Calculabilité & Complexité

• Qu'est-ce qu'un algorithme ?
• Que peut-on calculer ? Que ne peut-on pas calculer ? (Que signifie calculer ?).
• Indépendamment du temps nécessaire (mais fini tout de même).
• Nécessité d'un (ou plusieurs) modèle de calcul.
• Classifier les problèmes algorithmiques en fonction de la facilité à les calculer (ou à les vérifier).

Histoire ancienne

• Machine de Pascal (~ 1645). Première machine à calculer. Additions/soustractions. Retenues.
• Machine analytique de Charles Babbage. 1834. Prototype. Unité de Contrôle, unité de calcul, mémoire, E/S. Ada Lovelace première programmeuse.
• Gödel. Tout système formel suffisamment puissant est soit incomplet, soit contradictoire.
• Turing & Chuch. Le problème de l'arrêt est indécidable. 1936. Thèse de Church/Turing : les MT ou le λ-calcul sont des modèles universels.
• Fonction calculable : définition indépendante du langage et de l'ordinateur.

1. Dénombrabilité

• Ensemble : boîte abstraite dans laquelle on met ce qu'on veut.
• Applications, injections, surjections, bijections.
• Ensembles finis. Ensembles infinis.
— Exemples : $\{1, \dots, 6\}$, $\{a, \dots, z\}$, mots sur 64 bits, élèves, $\emptyset$;
— Exemples : $\mathbb{N}$, $\mathbb{R}$, nombres pairs, suites binaires.
• Deux ensembles finis peuvent être mis en bijection ssi ils ont même nombre d'éléments.
• Infini dénombrable : en bijection avec $\mathbb{N}$.
• On verra qu'il existe des ensembles infinis non dénombrables.
• Dénombrable : fini ou infini dénombrable.
• Les sous-ensembles de $\mathbb{N}$ sont des ensembles dénombrables (d'après la proposition suivante).

1.1. — Proposition. *Soit D un ensemble. Les conditions suivantes sont équivalentes :*
(a) L'ensemble D est dénombrable.
(b) Il existe une injection de D dans $\mathbb{N}$.
(c) Il existe une surjection de $\mathbb{N}$ sur D.

Démonstration — • (a) $\Rightarrow$ (c) : c'est clair. (Et aussi (a) $\Rightarrow$ (b) bien sûr.)
• (c) $\Rightarrow$ (b) : tout élément de D admet un ou plusieurs numéros. On sélectionne le plus petit d'entre eux. L'application qui à $d \in D$ associe ce plus petit numéro est une injection de D dans $\mathbb{N}$.
• (b) $\Rightarrow$ (a) : ordonner les éléments de D par leurs images et boucher les trous. On obtient une liste finie ou une bijection avec $\mathbb{N}$.

1.2. — Exercice. *Une réunion de deux ensembles dénombrables est dénombrable.*

1.3. — Proposition. *Le produit $\mathbb{N} \times \mathbb{N}$ est dénombrable.*

Démonstration — Numéroter les points (i, j) à partir du coin $(0, 0) \mapsto 0$ par lignes obliques ($i+j$ constant) en faisant croître $i + j$. A l'intérieur des lignes, prendre j croissant. Concrètement :

$$(i, j) \longmapsto \frac{(i + j)(i + j + 1)}{2} + j.$$

□

1.4. — Proposition. *Une réunion dénombrable d'ensembles dénombrables est dénombrable.*

Démonstration — Parce qu'on peut l'injecter dans $\mathbb{N} \times \mathbb{N}$. □

1.5. — Exemples d'ensembles dénombrables. • $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$.
• $\mathbb{N} \times \mathbb{N}$.
• A^* où A alphabet (ensemble fini). L'ensemble des programmes en langage C.
• L'ensemble des graphes.

1.6. — Rappel. Ensemble des parties de E : $\mathcal{P}(E)$.

