

Modèles de calcul

Année 2009–2010

M1, Univ. Bordeaux

26 novembre 2009

1 / 159

Objectifs (fiche UE)

Définir, **indépendamment de la technologie** :

- ▶ ce qui est **calculable** et ce qui ne l'est pas (théorie de la calculabilité) ;
- ▶ ce qui est calculable **efficacement** et ce qui ne l'est pas (théorie de la complexité).






3 / 159

Modalités du cours

- ▶ 12 cours, 3 groupes de TD (débutent le 14/9/09).
- ▶ N. Saheb, G. Sénizergues, A. Zvonkine.
- ▶ Contrôle continu (CC) obligatoire sauf dispense.
- ▶ Note finale session 1 :
 $\frac{2}{3}$ Examen (3h) + $\frac{1}{3}$ CC.
- ▶ Note finale session 2 :
 $\max(\text{Examen}, \frac{2}{3} \text{ Examen (3h)} + \frac{1}{3} \text{ CC})$.




2 / 159

Bibliographie

-  [J.E. Hopcroft, R. Motwani, J. D. Ullman.](#)
Introduction to Automata Theory, Languages & Computation.
[Addison-Wesley, 2005.](#)
-  [M. Sipser.](#)
Introduction to the Theory of Computation.
[PWS publishing Company, 1997.](#)
-  [O. Carton.](#)
Langages formels, Calculabilité et Complexité.
[Vuibert, 2008.](#)
-  [J.M. Autebert.](#)
Calculabilité et Décidabilité.
[Masson, 1992.](#)
-  [P. Wolper.](#)
Introduction à la calculabilité.
[InterÉditions, 1991.](#)

4 / 159

Bibliographie complémentaire

-  Ch. Papadimitriou.
Computational complexity.
Addison-Wesley, 1995.
-  M. Garey, D. Johnson.
Computers and intractability.
W.H. Freeman & Co, 1979.
-  J. E. Savage.
Models of computation.
Addison-Wesley, 1998.

5 / 159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

7 / 159

Plan du cours

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

6 / 159

Questions abordées dans ce cours

- ▶ Quels problèmes peut-on résoudre avec une machine (indépendamment de puissance de calcul des machines) ?
- ▶ Comment formaliser
 - ▶ ce qu'est un problème ?
 - ▶ ce qu'est une machine ?
- ▶ Quelles fonctions peut-on calculer avec une machine ?
- ▶ Comment comparer la complexité des problèmes ?
Y a-t-il des problèmes inhéremment difficiles ?

8 / 159

Bref historique

- 1900 Hilbert, 10^{ème} problème : peut-on décider, de façon mécanique, si une équation Diophantienne a une solution ?
- 1931 Gödel publie le théorème d'incomplétude.
- 1935 Turing formalise une définition de machine et de calcul.
- 1936 Church exhibe un problème non résoluble par machine.
- 1938 Kleene prouve l'équivalence entre machines de Turing, λ -calcul, fonctions récursives.
- 1947 Post & Markov prouvent qu'un problème posé par Thue en 1914 n'est pas résoluble mécaniquement.
- 1970 Matiyasevich répond négativement au 10^{ème} problème de Hilbert.
- 1971 Cook & Levin formalisent la notion de problème NP-complet.

9/159

Pourquoi ce cours (ou la théorie en général) ?

- pour comprendre les limites de la programmation, mais aussi...
- ... pour découvrir un côté esthétique du calcul - la théorie apporte le plus souvent un point de vue plus simple et plus élégant,
- ... pour chercher la pérennité : la technologie évolue rapidement, mais les concepts de base (théoriques) restent valides,
- ... pour entraîner l'abileté d'analyser et/ou décrire un problème avec clarté.

11/159

Bref historique (info fondamentale)

- 1936 la calculabilité émerge de la logique (Russel, Hilbert, Turing, Church sont logiciens)
- 1941 Zuse invente le premier ordinateur (Z3) ; suit ENIAC 1945 ; modèle de von Neumann
- 1950 premiers langages de programmation (Fortran, Cobol, Lisp)
- 1960 développement de la théorie des langages formels et des modèles de calcul (hiérarchie de Chomsky)
- 1970 développement de la théorie de la complexité, question P vs. NP.
- 70/80 algorithmique et structures de données
- 80/90 algorithmique parallèle, distribuée ; cryptographie

10/159

Exemples de problèmes de décision

- | | |
|------------|--|
| Problème 1 | Donnée Un nombre entier positif n en base 2.
Question n est-il pair ? |
| Problème 2 | D. Un nombre entier positif n en base 10.
Q. n est-il premier ? |
| Problème 3 | D. Une séquence DNA s et un motif p .
Q. p apparait-il dans s ? |
| Problème 4 | D. Un programme C.
Q. Le programme est-il syntaxiquement correct ? |

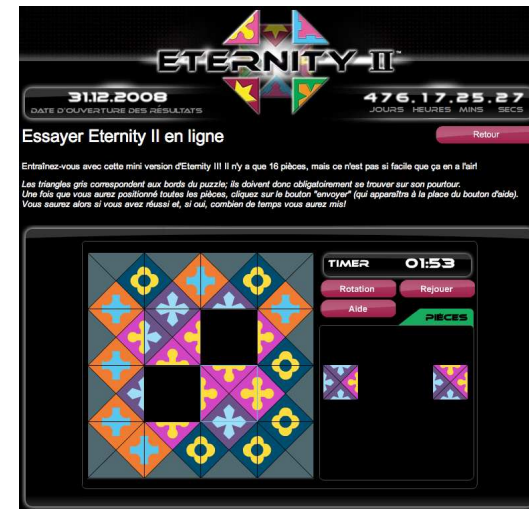
12/159

Exemples de problèmes de décision (2)

- Problème 5** **Donnée** Un graphe donné par une liste d'adjacence.
 Question Le graphe est-il 3-coloriable ?
- Problème 6** **D.** Un puzzle Eternity <http://fr.eternityii.com>.
 Q. Le puzzle a-t-il une solution ?
- Problème 7** **D.** Un programme C.
 Q. Le programme s'arrête-t-il sur au moins l'une de ses entrées ?
- Problème 8** **D.** Un programme C.
 Q. Le programme s'arrête-t-il sur toute entrée ?
- Problème 9** **D.** Des couples de mots $(u_1, v_1), \dots, (u_n, v_n)$.
 Q. Existe-t-il des entiers i_1, \dots, i_k tels que $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$?

13/159

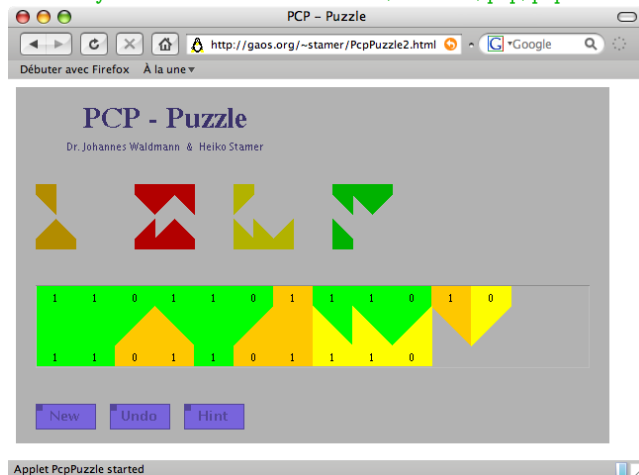
Problème 6 : Eternity II



14/159

Problème 9 : PCP, Problème de correspondance de Post

http://www.theory.informatik.uni-kassel.de/~stamer/pcp/pcpcontest_en.html



15/159

Notion de problème de décision

Un problème de décision est

- ▶ une **question**...
- ▶ ... portant sur un **ensemble** de données...
- ▶ ... dont une **description** est fixée...
- ▶ ... et dont la réponse est **OUI** ou **NON**.

Ne pas confondre **problème** et **algorithme le résolvant**.

Une **instance** du problème est la question posée sur une donnée particulière.

- ▶ On s'intéressera aussi aux problèmes calculatoires, dont la réponse n'est pas nécessairement binaire (OUI/NON).
- ▶ Exemple : **calculer** un 3-coloriage.

16/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

17/159

Propriétés des ensembles dénombrables

Proposition

- 1a. Toute partie de \mathbb{N} est au plus dénombrable.
- 1b. Toute partie d'un ensemble dénombrable est au plus dénombrable.
2. Soit $f : A \rightarrow B$ une application.
 - 2.1 Si f est injective et B dénombrable, alors A est au plus dénombrable.
 - 2.2 Si f est surjective et A dénombrable, alors B est au plus dénombrable.

19/159

Ensembles dénombrables

- ▶ On note $\mathbb{N} = \{0, 1, 2, \dots\}$ l'ensemble des entiers naturels.
- ▶ Un ensemble E est dit **dénombrable** s'il est en bijection avec \mathbb{N} .
- ▶ Un ensemble fini ou dénombrable est dit **au plus dénombrable**.
- ▶ **Exemples**. Les ensembles suivants sont dénombrables :
 1. L'ensemble \mathbb{N}^* des entiers strictement positifs,
 2. Pour un entier $k > 0$ donné, l'ensemble des entiers divisibles par k ,
 3. L'ensemble des entiers qui sont des puissances de 2,
 4. L'ensemble des entiers naturels qui sont des cubes,
 5. L'ensemble des nombres premiers,
 6. L'ensemble \mathbb{Z} des entiers relatifs,
 7. L'ensemble $\mathbb{N} \times \mathbb{N}$,
 8. L'ensemble \mathbb{Q} des rationnels.

18/159

Propriétés des ensembles dénombrables

Proposition

- 1a. Si A et B sont dénombrables, alors $A \times B$ l'est aussi.
- 1b. Si A_1, \dots, A_n sont dénombrables, $\prod_{i=1}^n A_i$ l'est aussi (récurrence).
2. Si J est au plus dénombrable, et si A_i est au plus dénombrable, pour $i \in J$, alors $\bigcup_{i \in J} A_i$ est dénombrable.
3. Si $A \neq \emptyset$ est au plus dénombrable, alors A^* est dénombrable.
4. L'ensemble des automates finis sur alphabet A fini est dénombrable.
5. L'ensemble des programmes C est dénombrable.

20/159

Quelques ensembles non dénombrables

Proposition Les ensembles suivants ne sont pas dénombrables :

1. l'ensemble des nombres réels ;
2. l'ensemble des suites infinies d'entiers ;
3. l'ensemble des parties de \mathbb{N} ;
4. l'ensemble des suites infinies de 0 ou 1 ;
5. l'ensemble des applications de \mathbb{N} dans $\{0, 1\}$.

Note Les trois derniers ensembles sont en bijection.

À une partie X de \mathbb{N} , on associe la suite $x = (x_n)_{n \geq 0}$ définie par

$$x_n = \begin{cases} 1 & \text{si } n \in X \\ 0 & \text{si } n \notin X \end{cases}$$

De même, à toute suite $x = (x_n)_{n \geq 0}$ à valeurs dans $\{0, 1\}$, on associe bijectivement l'application $f_x : \mathbb{N} \rightarrow \{0, 1\}$ définie par $f_x(n) = x_n$.

21/159

Un paradoxe ?

- ▶ On suppose qu'on dispose d'un ordinateur à mémoire infinie.
- ▶ On considère les programmes C qui écrivent une suite de 0 ou 1.
- ▶ On interprète une telle suite a_1, a_2, \dots comme le réel $0, a_1 a_2 \dots$.
- ▶ Un tel réel est dit **calculable**.
- ▶ L'ensemble de ces programmes est dénombrable : $\{P_1, P_2, \dots\}$.
- ▶ On considère le programme qui
 - ▶ lance P_1 jusqu'à ce que celui-ci s'apprête à écrire a_1 , et écrit à la place un entier différent de a_1 , puis,
 - ▶ lance P_2 jusqu'à ce que celui-ci s'apprête à écrire a_2 , et écrit à la place un entier différent de a_2 , etc.
- ▶ Notre programme écrit un réel calculable non calculable !
- ▶ Où est l'erreur ?

23/159

Quelques ensembles non dénombrables

L'argument ci-dessous, dû à Cantor, permet de montrer que l'ensemble $E = \{0, 1\}^{\mathbb{N}}$ des suites infinies de 0 ou 1 est non dénombrable.

- ▶ Soit $f : \mathbb{N} \rightarrow E$ une application.
- ▶ Soit $x = (x_n)_{n \geq 0}$ la suite infinie de 0 ou 1 définie par

$$x_n = 1 - f(n)_n$$

- ▶ Puisque $x_n \neq f(n)_n$, on a $x \neq f(n)$, et ce, pour tout $n \in \mathbb{N}$.
- ▶ Comme $x \in E$ n'est pas de la forme $f(n)$, f n'est pas surjective.
- ▶ En particulier, il n'y a aucune bijection de \mathbb{N} dans E .

22/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

24/159

Notion de problème : formalisation

- ▶ Un **problème de décision** est la donnée d'un ensemble (dénombrable) I d'**instances** et d'un sous-ensemble $P \subseteq I$ d'**instances positives**.
- ▶ **Exemples**
 - ▶ $I = \mathbb{N}, P = \{n \in \mathbb{N} \mid n \text{ est premier}\},$
 - ▶ $I = \{G \mid G \text{ graphe fini}\}, P = \{G \in I \mid G \text{ est 3-coloriable}\},$
 - ▶ $I = \{G \mid G \text{ grammaire hors contexte}\}, P = \{G \in I \mid G \text{ est ambiguë}\}.$
- ▶ Pour Σ fini, un **codage** des instances est une application **injective** $\langle \cdot \rangle : I \rightarrow \Sigma^*$ associant à chaque élément x de I un mot $\langle x \rangle \in \Sigma^*$.

