Leçon 915 - Classes de complexité : Exemples

[Carton]

CoNP = Pas de sujet

I - Modèles de calcul

Premier modèle : Une machine de Turing à un ruban.
On note uqv la configuration de la machine étant dans l'état q, ayant le mot uv sur son ruban et dont la tête de lecture pointe sur la première lettre de v.
La taille de la configuration uqv est $|uv|$.

Second modèle : Une machine de Turing à deux ruban. Le premier ruban sert à écrire le mot d'entrée. La machine ne peut écrire dessus. Le deuxième ruban est initialement vide.
On note $\begin{pmatrix} u_1 q v_1 \\ u_2 q v_2 \end{pmatrix}$ une configuration de cette machine.
La taille d'une configuration est $|u_2 v_2|$. (On ne compte pas la taille de l'entrée).

Déf 1 : Complexité d'un calcul.
Soit une machine de Turing M et soit un calcul $\gamma = C_0 \to C_1 \to \dots \to C_m$.
- Le temps de calcul $t_M(\gamma)$ est m.
- L'espace de calcul $s_M(\gamma)$ est $\max_{0 \le i \le m} |C_i|$.

Déf 2 : Complexité d'une entrée
Soit une machine de Turing M et soit mot w.
Notons Γ l'ensemble des calculs dont l'entrée est w
- Le temps de calcul de w est $\max_{\gamma \in \Gamma} t_M(\gamma) = t_M(w)$
- L'espace de calcul de w est $\max_{\gamma \in \Gamma} s_M(\gamma) = s_M(w)$

Déf 3 : Complexité
Soit une machine de Turing M.
- La complexité en temps de M et la fonction $t_M(n) = \max_{|w| \le n} t_M(w)$.
- De même, $s_M(n) = \max_{|w| \le n} s_M(w)$.

Prop 4. Soit M une machine à une bande de complexité $t_M(n)$ en temps et $s_M(n)$ en espace.
Il existe une machine M' à deux bandes en temps $t_M(n)$ et en espace $s_M(n)$ équivalente à M.

Prop 5: Soit M une machine à deux bandes, de complexité $t_M(n)$ en temps et $s_M(n)$, en espace.
Il existe une machine M' à une bande, en temps $O(t_M(n)(n + t_M(n)))$ et en espace $n + s_M(n)$, équivalente à M.

Prop 6: Pour toute machine M, il existe une constante K telle que : $s_M(n) \leqslant t_M(n)$ et $t_M(n) \leqslant 2^{K s_M(n)}$

II - Complexité en temps

Déf 7 : Soit A un problème décidé par une machine M déterministe
- Si il existe un polynôme P tel que pour tout $n \in \mathbb{N}$, $t_M(n) \leqslant P(n)$ Alors A appartient à la classe $\mathcal{P}$
- Si il existe un polynôme P tel que pour tout $n \in \mathbb{N}$, $t_M(n) \leqslant 2^{P(n)}$ Alors A appartient à la classe EXPTIME.

Ex 8 : Le problème PATH
Entrée : Un graphe G et deux sommets s et d
Sortie : Accepte ssi il y a un chemin dans G de s à d.
PATH $\in \mathcal{P}$

Ex 9 : Le problème k-HALT
Entrée : Une machine de Turing M et un entier k codé en binaire
Sortie : Accepte ssi M s'arrête toujours en au plus k étapes
k-HALT $\in$ EXPTIME

⚠ Complexité en espace → IR ne faut pas compter la taille de l'entrée.

Déf 10 : De la même manière que l'on a défini les classes P et EXPTIME pour des machines déterministe, on définit les classes NP et NEXPTIME pour des machines non déterministe

Ex 11 : Le problème SAT-CNF

Entrée : Une formule propositionnelle en CNF
Sortie : Accepte ssi la formule est satisfiable
$SAT^{CNF} \in NP$

Ex 12 : Le problème k-COL

Entrée : Un graphe G et un entier binaire k.
Sortie : Accepte ssi il existe $c : S \to \{1, \dots, k\}$ tq pour tout $(u,v) \in A$, $c(u) \neq c(v)$.
$k\text{-COL} \in NP$

Prop 13 : Pour toute classe déterministe DA et la classe non-déterministe associée NA, on a $DA \subseteq NA$.

