Remise à Niveau en Calculabilité et Complexité

Serenella Cerrito —— DEA I3

Supports de Cours

- Ces transparents...:-)
- S. Cerrito, H. Comon, C. Crepeau, *Calculabilité et Complexité*. Polycopié d'un ancien cours de maîtrise d'Informatique, fichier.ps accessible sur : `http://www.lri.fr/ serena/frpolycop.htm`
- R.L. Epstein, W. A. Carnielli, *Computability, Computable Functions, Logic, and the Foundations of Mathematics*, Wadsworth & Brooks/Cole, 1989.
- M, R. Garey, D. S. Johnson, *Computers and Intractability : A guide to the Theory of NP-Completeness*, Freeman, 1979.
- D. S. Johnson, *A Catalog of Complexity Classes*, chap. 2 du *Handbook of theoretical Computer Science*, Elsevier 1990

Calculabilité

Définition 0.1 *Une* machine de Turing *(déterministe) est un septuplet*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

où

- $Q = \{q_0, q_1, ..., q_k\}$ *est l'ensemble des* états *de la machine,*
- $\Gamma = \{c_0, c_1, ..., c_n\}$ *est l'alphabet de travail, (pour simplifier on suppose* $Q \cap \Gamma = \emptyset$*),*
- $B \in \Gamma$ *est un symbole spécial associé à une case vide (toutes les cases sauf un nombre fini contiennent* B*),*
- $\Sigma \subseteq \Gamma - \{B\}$ *est l'ensemble des symboles avec lesquels les mots fournis en entrée à la machine sont exprimés,*
- q_0 *est l'*état initial*,*
- $F \subseteq Q$ *est l'ensemble des* états acceptants *et*
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\triangleleft, \triangledown, \triangleright\}$ *est la* fonction de transition*.*

Une machine de Turing M *accepte* un mot m écrit dans son alphabet d'entrée (Σ) ssi, sur l'entrée m, elle s'arrête dans un de ses états acceptants.

Exemple

Alphabet d'entrée $= \{0, 1\}$, le seul état acceptant est q_4, la fonction de transition δ est définie par :

	$Q \times \Gamma$	0	1	X	Y	B
init	q_0	$(q_1, B, \triangleright)$	-	-	-	$(q_4, B, \triangleright)$
rech 1	q_1	$(q_1, 0, \triangleright)$	$(q_2, Y, \triangleleft)$	$(q_1, X, \triangleright)$	$(q_1, Y, \triangleright)$	-
rech 0	q_2	$(q_1, X, \triangleright)$	$(q_2, 1, \triangleleft)$	$(q_2, X, \triangleleft)$	$(q_2, Y, \triangleleft)$	$(q_3, B, \triangleright)$
1 fini ?	q_3	-	-	$(q_3, X, \triangleright)$	$(q_3, Y, \triangleright)$	$(q_4, B, \triangleright)$
final	q_4	-	-	-	-	-

Figure 1: la fonction de transition δ

Cette machine accepte le langage $\{0^n 1^n\}$.

Définition 0.2 *Un langage $L \subseteq \Sigma^*$ est* Turing-semi-calculable *s'il existe une machine de Turing M telle que l'ensemble des mots sur Σ acceptées est exactement L. L est* Turing-calculable *si de plus M s'arrête sur tous les mots $\omega \in \Sigma^*$.*

Modélisation de :

Problème (à reponse OUI/NON) "Semi-décidable" et "Décidable".

- Reformulation du problème :

  Le mot m appartient-il ou pas au langage$\{0^n 1^n \mid n \geq 0\}$?

  sous un autre format :

  Données : Un mot m sur l'alphabet $\{0, 1\}$

  Question : $m \in \{0^n 1^n \mid n \geq 0\}$?

- Seules réponses possibles : OUI, NON.

- La machine de Turing du troisième transparent montre que ce problème est DECIDABLE.
  (donc, a fortiori, il est aussi semi-décidable ! :-)

On peut aussi définir une machine de Turing qui, à la place de répondre à une question à réponse OUI/NON *calcule une fonction* $f : N^k \to N$.