25/159

Notion de problème

- ▶ Un **problème de décision** peut se voir comme une fonction

$$f_P : \Sigma^* \longrightarrow \{\text{OUI}, \text{NON}\}$$

calculant le langage L_P , au sens que $f_P(\langle x \rangle) = \text{OUI} \iff x \in P$.

- ▶ Plus généralement, on s'intéresse au calcul de **fonctions**

$$f : \Sigma_1^* \longrightarrow \Sigma_2^*$$

- ▶ **Exemples**
 - ▶ $\Sigma_1 = \Sigma_2 = \{0, 1\},$
 - ▶ un élément de $\{0, 1\}^*$ représente un entier codé en base 2
 - ▶ f calcule la fonction $n \mapsto 3^n$.

27/159

Notion de problème : formalisation

Exemples

- ▶ $I = \mathbb{N}, \Sigma = \{a\}$ et $\langle n \rangle = a^n$.
- ▶ $I = \mathbb{N}, \Sigma = \{0, 1\}$ et $\langle n \rangle =$ codage en base 2 de n .
- ▶ $I =$ graphes orientés $G = (V, E), \Sigma = \{0, 1, \#, \$\}.$
 $V = \{1, 2, \dots, n\}, E = \{(i_1, j_1), \dots, (i_m, j_m)\} :$

$$\langle (V, E) \rangle = \underbrace{\$ \cdots \$}_{n} \langle i_1 \rangle \# \langle j_1 \rangle \$ \langle i_2 \rangle \# \langle j_2 \rangle \$ \cdots \langle i_m \rangle \# \langle j_m \rangle$$

- ▶ Σ^* se partitionne en 3 ensembles :
 - ▶ Instances positives : $L_P = \{\langle x \rangle \mid x \in P\}.$
 - ▶ Instances négatives : $L_N = \{\langle x \rangle \mid x \in I \text{ et } x \notin P\}.$
 - ▶ Non instances : $\Sigma^* \setminus \{\langle x \rangle \mid x \in I\}.$

26/159

Qu'est-ce que c'est un ordinateur ?

- ▶ avant 1940 : circuits logiques (Boole, Shannon, etc)
- ▶ 1941, von Neumann : machines "modernes"
- ▶ 1950/60 : différencier la puissance des machines selon leur mémoire
Hiérarchie de Chomsky : automates finis, automates à pile, automates de mémoire linéaire, machines de Turing.
 Chomsky introduit sa hiérarchie avec des grammaires (calcul = **réécriture**).

28/159

Notion de machine

- ▶ Résoudre un problème consiste à déterminer pour $x \in I$, si $\langle x \rangle \in L_P$.
- ▶ On veut une machine « acceptant » les mots de L_P .
- ▶ **Automates** : mémoire bornée.
- ▶ **Automates à pile** : ne reconnaissent pas les langages
 - ▶ $\{a^n b^n c^n \mid n \geq 0\}$
 - ▶ $\{ww \mid w \in \{a, b\}^*\}$
- ▶ On peut reconnaître/accepter tous les langages $L \subseteq \Sigma^*$ avec un automate à nombre infini d'états.

29/159

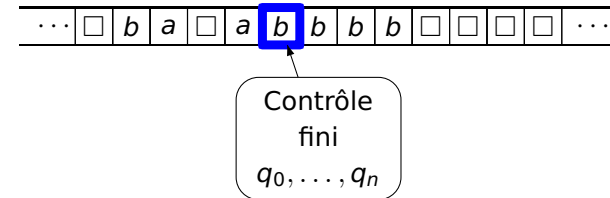
Machines de Turing : intuition

- ▶ Le nombre d'états d'une machine de Turing est **fini** (par comparaison, un ordinateur a un nombre fini de registres et les programmes sont finis).
- ▶ La bande représente la mémoire de la machine. Elle est **infinie** : sur un ordinateur, on peut ajouter des périphériques mémoire (disques...) de façon quasi-illimitée.
- ▶ L'accès à la mémoire est **séquentiel** : la machine peut bouger sa tête à droite ou à gauche d'une case à chaque étape.

31/159

Machines de Turing

- ▶ Les machines de Turing sont des abstractions des ordinateurs.
- ▶ Une machine de Turing comporte :



- ▶ Une **bande infinie** à droite et à gauche faite de cases consécutives.
- ▶ Dans chaque case se trouve un symbole, éventuellement blanc \square .
- ▶ Une tête de lecture-écriture.
- ▶ Un contrôle à nombre **fini** d'états.

30/159

Machines de Turing : formalisation

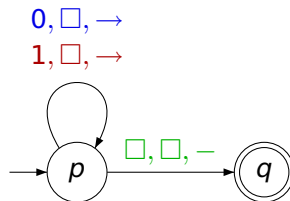
Une **Machine de Turing** (MT) à une bande $M = (Q, q_0, F, \Sigma, \Gamma, \delta)$ est donnée par

- ▶ Q : ensemble **fini** d'états.
- ▶ q_0 : état initial.
- ▶ $F \subseteq Q$: ensemble d'états finaux (ou acceptants).
- ▶ Γ : alphabet fini de la bande, avec $\square \in \Gamma$.
- ▶ Σ : alphabet d'entrée, avec $\Sigma \subseteq \Gamma \setminus \{\square\}$.
- ▶ δ : ensemble de transitions. Une transition est de la forme (p, a, q, b, d) , notée $p \xrightarrow{a, b, d} q$, avec
 - ▶ $p, q \in Q$,
 - ▶ $a, b \in \Gamma$,
 - ▶ $d \in \{\leftarrow, -, \rightarrow\}$.
- ▶ On supposera qu'aucune transition ne part d'un état de F .

32/159

Machines de Turing : représentation graphique

- ▶ On représente souvent une MT comme un automate.
- ▶ Seules changent les étiquettes des transitions.
- ▶ Exemple, avec $\Gamma = \{0, 1, \square\}$ et $\Sigma = \{0, 1\}$:



représente la machine avec $Q = \{p, q\}$, $q_0 = p$, $F = \{q\}$,
 $\delta = \{(p, 0, \square, \rightarrow, p), (p, 1, \square, \rightarrow, p), (p, \square, \square, \leftarrow, q)\}$

33/159

Fonctionnement d'une MT

- ▶ Initialement, un mot w est écrit sur la bande entouré de \square .
- ▶ Un calcul d'une MT sur w est une suite de **pas de calcul**.
- ▶ Cette suite peut être **finie** ou **infinie**.
- ▶ Le calcul commence
 - ▶ avec la tête de lecture-écriture sur la première lettre de w ,
 - ▶ dans l'état q_0 .
- ▶ Chaque pas de calcul consiste à appliquer une transition, si possible (il peut y avoir des choix : **non-déterminisme**).
- ▶ Le calcul ne s'arrête que si aucune transition n'est applicable.

34/159

Fonctionnement d'une MT

- ▶ Chaque pas consiste à appliquer une transition.
- ▶ Une transition de la forme $p \xrightarrow{a,b,d} q$ est possible seulement si
 1. la machine se trouve dans l'état p , et
 2. la lettre se trouvant sous la tête de lecture-écriture est a .
- ▶ Dans ce cas, l'application de la transition consiste à
 - ▶ changer l'état de contrôle qui devient q ,
 - ▶ remplacer le contenu de la case sous la tête de lecture-écriture par b ,
 - ▶ bouger la tête d'une case à gauche si $d = \leftarrow$, ou
 - ▶ bouger la tête d'une case à droite si $d = \rightarrow$, ou
 - ▶ ne pas bouger la tête si $d = -$.

35/159

Configurations et calculs

- ▶ Une **configuration** représente un instantané du calcul.
- ▶ La configuration uqv signifie que
 - ▶ L'état de contrôle est q
 - ▶ Le mot écrit sur la bande est uv , entouré par des \square ,
 - ▶ La tête de lecture est sur la première lettre de v .
- ▶ La configuration initiale sur w est donc q_0w .
- ▶ Pour 2 configurations C, C' , on écrit $C \vdash C'$ lorsqu'on obtient C' par application d'une transition à partir de C .

Un **calcul** d'une machine de Turing est une suite de configurations.

$$C_0 \vdash C_1 \vdash C_2 \vdash \dots$$

36/159

Calculs acceptants

Un calcul d'une machine de Turing est une suite de configurations.

$$C_0 \vdash C_1 \vdash C_2 \vdash \dots$$

3 cas possibles

- ▶ Le calcul est infini,
- ▶ Le calcul s'arrête sur un état final (de F),
- ▶ Le calcul s'arrête sur un état non final (pas de F).

37/159

Exemples de machines de Turing

- ▶ Machine qui effectue `while(true)` ;
- ▶ Machine qui efface son entrée et s'arrête.
- ▶ Machine qui accepte 0^*1^* ([automate fini](#)).
- ▶ Machine qui accepte $\{a^n b^n \mid n \geq 0\}$ ([automate à pile](#)).
- ▶ Machine qui accepte $\{a^{2^n} \mid n \geq 0\}$ ([automate à mémoire linéaire](#)).
- ▶ Machine qui accepte $\{a^n b^n c^n \mid n \geq 0\}$ ([automate à mémoire linéaire](#)).
- ▶ Machine qui accepte $\{ww \mid w \in \{0,1\}^*\}$ ([automate à mémoire linéaire](#)).

39/159

Langages acceptés

On peut utiliser une machine pour **accepter** des mots.

- ▶ Le langage $\mathcal{L}(M) \subseteq \Sigma^*$ des mots acceptés par une MT M est l'ensemble des mots w sur lesquels **il existe** un calcul **fini**

$$C_0 \vdash C_1 \vdash C_2 \vdash \dots \vdash C_n$$

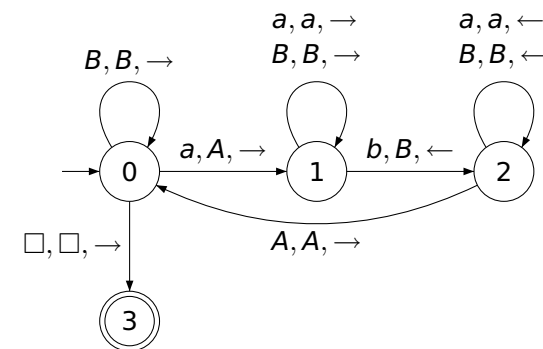
avec $C_0 = q_0 w$ (w est le mot **d'entrée**) et $C_n \in \Gamma^* F \Gamma^*$.

- ▶ 3 cas exclusifs : un calcul peut
 - ▶ soit s'arrêter sur un état acceptant,
 - ▶ soit s'arrêter sur un état non acceptant,
 - ▶ soit ne pas s'arrêter.
- ▶ On dit qu'une machine est **déterministe** si, pour tout $(p, a) \in Q \times \Gamma$, il existe **au plus** une transition de la forme $p \xrightarrow{a,b,d} q$.
- ▶ Si M est déterministe, elle n'admet qu'un calcul par entrée.

38/159

MT acceptant $(\{a^n b^n \mid n \geq 0\})^*$

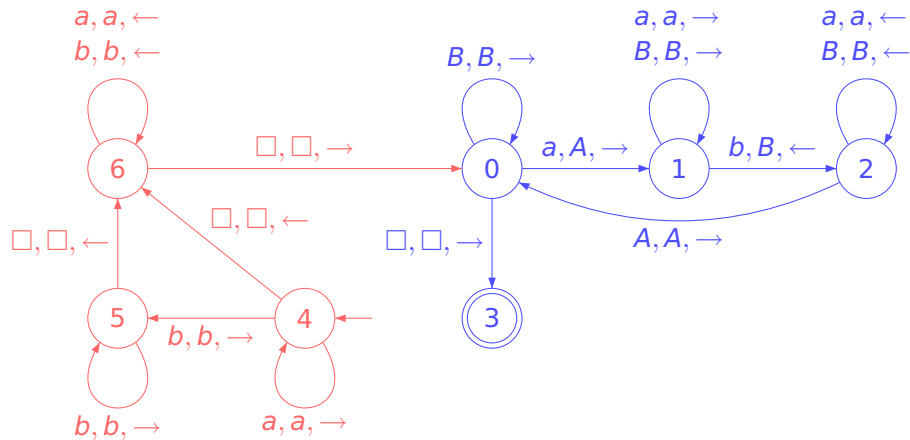
Idée : marquer le 1^{er} a et le 1^{er} b, et recommencer.



40/159

MT acceptant $\{a^n b^n \mid n \geq 0\}$

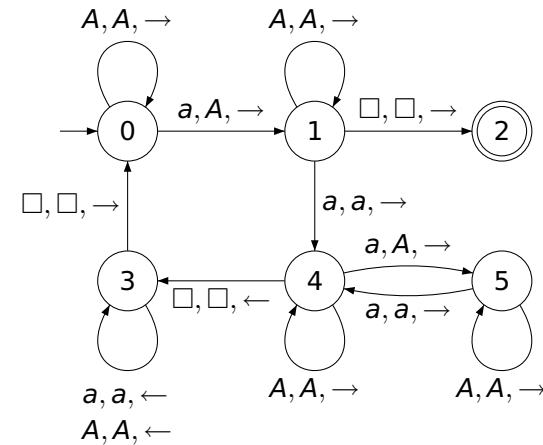
Idée : idem en vérifiant qu'on est dans $a^* b^*$.