Déf 14 : Pour toute classe A, on définit co-A comme la classe des langages L tels que $L^c \in A$

Ex 15 : Le problème VAL

Entrée : Une formule propositionnelle quelconque
Sortie : Accepte ssi la formule est valide
$VAL \in co\text{-}NP$

Prop 16 : Si DA est une classe déterministe, $DA = co\text{-}DA$.

Prop 17 : Inclusion des classes.

$P \subseteq^{NP}_{co\text{-}NP} \subsetneq EXPTIME \subseteq NEXPTIME$

III - Complexité en espace 2e modèle

Déf 18 : Soit A un problème décidé par une machine M déterministe

- Si il existe un polynôme P tel que pour tout $n \in \mathbb{N}$, $S_M(n) \leq P(n)$
  Alors A appartient à PSPACE
- Si il existe une constante K telle que pour tout $n \in \mathbb{N}$, $S_M(n) \leq K \log n$
  Alors A appartient à L

Ex 19 : Problème QBF

Entrée : Une formule booléenne quantifiée close
Sortie : Accepte si la formule est valide
$QBF \in PSPACE$

Ex 20 : $\{0^k 1^k \mid k \in \mathbb{N}\} \in L$

Déf 21 : On définit les classes non-déterministe NPSPACE et NL de la même façon que les classes PSPACE et L, mais en utilisant des machine non-déterministe.

Ex 22 : $PATH \in NL$

Théorème 23 : Savitch (DEV)

Soit M une machine non-déterministe à deux rubans telle que $S_M(n) \geq \log_2 n$.
Il existe M' une machine déterministe, équivalente à M, en espace $O(S_M(n)^2)$.

Corollaire 24 : $PSPACE = NPSPACE$

Prop 25 : $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$.

↳ Complexité logarithmique ? → ça a la place de le faire (autre modèle)

IV Réductions

Déf 26 : Soient A et B des problèmes codés par L_A et L_B sur les alphabet Σ_A et Σ_B.
Une réduction polynomiale de A à B est une fonction $f : \Sigma_A^* \to \Sigma_B^*$ calculable en temps polynomiale par une machine de Turing déterministe, telle que : $f(w) \in L_B \iff w \in L_A$.
On note alors $A \leqslant_p B$.

Remarque 27 : Il existe aussi des réductions logarithmique, mais leur définition nécessite d'introduire des machines particulière à trois rubans.

Déf 28 : Un problème A est dit NP-complet ssi :
- $A \in NP$
- $\forall B \in NP,\ B \leqslant_p A$ (NP-difficulté)

Théorème 29 : Cook (DEV)
SAT-CNF est NP-complet

Prop 30 : Si A est NP-difficile et $A \leqslant_p B$.
Alors B est NP-difficile

Corollaire 31 : k-COL est NP-complet.

Remarque 32 : La propriété 30 permet de montrer la NP-complétude de beaucoup d'autres problèmes, comme par exemple :
- La couverture par sommet — SAT pour toute formule prop
- L'existence de k-clique. — 3-SAT
- etc.

