

Pour chaque valeur de i entre 0 et n^k on ajoute à C les clauses:

$Q_{i,q} \Rightarrow \neg Q_{i,q'}$ pour tout ***q*** différent de ***q'***

Nombre de clauses: $O(n^k)$

SAT est NP-hard

Deuxième clauses: Unicité de la tête de lecture

SAT est NP-hard

Deuxième clauses: Unicité de la tête de lecture

Pour chaque valeur de i entre 0 et n^k on ajoute à C les clauses:

$H_{i,c} \Rightarrow \neg H_{i,c'}$ pour tout ***c*** différent de ***c'***

SAT est NP-hard

Deuxième clauses: Unicité de la tête de lecture

Pour chaque valeur de i entre 0 et n^k on ajoute à C les clauses:

$H_{i,c} \Rightarrow \neg H_{i,c'}$ pour tout ***c*** différent de ***c'***

Nombre de clauses: $O(n^{3k})$

SAT est NP-hard

Troisième clauses: Unicité du contenu des cases

SAT est NP-hard

Troisième clauses: Unicité du contenu des cases

Pour chaque valeur de i entre 0 et n^k et chaque valeur de c entre -n^k et n^k, on ajoute à C les clauses:

$S_{i,c,s} \Rightarrow \neg S_{i,c,s'}$ pour tout ***s*** différent de ***s'***

SAT est NP-hard

Troisième clauses: Unicité du contenu des cases

Pour chaque valeur de i entre 0 et n^k et chaque valeur de c entre -n^k et n^k, on ajoute à C les clauses:

$S_{i,c,s} \Rightarrow \neg S_{i,c,s'}$ pour tout ***s*** différent de ***s'***

Nombre de clauses: $O(n^{2k})$

SAT est NP-hard

Quatrième clauses: Initialisation

On fixe l'état initial et la position de la tête de lecture initiale en ajoutant les clauses Q_{0,q_0} et $H_{0,0}$.

SAT est NP-hard

Quatrième clauses: Initialisation

On fixe l'état initial et la position de la tête de lecture initiale en ajoutant les clauses Q_{0,q_0} et $H_{0,0}$.

Puis on fixe le contenu initial de la bande en ajoutant pour tout $\boldsymbol{c}$ entre $-\mathrm{n}^{\mathrm{k}}$ et n^{k} la clause $S_{0,s,c}$ pour un s bien choisi

SAT est NP-hard

Quatrième clauses: Initialisation

On fixe l'état initial et la position de la tête de lecture initiale en ajoutant les clauses Q_{0,q_0} et $H_{0,0}$.

Puis on fixe le contenu initial de la bande en ajoutant pour tout $\boldsymbol{c}$ entre $-\mathrm{n}^{\mathrm{k}}$ et n^{k} la clause $S_{0,s,c}$ pour un s bien choisi

Nombre de clauses: $O(n^k)$

SAT est NP-hard

Cinquième clauses: Etats finaux

On ajoute la clause:

$$\bigvee_{q_f \in Q_F} Q_{n^k, q_f}$$

SAT est NP-hard

Cinquième clauses: Etats finaux

On ajoute la clause:

$$\bigvee_{q_f \in Q_F} Q_{n^k, q_f}$$

Nombre de clauses: $\boldsymbol{O(1)}$

SAT est NP-hard

Sixième clauses: Transitions

On force maintenant notre machine virtuelle à effectuer une des transitions possibles à chaque incrémentation de i.

SAT est NP-hard

Sixième clauses: Transitions

On force maintenant notre machine virtuelle à effectuer une des transitions possibles à chaque incrémentation de i.

On procède en ajoutant pour tout i entre 0 et n^k-1, et pour tout c entre $-n^k$ et n^k:

$$\bigvee_{q_f \in Q_F} (Q_{i,q} \wedge H_{i,c} \wedge S_{i,s,c}) \Rightarrow (Q_{i+1,q'} \wedge H_{i+1,c+d} \wedge S_{i+1,s',c})$$

Pour toute élément $(\mathrm{q}\,,\mathrm{s}) \times (\mathrm{q}', \mathrm{s}', \mathrm{d})$ dans la relation de transition.

SAT est NP-hard

Sixième clauses: Transitions

On force maintenant notre machine virtuelle à effectuer une des transitions possibles à chaque incrémentation de i.

On procède en ajoutant pour tout i entre 0 et n^k-1, et pour tout c entre $-n^k$ et n^k:

$$\bigvee_{q_f \in Q_F} (Q_{i,q} \wedge H_{i,c} \ \wedge S_{i,s,c}) \Rightarrow (Q_{i+1,q'} \wedge H_{i+1,c+d} \ \wedge S_{i+1,s',c})$$

Pour toute élément $(\mathrm{q}\,,\mathrm{s}) \times (\mathrm{q}', \mathrm{s}', \mathrm{d})$ dans la relation de transition.