41/159

MT acceptant $\{a^{2^n} \mid n \geq 0\}$

Idée : marquer un a sur 2.



42/159

Les machines de Turing peuvent calculer

- ▶ On peut utiliser les MT pour **accepter** des langages ou **calculer**.
- ▶ Une MT déterministe acceptant un langage L calcule la fonction caractéristique de L définie par

$$f : \Sigma^* \rightarrow \{0, 1\}$$

$$w \mapsto \begin{cases} 0 & \text{si } w \notin L, \\ 1 & \text{si } w \in L. \end{cases}$$

- ▶ Plus généralement, on peut associer à une MT déterministe M une fonction $f_M : \Sigma^* \rightarrow \Gamma^*$
 - ▶ On écrit la donnée $w \in \Sigma^*$ sur la bande,
 - ▶ Si la MT s'arrête avec sur la bande le mot $z \in \Gamma^*$, la fonction est définie par $f_M(w) = z$.

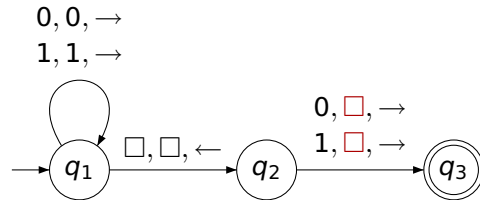
43/159

Exemples de machines de Turing

- ▶ Machine qui interprète son entrée comme un entier n , le remplace par $\lfloor n/2 \rfloor$ et s'arrête.
- ▶ Machine qui effectue l'incrément en binaire.
- ▶ Machine qui effectue l'addition de deux entiers en unaire.
- ▶ Machine qui effectue la multiplication de deux entiers en unaire.

44/159

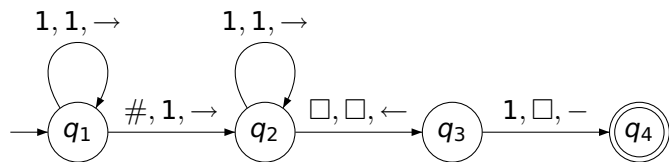
Calcul de $\lfloor n/2 \rfloor$



45/159

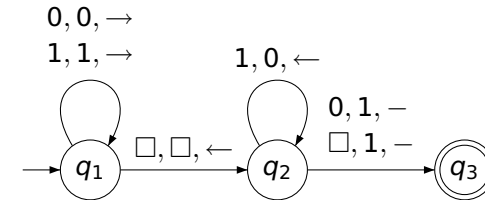
Addition en unaire

- Le mot d'entrée est de la forme $1^n \# 1^m$ interprété comme la donnée des entiers n et m .



47/159

Incrément en binaire



46/159

Il existe des langages non décidables

- Thèse de Church : les MT capturent tout ce qui est calculable.
- Un langage L est **décidable** (ou **récuratif**) s'il existe une MT qui
 - s'arrête sur F partant d'un mot de L ,
 - s'arrête sur $Q \setminus F$ partant d'un mot de $\Sigma^* \setminus L$.
- L'ensemble des langages sur $\{0, 1\}^*$ est non dénombrable.
- L'ensemble des machines de Turing est dénombrable.
- Il existe donc des langages **non décidables**.
- Peut-on décrire explicitement un tel langage ?

48/159

Variations des machines de Turing

On peut changer la définition des MT de plusieurs façons :

- ▶ bande finie sur la gauche et infinie sur la droite,
- ▶ un unique état acceptant, un unique état rejetant,
- ▶ calculs remplaçant la tête en début de bande. . .
... et terminant avec le mot d'entrée écrit sur la bande,
- ▶ machines déterministes,
- ▶ petit alphabet de bande : $\Gamma = \Sigma \cup \{\square\}$,
- ▶ machine à plusieurs bandes et plusieurs têtes.

Ces variations n'affectent pas ce que l'on peut accepter ou calculer.

49/159

Simulation : pour montrer l'équivalence des variantes

- ▶ Intuitivement, une machine M_2 simule une machine M_1 lorsque M_2 peut effectuer les mêmes calculs que M_1 .
- ▶ Un exemple (qui n'est pas général) où M_2 simule M_1 :

$$Q_1 \subseteq Q_2, \quad F_1 = F_2, \quad \Gamma_1 \subseteq \Gamma_2, \text{ et}$$

$$\forall C, C' \in \Gamma_1^* Q_1 \Gamma_1^*$$

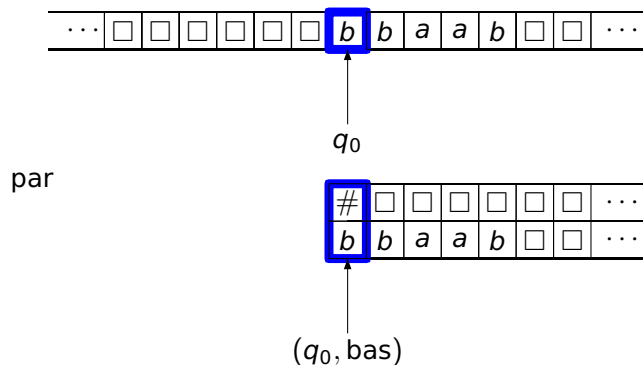
$$C \vdash_1 C' \text{ dans } M_1 \Leftrightarrow C \vdash_2 C_1 \vdash_2 C_2 \vdash_2 \dots \vdash_2 C_k \vdash_2 C' \\ \text{dans } M_2 \text{ avec } C_i \notin \Gamma_2^* Q_1 \Gamma_2^*$$

- ▶ M_2 simule M_1 si elle peut effectuer les mêmes passages entre configurations (éventuellement par plusieurs transitions).

50/159

Bande finie à gauche

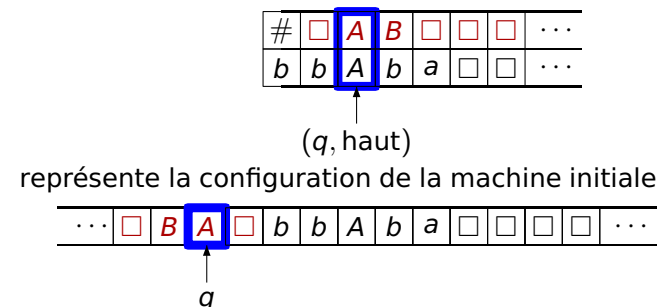
- ▶ On « replie » la bande.
- ▶ On représente la configuration initiale



51/159

Bande finie à gauche

- ▶ On a $\Gamma_2 = (\{\#\} \cup \Gamma_1) \times \Gamma_1$.
- ▶ $\#$ est un nouveau symbole servant à repérer le début de bande.
- ▶ On a $Q_2 = Q_1 \times \{\text{haut}, \text{bas}\}$.
- ▶ Une configuration de la nouvelle machine est du type



- ▶ Reste à écrire les transitions.

52/159

Bande finie vs. bi-infinie

- ▶ Inversement, toute machine travaillant sur une bande finie peut être simulée par une machine à bande bi-infinie.

53/159

Composition des MT

- ▶ En connectant l'état final (supposé unique) d'une machine M_1 à l'état initial d'une machine M_2 , on compose les fonctions calculées par les deux machines.
- ▶ Si M_1 termine en restaurant le mot d'entrée, on exécute la séquence $M_1; M_2$.
- ▶ **Exemple** : codage unaire \rightarrow binaire en utilisant l'incréméntation.

55/159

Un unique état acceptant, un unique état rejetant

A partir de $M = (Q, q_0, F, \Sigma, \Gamma, \delta)$, on construit $M' = (Q', q'_0, F', \Sigma', \Gamma', \delta')$.

- ▶ On ajoute un état OK, seul état acceptant de la nouvelle machine M' ,
- ▶ ... et des transitions de F vers OK.
- ▶ $Q' = Q \uplus \{\text{OK}\}$, $q'_0 = q_0$, $F' = \{\text{OK}\}$, $\Sigma' = \Sigma$, $\Gamma' = \Gamma$,
 $\delta' = \delta \cup (F \times \Gamma \times \{\text{OK}\} \times \Gamma \times \{-\})$.

On peut de même transformer M pour que :

- ▶ il y ait un **unique état rejetant KO**.
- ▶ tout calcul de M qui s'arrête remplace la tête en **début** de mot écrit.
- ▶ M sauvegarde son mot d'entrée et le **restaure** sur la bande quand elle s'arrête.

54/159

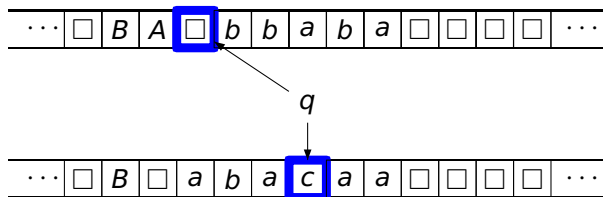
Langage des machines de Turing

- ▶ Les simplifications précédentes permettent de composer les MT.
- ▶ Composition séquentielle : $M_1; M_2$.
- ▶ Test : **Si** M_1 **alors** M_2 **sinon** M_3 .
- ▶ Boucle : **Tant que** M_1 **faire** M_2 .
- ▶ Instructions de base
 - ▶ test de lecture $R == a$,
 - ▶ écriture de a : $W(a)$,
 - ▶ Déplacements : G ou D,
 - ▶ arrêt OK ou KO.

56/159

Plusieurs bandes

- ▶ On peut utiliser plusieurs bandes.
- ▶ Initialement, le mot d'entrée est écrit sur une bande,
- ▶ Chaque transition lit un symbole sur chaque bande, et en fonction des symboles lus et de l'état, la machine
 - ▶ change d'état,
 - ▶ écrit un nouveau symbole sous chacune des têtes.
 - ▶ déplace chaque tête indépendamment.



57/159

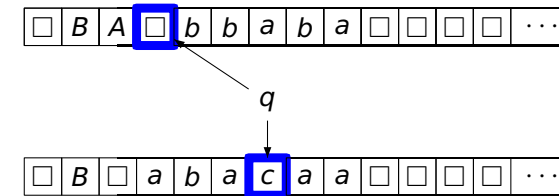
Plusieurs bandes

- ▶ **Idée** de la simulation : position des têtes stockée dans la bande.
- ▶ Chaque transition de la machine originale est simulée par
 - ▶ un aller sur la bande pour récupérer les symboles sous les têtes,
 - ▶ un retour pour simuler la transition.
- ▶ Contrairement à la simulation de bande infinie par bande finie, plusieurs mouvements sont nécessaires pour simuler un seul mouvement.
- ▶ La nouvelle machine est intuitivement « plus lente ».
- ▶ De combien ? [Retour aux classes P et NP](#)

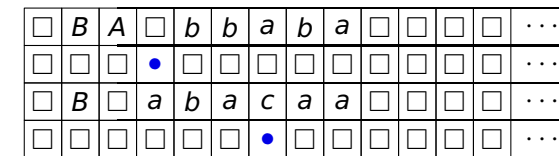
59/159

Plusieurs bandes

- ▶ Simulation possible : regrouper les bandes.



est représentée par



58/159

Machines déterministes

- ▶ Comme pour les automates finis, pour toute machine M , on peut construire une machine déterministe qui simule M .
- ▶ Idée : on parcourt l'arbre des calculs en **largeur**.
- ▶ On note r le nombre maximal de choix dans M .
- ▶ On utilise 3 bandes :
 - ▶ bande 1 : sauvegarde la valeur du mot d'entrée w .
 - ▶ bande 2 : génère dans l'ordre hiérarchique les suites finies d'entiers sur $\{1, \dots, r\}$.
 - ▶ bande 3 : effectue le calcul de M sur w correspondant à la suite de choix écrite sur la bande 2.

[Retour aux classes P et NP](#)

60/159

$$\Gamma = \Sigma \cup \{\square\}$$

- ▶ On peut faire en sorte que Γ n'utilise pas de symboles supplémentaires, à part \square .
- ▶ Idée : coder chaque symbole de Γ sur $\Sigma \cup \{\square\}$.
- ▶ Représenter
 - ▶ 0 par $(0, \square, \dots, \square)$,
 - ▶ 1 par $(1, \square, \dots, \square)$,
 - ▶ \square par $(\square, \square, \dots, \square)$,

61/159

Machines RAM

Une machine RAM est une machine possédant

- ▶ Un programme, qui est une suite finie d'instructions I_0, I_1, I_2, \dots
- ▶ Une suite infinie de registres R_0, R_1, R_2, \dots
- ▶ Un registre spécial : compteur de programme **PC** (program counter).
- ▶ Une bande d'entrée sur laquelle la machine lit ses données.
- ▶ Une bande de sortie sur laquelle la machine écrit ses résultats.

63/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

62/159

Machines RAM : registres

- ▶ Chaque registre peut mémoriser n'importe quel entier positif ou nul.
- ▶ Initialement, tous les registres ont comme valeur 0.
- ▶ Le registre **PC** contient le numéro de la prochaine instruction à exécuter.
- ▶ Les autres registres contiennent des valeurs (entières positives) manipulées par la machine au cours du calcul.