Voici une machine qui calcule $n + m$ lorsqu'on lui donne $0^n 1 0^m$ sur son ruban:

$$M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \emptyset)$$

où δ est donnée sous forme de table :

$Q \times \Gamma$	0	1	B
q_0	$(q_0, 0, \triangleright)$	$(q_0, 0, \triangleright)$	$(q_1, B, \triangleleft)$
q_1	$(q_1, B, \triangledown)$	-	-

Figure 2: la fonction δ

Une définition précise....

Définition 0.3 *Une fonction partielle* $f : N^k \to N$ *est* Turing-semi-calculable *s'il existe une machine* M *telle que pour tous les* $\vec{n}$ *où* $f(\vec{n})$ *est définie,* $M(\vec{n}) = f(\vec{n})$*. Une fonction totale* $f : N^k \to N$ *est* Turing-calculable *s'il existe une machine* M *telle que pour tout* $\vec{n}$ *on a* $M(\vec{n}) = f(\vec{n})$*.*

($M(\vec{n})$ est le mot que la machine M laisse sur son ruban à la fin de son calcul sur l'entrée $\vec{n}$; ce mot représente un entier.)

Modélisation de la notion de fonction “CALCULABLE”, ou “PROGRAMMABLE”, ou “telle qu'il existe un ALGORITHME de calcul”...

NB : Il existe des fonctions qui ne sont pas calculables !! (voir la suite du feuilleton...:-)

D'autres modèles de calcul **équivalents** aux machines de Turing :

- Fonctions *Récursives* (Herbrand, Gödel)
- λ-calcul non typé (Church)
- Systèmes de Post (Post)
- Une certaine axiomatisation logique de l'arithmétique (Gödel))
- ...

- Au lieu d'**imposer** que δ (transition) soit une FONCTION, on peut juste demander qu'elle soit une RELATION...
- On obtient alors la notion de machine de Turing **non-déterministe**
- Problèmes à réponse OUI/NON : une machine non-déterministe accepte un mot m ssi *il existe un calcul* qui mène à un état acceptant.
- On peut toujours déterminiser une machine non-déterministe. En gardant la même complexité de calcul ? Cela, est une autre histoire...
- NB : Une machine déterministe est un cas particulier de machine non-déterministe.

Le PROBLEME DE L'ARRET

Données Un couple $\langle M, m\rangle$, où M est une machine de Turing et m est un mot de son alphabet d'entrée.

Question M s'arrête-elle ou pas sur l'entrée m ?

Ce problème est INDECIDABLE.

Idée de la preuve d'indécidabilité du problème de l'arrêt (par l'absurde).

Hypothèse : Il existe une machine, MAGIQUE, qui, sur une donnée du type $\langle M, w \rangle$, s'arrête dans un état acceptant q_{OUI} si M s'arrête sur la donnée w, et s'arrête dans un état (non-acceptant) q_{NON} si M ne s'arrête pas sur la donnée w

Alors, il existe aussi une machine MAGIQUE2 qui sur une donnée du type (M, w), ne s'arrête pas si M s'arrête sur la donnée w et s'arrête si M ne s'arrête **pas** sur la donnée w. (Idée : ajouter un nouvel état q_{boucle} et de nouvelles transitions $\delta(q_{OUI}, X) = (q_{boucle}, X, \triangleright)$, pour tout symbole de Γ, aux instructions de MAGIQUE).

Alors, étant donnée une machine M, il existe aussi une machine MAGIQUE3 qui prend un mot $< M >$ en entrée, où $< M >$ est **le code d'une machine de Turing** M, et qui s'arrête sur $< M >$ ssi la machine M s'arrête sur le mot $< M >$ (Idée : MAGIQUE3 a des instructions permettant de transformer le mot $< M >$ dans le mot $(< M >, < M >)$, de revenir au début de ce mot, et de rentrer dans l'état q_0 de M_2; elle a, de plus, toutes les instructions de $M2$).

On obtient : $M3$ s'arrête sur l'entrée $< M3 >$ ssi $M3$ ne s'arrête pas sur l'entrée $< M3 >$. Contradiction !

