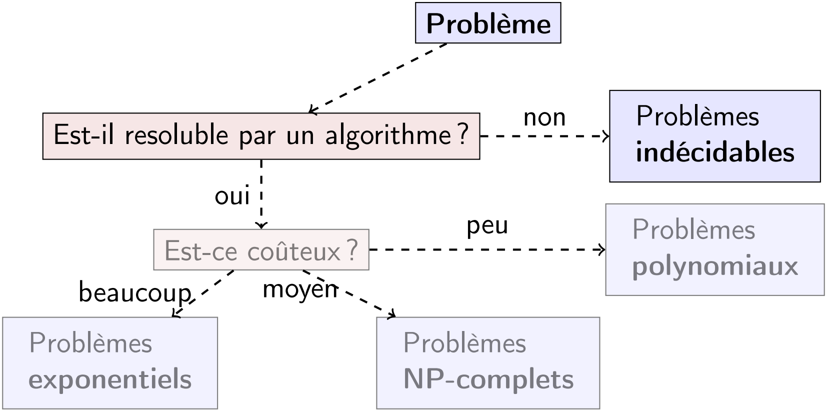
Calculabilité et NP-Complétude

Partie II : Classes de complexité

Master 2

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)

Plan

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Un problème est dans la classe **P** (polynomial) s'il existe une machine de Turing **déterministe** polynomiale le décidant.

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

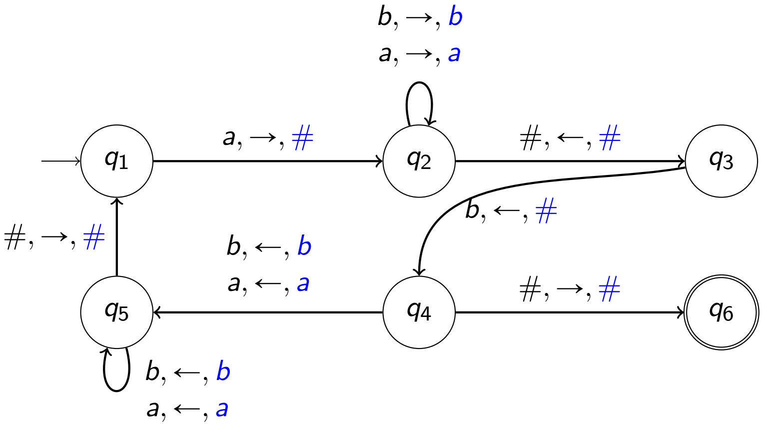
Un problème est dans la classe **P** (polynomial) s'il existe une machine de Turing **déterministe** polynomiale le décidant.
Un problème est dans la classe **P** (polynomial) s'il existe un programme de complexité polynomiale pour le résoudre.

Exemple 1

On considère sur $\{a, b\}$ le langage des mots de a^*b^* ayant autant de a que de b.

Exemple 1

On considère sur $\{a, b\}$ le langage des mots de a^*b^* ayant autant de a que de b.



Le calcul maximal a une longueur de l'ordre de $|w|^2$. La machine est déterministe polynomiale. Le problème est dans **P**.

Exemple 2

On considère sur $\{a, b\}$ le langage des mots ayant autant de a que de b.

```
def NaNb(w): #w est une chaine ne comportant que des a et des b
    compteur=0
    for x in w :
        if x=='a':
            compteur+=1
        else :
            compteur-=1
return (compteur == 0)
```

Le nombre d'opérations est proportionel à la longueur de w. Le problème est dans **P**.

Exemple 3

On s'interesse au problème suivant : *Etant donné un entier en binaire, est-il premier ?*

```
def Prime(n):
    for x in range(2,n-1):
        if n%x == 0 :
            return False
    return True
```

- L'algorithme précédent est-il polynomial ?

Exemple 3

On s'interesse au problème suivant : *Etant donné un entier en binaire, est-il premier ?*

```
def Prime(n):
    for x in range(2,n-1):
        if n%x == 0 :
            return False
    return True
```

- L'algorithme précédent est-il polynomial ?
- Quelle est la taille de l'entrée (en fonction de n) ?
- Combien y a-t-il d'instructions (en fonction de n) ?
- L'algorithme précédent est-il polynomial ?

EXPTIME

Définition

Un problème est dans **EXPTIME** s'il existe une machine de Turing déterministe qui le décide en temps $O(2^{P(n)})$ où n est la taille de l'entrée et P un polynôme (qui ne dépend que du problème).