64/159

Machines RAM : instructions

Les instructions sont de 4 types :

1. Manipulations de registres.
2. Opérations arithmétiques.
3. Sauts et arrêt.
4. Entrées/sorties.

65/159

Machines RAM : sauts et arrêt

- ▶ **jump** $\langle k \rangle$: saut inconditionnel à l'instruction I_k .
- ▶ **jzero** $\langle k, n \rangle$: saut à l'instruction I_k si la valeur du registre R_n est 0.
- ▶ **stop** : fin du programme.

En pratique, on se permet de mettre des étiquettes au niveau d'instructions vers lesquelles on veut aller.

67/159

Machines RAM : Opérations arithmétiques

- ▶ **incr**(n) : incrémente de 1 la valeur contenue dans le registre R_n .
- ▶ **decr**(n) : décrémente de 1 la valeur contenue dans le registre R_n , si elle est strictement positive. Ne fait rien sinon.

66/159

Machines RAM : E/S

Sur la bande d'entrée se trouve une suite d'entiers (les entrées du programme).

- ▶ **read**(n) : lit la valeur courante de la bande d'entrée et la met dans le registre R_n . La prochaine valeur lue sera la suivante de la bande d'entrée.
- ▶ **write**(n) : écrit sur la bande de sortie la valeur contenue dans le registre R_n .

68/159

Exécution d'une machine RAM

- ▶ La machine exécute les instructions en commençant par I_0 .
- ▶ Une fois l'instruction I_k exécutée, c'est l'instruction I_{k+1} qui est exécutée, sauf après une instruction de saut ou **stop**.

69/159

Machines RAM : simulation par machines de Turing

Une MT peut simuler une RAM : tout programme RAM peut être réalisé par machine de Turing à plusieurs bandes :

- ▶ Une bande d'entrée sur laquelle on met les données, séparées par \square .
- ▶ Une bande de sortie.
- ▶ Une bande pour chaque registre (codage unaire par exemple).

71/159

Jeux d'instruction des machines RAM vs. processeurs

Les machines RAM

- + peuvent manipuler des entiers arbitraires,
- + peuvent utiliser un nombre arbitraire de registres.
- ont un jeu d'instructions plus restreint que celui des processeurs habituels.

Mais on peut reprogrammer les instructions manquantes.

- ▶ $R_m := R_n, R_n := 0$.
- ▶ Opérations arithmétiques (addition, multiplication, soustraction « tronquée », ...).
- ▶ Appel de sous-programme : **call/return**
 \rightsquigarrow nécessite un décalage de registres.

70/159

Machines RAM : simulation par machines de Turing

- ▶ Chaque instruction RAM est codée par un "sous-programme" de MT.
- ▶ Les instructions sont reliées entre elles grâce aux états. Dans l'état de la MT on code l'instruction RAM I_k simulée.

Exemples.

- ▶ **incr/decr** : machines incrémentation/décrémentation déjà vues, travaillant sur la bande correspondante au registre.
- ▶ **jump $\langle k \rangle$** : correspond aux transitions $j \xrightarrow{x/x/-} k$ pour tout $x \in \Gamma$.
- ▶ **jzero $\langle k, n \rangle$** : idem si la bande du registre R_n est vide. Sinon, aller en $j + 1$.
- ▶ **stop** : aller dans l'état **OK**.

72/159

À quoi sert cette simulation ?

- ▶ On peut compiler un programme C (sans entrée-sortie) vers un programme RAM.
- ▶ La simulation précédente montre qu'on peut compiler tout programme RAM en machine de Turing (déterministe).
- ▶ Tout programme sans entrée-sortie dans n'importe quel langage de programmation se compile en une machine de Turing.

73/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

75/159

Simulation inverse

Inversement, on peut simuler toute machine de Turing par une RAM. Idée de simulation, pour machine à bande finie à gauche :

- ▶ Mémoriser chaque configuration uqv par 2 entiers + état courant q (codé par exemple par l'instruction RAM courante).
- ▶ Le mot $u \in \Gamma^*$ est codé par le premier entier (bit le moins significatif à droite), et le mot v par le deuxième entier (bit le moins significatif à gauche).
- ▶ Chaque instruction de MT est simulée par un bout de code RAM. Par exemple, un mouvement de tête vers la droite signifie une multiplication du premier entier (qui gagne un bit) et une division du deuxième entier (qui perd un bit).
- ▶ Suite en TD...

74/159

Définitions

- ▶ Un langage L est **semi-décidable** (ou **rékursivement énumérable**) s'il est le langage accepté par une machine de Turing.
- ▶ **RE** = classe des langages semi-décidables.
- ▶ **Note** Sur un mot $\Sigma^* \setminus L$, la machine de Turing peut ne pas s'arrêter : **3 possibilités OUI/NON/ ?**
- ▶ Un langage L est **décidable** (ou **rékursif**) s'il est le langage accepté par une machine de Turing **qui s'arrête sur toute entrée** : **2 possibilités OUI/NON**
- ▶ **R** = classe des langages décidables.

76/159

Lagages RE et R

- ▶ On a clairement $R \subseteq RE$.
- ▶ La classe des langages RE est fermée par union.
- ▶ La classe des langages R est fermée par union et complément.
- ▶ Si L et $\Sigma^* \setminus L$ sont dans RE, alors ils sont dans R.
 - ▶ On construit une machine qui simule en parallèle les machines M et N acceptant L et $\Sigma^* \setminus L$ (elle alterne un pas de calcul de M , un pas de calcul de N).
 - ▶ Cette machine s'arrête si l'une des machines M et N s'arrête.
 - ▶ Un mot appartient soit à L , soit à $\Sigma^* \setminus L$, donc elle **s'arrête toujours**.

77/159

Le langage diagonal

- ▶ On numérote les mots de $\Sigma = \{0, 1\}^*$ dans l'ordre **hiérarchique**.
 - ▶ longueur d'abord,
 - ▶ ordre lexicographique pour une longueur donnée.

$w_0 = \epsilon, w_1 = 0, w_2 = 1, w_3 = 00, w_4 = 01, w_5 = 10, w_6 = 11, \dots$

- ▶ **Note** Une MT peut calculer le numéro d'un mot.
- ▶ On note M_i la machine telle que $\langle M_i \rangle = w_i$ (par convention $\mathcal{L}(M_i) = \emptyset$ si w_i n'est pas le code d'une machine de Turing).
- ▶ **Proposition** Le langage

$$L_d = \{w_i \mid w_i \notin \mathcal{L}(M_i)\}$$

n'est **pas** RE.

79/159

Codage des MT

- ▶ On travaille sur $\Sigma = \{0, 1\}$, et on peut supposer $\Gamma = \{0, 1, \square\}$.
- ▶ On ne considère que les MT à un unique état OK.
- ▶ Toute MT de ce type peut se coder sur l'alphabet $\{0, 1\}$.
- ▶ On code une transition $p_i \xrightarrow{a_j, a_k, d_\ell} q_m$ par $0^i 10^j 10^k 10^\ell 10^m$.
- ▶ On code la MT M par la suite de ces transitions, séparées entre elles par 11, avec deux blocs 111 en début et fin.
- ▶ On note ce code $\langle M \rangle$.
- ▶ Si w est un mot, on note $\langle M, w \rangle$ le code de M suivi par w .

78/159

Réductions

- ▶ Soient P_A et P_B deux problèmes.
- ▶ On note L_A l'ensemble des instances positives de P_A .
- ▶ On note L_B l'ensemble des instances positives de P_B .
- ▶ Une réduction de P_A vers P_B est une **fonction calculable par MT** $f : \Sigma^* \rightarrow \Sigma^*$ telle que

$$x \in L_A \iff f(x) \in L_B.$$

- ▶ On note $P_A \leq P_B$ (P_A **se réduit** à P_B)
- ▶ L'existence d'une réduction de P_A vers P_B assure que
 - ▶ si P_B est décidable, P_A l'est aussi,
 - ▶ si P_A est indécidable, P_B l'est aussi.

80/159

Réductions - quelques avertissements

- ▶ “ P_A se réduit à P_B ” ne signifie **PAS** que le problème P_B est plus facile que P_A . Cela signifie plutôt que la recherche d’une solution pour P_A sur une instance x donnée peut être ramenée à la recherche d’une solution pour P_B sur l’instance $f(x)$.
- ▶ La notion de réduction est **symétrique** : x est une instance positive de P_A **SI ET SEULEMENT SI** $f(x)$ est une instance positive de P_B .

81/159

Simple-PCP

- ▶ Dans **Simple-PCP** on a les restrictions suivantes :
 1. u_i et v_i ont la **même longueur**, pour tout $2 \leq i \leq n-1$ (on suppose $n \neq 1$).
 2. La solution de PCP doit commencer par l’indice 1 et terminer par l’indice n . De plus, ces indices ne peuvent pas être utilisés au milieu.
- ▶ Exemple : $(u_1, v_1) = (a, ab)$, $(u_2, v_2) = (ba, aa)$ et $(u_3, v_3) = (aa, a)$. La séquence 1,2,3 est une solution de **Simple-PCP**.
- ▶ Rq. 1 : pour qu’une solution existe, il faut que $|u_1| - |v_1| = |v_n| - |u_n|$. Soit donc $k := |u_1| - |v_1| = |v_n| - |u_n|$ et supposons que $k > 0$.
- ▶ Rq. 2 : Pour chaque séquence $1 = i_1, i_2, \dots, i_k$ où $i_j \neq 1, n$ pour tout j , on a :

$$|u_{i_1} \dots u_{i_k}| - |v_{i_1} \dots v_{i_k}| = k$$

83/159

Problème de correspondance de Post (1946)

- ▶ **Donnée** : n paires de mots $(u_1, v_1), \dots, (u_n, v_n)$.
- ▶ **Question** : existe-il une suite finie i_1, \dots, i_k telle que

$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$$

[A noter : les suites d’indices sont les mêmes.]

- ▶ \rightsquigarrow On peut voir les couples de mots comme des dominos.

a	aa	ba	bab
ab	a	aa	$abba$

- ▶ Une solution : $a.bab.ba.aa.aa = ab.abba.aa.a.a$
- ▶ Suite d’indices : 1, 4, 3, 2, 2.

PCP est un problème indécidable (voir TD).

82/159

Réduction de Simple-PCP au problème d’accessibilité

- ▶ On peut donc chercher une solution de **Simple-PCP** dans un graphe orienté **fini** : les sommets sont les mots de longueur k . On a un arc de u vers v s’il existe un couple (u_j, v_j) tel que $u u_j = v_j v$.
Le sommet de départ est le mot w tel que $u_1 = v_1 w$ et le sommet d’arrivée est w' tel que $w' u_n = v_n$. L’instance de **Simple-PCP** donnée a une solution **si et seulement si** il existe un chemin de w à w' .
- ▶ On a donc réduit **Simple-PCP** au problème d’accessibilité dans les graphes orientés. Comme ce dernier problème est décidable, **Simple-PCP** l’est aussi.
- ▶ C’est possible de réduire aussi dans le sens inverse, du problème d’accessibilité à **Simple-PCP**.

84/159

Réductions : deuxième exemple

Problème de l'arrêt borné :

- ▶ **Données** : MT M , entrée w et entier k .
- ▶ **Question** : Est-ce que M accepte w en moins de k pas ?

L'arrêt borné se réduit aussi à un problème d'accessibilité : on construit un graphe orienté $G(M, w, k)$ dont les sommets sont toutes les configurations uqv de taille au plus k (donc $|uv| \leq k$). On met des arcs $uqv \rightarrow u'q'v'$ si $uqv \vdash_M u'q'v'$.

M accepte w en moins de k pas de calcul si et seulement s'il existe un chemin de longueur au plus k dans $G(M, w, k)$ de la configuration initiale $c_0(w)$ à une configuration finale.

85/159

Un autre langage indécidable

- ▶ Le langage

$$L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$$

n'est pas RE.

- ▶ On montre d'abord que $L_{\neq \emptyset} := \Sigma^* \setminus L_\emptyset$ est RE.
- ▶ Si L_\emptyset était dans RE, il serait dans R. Alors L_u serait aussi dans R :
- ▶ On construit une réduction $L_u \leq L_{\neq \emptyset}$.
 - ▶ À partir de M et w , on construit M' qui efface son entrée et lance M sur w .
 - ▶ M' accepte si et seulement si M s'arrête sur OK.

87/159

Langage et machine universelle

- ▶ Le langage (universel)

$$L_u = \{\langle M, w \rangle \mid M \text{ accepte } w\}$$

est RE, mais pas R.

- ▶ S'il était dans R, alors $\Sigma^* \setminus L_d$ le serait aussi.
- ▶ À partir d'une machine de Turing hypothétique M qui s'arrête sur toute entrée et telle que $\mathcal{L}(M) = L_u$, on **construit** une machine de Turing M' qui s'arrête sur toute entrée et telle que $\mathcal{L}(M') = \Sigma^* \setminus L_d$.
- ▶ Or, une telle machine M' n'existe pas, car sinon, L_d serait dans R.
- ▶ On a construit une **réduction** de $\Sigma^* \setminus L_d$ à L_u : $\Sigma^* \setminus L_d \leq L_u$.