Soit f_{arr} une fonction : Machines de Turing $\times$ Mots $\rightarrow \{0,1\}$
telle que :
$f(machine, mot) = 1$ ssi *machine* s'arrête sur *mot*,
$f(machine, mot) = 0$ sinon.

Puisque le problème de l'arrêt n'est pas décidable, cette fonction est bien définie mais elle **n'est pas calculable**. Impossible de la programmer...

Dommage, un programme capable de tester si un autre programme est en train de boucler sur son entrée aurait été utile... :-)

- On peut REDUIRE le problème de l'arrêt au problème de la **satisfiabilité d'une formule du calcul des prédicats du premier ordre**.
- Idée : On associe à M et m une formule $F_{M,m}$ telle que $F_{M,m}$ est satisfiable ssi M ne s'arrête pas sur m. Donc, SI on pouvait décider la satisfiabilité, alors on pourrait décider l'arrêt aussi...
- *Donc* : **La satisfiabilité d'une formule du calcul des prédicats du premier ordre n'est pas décidable !**
- La validité d'une formule du calcul des prédicats du premier ordre n'est pas décidable, mais elle est semi-décidable (penser à un système de preuve complet, la résolution, par exemple).

Un autre exemple de problème indécidable que vous allez rencontrer dans vos cours : l'inclusion des requêtes en Datalog.

Vous allez rencontrer aussi des problèmes décidables, mais très, très durs...

Complexité

- Mesure de la **complexité en temps** d'une machine de Turing non-déterministe qui s'arrête sur toute entrée : nombre de transitions maximum, pour chaque branche de l'arbre des calculs possibles.
- Si, pour toute entrée de longueur n la machine fait au max $f(n)$ transitions, on dira que sa complexité en temps est bornée par f. Attention : Pire des Cas !
- Mesure de la **complexité en espace** d'une machine de Turing non-déterministe qui s'arrête sur toute entrée : nombre maximum de cases de ruban visitées, pour chaque branche de l'arbre des calculs possibles.
- Si, pour toute entrée de longueur n la machine visite au max $f(n)$ cases, on dira que sa complexité en espace est bornée par f. Attention : Pire des Cas !

La classe de complexité **P** est la classe des problèmes (à réponse OUI/NON) définie par :
P=$\{Prob \mid$ il existe au moins une machine de Turing **déterministe** M qui décide $Prob$ et telle que la complexité en **temps** de M est bornée par un fonction **polynomiale** de la taille de l'entrée $\}$

Exemples de problèmes de la classe **P**

- Le problème de la subsomption pour certaines logiques de description (voir cours de M.C.Rousset)
- Le problème de savoir si une formule propositionnelle F est vraie pour une interprétation **donnée** $\mathcal{I}$. On notera ce problème : **Vrai ?**.

La classe de complexité **NP** est la classe de ces problèmes (à réponse OUI/NON) définie par :
P=$\{Prob \mid$ il existe au moins une machine de Turing **non-déterministe** M qui "décide" $Prob$ et telle que la complexité en **temps** de M est bornée par un fonction **polynomiale** de la taille de l'entrée$\}$

- "décide" = ???
  Exemple : Le problème de la **satisfiabilité d'une formule de la logique propositionnelle** (**SAT**) appartient à **NP**.
  Idée : on peut définir une machine non-déterministe dont l'arbre de calcul a une profondeur polynomiale et [contient au moins une branche qui accepte F] ssi [il existe au moins une interprétation $\mathcal{I}$ qui rend F vraie].
- **Intuition** : Un problème de **NP** est tel que, au moins, on sait **tester** en temps polynomial si une *solution candidate* est une vraie solution...

- $\mathbf{P} \subseteq \mathbf{NP}$? Certes...
- $\mathbf{NP} \subseteq \mathbf{P}$? On ne sait pas !!
  Donc on ne sait pas si $\mathbf{P} \neq \mathbf{NP}$
- On veut caractériser les problèmes $Prob$ tels que *si* $\mathbf{P} \neq \mathbf{NP}$, alors $Prob \in \mathbf{NP} \setminus \mathbf{P}$... Ce sont les problèmes dits **NP-complets**.
  Il s'agit de problemes qui, "probablement", ne sont pas "faisables".