On peut définir des complexités encore plus élevées. On sait que :

$$\mathbf{P} \subsetneq \mathbf{EXPTIME} \subseteq \text{Décidables}$$

.

NP

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

NP

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Un problème est dans la classe **NP** (polynomial non-déterministe) s'il existe une machine de Turing **non déterministe** polynomiale le décidant.

Exemple

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbabababaaa*.

Exemple

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababab aaa*.

Réfléchir à un aglorithme (au sens classique) pour ce problème. Quelle est sa complexité ?

Exemple (machine de Turing)

Données $w_0\$w_1\$\dots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

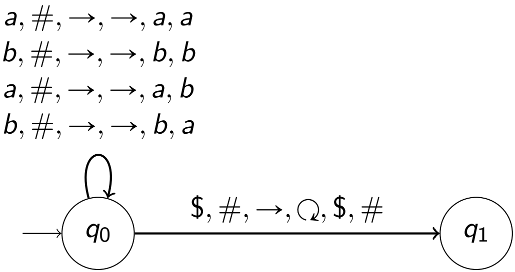
Trouver une machine non déterminste polynomiale.

Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

On commence par lire w_0 et on écrit de façon non déterministe a ou b sur le second ruban. On obtient un mot x de la taille de w_0.

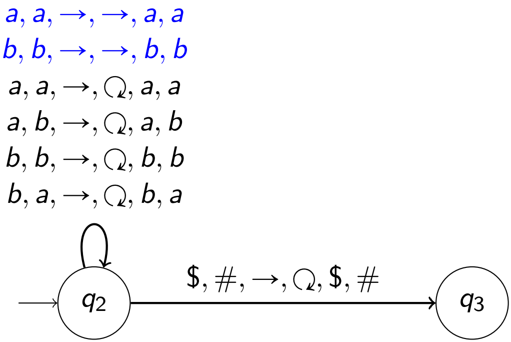


Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

On vérifie que le mot x sur le second ruban est sous-mot de tous les autres.



Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

La machine est non-déterministe polynomiale, le problème est dans **NP**.

Inclusions

On peut démontrer que les machines de Turing sont déterminisables avec un coût exponentiel.

On sait alors que :

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME} \subseteq \text{Décidables}$$

.
Ces classes sont indépendants du nombre de rubans (et même en pratique du modèle de calcul utilisé). La grande question est :

$$\mathsf{P{=}NP\,?}$$

Exercice

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- Chaque w_i pour $i > 0$ est un sous mot de x ?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababababaaa*.

Exercice

Données $w_0\$w_1\$\dots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- Chaque w_i pour $i > 0$ est un sous mot de x ?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababaaa*.

Décrire une machine de Turing non-déterministe polynomiale pour répondre à ce problème.

Plan

Machine de Turing à entrée/sortie

Machines à entrées/sorties

Une machine de Turing à entrée/sortie est une machine qui possède

- Un ruban d'entrée en lecture seule (on ne peut pas réécrire),
- Un ruban de sortie en écriture seule (on ne peut se déplacer que vers la droite),
- des rubans de travail (éventuellement aucun) sans restriction.

L'**espace** utilisé pour un calcul maximal d'une machine à entrée-sortie est les maximum des somme des longeurs des mots écrits sur les rubans de travail lors de ce calcul (on compte les # si la tête de lecture se déplace sans écrire).

PSPACE, NPSPACE, NLOGSPACE

- **PSPACE** : machine de Turing déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en la taille de l'entrée.
- **NPSPACE** : même chose sans déterminisme.
- **NLOGSPACE** : machine de Turing non-déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en le logarithme de la taille de l'entrée.

Exemple de langage **NLOGSPACE** : $\{a^n b^n \mid n > 0\}$

PSPACE, NPSPACE, NLOGSPACE

- **PSPACE** : machine de Turing déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en la taille de l'entrée.
- **NPSPACE** : même chose sans déterminisme.
- **NLOGSPACE** : machine de Turing non-déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en le logarithme de la taille de l'entrée.

Exemple de langage **NLOGSPACE** : $\{a^n b^n \mid n > 0\}$

On reverra **NLOGSPACE** dans la section *Algorithmique non-déterministe*.