86/159

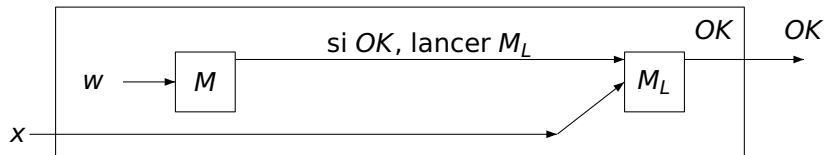
Théorème de Rice

- ▶ Soit \mathcal{E} l'ensemble des langages RE de $\{0, 1\}^*$.
- ▶ Une **propriété des langages RE** est un sous-ensemble \mathcal{P} de \mathcal{E} .
- ▶ Une propriété $\mathcal{P} \subseteq \mathcal{E}$ est **triviale** si $\mathcal{P} = \emptyset$ ou $\mathcal{P} = \mathcal{E}$.
- ▶ **Attention** Ne pas confondre $\mathcal{P} = \emptyset$ (\mathcal{P} ne contient aucun langage) et $\mathcal{P} = \{\emptyset\}$ (\mathcal{P} ne contient que le langage vide).

88/159

Théorème de Rice

- ▶ Toute propriété non triviale \mathcal{P} des langages RE est non décidable.
- ▶ **Attention** : il s'agit d'une propriété des langages, non des MT.
- ▶ **Preuve** A nouveau une réduction à partir de L_U .
- ▶ Quitte à changer \mathcal{P} et $\mathcal{E} \setminus \mathcal{P}$, on peut supposer $\emptyset \notin \mathcal{P}$.
- ▶ Comme $\mathcal{P} \neq \emptyset$, il existe $L \in \mathcal{P}$. Soit M_L telle que $\mathcal{L}(M_L) = L$.
- ▶ À partir de M, w donnés, on construit (par machine de Turing !) la MT suivante :



- ▶ Cette machine accepte $\emptyset \notin \mathcal{P}$ si $\langle M, w \rangle \notin L_U$, et $L \in \mathcal{P}$ sinon.
- ▶ Donc si \mathcal{P} était dans R, L_U le serait aussi.

89/159

Problème de l'arrêt

Les problèmes suivants sont **indécidables** :

- ▶ étant donnée une MT M et un mot w , M s'arrête-t-elle sur w ?
- ▶ étant donnée une MT M , s'arrête-t-elle sur ε ?
- ▶ étant donnée une MT M , s'arrête-t-elle sur au moins une entrée ?
- ▶ étant donnée une MT M , s'arrête-t-elle sur toute entrée ?
- ▶ étant donnée une MT M et un état q de M , M utilise-t-elle l'état q sur l'entrée ε ?

90/159

L'indécidabilité hors du monde des MT : le PCP

- ▶ Problème de correspondance de Post (1946).
- ▶ **Donnée** : n paires de mots $(u_1, v_1), \dots, (u_n, v_n)$.
- ▶ **Question** : existe-il une suite finie i_1, \dots, i_k telle que

$$u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$$

[A noter : les suites d'indices sont les mêmes.]

- ▶ \rightsquigarrow On peut voir les couples de mots comme des dominos.

a	aa	ba	bab
ab	a	aa	abba

- ▶ Une solution : $a.bab.ba.aa.aa = ab.abba.aa.a.a$
- ▶ Suite d'indices : 1, 4, 3, 2, 2.

91/159

Le PCP modifié (PCPM)

- ▶ Problème de correspondance de Post (1946).
- ▶ **Donnée** : n paires de mots $(u_1, v_1), \dots, (u_n, v_n)$.
- ▶ **Question** : existe-il une suite finie i_1, \dots, i_k telle que $i_1 = 1$ et

$$u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$$

A noter : les suites d'indices sont les mêmes, ...
... et le premier indice est 1.

92/159

Indécidabilité du PCP et PCPM

- ▶ On montre que

$$L_u \leq \text{PCPM} \leq \text{PCP}.$$

- ▶ Comme le langage universel L_u est non décidable, il en est de même de PCPM et de PCP.
- ▶ Accessoirement, on peut montrer que $\text{PCP} \leq \text{PCPM}$.

93/159

PCPM \leq PCP : plus difficile

- ▶ Supposons donné un algorithme pour résoudre le PCP.
- ▶ On introduit une nouvelle lettre \$, et pour $a_1 \cdots a_k \in A^+$, soient $p(a_1 \cdots a_k) = \$a_1 \cdots \a_k et $s(a_1 \cdots a_k) = a_1\$ \cdots a_k\$$.
- ▶ Soit $(u_1, v_1), \dots, (u_n, v_n)$ une instance de PCPM.
- ▶ Soient les $2n + 1$ mots suivants :

$$\begin{aligned} x_i &= p(u_i), & y_i &= s(v_i) \\ x_{n+i} &= p(u_i)\$, & y_{n+i} &= s(v_i) \\ x_{2n+1} &= p(u_1), & y_{2n+1} &= \$s(v_1). \end{aligned}$$

- ▶ Le PCPM sur l'instance $((u_\ell, v_\ell))_{1 \leq \ell \leq n}$ a une solution si et seulement si le PCP sur l'instance $((x_\ell, y_\ell))_{1 \leq \ell \leq 2n+1}$ en a une.

95/159

PCP \leq PCPM

- ▶ Si on a un algorithme pour résoudre PCPM, on a un algorithme pour résoudre PCP.
- ▶ Il suffit de résoudre n PCPM différents, selon le mot avec lequel on commence.

94/159

Indécidabilité du PCPM

- ▶ On rappelle que $L_u = \{\langle M, w \rangle \mid M \text{ accepte } w\}$ est indécidable.
- ▶ On montre $L_u \leq \text{PCPM}$.
- ▶ Étant donné une MT M et un mot w , on construit une instance $((u_\ell, v_\ell))_{1 \leq \ell \leq n}$ de PCPM telle que

$$\langle M, w \rangle \in L_u \iff \text{PCP sur l'instance } ((u_\ell, v_\ell))_{1 \leq \ell \leq n} \text{ a une solution}.$$

- ▶ On peut supposer que
 - ▶ le seul état d'arrêt de M est q_{OK} ,
 - ▶ M déplace sa tête à chaque transition.

96/159

La réduction $L \leq \text{PCPM}$

- ▶ **Idée** : la seule solution sera la suite des configurations sur w .
- ▶ La partie **bleue** est en retard d'une configuration.
- ▶ Domino 1 : $(\#, \#q_0w\#)$. Les autres dominos :
- ▶ Dominos de copie : (a, a) , $(\#, \#)$,
- ▶ Dominos de transitions :
- ▶ Si $p \xrightarrow{a,b,\rightarrow} q \in \delta$, on met un domino (pa, bq) .
- ▶ Si $p \xrightarrow{a,b,\leftarrow} q \in \delta$, on met un domino (xpa, qxb) pour tout $x \in \Gamma$.
- ▶ Si $p \xrightarrow{\square,b,\rightarrow} q \in \delta$, on met un domino $(p\#, bq\#)$.
- ▶ Si $p \xrightarrow{\square,b,\leftarrow} q \in \delta$, on met un domino $(xp\#, qxb\#)$ pour tout $x \in \Gamma$.
- ▶ Dominos de synchronisation en fin de calcul :
 - ▶ pour chaque $a, b \in \Gamma$: (aq_{OK}, q_{OK}) , $(q_{OK}b, q_{OK})$,
 - ▶ $(q_{OK}\#\#, \#)$.

97/159

Révisions : vrai/faux

- (a) L'ensemble des graphes orientés finis est dénombrable.
- (b) Tout langage fini est décidable.
- (c) Pour tous $K, L \subseteq \Sigma^*$ avec $\Sigma = \{0, 1\}$, si K se réduit à L , alors $\Sigma^* \setminus K$ se réduit à $\Sigma^* \setminus L$.
- (d) Tout langage sur un alphabet à une lettre est décidable.
- (e) Étant donné une machine de Turing M , un mot w et un entier k , on peut décider si M accepte w en au plus k pas de calcul.
- (f) Étant données deux machines de Turing M_1 et M_2 , on peut décider si $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$.
- (g) Le langage des mots sur l'alphabet ASCII représentant un programme C syntaxiquement correct est décidable.
- (h) Le complément de tout ensemble semi-décidable est aussi semi-décidable.

99/159

Quelques autres problèmes indécidables

Les problèmes suivants sont indécidables :

- ▶ Étant donné un jeu fini de tuiles carrées, avec conditions de compatibilité entre côtés, déterminer si on peut paver le $1/4$ de plan.
- ▶ Étant donnée une grammaire, déterminer si elle est ambiguë.
- ▶ Étant donné un nombre fini de matrices 3×3 à coefficients entiers, déterminer si un produit permet d'annuler la composante $(3,2)$.
- ▶ Étant donnée une suite calculable d'entiers, déterminer si elle converge.

98/159

L'indécidabilité hors du monde des MT : le 10e problème de Hilbert

- ▶ 10e problème de Hilbert : résolution d'équations diophantiennes.
- ▶ Exemple : est-ce que l'équation $x^3 + 3x^2y + 3xy^2 + y^3$ a une solution avec x, y entiers ?
- ▶ Le problème, formulé par D. Hilbert en 1900, a reçu une solution négative (**indécidabilité**) par Yu. Matyasevich (suite aux avancées faites par J. Robinson, H. Putnam et M. Davis).
- ▶ Ce résultat d'indécidabilité raffine le **théorème d'incomplétude de Gödel**, qui dit que la théorie du premier ordre de l'arithmétique (formules avec quantificateurs, addition/multiplication, coefficients et variables entiers) est indécidable.

100/159

Equations diophantiennes

- ▶ Une équation diophantienne a la forme $P(x_1, \dots, x_n) = 0$, avec P **polynôme** avec coefficients entiers.
- ▶ **Dioph(\mathbb{Z})** demande si une équation diophantienne a une solution avec tous les x_i **entiers**; **Dioph(\mathbb{N})** demande l'existence d'une solution avec les x_i **entiers positifs**.
- ▶ Une équation diophantienne **exponentielle** permet de plus l'**exponentiation**. Exemple : $2x^y - 7x^2 = 1$.
- ▶ **ExpDioph(\mathbb{N})** demande si une équation diophantienne exponentielle a une solution avec tous les x_i **entiers positifs**.
- ▶ La solution de Matyasevich est de réduire le problème de l'arrêt (des RAM par exemple) à ExpDioph(\mathbb{N}). Ensuite elle réduit ExpDioph(\mathbb{N}) à Dioph(\mathbb{N}) par des méthodes de la théorie des nombres. Nous montrons ici juste la première réduction (facile).

101/159

Coder l'arrêt d'une RAM en arithmétique

- ▶ Pour le codage on va utiliser les variables (entières) suivantes :
 - ▶ B : sa valeur est la **base** pour le codage de séquences d'entiers (voir transparent précédent).
 - ▶ L : sa valeur est le **nombre de pas** de calcul de la RAM P avant d'atteindre stop. Les séquences d'entiers qu'on va coder en base B seront de longueur $L + 1$ (pas : $0, 1, \dots, L$).
 - ▶ W_p ($1 \leq p \leq k$) : pour chaque registre R_p de P . La valeur de W_p représente la suite (n_0^p, \dots, n_L^p) des **contenus de R_p** au cours du calcul : n_ℓ^p est la valeur de R_p au pas ℓ du calcul.
 - ▶ L_i ($1 \leq i \leq m$) : pour chaque instruction I_i de P . La valeur de L_i représente une séquence (s_0^i, \dots, s_L^i) telle que $s_\ell^i = 1$ si **PC = I_i** au pas ℓ , et $s_\ell^i = 0$ sinon.
- ▶ **Exemple** : Soient $B = 10$, $L = 5$. Si R_1 contient les valeurs **0,1,2,1,1,0** au cours du calcul, alors $W_1 = 11210$. Si l'instruction I_1 est exécutée au pas **2, 3 et 5**, alors $L_1 = 101100$.

103/159

Coder l'arrêt d'une RAM en arithmétique

- ▶ On part du modèle RAM avec instructions incr, decr, jump, jzero, stop. De plus, on suppose que la RAM s'arrête avec tous les registres nuls, et que decr ne s'applique que sur des registres de valeur non-nulle. On veut coder l'**arrêt** d'une RAM P (sans entrée/sortie).
- ▶ Soit P une suite de m instructions, étiquetées $1, \dots, m$, avec la dernière instruction égale à stop (ceci est l'unique occurrence de stop).
- ▶ On va coder une **séquence** finie d'entiers (n_0, \dots, n_ℓ) par un **seul** entier $\sum_{i=0}^{\ell} n_i B^i$ de telle façon que B (une variable) est plus grand que toutes les valeurs des n_i (ainsi, on peut décoder n_i en tant que coefficient de B^i). La valeur de B doit être aussi suffisamment grande pour que par exemple quand on somme (les valeurs codant) 2 séquences, il suffit d'additionner les coefficients base B .

102/159

Coder l'arrêt d'une RAM en arithmétique

- ▶ En utilisant des équations (exponentielles) on va demander que B est une grande puissance de 2 :

$$B = 2^K \quad \wedge \quad B > \max\{k, m, 2 \cdot L\}$$

En particulier, $B > 2L$ signifie que B dépasse le double de la valeur maximale d'un registre (après L pas de calcul).