1.7. — Proposition. *L'ensemble des parties de $\mathbb{N}$ n'est pas dénombrable.*

Démonstration — Procédé diagonal. Si l'ensemble $\mathcal{P}(\mathbb{N})$ était (infini) dénombrable, on pourrait numéroter les parties $P_i \subseteq \mathbb{N}$, avec $i \in \mathbb{N}$. On pose

$$X = \{n \in \mathbb{N} \mid n \notin P_n\}.$$

On voit que X est distinct de chaque P_n (car n appartient à P_n ou à X mais pas aux deux). Contradiction. □

• Autres ensembles non dénombrables : les suites infinies d'entiers ou de bits. L'ensemble des nombres réels. L'ensemble des fonctions de $\mathbb{N}$ dans $\{0, 1\}$.

1.8. — Proposition. *Il existe des fonctions non calculables.*

Démonstration — Parce que le nombre de programmes est dénombrable. □

Langages

1.9. — Définition. Un *alphabet* est un ensemble fini (de symboles).

1.10. — Exemples. Alphabet $\{a, \ldots, z\}$. Alphabet binaire $\{0, 1\}$.

1.11. — Notations. • A^n est l'ensemble des mots sur A de longueur n.
• A^* est l'ensemble des mots (finis) sur A.

1.12. — Définition. Un langage sur A est un sous-ensemble de A^*.

1.13. — Remarques. Soit A alphabet non vide. L'ensemble A^* est infini dénombrable. L'ensemble des langages sur A est non dénombrable.

1.14. — Exemples. $(ab)^*$, $\{a^n b^n \mid n \in \mathbb{N}\}$. Expressions bien parenthésées sur $A = \{(,)\}$.

2. Calculabilité

Machines de Turing

Introduites par Alan Turing en 1936.

2.1. — Description. Une *machine de Turing* est la donnée de :
• Un alphabet A. Aux symboles de A, on adjoint un caractère spécial « blanc » noté □. Je note A' l'alphabet $A \cup \{\square\}$.

• Une bande infinie (dans les deux sens) formée de cellules juxtaposées. Cette bande symbolise la mémoire (extensible) d'un ordinateur. Chaque cellule contient exactement un caractère (éventuellement blanc). Seules un nombre fini de cellules contiennent un caractère non blanc.
• Une tête de lecture. Elle est positionnée sur une cellule et peut se déplacer sur commande à une cellule adjacente (vers la droite ou vers la gauche).
• Un ensemble fini Q d'états, l'un d'entre eux étant appelé l'état *initial*.

• Un programme ou *fonction de transition*. Ce programme est composé d'un tableau d'instructions (ou transitions), indexé par Q et A'. Pour chaque couple $(q, s) \in Q \times A'$, le programme possède au plus une instruction qui sera exécutée quand la machine sera dans l'état q et que la tête lira le symbole s. Cette instruction est codée sous la forme (q, s, q', s', d) avec $q' \in Q$, $s' \in A'$ et $d \in \{L, N, R\}$. L'exécution de cette instruction consiste à écrire le symbole s' (à la place du symbole s), à déplacer la tête d'une case (L : à gauche ; R : à droite ; N : ne pas déplacer) et à placer la machine dans l'état q'.

Calcul par une machine de Turing

• Les données sont inscrites sur la bande (sous la forme de mots sur A séparés par des □).
• La machine est initialement dans l'état initial.
• Convention usuelle : la machine démarre avec la tête de lecture sur le symbole non blanc le plus à gauche.
• La machine s'arrête lorsqu'elle ne possède pas de transition associée au symbole et à l'état courants.
• Le *résultat* de la machine est alors constitué des mots présents sur la bande (séparés par des □).

2.2. — Exemple (incrémenteur). Cette machine additionne 1 à la donnée présentée en entrée, écrite en binaire.

$$A = \{0, 1\}, \qquad Q = \{q_0, q_1, q_2\}, \quad \text{l'état initial étant } q_0.$$

Sa fonction de transition est donnée par le tableau suivant :

	0	1	□
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, □, L)$
q_1	$(q_2, 1, L)$	$(q_1, 0, L)$	$(q_2, 1, L)$

2.3. — Thèse (Church/Turing). *Toute fonction physiquement calculable l'est par une machine de Turing. La machine de Turing est un modèle d'ordinateur universel.*

Machines de Turing à plusieurs bandes

• Une tête de lecture par bande.
• Transitions : $(q, a_1, b_1, D_1, a_2, b_2, D_2, \ldots, q')$ où a_i symbole lu, b_i symbole écrit, D_i déplacement sur la bande i.