Il faudrait également des clauses pour garantir que les cases ne s'écrivent pas toutes seules.

SAT est NP-hard

Sixième clauses: Transitions

On force maintenant notre machine virtuelle à effectuer une des transitions possibles à chaque incrémentation de i.

On procède en ajoutant pour tout i entre 0 et n^k-1, et pour tout c entre $-n^k$ et n^k:

$$\bigvee_{q_f \in Q_F} (Q_{i,q} \wedge H_{i,c} \ \wedge S_{i,s,c}) \Rightarrow (Q_{i+1,q'} \wedge H_{i+1,c+d} \ \wedge S_{i+1,s',c})$$

Pour toute élément $(\mathrm{q}\,,\mathrm{s}) \times (\mathrm{q}', \mathrm{s}', \mathrm{d})$ dans la relation de transition.

Il faudrait également des clauses pour garantir que les cases ne s’écrivent pas toutes seules.

Nombre de clauses: $\boldsymbol{O(n^{2k})}$

SAT est NP-hard

Conclusion

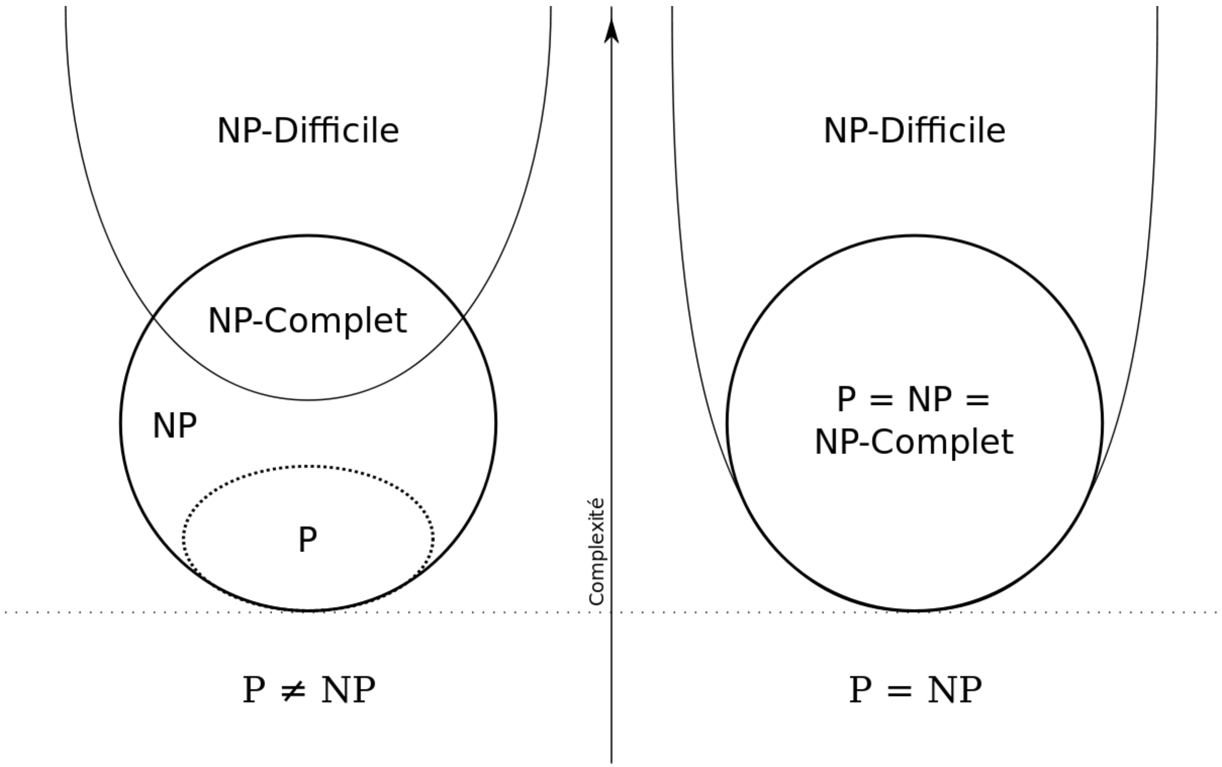
On a explicité la fonction h_M et on a bien:

1) $h_M(w)$ est une instance acceptante si *w* est une instance acceptante

2) *w* est une instance acceptante si $h_M(w)$ est une instance acceptante

P=NP?