- ▶ Rq : avec $B = 2^K$, tout nombre en base B peut s'écrire en binaire comme suite de blocs 0/1 de longueur K .
- ▶ Pour deux entiers x, y on écrit $x \trianglelefteq y$ si – en **représentation binaire** – chaque bit de x est plus petit ou égal au bit correspondant de y . Exemple : $4 \trianglelefteq 14$, car $\text{bin}(4) = 100, \text{bin}(14) = 1110$. Mais $4 \not\trianglelefteq 8$.
- ▶ On utilisera une variable T pour désigner la séquence constituée que de 1 (en base B) : $T = \sum_{i=0}^L B^i$. Cette variable est l'unique solution de l'équation $1 + (B - 1)T = B^{L+1}$.

104/159

Coder l'arrêt d'une RAM en arithmétique

- ▶ La relation $L_i \leq T$ exprime que la séquence L_i (en base B) est constituée que des 0 et 1.
- ▶ En combinaison avec la relation précédente, $\sum_{i=1}^m L_i = T$ exprime qu'à chaque pas de calcul, une unique instruction est exécutée.
- ▶ La relation $1 \leq L_1$ garantit que la première instruction est I_1 . La relation $B^L \leq L_m$ garantit que la dernière instruction est I_m (stop).
- ▶ La relation $W_p \leq B^{L+1} - B$ garantit que la valeur initiale de R_p est 0.
- ▶ Si $I_i : \text{jump}(k)$ alors on garantit avec $B \cdot L_i \leq L_k$ que I_i est suivi par I_k . La multiplication par B fait le lien entre le pas ℓ et le pas $\ell + 1$.

105/159

ExpDioph(\mathbb{N})

- ▶ Finalement, on traduit la relation \leq par des équations exponentielles.
- ▶ D'abord, on peut montrer que $x \leq y$ ssi $\binom{x}{y}$ est impair.
- ▶ Le coefficient binomial $\binom{x}{y}$ est le coefficient de U^y dans la représentation en base U de $(1+U)^x$ (il faut choisir U plus grand que $\binom{x}{y}$, par exemple $U > 2^x$). On a que $\binom{x}{y}$ est l'unique solution z des équations suivantes :

$$U = 2^x + 1 \quad \wedge \quad V < U^y \quad \wedge \quad z < U \quad \wedge \quad (1+U)^x = WU^{y+1} + zU^y + V$$

- ▶ Pour résumer : on obtient un système d'équations exponentielles (qu'on peut réduire à une unique équation – comment ?), en utilisant les variables B, L, W_p, L_i, T ainsi que des variables auxiliaires t.q. les U, V, W qui traduisent les coefficients binomiaux.

107/159

Coder l'arrêt d'une RAM en arithmétique

- ▶ Pareil pour $I_i : \text{incr}(j)$ (ou decr) : $B \cdot L_i \leq L_{i+1}$ (après I_i on exécute I_{i+1}). On rajoute que les valeurs de R_p sont cohérentes :

$$W_p = B \cdot (W_p + \sum_i L_i - \sum_{i'} L_{i'})$$

On somme ici sur tous les i t.q. $I_i : \text{incr}(p)$ et sur tous les i' t.q. $I_{i'} : \text{decr}(p)$.

- ▶ Soit maintenant $I_i : \text{jzero}(j, p)$ (si R_p vaut zéro, continuer avec I_j , sinon avec I_{i+1}).
La relation $B \cdot L_i \leq L_j + L_{i+1}$ garantit que I_i est suivie soit par I_j , ou par I_{i+1} . La relation $B \cdot L_i \leq L_{i+1} + B \cdot T - 2 \cdot W_p$ traduit le test $R_p = ? 0$.

106/159

Révisions : problèmes indécidables

Les problèmes suivants sont indécidables :

1. P1 : Étant donnée une MT M , décider si elle accepte au moins 7 mots.
2. P2 : Étant donnée une MT M , décider si elle s'arrête toujours avec écrit sur sa bande 2008.
3. P3 : Étant donnée une MT M et un état p de M , décider si M utilise l'état p sur au moins une entrée.
4. P4 : Étant donnée une MT M et un état p de M , décider si M utilise l'état p sur toute une entrée.
5. P5 : Étant donnée une MT M , décider si elle accepte n'importe quel mot représentant (en binaire) un entier premier.

108/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

109/159

Comparaison de problèmes

► Problèmes **difficiles** :

1. SAT : savoir si une formule booléenne (sans quantificateurs) a une valuation satisfaisante (voir <http://www.dwheeler.com/essays/minisat-user-guide.html>).
2. Graphes hamiltoniens : est-ce qu'un graphe possède un circuit qui passe par chaque sommet exactement une fois (voir <http://www.tsp.gatech.edu/index.html>).
3. 3-colorabilité

► Problèmes encore **plus difficiles** :

1. Certains jeux à 2 joueurs.
2. QSAT : savoir si une formule booléenne quantifiée est valide.
3. Savoir si l'intersection de n automates finis est non-vide.

- **Difficile** veut dire qu'on ne connaît pas d'algorithmes polynomiaux pour ces problèmes. La théorie de la **complexité** classe les problèmes selon les ressources (en temps et/ou mémoire) **nécessaires** pour les résoudre.

111/159

Comparaison de problèmes

Spectre large des niveaux de difficulté :

► Problèmes **faciles** (sur les graphes) :

1. Accessibilité : étant donné un graphe et 2 sommets s, t , est-ce qu'il y a un chemin de s à t ? (solution : Dijkstra, Floyd-Warshall)
2. Connexité : est-ce qu'un graphe donné est connexe ? (solution : accessibilité)
3. Graphes eulériens : est-ce qu'un graphe possède un circuit qui passe par chaque arc exactement une fois ? (solution : connexité)
4. 2-colorabilité : est-ce qu'on peut colorier un graphe avec 2 couleurs t.q. pour chaque arête uv , les 2 sommets u, v ont des couleurs différentes ? (solution : DFS)

- **Facile** veut dire qu'on connaît des algorithmes **polynomiaux** pour résoudre ces problèmes.

110/159

Littéraux, clauses, et formules CNF

Étant données des variables x_1, x_2, \dots :

- un littéral est soit une variable x_i , soit la négation d'une variable $\neg x_i$.

- Une **clause** est une disjonction de littéraux.

Exemple : $x_1 \vee \neg x_2 \vee \neg x_4 \vee x_5$.

- Une **3-clause** est une clause avec 3 littéraux différents.

Exemple : $x_1 \vee \neg x_2 \vee \neg x_4$.

- Une formule CNF est une conjonction de clauses.

- Une formule 3-CNF est une conjonction de 3-clauses.

Exemple : $(x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_3)$.

112/159

SAT

Le problème **SAT** est le suivant :

- ▶ **Donnée** : une formule **CNF** sur des variables x_1, x_2, \dots, x_n .
- ▶ **Question** : existe-t-il une **valuation** des variables $\sigma : \{x_i \mid 1 \leq i \leq n\} \rightarrow \{\text{tt}, \text{ff}\}$ qui rend la formule vraie ?

113/159

Réduction SAT vers 3-SAT

- ▶ À toute instance φ de SAT, on associe une instance $\tilde{\varphi}$ de 3-SAT tq.

φ est satisfaisable $\iff \tilde{\varphi}$ est satisfaisable.

Remarque. On va construire $\tilde{\varphi}$ en temps polynomial par rapport à $|\varphi|$.

115/159

3-SAT

Le problème **3-SAT** est le suivant :

- ▶ **Donnée** : une formule **3-CNF** sur des variables x_1, x_2, \dots, x_n .
- ▶ **Question** : existe-t-il une **valuation** des variables $\sigma : \{x_i \mid 1 \leq i \leq n\} \rightarrow \{\text{tt}, \text{ff}\}$ qui rend la formule vraie ?

Le problème **3-SAT** est donc moins général que **SAT**.

114/159

Réduction SAT vers 3-SAT

Si $\varphi = c_1 \wedge \dots \wedge c_k$ où chaque c_i est une clause, on construit $\tilde{\varphi} = \varphi_1 \wedge \dots \wedge \varphi_k$, où

- ▶ Chaque φ_i est une conjonction de 3-clauses,
- ▶ φ_i utilise les variables de c_i , + éventuellement de nouvelles variables.
- ▶ Si une affectation des variables rend c_i vraie, on peut la compléter pour rendre φ_i vraie.
- ▶ Inversement, si une affectation des variables rend φ_i vraie, sa restriction aux variables de c_i rend c_i vraie.

116/159

Réduction SAT vers 3-SAT : construction de φ_i

- Si $c_i = \ell_1$ (un littéral), on ajoute 2 variables y_i, z_i et

$$\varphi_i = (\ell_1 \vee y_i \vee z_i) \wedge (\ell_1 \vee \neg y_i \vee z_i) \wedge (\ell_1 \vee y_i \vee \neg z_i) \wedge (\ell_1 \vee \neg y_i \vee \neg z_i).$$

- Si $c_i = \ell_1 \vee \ell_2$ (2 littéraux), on ajoute 1 variable y_i et

$$\varphi_i = (y_i \vee \ell_1 \vee \ell_2) \wedge (\neg y_i \vee \ell_1 \vee \ell_2).$$

- Si c_i est une 3-clause : $\varphi_i = c_i$.
- Si $c_i = \ell_1 \vee \dots \vee \ell_k$ avec $k \geq 4$, on ajoute $k - 3$ variables $t_{i,1}, \dots, t_{i,k-3}$

$$\varphi_i = (t_{i,1} \vee \ell_1 \vee \ell_2) \wedge (\neg t_{i,1} \vee \ell_3 \vee t_{i,2}) \wedge (\neg t_{i,2} \vee \ell_4 \vee t_{i,3}) \wedge \dots \wedge (\neg t_{i,k-3} \vee \ell_{k-1} \vee \ell_k)$$

117/159

Réduction SAT vers 3-SAT

On vérifie qu'avec la construction précédente :

- Si une affectation des variables rend chaque c_i vraie, on la complète facilement pour rendre chaque φ_i vraie.
- Inversement, si une affectation des variables rend chaque φ_i vraie, la restriction de cette affectation aux variables de c_i rend c_i vraie. Donc

c_i est satisfaisable

\iff

φ_i est satisfaisable avec les mêmes valeurs pour les variables de c_i .

- Comme les variables ajoutées dans φ_i n'apparaissent que dans φ_i :

φ est satisfaisable $\iff \tilde{\varphi}$ est satisfaisable.

119/159

Réduction SAT vers 3-SAT : exemple

- $\varphi = (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4 \vee x_5) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_4)$, alors

- La construction donne

$$\begin{aligned} \tilde{\varphi} = & (t_{1,1} \vee x_1 \vee \neg x_2) \wedge (\neg t_{1,1} \vee x_3 \vee t_{1,2}) \wedge (\neg t_{1,2} \vee \neg x_4 \vee x_5) \\ & \wedge (y_2 \vee x_1 \vee \neg x_2) \wedge (\neg y_2 \vee x_1 \vee \neg x_2) \\ & \wedge (\neg x_1 \vee x_2 \vee x_4) \end{aligned}$$

118/159

Réduction SAT vers 3-SAT

Récapitulatif. A partir de φ CNF, on a construit $\tilde{\varphi}$ 3-CNF telle que

φ est satisfaisable $\iff \tilde{\varphi}$ est satisfaisable.

On a donc

$SAT \leq 3-SAT.$

Inversement, comme 3-SAT est un problème moins général que SAT :

$3-SAT \leq SAT.$

120/159

3-coloration

Le problème **3-coloration** est le suivant :

- **Donnée** : un graphe non orienté G .
- **Question** : existe-t-il une 3-coloration de G ?

121/159

Réduction 3-SAT vers 3-coloration

- À toute instance φ de 3-SAT, on associe une instance G_φ de 3-coloration tq.

φ est satisfaisable $\iff G_\varphi$ admet une 3-coloration.

On utilise des sous-graphes (appelés *gadgets*) pour coder

- les littéraux vrais dans une évaluation qui satisfait φ ,
- les opérateurs logiques \wedge et \vee .

123/159

Réduction 3-coloration vers 3-SAT

- À toute instance $G = (V, E)$ de 3-coloration on associe une formule φ_G tq.
- G est 3-coloriable $\iff \varphi_G$ est satisfaisable
- Littéraux : $\{v_i \mid v \in V, i \in \{1, 2, 3\}\}$
 (les couleurs sont 1,2,3 ; v_i vrai signifiera que le sommet v est colorié par la couleur i)
- Clauses :

1. Chaque sommet est colorié par une (et une seule) couleur :

$$\bigwedge_{v \in V} ((v_1 \vee v_2 \vee v_3) \wedge \bigwedge_{i \neq j} (\overline{v_i} \vee \overline{v_j}))$$

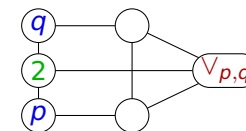
2. Deux sommets voisins n'ont pas la même couleur :

$$\bigwedge_{uv \in E, i \in \{1, 2, 3\}} (\overline{u_i} \vee \overline{v_i})$$

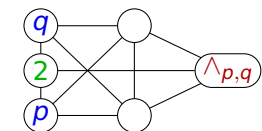
122/159

3-SAT \leq 3-coloration

- On utilise 3 sommets particuliers 0, 1, 2 reliés entre eux, qu'on peut supposer, quitte à renommer les couleurs, coloriés par 0, 1, 2.
- Pour chaque variable x_i : 2 sommets x_i et $\neg x_i$ reliés entre eux et à 2.
- Opérateurs : OU $p \vee q$ codé par :



ET $p \wedge q$ codé par :



en reliant $V_{p,q}$ et $A_{p,q}$ au sommet 2 (p et q le sont inductivement).