2.4. — Théorème. *Toute machine de Turing M à plusieurs bandes peut être simulée par une machine M' à une seule bande. De plus, si M effectue t transitions dans son calcul, alors M' peut être conçue pour effectuer au plus ct^2 transitions pour le même calcul (c constante).*

Langages acceptés

• Un ou plusieurs états peuvent être désignés comme états *finaux*. Ils peuvent se subdiviser en états d'acceptation et en états de rejet.
• Dans ce cas on dit que la machine *accepte* le mot (unique) écrit sur la bande si et seulement si elle s'arrête dans un état d'acceptation. Le mot est dit *rejeté* si la machine s'arrête dans un état de rejet ou bien ne s'arrête pas.
• Le *langage accepté* ou *reconnu* par une machine de Turing est formé des mots acceptés (lol).

2.5. — Exemple (testeur de poids pair). Cette machine s'arrête dans un état différent selon que le nombre binaire entré possède un nombre pair (état q_y) ou impair (état q_n) de 1.

$$A = \{0, 1\}, \qquad Q = \{q_0, q_1, q_y, q_n\}, \quad \text{l'état initial étant } q_0.$$

Sa fonction de transition est donnée par le tableau suivant :

	0	1	□
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_y, □, R)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_n, □, R)$

Machines à registres

• Elles sont constituées d'un nombre fini de registres.
• Chaque registre contient un entier naturel (non borné).
• Graphe avec un sommet « début » (un arc sortant), un (ou plusieurs) sommet « fin » (sans arc sortant), et un ou plusieurs sommets pour chaque registre.
• Registre de type R^+ : Un ou plusieurs arcs entrants et un arc sortant. Le registre est incrémenté à chaque passage.
• Registre de type R^- : Un ou plusieurs arcs entrants et deux arcs sortants dont l'un d'entre eux est labellé 0. Si contenu non nul, on décrémente et on sort par la sortie non labellée. Si contenu nul, on sort par la sortie 0.
• Exemple : Addition de deux entiers.
• Une machine à registres peut être simulée par une machine de Turing à plusieurs bandes (alphabet unaire, une bande par registre, un état par sommet de l'automate).
• Pour simuler la machine à registres d'addition : $(a, b) \mapsto (a + b, 0)$ on a la machine de Turing unaire :

$$(q_a, 1, 1, L, x, x, N, q_a), \quad (q_a, \square, 1, N, x, x, N, q_b), \quad (q_b, x, x, N, 1, \square, R, q_a), \quad (q_b, x, x, N, \square, \square, N, q_f)$$

où x représente $\square$ ou 1 et où q_b est l'état initial.

Langages décidables

• RE : récursivement énumérable, semi-décidable, acceptable, reconnaissable. Il existe une machine de Turing qui reconnaît le langage (s'arrête dans l'état q_y si est seulement si $w \in L$).
• R : récursif ou décidable. Il existe une machine de Turing qui s'arrête dans l'état q_y si $w \in L$, dans q_n sinon.
• Indécidable : contraire de décidable...
• Le complémentaire d'un langage est un langage.
• Le complémentaire d'un langage décidable est décidable. (Pas vrai pour RE.)
• Un langage L est décidable si et seulement si L et $\overline{L}$ sont tous deux RE.
• On énumère les mots $(w_n)_{n\in\mathbb{N}}$ sur un alphabet, et les machines $(M_n)_{n\in\mathbb{N}}$. On définit le langage

$$L_{\text{diag}} = \{w_n \mid M_n \text{ accepte } w_n\}.$$

2.6. — Proposition. *(a) Le langage L_{diag} est reconnaissable, mais pas décidable.*
(b) Le langage $\overline{L_{\text{diag}}}$ n'est pas reconnaissable.