Inclusions

Il faut du temps pour occuper l'espace :

$$\mathbf{P} \subseteq \mathbf{PSPACE} \quad \text{et} \quad \mathbf{NP} \subseteq \mathbf{NPSPACE}$$

Inclusions

Il faut du temps pour occuper l'espace :

$$\mathsf{P} \subseteq \mathsf{PSPACE} \quad \text{et} \quad \mathsf{NP} \subseteq \mathsf{NPSPACE}$$

Théorème de Savitch

$\mathsf{PSPACE} = \mathsf{NPSPACE}$

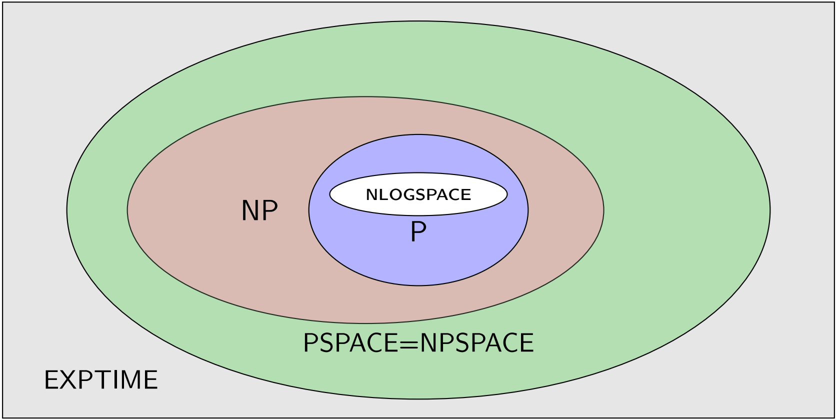
Inclusions

Il faut du temps pour occuper l'espace :

$$\mathbf{P} \subseteq \mathbf{PSPACE} \quad \text{et} \quad \mathbf{NP} \subseteq \mathbf{NPSPACE}$$

Théorème de Savitch

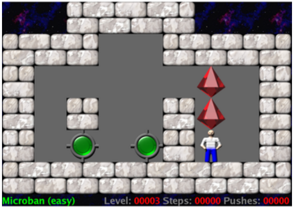
PSPACE $=$ **NPSPACE**



Exemples

Exemples de problèmes dans **PSPACE** que l'on *pense* ne pas être dans **NP**.

- Etant donnés deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$, reconnaissent-ils le même langage ?
- Etant donnés deux expression régulières, codent-elles le même langage ?
- Sokoban



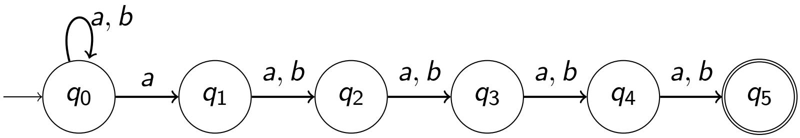
Plan

Exercice

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.

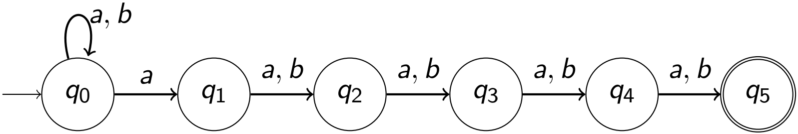
Exercice

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



Exercice

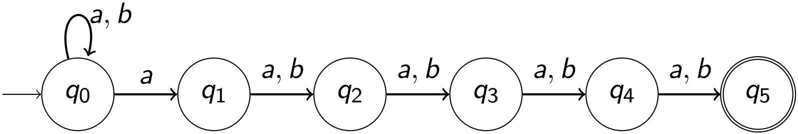
- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



- Dessiner un automate déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.

Exercice

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



- Dessiner un automate déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.
Il faut au moins 32 états...

Moral de l'exercice

Grace à l'automate non déterministe de l'exercice on peut savoir sans faire une fastidieuse déterminisation que :

- Le langage est reconnaissable par un automate (déterministe),
- Qu'il existe un tel automate avec au plus 64 états.

L'algorithmique non-déterministe permet d'avoir accès facilement à des information sur la complexité d'un problème, sans avoir à le résoudre explicitement.
Si on a besoin d'un programme effectif, il faudra faire le travail... mais on aura des indications.

Algorithmes non déterministes

Les machines de Turing non-déterministes sont à l'algorithmique non-déterministe ce que les machines de Turing détermnistes sont à l'algorithmique : super pour la théorie, mais peu utilisables en pratique.