Déf 33 : Un problème A est PSPACE-complet ssi :
- $A \in PSPACE$
- $\forall B \in PSPACE,\ B \leqslant_p A$

Ex 34 : QBF est PSPACE-complet

comme ensemble d'états. Les transitions de l'automate $\mathcal{A}_\varphi$ sont données par

$$q_{i,j} \cdot a = \begin{cases} 0 & \text{si } x_j \text{ apparaît dans } c_i \\ q_{i,j+1} & \text{sinon} \end{cases} \qquad \text{pour } 1 \leq i \leq k \text{ et } 1 \leq j \leq n$$

$$q_{i,j} \cdot b = \begin{cases} 0 & \text{si } \neg x_j \text{ apparaît dans } c_i \\ q_{i,j+1} & \text{sinon} \end{cases} \qquad \text{pour } 1 \leq i \leq k \text{ et } 1 \leq j \leq n$$

$$q_{i,n+1} \cdot a = q_{i,n+1} \cdot b = 0 \qquad \text{pour } 1 \leq i \leq k$$

$$0 \cdot a = 0 \cdot b = 0.$$

L'automate ainsi construit est synchronisant et son délai de synchronisation est inférieur à n si et seulement si la formule φ est satisfiable.

6. Le problème de savoir si le délai de synchronisation est au plus un entier donné se réduit facilement au problème de savoir si le délai de synchronisation est exactement un entier donné. Ce dernier problème est donc tout autant NP-difficile. Savoir si le délai de synchronisation de l'automate $\mathcal{A}_\varphi$ est $n+1$ permet de décider si la formule φ n'est pas satisfiable. Ce problème est donc aussi co-NP-difficile.

4.3 Complexité en espace

On rappelle que la complexité en espace d'un calcul est le nombre de cellules de la bande qui sont utilisées, c'est-à-dire la taille maximale d'une configuration apparaissant dans le calcul. La fonction de complexité en espace d'une machine $\mathcal{M}$ est la fonction $s_\mathcal{M}$ qui donne pour chaque entier n la complexité en espace maximale d'un calcul sur une entrée de taille n (cf. définition 4.1 p. 193).

4.3.1 Changement de modèle

Il a été vu que les changements de modèles peuvent avoir une incidence plus ou moins grande sur la complexité en temps. Le passage de plusieurs bandes à une seule bande donne une explosion quadratique (cf. proposition 4.4) alors que le passage d'une machine non déterministe à une machine déterministe impose une explosion exponentielle (cf. proposition 4.5). Au contraire, les changements de modèle ont moins d'incidence sur la complexité en espace.

Le passage d'une machine à bande bi-infinie à bande infinie ne change pas la complexité en espace puisque l'espace utilisé reste identique. En effet, la simulation est effectuée en repliant la bande sur elle-même. L'espace utilisé n'est pas augmenté.

Le passage d'une machine à k bandes à une machine à une seule bande peut être réalisé en mettant bout à bout les contenus des k bandes sur l'unique bande séparés par un caractère spécial. Le nombre de cases de bande utilisées est égal à la somme des cases utilisées sur les k bandes. L'espace reste encore inchangé.

Le passage d'une machine non déterministe à une machine déterministe change la complexité en espace mais de façon plus modérée que la complexité en temps. Le théorème suivant énonce qu'il est possible de simuler une machine non déterministe avec une machine déterministe en utilisant seulement le carré de l'espace.

Théorème 4.30 (Savitch 1970). *Soit s une fonction de $\mathbb{N}$ dans $\mathbb{R}^+$ telle que $s(n) \geq n$ pour tout n assez grand. Toute machine non déterministe qui fonctionne en espace $s(n)$ est équivalente à une machine de Turing déterministe en espace $O(s^2(n))$.*

Preuve. Soit $\mathcal{M}$ une machine de Turing fonctionnant en espace $s(n)$. Sans perte de généralité, on peut supposer que cette machine possède un seul état final q_f. On modifie la machine pour qu'avant d'accepter, elle efface la bande en remplaçant chaque symbole par le symbole blanc #. L'objectif de cette transformation est d'avoir une seule configuration acceptante égale à q_f.