Démonstration — (b) Si une des machines reconnaît $\overline{L_{\text{diag}}}$, il s'agit de l'une des M_{n_0}. Cela signifie que M_{n_0} accepte w_{n_0} ssi $w_{n_0} \notin L_{\text{diag}}$. Pourtant, par définition de L_{diag}, on a au contraire M_{n_0} accepte w_{n_0} ssi $w_{n_0} \in L_{\text{diag}}$.
(a) Comme $\overline{L_{\text{diag}}}$ n'est pas reconnaissable, L_{diag} n'est pas décidable. Tout de même, la machine qui sélectionne M_n et l'exécute sur w_n (pour w_n en entrée) reconnait L_{diag}. □

Fonctions calculables

• Fonction (partielle) de E dans E : pas nécessairement définie sur E tout entier.
• Si f n'est pas définie en x on dit que f diverge en x et on note $f(x) \uparrow$.
• Si f est définie en x on dit que f converge en x et on note $f(x) \downarrow$ ou, plus précisément, $f(x) = y$.
• Exemple : $n \mapsto n - 1$, pour $E = \mathbb{N}$ (représenté en alphabet binaire).
• Deux fonctions partielles f et g sont égales si et seulement si elles divergent sur les mêmes entrées et $f(x) = g(x)$ pour tous les x pour lesquelles elles convergent. On note $f = g$.
• A une machine de Turing M (ou à registres) on peut associer la fonction partielle f calculée par M. Cette fonction diverge lorsque M ne s'arrête pas. On notera cette fonction f_M.
• Une fonction f est *calculable* dans un modèle (Turing, registres, ...) si il existe une machine qui calcule f (càd $f = f_M$).

Machine de Turing universelle

• Une machine de Turing peut être décrite à l'aide d'un nombre fini de symboles sur un alphabet convenable. Pour M machine de Turing, je note $\langle M \rangle$ le mot correspondant.

• On peut imposer l'alphabet $\{0, 1\}$ pour écrire $\langle M \rangle$ (encodage ascii par exemple). A tout mot qui ne respecte pas la syntaxe $\langle M \rangle$, on associe la machine triviale qui boucle toujours (par exemple).

• La machine universelle U prend en entrée la description $\langle M \rangle$ d'une machine de Turing et une donnée (séquence de mots) d et simule l'exécution de M sur d (calcule le même résultat ; accepte les mêmes mots) :

$$U(\langle M \rangle, d) = M(d).$$

Problème de l'arrêt

• Problème de l'arrêt : existe-t-il une machine S telle que $S(\langle M \rangle, w)$ décide si M s'arrête sur w. On fixe l'alphabet $\{0, 1\}$. Si S existait alors il existerait une machine D telle que

Si $S(w, w)$ répond oui alors D entre dans une boucle infinie; sinon D accepte w.

L'appel $S(\langle D \rangle, \langle D \rangle)$ révèle une contradiction :

— S accepte $(\langle D \rangle, \langle D \rangle)$ si et seulement si D s'arrête sur $\langle D \rangle$, par définition de S.

— S accepte $(\langle D \rangle, \langle D \rangle)$ si et seulement si D n'accepte pas $\langle D \rangle$, par définition de D. □

3. Complexité

Comparaisons asymptotiques (Notations Bachmann/Landau)

Ici, f et g fonctions positives.

• $f(n) \in O(g(n))$ lorsque $f(n) \leqslant \alpha g(n)$ pour une constante α et n assez grand.

• $f(n) \in o(g(n))$ lorsque, pour tout $\epsilon > 0$, $f(n) \leqslant \epsilon g(n)$ et n assez grand.

• $f(n) \sim g(n))$ lorsque $f(n)/g(n)$ tend vers 1 (on suppose $g(n) \neq 0$).

• $f(n) \in \Omega(g(n))$ lorsque $\alpha g(n) \leqslant f(n)$ pour une constante $\alpha > 0$ et n assez grand.