Algorithmes non déterministes

Les machines de Turing non-déterministes sont à l'algorithmique non-déterministe ce que les machines de Turing détermnistes sont à l'algorithmique : super pour la théorie, mais peu utilisables en pratique.

- Dans un algorithmes non-determinste, on a le droit d'utiliser la fontion Choisir(X) où X est un ensemble de caractères fixer. Elle retourne, de façon non déterministe, un élément de X.
- Un algorithme non déterministe décide un problème si
  - Pour toute instance positive du problème, il existe des choix possibles qui font que l'algorithme retourne vrai.
  - Pour toute instance négative du problème, quelque soient les choix, l'algorithme retourne faux.

A mettre en parallèle avec l'existence d'un calcul réussi.

Exemple 1

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

Exemple 1

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

Si n n'est pas premier, en choisissant le codage binaire d'un diviseur strict de n, l'algorithme répondra Vrai.
Si n est premier, l'algorithme répondra toujours Faux.

Exemple 1

Donnée : un entier *n* codé en binaire
Question : *n* est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

L'algorithme utilise un temps polynomial : le problème est dans **NP**.

Exemple 1 (suite)

En pratique on s'autorise des raccourcis (en s'assurant qu'on peut toujours revenir aux choix de caractères).

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
        d=Choisir(Entier plus petit que n)
        if d==0 or d==1 or d==n:
            return False
        if n%d == 0:
            return True
        return False
```

Exemple 2

Donnée : un graphe orienté (V, E), p et q deux sommets.
Question : Existe-t-il un chemin de p à q ?

```
def Accessible(V,E,p,q):
    s=p
    for x in range(len(V)):
        if s==q :
            return True
        r=Choisir(V)
        if (s,r) not in E:
            return False
        s=r
    return False
```

Exemple 2

Donnée : un graphe orienté (V, E), p et q deux sommets.
Question : Existe-t-il un chemin de p à q ?

```
def Accessible(V,E,p,q):
    s=p
    for x in range(len(V)):
        if s==q :
            return True
        r=Choisir(V)
        if (s,r) not in E:
            return False
        s=r
    return False
```

On utilise deux variables locales pour les sommet r et s et une pour le compteur x.
Chacune utilise un espace logarithmique en la taille de l'entrée.
Le problème est dans **NLOGSPACE**.

Exercices

1. Montrer que le problème suivant est dans **NP**.

SAT

Donnée : Une formule booléenne $F(x_1, \ldots, x_k)$
Question : Existe-t-il une instanciation des variables qui rende la formule vraie.
(Remarque : la méthode des tables de vérité est exponentielle en k)

2. Montrer que le problème suivant est dans **NP**.

Chemin Hamiltonien

Donnée : Un graphe fini orienté $G = (V, E)$
Question : Existe-t-il un chemin dans G G qui passe une et une seule fois par tous les sommets.

Exercices

1. Montrer que le problème suivant est dans **NLOGSPACE**.

Triangle

Donnée : Un graphe fini non orienté $G = (V, E)$
Question : Existe-t-il trois sommets qui forment un triangle ? (en relationd eux à deux).

2. Montrer que le problème esuivant est dans **NLOGSPACE**.

Chemin Hamiltonien

Donnée : Un graphe fini orienté $G = (V, E)$
Question : Existe-t-il un circuit dans G.

Exercice

.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.

Exercice

.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.
3. Montrer que **Membership** est dans **NLOGSPACE**. Que peut-t-on en déduire sur la complexité déterministe du problème ?

Exercice

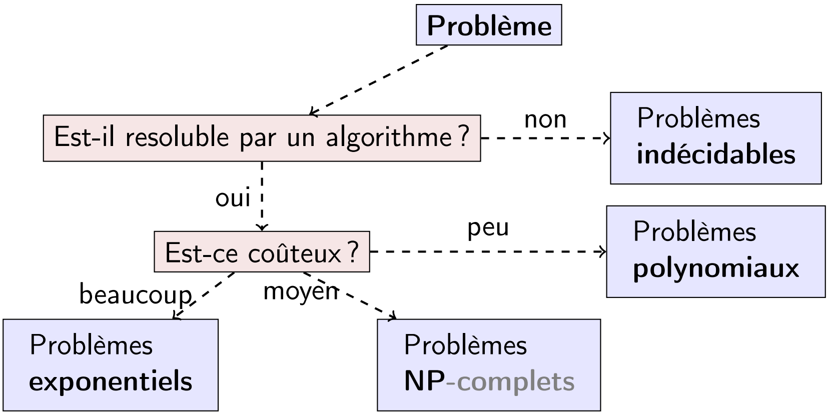
.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.
3. Montrer que **Membership** est dans **NLOGSPACE**. Que peut-t-on en déduire sur la complexité déterministe du problème ?
4. Donner un algorithme polynomial pour **Membership**.

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)

Plan

Problemes complémentaires

Soit Pr un problème. Le problème complémentaire de Pr est le problème associé au complémentaire du langage de Pr.

Problemes complémentaires

Soit Pr un problème. Le problème complémentaire de Pr est le problème associé au complémentaire du langage de Pr.

Problème Pr

Donnée : un mot w sur $\{a, b\}$.
Question : Est-ce que w a autant de a que de b ?

Problème complémentaire de Pr

Donnée : un mot w sur $\{a, b\}$.
Question : Est-ce que w n'a pas autant de a que de b ?

Mais à quoi ça sert ?

Problemes complémentaires et classes déterministes

On note co-$\mathcal{C}$ la classe des langages complémentaires des langages de la classe $\mathcal{C}$.