- $V_{p,q}$ coloriable par 1 si et seulement si p OU q sont coloriés 1.
- $A_{p,q}$ coloriable par 1 si et seulement si p ET q sont coloriés 1.
- Sommet « résultat » relié à 0 (et 2 par la construction précédente).

124/159

Clique et ensemble indépendant

Dans un graphe G non orienté

- ▶ Une **clique** pour G est un ensemble de sommets tous reliés 2 à 2.

Le problème **Clique** est le suivant :

- ▶ **Donnée** : un graphe G non orienté et un entier $K > 0$.
- ▶ **Question** : existe-t-il une clique de G de taille K ?

125/159

Réduction 3-SAT vers clique

- ▶ Soit $\varphi = (\ell_0 \vee \ell_1 \vee \ell_2) \wedge \dots \wedge (\ell_{3k-3} \wedge \ell_{3k-2} \wedge \ell_{3k-1})$.
- ▶ Le graphe G_φ a $3k$ sommets ℓ_1, \dots, ℓ_{3k} .
- ▶ Deux sommets ℓ_i, ℓ_k sont reliés si
 - ▶ ils ne proviennent pas de la même clause ($i/3 \neq k/3$), et si
 - ▶ ils ne sont pas de la forme $\ell, \neg \ell$.
- ▶ On choisit l'entier K_φ égal à k .
- ▶ On vérifie que G_φ a une clique de taille K_φ ssi φ est satisfaisable.

127/159

Réduction 3-SAT vers clique

- ▶ À toute instance φ de 3-SAT, on associe une instance G_φ, K_φ de **Clique** tq.

φ est satisfaisable $\iff G_\varphi$ a une clique de taille K_φ .

et tq. on peut construire G_φ, K_φ en temps polynomial par rapport à $|\varphi|$.

126/159

Plan

Présentation, bref historique

Ensembles dénombrables. Un paradoxe

Machines de Turing

Machines RAM

Problèmes indécidables

Réductions : logique, graphes, et problèmes sur entiers

Classes de complexité : P, NP, PSPACE.

128/159

Notations - rappels

- ▶ On s'intéresse uniquement aux mesures **asymptotiques** des ressources de calcul (temps/mémoire).
- ▶ **Exemple** Un algorithme de complexité $\leq 5n^2 - 7n + 10$ est qualifié comme algorithme **quadratique**; un algorithme de complexité $\leq 10^6 n^3$ comme algorithme **cubique**, etc.
- ▶ **Notation grand-O et petit-o** :
Soient $f, g : \mathbb{N} \rightarrow \mathbb{N}$. On écrit :
 - ▶ $f \in o(g)$ si pour tout $c > 0$ il existe $n_c \in \mathbb{N}$ tel que $f(n) \leq c \cdot g(n)$ pour tout $n \geq n_c$.
 - ▶ $f \in \mathcal{O}(g)$ s'il existent $c > 0$ et $n_0 \in \mathbb{N}$ tels que $f(n) \leq c \cdot g(n)$ pour tout $n \geq n_0$.
 - ▶ $f \in \Theta(g)$ si $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(f)$.
- ▶ Linéaire = $\Theta(n)$, quadratique = $\Theta(n^2)$, etc.

129/159

$\{a^n b^n \mid n \geq 0\}$ - quelle complexité en temps ?

On peut faire mieux :

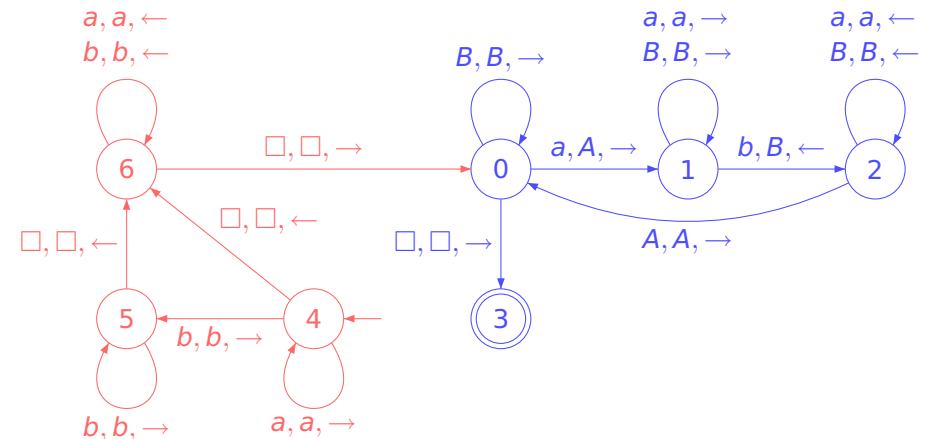
- ▶ Vérifier que l'entrée est dans a^*b^* : $\mathcal{O}(n)$.
- ▶ Répéter tant qu'il reste des a ou des b non-marqués :
 - ▶ Vérifier que le nombre total de symboles non-marqués est pair. Rejeter s'il est impair.
 - ▶ Marquer un a sur deux, et un b sur deux (en commençant par les premières occurrences).
- ▶ Si tous les a et tous les b sont marqués, accepter. Rejeter sinon.

Le temps de calcul est de $\mathcal{O}(n \log(n))$: le nombre d'itérations de la boucle interne est logarithmique (après la première itération $\frac{1}{2}$ des a sont marqués, après la deuxième itération $\frac{3}{4}$ sont marqués, etc. En fait cet algorithme compte le nombre de a (et de b) en binaire.

131/159

$\{a^n b^n \mid n \geq 0\}$ - quelle complexité en temps ?

Vérifier qu'on est dans a^*b^* : $\mathcal{O}(n)$. Vérifier que le **nombre de a et de b est le même** : $\mathcal{O}(n^2)$.



130/159

$\{a^n b^n \mid n \geq 0\}$ - quelle complexité en temps ?

On peut faire encore mieux - en utilisant plusieurs bandes :¹

- ▶ Vérifier que l'entrée est dans a^*b^* : $\mathcal{O}(n)$.
- ▶ Lire l'entrée de gauche à droite, en copiant les a sur la deuxième bande.
- ▶ Continuer de lire les b de l'entrée. Pour chaque b lu, on lit un a sur la deuxième bande. Si les nombres sont égaux, accepter.

Cet algorithme est linéaire (de complexité $\mathcal{O}(n)$).

1. Sur une seule bande, tout langage reconnu en $\mathcal{O}(n \log(n))$ est régulier.

132/159

Complexité en temps

- ▶ Soit M une machine de Turing (s'arrêtant sur toute entrée).
- ▶ Soit $\text{step}_M(w)$ le **nombre maximal** d'instructions exécutées sur l'entrée w jusqu'à l'arrêt de M .
 M étant non-déterministe, $\text{step}_M(w)$ représente la profondeur de l'arbre de calcul de M sur w .
- ▶ Soit $f_M(n) = \max \{ \text{step}_M(w) \mid |w| = n \}$.
- ▶ $f_M(n)$ mesure le temps passé par M sur une entrée de taille n , **dans le pire des cas**.

133/159

Déterminisme vs. non-déterminisme

Soit $f : \mathbb{N} \rightarrow \mathbb{N}$ t.q. $f(n) > n$.

- ▶ Toute MT M **non-déterministe** t.q. $f_M \leq f$ peut être simulée par une MT N **déterministe** t.q. $f_N \leq 2^{\mathcal{O}(f)}$.

En algorithmique on distingue les temps de calcul : linéaire, $\mathcal{O}(n \log(n))$, quadratique, $\mathcal{O}(n^6)$, etc. En théorie de la complexité on classe les problèmes de manière moins fine. L'objectif est de pouvoir **déterminer** la complexité **intrinsèque** d'un problème.

L'origine de la théorie de la complexité :

- ▶ **P** = $\bigcup_{k \geq 1} \text{DTIME}(n^k)$ **temps polynomial déterministe**
- ▶ **NP** = $\bigcup_{k \geq 1} \text{NTIME}(n^k)$ **temps polynomial non-déterministe**

135/159

Les classes de complexité $\text{DTIME}(f)$ et $\text{NTIME}(f)$

Soit $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(n) > n$.

- ▶ On note par **$\text{NTIME}(f)$** :

$$\{L \mid L \text{ est accepté par une MT } M \text{ t.q. } f_M \leq f\}$$

- ▶ On note par **$\text{DTIME}(f)$** :

$$\{L \mid L \text{ est accepté par une MT } M \text{ déterministe t.q. } f_M \leq f\}$$

Remarque : on peut définir de manière analogue le temps de calcul sur une RAM (ou d'autres modèles de calcul), ainsi que les classes DTIME et NTIME .

- ▶ **Thèse de Church qualitative** : la complexité du même algorithme sur des modèles de calcul "comparables" varie seulement de façon polynomiale.

Exemple : Toute MT M à plusieurs bandes t.q. $f_M \leq f$ peut être simulée par une MT N à une bande, t.q. $f_N \leq \mathcal{O}(f^2)$.

134/159

Classe **P** : exemples

- ▶ **Accessibilité** : étant donné un graphe et 2 sommets s, t , est-ce qu'il y a un chemin de s à t ? (solution : Dijkstra $\mathcal{O}(n^2 + m \log(n))$, Floyd-Warshall $\mathcal{O}(n^3)$)
- ▶ **Connexité** : est-ce qu'un graphe donné est connexe? (solution : accessibilité $\mathcal{O}(n + m)$)
- ▶ **Acyclicité** : est-ce qu'un graphe donné contient un cycle? (solution : accessibilité)
- ▶ **Graphes eulériens** : est-ce qu'un graphe possède un circuit qui passe par chaque arc exactement une fois? (solution : connexité)
- ▶ **2-colorabilité** : est-ce qu'on peut colorier un graphe avec 2 couleurs t.q. pour chaque arête uv , les 2 sommets u, v ont des couleurs différentes? (solution : DFS)
- ▶ **L'appartenance d'un mot à un langage hors-contexte** ($\mathcal{O}(n^3)$).

136/159

Les classes **P** et **NP**

NP = $\{\mathcal{L}(M) \mid \exists p \text{ polynôme tel que } f_M(n) \leq p(n)\}$.

P = $\{\mathcal{L}(M) \mid \exists p \text{ polynôme tel que } f_M(n) \leq p(n) \text{ et } M \text{ déterministe}\}$.

- ▶ **P** \subseteq **NP**.
- ▶ Déterminer si l'inclusion ci-dessus est stricte est un problème ouvert.
- ▶ **NP** peut être défini par l'**existence** de **preuve** qui peut être vérifié en temps polynomial :
- ▶ **Exemple** : la preuve pour la satisfaisabilité d'une formule booléenne est une **valuation des variables** qui rend la formule vraie.

137/159

Classe **NP**

Un langage L est dans **NP** si et seulement s'il existe un polynôme p et un langage $K \in \mathbf{P}$ tels que :

$$x \in L \iff \text{il existe } y \in \Sigma^{\leq p(|x|)} : \langle x, y \rangle \in K$$

Le mot y est une **preuve** (de taille polynomiale) pour l'appartenance de x à L . Le langage K est un **vérifieur polynomial** des preuves.

- ▶ Vocabulaire : on montre souvent l'appartenance de L dans **NP** en (1) **devinant** une preuve y pour l'entrée x et (2) **vérifiant** que y est bien une preuve.
- ▶ On vérifie facilement que les problèmes **3-COLORATION, CLIQUE, CYCLE-HAMILTONIEN, ...** sont dans la classe **NP**.

138/159

Réductions polynomiales

- ▶ Soient A et B deux langages de Σ^* .
- ▶ Une **réduction polynomiale** de A vers B est une fonction $f : \Sigma^* \rightarrow \Sigma^*$ telle que
 - ▶ f est **calculable par une MT déterministe en temps polynomial**.
 - ▶ On a l'équivalence suivante :

$$x \in A \iff f(x) \in B.$$

- ▶ On note $A \leq_P B$ (A **se réduit** à B de façon polynomiale).
- ▶ L'existence d'une réduction polynomiale de A vers B assure que
 - ▶ si $B \in \mathbf{P}$, alors $A \in \mathbf{P}$.
 - ▶ si $B \in \mathbf{NP}$, alors $A \in \mathbf{NP}$.

On dit aussi que chacune des classes **P** et **NP** est **fermée** par réductions polynomiales.

139/159

Réductions polynomiales

Plusieurs réductions vues précédemment sont bien polynomiales :

$$\text{SAT} \leq 3\text{-SAT} \leq 3\text{-COLORATION} \leq \text{SAT} \\ 3\text{-SAT} \leq \text{CLIQUE}$$

140/159

Complexité en temps : résumé

- ▶ On s'intéresse à des machines de Turing s'arrêtant sur toute entrée.
- ▶ Les machines non-déterministes ont plusieurs calculs sur un mot (\rightarrow **arbre de calculs**).
- ▶ fonction de complexité f_M d'une machine M associe à tout entier n le temps du plus long calcul de M sur une entrée de taille n .
- ▶ **P** = langages décidés en temps polynomial par une MT **déterministe**.
- ▶ **NP** = langages décidés en temps polynomial par une MT.
- ▶ On ne sait pas si **P** = **NP**.
- ▶ Réduction polynomiale : réduction qui se calcule en temps polynomial.
Exemple : de **SAT** vers **3-COLORATION**.