• $f(n) \in \Theta(g(n))$ lorsque $\alpha g(n) \leqslant f(n) \leqslant \beta g(n)$ pour deux constantes $\alpha > 0, \beta$ et n assez grand (à la fois O et Ω)

Coûts polynomiaux, exponentiels, logarithmiques : $\Theta(n^k)$, $\Theta(\mathsf{exp}(n^k))$, $\Theta(\mathsf{ln}(n))$.

La classe P

• Problèmes polynomiaux : classe des problèmes que l'on peut résoudre en temps polynomial (sur une Machine de Turing). Taille de la donnée (longueur sur la bande). Temps = nombre de transitions. Polynomial aussi sur un ordinateur moderne. $c \leqslant |x|^n$ pour un $n \in \mathbb{N}$.

• Exemples. Addition (linéaire). Multiplication (quadratique). Division euclidienne (quadratique). Exponentiation (cubique). Tri : en $n\,\mathsf{ln}(n)$.

• Contre-exemple. Trouver un diviseur non trivial d'un entier composé ne semble pas être polynomial (mais ce n'est pas prouvé).

• Problèmes de décision. Les entrées positives forment un langage.

• Classe P : problèmes de décision (langages) décidables en temps polynomial.

3.1. — Définition. Un langage L sur un alphabet A est de classe P si il existe une machine de Turing M et un polynôme q tels que :

— La machine M **décide** le langage L (pour $x \in L$, on a $M(x) \to q_y$; pour $x \notin L$ on a $M(x) \to q_n$).

— Le temps de calcul $M(x)$ majoré par $q(|x|)$.

• Exemples. Comparer deux entiers ($a - b \geqslant 0$). Divisibilité $a \mid b$. Vérifier si deux entiers sont premiers entre eux $\mathsf{pgcd}(a, b) = 1$). Déterminer si n est premier (c'est non trivial).

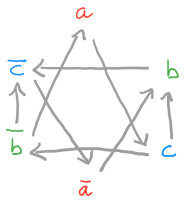
Problème SAT

• Variables booléennes (en nombre fini).
• Clauses $a_i \vee a_j \vee a_k \vee \cdots$ avec ou sans barres (complément). n-clause si n termes. Clause satisfaite on non par des valeurs booléennes.
• Liste finie de clauses à satisfaire simultanément.
Lorsque toutes les clauses ont n termes, on parle de problème n-SAT.

3.2. — **Exemple.** Instance de problème 2-SAT :

$$a \vee b, \qquad \overline{a} \vee c, \qquad \overline{c} \vee b, \qquad \overline{b} \vee \overline{c}.$$

• La clause $a \vee b$ est équivalente à deux implications : $\overline{a} \Rightarrow b$ et $\overline{b} \Rightarrow a$. On peut faire un graphe avec sommets $a, b, c, \overline{a}, \overline{b}, \overline{c}$.



• Si on a une boucle $x \Rightarrow \overline{x}$ et $\overline{x} \Rightarrow x$ alors on ne peut satisfaire l'instance. Dans l'exemple, on a

$$a \Rightarrow \overline{a}, \qquad \overline{b} \Rightarrow b, \qquad c \Rightarrow \overline{c}$$

mais on a pas les implications contraires.
• Solution de cette instance : $(a, b, c) = (0, 1, 0)$.

3.3. — **Proposition.** *La condition ci-dessus est nécessaire et suffisante.* □

3.4. — **Corollaire.** *Le problème 2-SAT est de classe P.*

Démonstration — Parce que la recherche des chemins $x \mapsto \overline{x}$ (ou le contraire) revient à calculer des puissances de matrices d'adjacence (jusqu'à l'exposant $2k$). □
• Pas vrai pour 3-SAT (sauf collapse).