On commence par définir une fonction ACCESS qui prend en paramètre deux configurations C et C' de $\mathcal{M}$ ainsi que deux entiers t et r. La fonction retourne vrai s'il existe un calcul de C à C' de longueur au plus t qui n'utilise que des configurations intermédiaires de taille au plus r. La fonction est récursive et fonctionne de la façon suivante. Si $t = 0$ ou $t = 1$, elle se contente de tester si C et C' sont égales ou s'il existe une étape de calcul de de C à C'. Sinon, elle parcourt toutes les configurations C'' de taille au plus r et pour chaque configuration C'', elle teste s'il existe un calcul dont C'' est la configuration médiane.

```
1: ACCESS(C, C', t, r)
2: if t = 0 then
3:   return C = C'
4: else if t = 1 then
5:   return C = C' or C → C'
6: else
7:   for all C'' de taille r do
8:     if ACCESS(C, C'', ⌈t/2⌉, r) and ACCESS(C'', C', ⌊t/2⌋, r) then
9:       return true
10:  return false
```

Algorithme 11: Fonction ACCESS

Nous analysons maintenant la complexité en espace de la fonction ACCESS. La valeur de t est divisée par 2 à chaque appel récursif. Le nombre maximal d'appels imbriqués de la fonction est donc borné par $\log_2 t$. Chaque appel récursif utilise trois variables C, C' et C'' de taille au plus r et un entier t qui nécessite un espace au plus $\log_2 t$ s'il est codé en binaire. La variable r reste constante et on peut donc considérer qu'il n'y a en fait qu'une seule instance de cette variable qui occupe un espace $\log_2 r$. L'espace utilisé globalement est donc borné par $O(\log_2 r + (r + \log_2 t) \log_2 t)$.

Nous allons maintenant utiliser la fonction ACCESS pour résoudre le problème. Dans un premier temps, nous supposons pour simplifier que la fonction $s(n)$ est calculable en utilisant un espace au plus $s(n)$. Puisque la machine $\mathcal{M}$ fonctionne en espace $s(n)$, il existe, d'après le lemme 4.2 une constante K telle $t_\mathcal{M}(n) \leq 2^{Ks(n)}$. Soit w une entrée de taille n. Puisque q_f est l'unique configuration acceptante de $\mathcal{M}$, on a l'équivalence suivante.

$$w \in L(\mathcal{M}) \iff \text{ACCESS}(q_0 w, q_f, 2^{Ks(n)}, s(n))$$

Il est alors facile de donner une machine déterministe $\mathcal{M}'$ équivalente à $\mathcal{M}$. Pour chaque entrée w de taille n, la machine $\mathcal{M}'$ calcule la valeur de $s(n)$ puis utilise la fonction

ACCESS avec $t = 2^{Ks(n)}$ et $r = s(n)$. L'espace utilisé est alors borné par $O(K^2 s^2(n)) = O(s^2(n))$.

Nous ne supposons plus que la fonction $s(n)$ est calculable et nous allons utiliser à nouveau la fonction ACCESS pour contourner cette difficulté. Soit w une entrée de taille n et soit m la taille maximale d'une configuration accessible à partir de la configuration initiale $q_0 w$. Nous donnons ci-dessous un algorithme qui calcule la valeur de m en utilisant un espace borné par $O(m^2)$. Comme $m \leq s(n)$, cet espace est donc borné par $O(s^2(n))$.

Pour chaque entier $k \geq n+1$, on note N_k, le nombre de configurations de taille au plus k accessibles par un calcul à partir de $q_0 w$ en utilisant des configurations de taille au plus k. Par définition, la suite $(N_k)_{k \geq 0}$ est croissante et elle est bornée par le nombre de configurations de taille au plus $s(n)$. Il existe donc un entier k tel que $N_{k+1} = N_k$. Réciproquement, si k est le plus petit entier tel que $N_{k+1} = N_k$, alors k est égal à m et N_k est le nombre total de configurations accessibles à partir de $q_0 w$. Supposons par l'absurde qu'il existe une configuration accessible à partir de $q_0 w$ avec un calcul utilisant des configurations de taille au moins $k+1$. On considère la première configuration de ce calcul de taille $k+1$. Cette configuration existe car la taille des configurations ne varie que d'une unité au plus à chaque étape de calcul. Comme cette configuration est de taille $k+1$ et que celles qui la précèdent dans le calcul sont de taille au plus k, elle prouve que $N_{k+1} > N_k$, d'où la contradiction. L'algorithme suivant calcule la valeur de m.