Définition 0.4 *On dit que* $L_1 \subseteq \Sigma_1^*$ *se réduit polynomialement à* $L_2 \subseteq \Sigma_2^*$ *(noté* $L_1 \leq_{\mathcal{P}} L_2$*) s'il existe une fonction* $f : \Sigma_1^* \rightarrow \Sigma_2^*$ *telle que*

1. f *est calculable en temps polynômial par rapport à la taille de sa donnée*

2. *pour tout* $x \in \Sigma_1^*$ *on a*

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Exemple de réduction polinomiale

$L1$: (mots qui décrivent) l'ensemble
$\{\langle G_1, k_1\rangle \ \mid \ G_1 = (S, A)$ est un graphe non-orienté, k_1 est un entier poisitif $\leq \mid S \mid$, et il existe un $S_1 \subseteq S$ tel que :
$\mid S_1 \mid \leq k_1$ et $\forall \langle s, s'\rangle \in A$, on a $s \in S_1 \vee s' \in S_1\}$.
(Pb. dit *vertex covering*)

$L2$: (mots qui décrivent) l'ensemble
$\{\langle G_2, k_2\rangle \ \mid \ G_2 = (S, A)$ est un graphe non-orienté, k_2 est un entier poisitif $\leq \mid S \mid$, et il existe un $S_2 \subseteq S$ tel que :
$\mid S_2 \mid \geq k_2$ et $\forall s, s' \in S$, on a $\langle s, s'\rangle \notin A\}$.
(Pb. dit *stable*)

Exemple de réduction polinomiale, suite

$L_1 \leq_{\mathcal{P}} L_2$.
Il suffit de prendre f:
$Graphes \times N \ \to Graphes \times N$
telle que $f(\langle (S, A),\ k \rangle) = \langle (S, A),\ \mid S \mid -k \rangle$

Idée : S_1 est un *vertex covering* de taille bornée par k du graphe (S, A) ssi $S \setminus S_1$ est un *stable* de taille supérieure ou égale à $\mid S \mid -k$ du graphe associé par f à (S, A).

La fonction f se calcule en temps polynomial par rapport au nombre de sommets du graphe (S, A) et à k.

Théorème 0.5 *Si* $L_2 \in \mathcal{P}$ *et* $L_1 \leq_{\mathcal{P}} L_2$ *alors* $L_1 \in \mathcal{P}$.

- Un problème $Prob$ est dit **NP-facile** s'il appartient à **NP**.
- un problème $Prob$ est dit **NP-difficile** si, quelque soit le problème $Prob'$ de **NP**, $Prob'$ se réduit polynômialement à $prob$
- Un problème est dit **NP-complet** s'il est **NP-facile** et **NP-difficile**.

- **SAT** est le problème de la satisfiabilité d'un ensemble de clauses **propositionnelles**. Il est décidable.
- **SAT** est **NP-complet** (Théorème de Cook)
- Idée (vague) de la preuve : montrer comme associer, en temps polynômial, à toute machine non-déterministe M bornée en temps par un polynôme p et à toute entrée m une formule propositionnelle $F_{M,m}$ qui est satisfiable ssi M accepte m.
- Remarquer la différence entre **SAT** et **Vrai ?** !!!

- La classe **co-NP** est définie par :
  co-NP $= \{P \mid P$ est le complément d'un problème P' qui appartient à **NP**$\}$
- Puisque une formule A est valide ssi sa négation n'est pas satisfiable, le problème **Val** (validité propositionnelle) appartient à **co-NP**.
- On ne sait pas si **co-NP** $=$ **NP**
- Si **NP** $\neq$ **co-NP** alors **P** $\neq$ **NP**.
- On pourrait avoir **P** $\neq$ **NP** sans avoir **NP** $\neq$ **co-NP**.