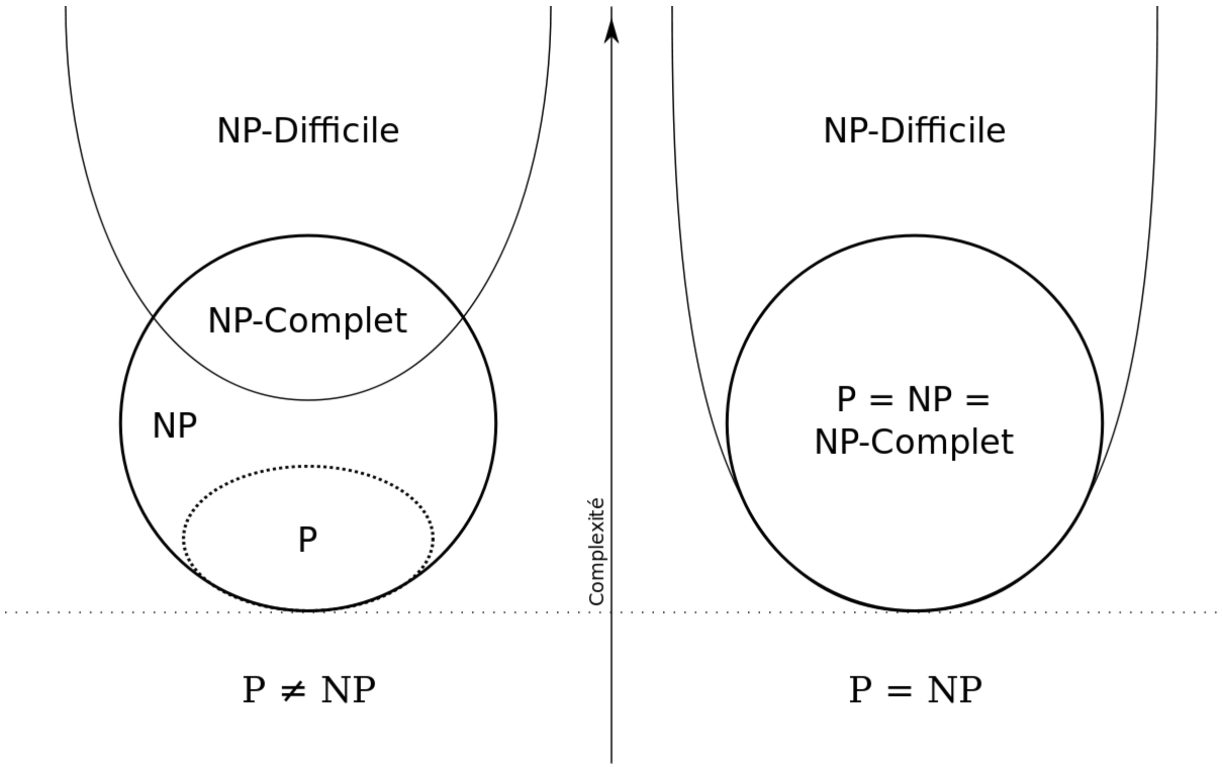
Supposons qu'on est capable de
résoudre SAT en temps polynomial.



P=NP?

Supposons qu’on est capable de résoudre SAT en temps polynomial.

Puisque SAT est NPC tous les problèmes de NP se réduisent à SAT en temps polynomial.