La classe NP

3.5. — **Définition.** Un langage L sur un alphabet A est de classe NP si il existe une machine de Turing M et des polynômes p, q tels que :
— Pour tout $x \in L$, il existe un témoin $w \in A^*$, de taille $\leqslant p(|x|)$, tel que $M(x, w) \to q_y$.
— Pour tout $x \notin L$ et pour tout $w \in A^*$ de taille $\leqslant p(|x|)$, la machine $M(x, w) \to q_n$.
— Le temps de calcul $M(x, w)$ est $\leqslant q(|x|)$.

3.6. — **Exemple.** Le langage SAT (le langage des formules logiques satisfaisables) est de classe NP.

3.7. — **Exemples.** Le langage des nombres composés est de classe NP. Le langage *subset sum* est de classe NP.

• On a l'inclusion P $\subseteq$ NP.
• C'est pas symétrique. Un problème de décision D est co-NP si $\overline{D}$ est NP.
• On a l'inclusion P $\subseteq$ NP $\cap$ co-NP.

Machines de Turing non déterministes

• Il peut y avoir plusieurs (deux) transitions qui commencent par le même couple d'entrée (q, a) :

$$(q, a, q'_0, a'_0, D_0) \qquad \text{et} \qquad (q, a, q'_1, a'_1, D_1).$$

• On dit que la machine accepte la donnée x si l'une (au moins) des séquences de choix mène à un état d'acceptation.

• Exemple. La machine de Turing

$$(q_0, \square, q_1, 1, D), \qquad (q_1, \square, q_2, 1, D), \qquad (q_2, \square, q_2, 1, D), \qquad (q_2, \square, q_f, \square, N).$$

écrit un nombre aléatoire $\geqslant 2$ en unaire sur la bande vide. On peut l'utiliser pour construire une machine qui écrit deux nombres aléatoires $\geqslant 2$, les multiplie, compare le résultat à une donnée initiale, et accepte si égalité. Cette machine accepte le langage des nombres composés.

• On dit que la machine ND reconnaît un langage en temps $t(n)$ si, pour chaque mot x de L de taille n, il existe une séquence de choix qui accepte x en temps majoré par $t(n)$.

3.8. — Définition. *Un langage L est de classe NTIME$\big(t(n)\big)$ si il existe une machine de Turing ND et de coût $\leqslant t(n)$ qui reconnait L.*

3.9. — Fait. *Pour N machine de Turing non déterministe, et w séquence de choix, il existe une machine de Turing M (déterministe) telle que $M(x, w)$ effectue le même calcul que $N(x)$ dirigée par la séquence w.*

Définition de NP avec machines non déterministes

3.10. — Théorème. *NP $= \cup_{c\in\mathbb{N}}$NTIME(n^c).*

Démonstration — • Si L est dans NTIME(n^c) alors la séquence des choix est un témoin (polynomial) pour chaque $x \in L$.

• Si L est NP, on a une machine M qui répond en temps polynomial $q(n)$ avec des témoins de taille polynomiale $p(n)$. On construit une machine non déterministe. Elle génère en parallèle tous les $w \in A^{p(n)}$:

$$(q_i, \square, q_{i+1}, a, D) \qquad \text{pour tout } a \text{ dans l'alphabet.}$$

Elle appelle ensuite $M(x, w)$. □

3.11. — Théorème. *Si L est une langage reconnu par une machine N de Turing ND en temps $t(n)$, alors il existe une machine de Turing (déterministe) M qui reconnait L, en temps $c^{t(n)}$ pour une constante $c > 1$.*

Démonstration — On peut définir une machine M à 3 bandes. La première pour contenir la donnée initiale (en lecture seule) ; la deuxième qui encode une séquence dans l'arbre des choix (il faudra énumérer les séquences) ; et la troisième qui permet de travailler sur la donnée (et qui sera initialisée à chaque itération pour la séquence de choix suivante). □

SAT est NP-complet

3.12. — Définition (réduction polynomiale). Soient L_1 et L_2 deux langages sur un même alphabet A. On dit que L_1 est polynomialement réductible à L_2 (noté $L_1 \leqslant_p L_2$) si il existe une fonction f de A^* dans lui-même, calculable en temps polynomial telle que

$$x \in L_1 \qquad \text{si et seulement si} \qquad f(x) \in L_2.$$

• Concrètement : toute méthode permettant de reconnaître L_2 permet de reconnaître L_1, avec un surcoût seulement polynomial.