Exemples de Problèmes "durs"

- Le problème (voir cours de M.C Rousset) :
  KB $\cup \neg$p *est-il satisfiable ?*
  où KB est un ensemble de clauses propositionnelles quelconque
  où p est une clause propositionnelle est **NP**-complet, car c'est **SAT** !
- Donc le problème : "$KB \models p$?" est co-NP-complet.
  (C'est **Val**).
- Le problème de l'inclusion pour *certaines* logiques de description (voir cours de M.C Rousset) est **NP**-complet.
- Le problème de l'équivalence de 2 expressions algébriques en BD relationnelle (voir cours de N. Spyratos) est **NP**-complet.

Pour prouver qu'un problème donné *Prob* est **NP-complet**, prouver :

1. Qu'il est **NP-facile**. D'habitude, c'est facile... :-)
2. Qu'il est **NP-difficile**, **en exhibant une réduction polynomiale d'un autre problème déjà connu comme étant NP-complet** à *Prob*.
   NE PAS SE TROMPER SUR L'ORIENTATION DE LA REDUCTION !

- La classe de complexité **PSPACE** est la classe des problèmes (à réponse OUI/NON) définie par :
  PSPACE=$\{Prob \mid$ il existe au moins une machine de Turing **déterministe** M qui décide $Prob$ et telle que la complexité en **espace** de M est bornée par un fonction **polynomiale** de la taille de l'entrée $\}$
- La classe de complexité **NPSPACE** est la classe de ces problèmes (à réponse OUI/NON) définie par :
  NPSPACE=$\{Prob \mid$ il existe au moins une machine de Turing **non-déterministe** M qui "décide" $Prob$ et telle que la complexité en **espace** de M est bornée par un fonction **polynômiale** de la taille de l'entrée$\}$
- On sait que **PSPACE**= **NPSPACE** (théorème de Savitch)

- Un problème P est **PSPACE**-complet s'il appartient à **PSPACE** et *tout* problème $P' \in$ **PSPACE** se réduit polynomialment à P
- Le fait qu'un problème P soit **PSPACE**-complet est une indication du fait qu'il est "dur", probablement encore plus dur que s'il était juste **NP**-complet :
  - **NP** $\subseteq$ **PSPACE**
  - même si **P** $=$ **NP**, on pourrait avoir **P** $\neq$ **PSPACE**.

Un problème **PSPACE** complet : **Q**uantified **B**oolean **F**ormulae.

Données : Une expression de la forme :

$$Q_1 b_1 ....Q_n b_n F$$

où F est une formule de la logique propositionnelle dont les lettres propositionnelles sont notées $b_1, ...b_n$ et Q_i est ou bien $\forall$ ou bien $\exists$. (La valeur des b_i est dans $Bool = \{V, F\}$)

Question : $Qb_1 ....Qb_n F$ est vraie ?

NB : SAT est un cas particulier de QBF.

- La classe **EXPTIME** est la classe des problèmes (à réponse OUI/NON) décidés par une machine de Turing déterministe dont la complexité en temps est bornée par $2^{p(n)}$, où n est la taille de l'entrée et p est un polynôme.
- On sait : $\mathbf{NP} \subset \mathbf{EXPTIME}$

  Si *prob* est **EXPTIME**-complet, alors $prob \notin \mathbf{NP}$
- Les problèmes **EXPTIME**-complets (certains problèmes du jeu d'échecs, par ex.) sont *certainement* "pas faisables".

EXERCICE

En sachant que le problème du **Stable**, défini par :
Données : Un graphe (non orienté) $G = (S, A)$ et un entier positif k
Question : G contient-il un ensemble S contenant au moins k sommets tel si s_1 et s_2 appartiennent à A alors s_1 et s_2 ne sont pas reliés ?
est **NP**-complet, montrer que le problème **sat-modifié** :
Données : Un ensemble de clauses propositionnelles B où chaque clause contient au plus 2 littéraux, et un entier positif k'
Question : B est-il satisfiable par une interprétation $\mathcal{I}$
dans laquelle au plus k' lettres propositionnelles prennent la valeur V ?
est **NP**-complet.