```
def MonAlgoEnPython(w):
    ....
    return True  # Changer en False
    ....
    return False # Changer en True
```

Pour une classe $\mathcal{C}$ de langages définie par des machines déterministes (sans conditions sur les états finaux), alors co-$\mathcal{C} = \mathcal{C}$.

On a **P** $= \mathrm{co}-$**P**, **PSPACE** $= \mathrm{co}-$**PSPACE**, etc.

Problemes complémentaires et classes non-déterministes

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

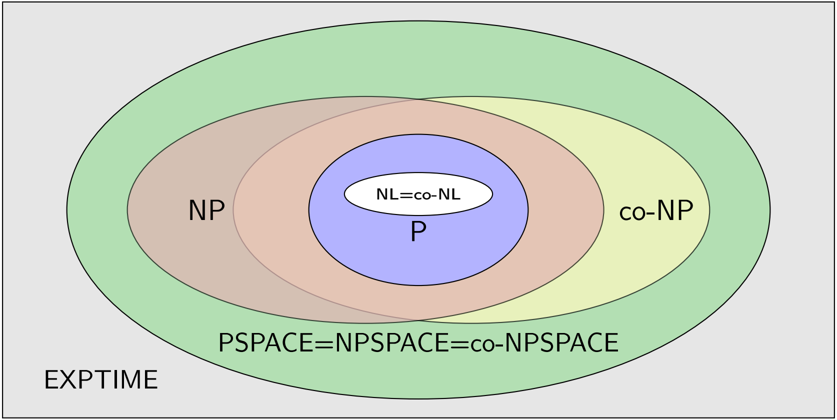
```
def NotPrime(n): #n est une chaine de 0,1
        d=Choisir(Entier plus petit que n)
        if d==0 or d==1 or d==n:
            return False # True
        if n%d == 0:
            return True  # False
        return False
```

Si on change `True` en `False` et réciproquement, ça ne donne pas un algorithme non déterministe pour le problème complémentaire (dans l'exemple, tous les entiers sont acceptés).

Hierarchie

On peut démontrer que :

$\mathbf{NLOGSPACE} = \mathbf{co} - \mathbf{NLOGSPACE}$ et $\mathbf{NPSPACE} = \mathbf{co} - \mathbf{NPSPACE}$



Problème difficile/complet pour une classe

Pour une classe de complexité $\mathcal{C}$, un problème est dit difficile s'il est *plus coûteux à résoudre* que tout problème de $\mathcal{C}$. On dit qu'il est $\mathcal{C}$-difficile.

Un problème est complet pour $\mathcal{C}$ ou $\mathcal{C}$-complet si

- Il est dans $\mathcal{C}$ et
- Il est $\mathcal{C}$-difficile.

Qu'entend-on par *plus coûteux à résoudre* ?

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

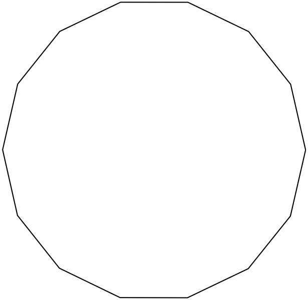
Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



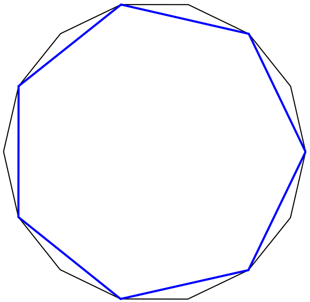
Non. Sinon on trace un heptagone.

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



Non. Sinon on trace un heptagone.

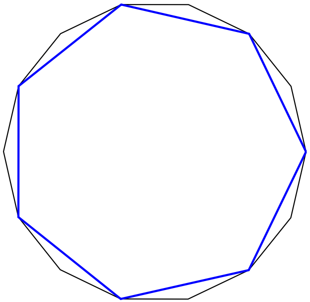
On relie à la règle un point sur deux.

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



Non. Sinon on trace un heptagone.

On relie à la règle un point sur deux.

On obtientrait une construction d'un heptagone à la règle et au compas.
Cela contredit le résultat.

Réduction : un exemple d'indécidabilité

On admet que le problème de l'arrêt est indécidable.

Problème de l'arrêt

Donnée : Un programme M et une entrée w de se programme.
Question : L'execution de M sur w termine-t-elle ?

Comment en déduire que le problème suivant est indécidable ?

Problème de l'arrêt universel

Donnée : Un programme M.
Question : L'execution de M termine-t-elle pour toute entrée ?

Réduction : un exemple d'indécidabilité

On admet que le problème de l'arrêt est indécidable.

Problème de l'arrêt

Donnée : Un programme M et une entrée w de se programme.
Question : L'execution de M sur w termine-t-elle ?

Comment en déduire que le problème suivant est indécidable ?

Problème de l'arrêt universel

Donnée : Un programme M.
Question : L'execution de M termine-t-elle pour toute entrée ?

Supposons que l'on a un programme `ArretUniversel` pour le problème AU.