141/159

Problèmes de décision / calcul de solutions / optimisation

- ▶ Les problèmes vus jusqu'à présent (SAT, 3-coloration, etc) sont des problèmes de **décision** : réponse OUI/NON.
- ▶ Problèmes de **calcul de solution** : si une formule est satisfaisable, alors calculer une valuation satisfaisante. Si un graphe est 3-coloriable, alors calculer un coloriage à 3 couleurs...
- ▶ Problèmes d'**optimisation** : ayant un graphe pondéré sur les arcs, calculer un cycle hamiltonien de poids minimal (TSP).

143/159

Problèmes **NP**-complets

- ▶ Un langage L (ou problème) est **NP-complet** si
 1. $L \in \mathbf{NP}$, et
 2. pour tout langage $K \in \mathbf{NP}$, on a $K \leq_P L$. (On dit aussi que L est **NP-difficile**.)
- ▶ On va montrer qu'il existe des problèmes **NP-complets**.
- ▶ Ne pas confondre les termes **NP** et **NP-complet**. Tout problème $L \in \mathbf{P}$ est aussi dans **NP**. MAIS, si vous réussissez à montrer l'appartenance d'un problème **NP-complet** à **P**, vous pouvez remporter un prix de 1 Mio.\$
- ▶ Un problème **NP-difficile** peut être indécidable (**NP-difficile** signifie seulement une borne inf sur la complexité.)

142/159

Problèmes de décision / calcul de solutions / optimisation

Considérons le problème SAT et supposons qu'il ait un **algorithme polynomial A** pour le résoudre. Alors on peut construire l'algorithme suivant, qui calcule pour une formule donnée φ une valuation satisfaisante σ (s'il en existe une) :

- ▶ La formule donnée : $\varphi(x_1, \dots, x_n)$.
- ▶ Si $A(\varphi)$ retourne "non", alors retourner "non-satisfaisable".
- ▶ Pour $i = 1$ jusqu'à n faire :
 - ▶ Si $A(\varphi(b_1, \dots, b_{i-1}, \text{true}, x_{i+1}, \dots, x_n))$ retourne "oui", alors $b_i := \text{true}$, sinon $b_i = \text{false}$.
- ▶ Retourner (b_1, \dots, b_n) .

144/159

Réductions de Turing

On remarque que l'algorithme précédent est en fait une **réduction** du problème de calcul de solution au problème de décision. Ce type de réduction s'appelle **Turing (polynomiale)**. Comme les réductions de type "many-one", celles de type Turing préservent la décidabilité, et **P** et **NP** sont fermées par les réductions de Turing polynomiales :

- ▶ $A \leq_T B$ signifie que A se réduit à B par une réduction de type Turing (cad, il existe un algorithme pour A , qui utilise un algorithme pour B de façon "boîte noire").
On écrit $A \leq_T^p B$ si A se réduit à B par une réduction Turing polynomiale (cad, l'algorithme pour A - sans compter le temps d'exécution des appels de B - est polynomial).
- ▶ Si $A \leq_T B$ et B est décidable, alors A est décidable.
- ▶ Si $A \leq_T^p B$ et $B \in \mathbf{P}$, alors $A \in \mathbf{P}$.
- ▶ Si $A \leq_T^p B$ et $B \in \mathbf{NP}$, alors $A \in \mathbf{NP}$.

145/159

Théorème de Cook-Levin — principe de la preuve

- ▶ On part d'un langage quelconque de **NP**, décidé par une MT M à une bande (bornée à gauche).
- ▶ On construit à partir de M et d'un mot d'entrée w une formule $\varphi_{M,w}$ de taille polynomiale, et telle que

$$M \text{ accepte } w \iff \varphi_{M,w} \text{ est satisfaisable.}$$

147/159

Théorème de Cook-Levin

- ▶ **Théorème (Cook, Levin)** SAT est **NP**-complet.
- ▶ **Conséquence.** D'après les réductions polynomiales vues précédemment, les problèmes 3-SAT, 3-COLORATION, et CLIQUE sont **NP**-complets.

146/159

Théorème de Cook-Levin — idée de preuve

- ▶ Tout calcul de M sur w prend $p(n)$ étapes (instants), où p est un polynôme et $n = |w|$.
- ▶ Donc tout calcul de M sur w utilise au plus $p(n) + 1$ cases.
- ▶ On introduit des variables, avec leur signification intuitive
 - ▶ $Q(i, q)$: **vrai** ssi l'état de contrôle de M à l'instant i est q .
 - ▶ $P(i, j)$: **vrai** ssi la position de la tête de M à l'instant i est j .
 - ▶ $L(i, j, A)$: **vrai** ssi le symbole à l'instant i dans la case j est A .
- ▶ **Note.** Seulement un nombre **polynomial** de variables !

148/159

Théorème de Cook-Levin — idée de preuve

- ▶ Grâce aux variables Q, P, L , on encode le calcul : À tout instant
 - ▶ on se trouve dans un et un seul état,
 - ▶ une et une seule case est lue,
 - ▶ il y a une et une seule lettre dans chaque case.
- ▶ On code également que
 - ▶ à l'instant 0, la configuration est $q_0 w B^p(|w|+1-|w|)$.
 - ▶ à l'instant $p(n)$, la machine est dans l'état OK.
 - ▶ pour tout i : le passage entre les valeurs $Q(i, \cdot), P(i, \cdot), L(i, \cdot, \cdot)$ et les valeurs $Q(i+1, \cdot), P(i+1, \cdot), L(i+1, \cdot, \cdot)$ correspond bien à une transition de M .

149/159

Partition

Le problème **Partition** est le suivant :

- ▶ **Donnée** : des entiers $x_1, \dots, x_k > 0$.
- ▶ **Question** : existe-t-il $X \subseteq \{1, \dots, k\}$ tel que

$$\sum_{i \in X} x_i = \sum_{i \notin X} x_i.$$

C'est clairement dans **NP** : on devine X et on teste.

151/159

Un autre problème **NP**-complet : Somme d'entiers

Le problème **Somme d'entiers** est le suivant :

- ▶ **Donnée** : des entiers $x_1, \dots, x_k > 0$ et un entier s .
- ▶ **Question** : existe-t-il $1 \leq i_1 < i_2 < \dots < i_p \leq k$ tels que

$$x_{i_1} + \dots + x_{i_p} = s.$$

C'est clairement dans **NP** : on devine i_1, \dots, i_p et on teste.

On montre que c'est **NP**-complet par une réduction $3\text{-SAT} \leq \text{Somme d'entiers}$.

150/159

Réduction Somme d'entiers vers Partition

Soit x_1, \dots, x_k, s une instance de **Somme d'entiers**. Soit $x = \sum x_i$. On construit (en temps polynomial) l'instance $x_1, \dots, x_k, x - 2s$ de **Partition**.

- ▶ Si **Somme d'entiers** a une solution sur x_1, \dots, x_k, s , **Partition** a une solution sur $x_1, \dots, x_k, x - 2s$.
- ▶ Inversement, si **Partition** a une solution sur $x_1, \dots, x_k, x - 2s$, **Somme d'entiers** a une solution sur x_1, \dots, x_k, s .

152/159

Réduction 3-SAT vers Somme d'entiers

- ▶ On construit à partir d'une **formule 3-CNF** $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ avec variables x_1, \dots, x_n une instance de **Somme d'entiers** (x_1, \dots, x_k, s) t.q. φ est satisfaisable ssi (x_1, \dots, x_k, s) a une solution.
- ▶ Idée : on code les littéraux et les clauses de φ par de (très) grands entiers en base 10. Le codage est défini de telle façon que quand on fait des sommes, il n'y a pas de retenue (c-à-d, les "bits" sont décodables).
- ▶ A chaque variable x_i correspondent les 2 entiers y_i, z_i . Les entiers y_i, z_i sont de longueur $m + i$ et commencent chacun par 10^{i-1} . Les **m** derniers "bits" codent les clauses : pour y_i le j dernier "bit" est 1 si x_i apparaît dans C_j , et 0 sinon ; pour z_i le j dernier "bit" est 1 si $\overline{x_i}$ apparaît dans C_j , et 0 sinon.

153/159

Complexité en espace

- ▶ Soit M une machine de Turing (s'arrêtant sur toute entrée).
- ▶ Soit $\text{space}_M(w)$ le **nombre maximal** de cases visitées sur les bandes, sur l'entrée w jusqu'à l'arrêt de M .
- ▶ Soit $s_M(n) = \max \{ \text{space}_M(w) \mid |w| = n \}$.
- ▶ $s_M(n)$ mesure la mémoire utilisée par M sur une entrée de taille n , **dans le pire des cas**.
- ▶ On note par **DSPACE**(s) la classe des langages acceptés par une MT déterministe M t.q. $s_M \leq s$. On note par **NSPACE**(s) la classe des langages acceptés par une MT M t.q. $s_M \leq s$.

155/159

Réduction 3-SAT vers Somme d'entiers

- ▶ Exemple : Pour $\varphi = (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$ on a $y_1 = \mathbf{110}$, $z_1 = \mathbf{101}$, $y_2 = \mathbf{1001}$, $z_2 = \mathbf{1000}$, etc.
- ▶ A chaque clause C_j correspondent les 2 entiers $t_j = w_j = 10^{m-j}$.
- ▶ L'entier $s = \underbrace{1 \dots 1}_n \underbrace{3 \dots 3}_m$.
- ▶ Pour produire le bloc 1^n dans s il faut choisir exactement un de y_i, z_i (pour tout $1 \leq i \leq n$). Ceci revient à choisir une valuation des variables – y_i signifie x_i vrai, et z_i signifie x_i faux.
- ▶ Pour produire le bloc 3^m dans s il faut que pour chaque clause, au moins un des littéraux soit vrai. On complète jusqu'à 3 en utilisant les entiers t_j, w_j .
- ▶ L'instance de "Somme d'entiers" peut se calculer en temps polynomial.

154/159

La classe PSPACE

- ▶ On définit **PSPACE** = $\bigcup_{k \geq 1} \mathbf{DSPACE}(n^k) = \bigcup_{k \geq 1} \mathbf{NSPACE}(n^k)$.
- ▶ $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} = \bigcup_{k \geq 1} \mathbf{DTIME}(n^k)$
- ▶ Exemples de problèmes dans **PSPACE** :
 - Entrée** : n automates finis \mathcal{A}_i .
 - Question** : est-ce que $\bigcap_{i=1}^n L(\mathcal{A}_i) \neq \emptyset$?
 - Entrée** : expressions rationnelles R_1, R_2 .
 - Question** : est-ce que R_1, R_2 décrivent le même langage ?
 - Entrée** : formule booléenne quantifiée φ .
 - Question** : est-ce que φ est valide ?
 - Entrée** : graphe orienté G , sommet de départ s : jeu à 2 joueurs J_1, J_2 qui jouent de façon alternée en formant un chemin simple dans G (à partir de s) - le premier qui ne peut plus jouer, perd.
 - Question** : est-ce que J_1 a une stratégie gagnante ?

156/159

PSPACE et preuves interactives

- Rappel : **NP** représente la classe des problèmes pour lesquels on peut **vérifier la preuve** de manière efficace.
- 2 acteurs : **prouveur P** et **vérifieur V** . Le but de P est de convaincre V que l'entrée w appartient à un langage donné L .
- Exemple : $L = \text{SAT}$. Sur une entrée φ , le prouveur P présente à V une valuation σ des variables de φ . Le vérifieur V est convaincu, si σ est satisfaisante.
- Question : est-ce qu'on peut faire de même pour le complément de SAT? Réponse : oui, si on augmente les capacités de P et V .
 1. P et V pourront communiquer de façon bi-directionnelle.
 2. V pourra faire des choix probabilistes ; il devra être convaincu par P avec grande probabilité seulement.

157/159

Preuves interactives

- Un **système de preuves interactives** est une paire (P, V) t.q. :
 1. V est un algorithme polynomial **probabiliste**,
 2. P et V échangent un nombre polynomial de messages (de taille polynomiale).
 3. Un langage L est décidé par une paire (P, V) si pour toute entrée w :
 - $w \in L$ implique que V accepte (après communication avec P) avec proba $\geq 2/3$,
 - $w \notin L$ implique que V accepte (après communication avec P) avec proba $< 1/3$.
- Rq : on ne restreint pas P .

158/159

PSPACE et preuves interactives

- Exemple : Non-isomorphie de graphes. On veut vérifier que 2 graphes non-orientés G_1, G_2 ne sont pas isomorphes.
 - V choisit de façon aléatoire (et cachée) G_i ($i \in \{1, 2\}$) et applique une permutation sur G_i , en obtenant le graphe H . Il envoie H à P , en lui demandant l'indice i .
 - P envoie $j \in \{1, 2\}$.
 - V accepte ssi $i = j$.
- **PSPACE** est la classe des langages qui peuvent être décidés par des preuves interactives.
- Rq : **NP** ainsi que co-**NP** (la classe des compléments de langages dans **NP**) sont inclus dans **PSPACE**.

159/159