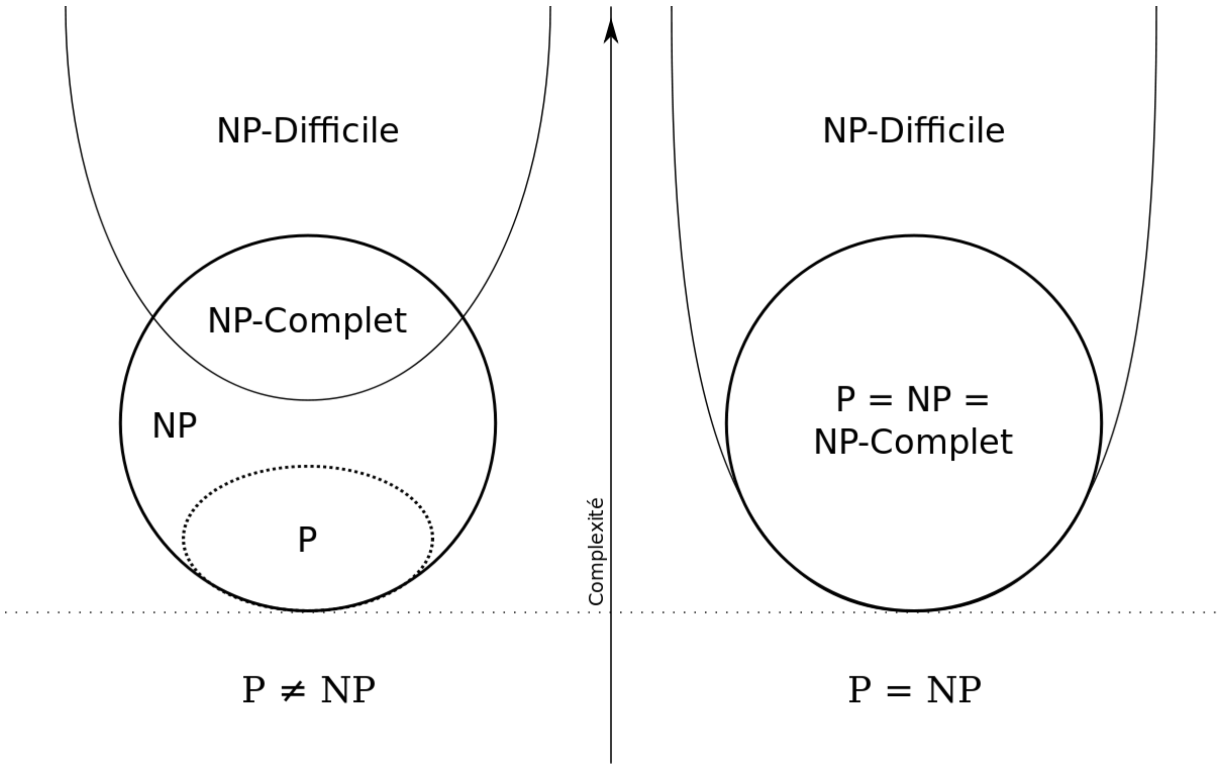


P=NP?

Supposons qu'on est capable de résoudre SAT en temps polynomial.

Puisque SAT est NPC tous les problèmes de NP se réduisent à SAT en temps polynomial.

On est donc capable de résoudre tous les problèmes de NP en temps polynomial, et donc P=NP.



Shor et le problème de factorisation

$$\textsf{FACTORING} := \{\langle N, L, U\rangle : \exists \text{prime } p \text{ s.t. } p \mid N, L \le p \le U\}.$$

That is, the factoring decision problem is to decide whether N has a prime factor between L and U.

The factoring search problem is: Given $N \geq 2$, find primes $p_1, \ldots, p_n$ (not necessarily distinct) such that $p_1 \cdot \ldots \cdot p_n = N$.

Shor et le problème de factorisation

$$\text{FACTORING} := \{\langle N, L, U\rangle : \exists \text{prime } p \text{ s.t. } p \mid N, L \le p \le U\}.$$

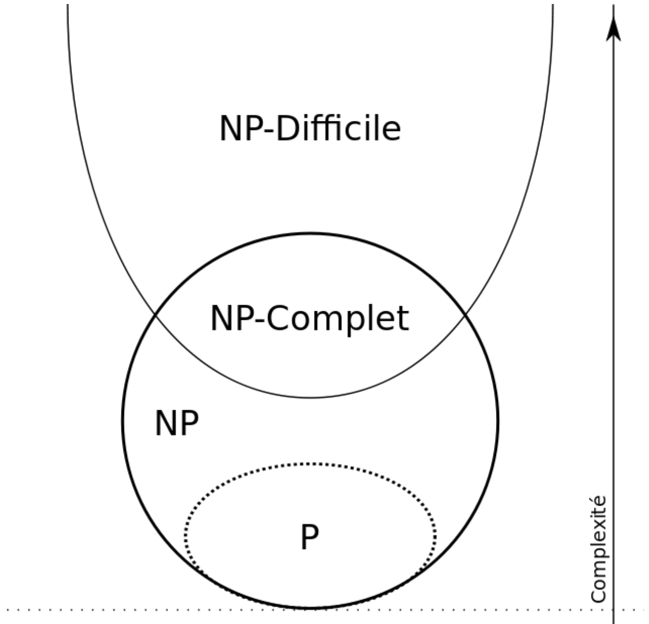
That is, the factoring decision problem is to decide whether N has a prime factor between L and U.

The factoring search problem is: Given $N \ge 2$, find primes $p_1, \ldots, p_n$ (not necessarily distinct) such that $p_1 \cdot \ldots \cdot p_n = N$.

Deux problèmes NPI importants:

Le problème de factorisation (RSA)

Le problème du logarithme discret (Diffie Hellman)



Complexité quantique