```
def Arret(M,w):
    def NewM(x):    #NewM s'arrete pour toutes les entrees
        if x==w:    #ssi NewM(w) s'arrete ssi M(w) s'arrete
            return M(w)
        return False
    return ArretUniversel(NewM)
```

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrée de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrée de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

1. Si φ et B sont décidables, alors A est décidable.

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrée de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

1. Si φ et B sont décidables, alors A est décidable.
2. Si φ et B sont calculable en temps polynomial, alors A est calculable en temps polynomial. On parle alors de réduction polynomiale et on note

$$A <<_{\mathbf{P}} B$$

NP-complet

Un problème B est **NP**-complet si

- Il est dans **NP**,
- Pour tout problème A de **NP**,

$$A <<_{\mathbf{P}} B$$

NP-complet

Un problème B est **NP**-complet si

- Il est dans **NP**,
- Pour tout problème A de **NP**,

$$A <<_{\mathbf{P}} B$$

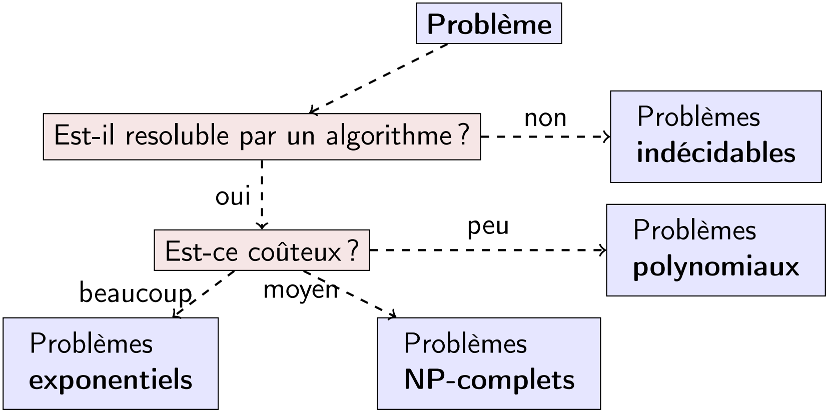
Le second point est difficile à monter, il y a une infinité de problèmes dans **NP**. En général, on montrer que

- Il est dans **NP**,
- Il existse un problème C **NP**-complet tel que ,

$$C <<_{\mathbf{P}} B$$

(reste à trouver un premier problème NP-Complet, ce qu'on verra au prochain cours.)

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.
Solveurs SAT
Solveurs IP (VS)