3.13. — Théorème. *Pour tout langage L dans NP, on a $L \leqslant_p$ SAT.*

Démonstration — Soit M une machine de Turing ND qui reconnaît L (en temps polynomial, etc). On indexe les lettres de l'alphabet par i (avec $0 \leqslant i \leqslant I$), les étapes du calcul par t (avec $0 \leqslant t \leqslant T$), les cellules de la bande par s (avec $-T \leqslant s \leqslant T$) et les états de M par k (avec $0 \leqslant k \leqslant K$).

On introduit les $(2T+1)(T+1)(I+1)$ littéraux

$$P_{s,t,i} : \text{vrai lorsque la cellule } s \text{ contient le symbole } a_i \text{ à l'étape } t$$

et les $(T+1)(K+1)$ littéraux

$$Q_{t,k} : \text{vrai si } M \text{ est dans l'état } q_k \text{ à l'étape } t$$

ainsi que les $(2T+1)(T+1)$ littéraux

$$R_{s,t} : \text{vrai si la tête est en position } s \text{ à l'étape } t.$$

On introduit les $(T+1)2T(2T+1)/2$ clauses

$$\bigvee_{-T\leqslant s\leqslant T} R_{s,t} \quad \text{et} \quad \overline{R_{s,t}} \vee \overline{R_{s',t}} \qquad \text{pour } s \neq s'$$

car à chaque étape, la tête est placée sur une cellule et une seule. Aussi les $(2T+1)(T+1)I(I+1)/2$ clauses

$$\bigvee_{0\leqslant i\leqslant I} P_{s,t,i} \quad \text{et} \quad \overline{P_{s,t,i}} \vee \overline{P_{s,t,i'}} \qquad \text{pour } i \neq i'$$

car à chaque étape et dans chaque cellule, il y a un unique symbole. Puis les $(T+1)K(K+1)/2$ clauses

$$\bigvee_{0\leqslant k\leqslant K} Q_{t,k} \quad \text{et} \quad \overline{Q_{t,k}} \vee \overline{Q_{t,k'}} \qquad \text{pour } k \neq k'$$

car à chaque étape, la machine M est dans un unique état. Aussi les conditions initiales

$$Q_{0,0}, \qquad R_{0,0}, \qquad P_{s,0,i_s} \qquad \text{pour } -T \leqslant s \leqslant T$$

qui disent que M est dans l'état q_0 et sur la cellule 0 à l'étape 0, que la bande est initialisée avec la donnée $a_{i_{-T}} \cdots a_{i_T}$. Enfin

$$\bigvee_{0\leqslant t\leqslant T} Q_{t,K}$$

qui dit que la machine finit par atteindre l'état d'acceptation q_K. Il faut aussi traduire chaque transition $(q_k, a_i, q_{k'}, a_{i'}, D)$ par les $T+1$ implications

$$Q_{t,k} \wedge R_{s,t} \wedge P_{s,t,i} \Rightarrow Q_{t+1,k'} \wedge P_{s,t+1,i'} \wedge R_{s\pm1,t+1}$$

converties chacune en 3 clauses en utilisant les formules $a \wedge a' \Rightarrow b \wedge b'$ par $\overline{a} \vee \overline{a'} \vee b$ et $\overline{a} \vee \overline{a'} \vee b'$. (Puisque la machine est non déterministe, les implications ont en fait des $\vee$ de triplets en partie droite, ce qui complique un peu plus les choses…)

Cette construction a un coût polynomial et réduit le langage L à une instance de SAT. □

• Les problèmes de décision polynomialement équivalents à SAT sont dits NP-complets.

• Problème P / NP : les classes de complexité P et NP sont-elles distinctes ou pas ? Cette question a été énoncée en 1971 par S. Cook et reste toujours sans réponse.