Input w de taille n
1: $k \leftarrow n$
2: $i \leftarrow 0$
3: **repeat**
4: $\quad k \leftarrow k+1$
5: $\quad N \leftarrow i$
6: $\quad i \leftarrow 0$
7: $\quad$ **for all** C de taille au plus k **do**
8: $\quad\quad$ **if** ACCESS$(q_0 w, C, 2^{Kk}, k)$ **then**
9: $\quad\quad\quad i \leftarrow i+1$
10: **until** $i = N$
11: return k

Algorithme 12: Calcul de m

Comme k est borné par m, l'espace utilisé par chaque appel à ACCESS est borné par $O(m^2)$. L'espace nécessaire aux variables i et N est borné par m si ces entiers sont codés en binaire. L'espace global utilisé par l'algorithme précédent est donc au plus $O(m^2)$.

Une machine déterministe $\mathcal{M}'$ équivalente à $\mathcal{M}$ fonctionne de la façon suivante. Elle calcule d'abord la valeur de m avec l'algorithme précédent puis utilise ensuite la fonction ACCESS avec $t = 2^{Km}$ et $r = m$ pour décider s'il y a un calcul de $q_0 w$ à la configuration acceptante q_f. □

La preuve du théorème de Savitch appelle quelques commentaires. Il faut d'abord remarquer une certaine similitude entre la fonction ACCESS et l'algorithme de McNaughton et Yamada pour calculer une expression rationnelle équivalente à un automate fini. L'algorithme pour calculer la valeur de m peut aussi être rapproché des algorithmes donnés dans la preuve du théorème de Immerman et Szelepcsényi avec toutefois la différence

que ces derniers sont non déterministes.

Dans l'analyse de la complexité en espace de la fonction ACCESS, il faut comprendre que c'est le nombre maximal d'appels imbriqués qui importe et non pas le nombre total d'appels qui interviendrait dans la complexité en temps. Par appel imbriqué d'une fonction récursive, on entend un appel actif, c'est-à-dire qui a commencé à s'exécuter mais ne s'est pas terminé car il a provoqué un autre appel. Le nombre maximal d'appels imbriqués correspond à la hauteur maximale de la pile utilisée pour stocker les variables locales et les adresses de retour.

4.3.2 Classes de complexité en espace

Comme pour le temps, on introduit des classes de problèmes déterminées par l'espace nécessaire à leur décision.

Définition 4.31. Pour une fonction $f : \mathbb{N} \to \mathbb{R}^+$, on définit les classes
- SPACE$(f(n))$ est l'ensemble des problèmes décidés par une machine de Turing *déterministe* (à plusieurs bandes) en espace $O(f(n))$.
- NSPACE$(f(n))$ est l'ensemble des problèmes décidés par une machine de Turing *non déterministe* (à plusieurs bandes) en espace $O(f(n))$.

Le théorème de Savitch garantit qu'il n'est pas nécessaire de distinguer les machines déterministes et non déterministes pour l'espace polynomial ou exponentiel. On définit les classes suivantes.

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k) = \bigcup_{k \geq 0} \text{NSPACE}(n^k)$$
$$\text{EXPSPACE} = \bigcup_{k \geq 0} \text{SPACE}(2^{n^k}) = \bigcup_{k \geq 0} \text{NSPACE}(2^{n^k})$$

4.3.3 Complexités en temps et en espace

Nous étudions maintenant quelques relations entre les classes de complexité en temps et en espace. La proposition suivante résume les relations entre les différentes classes de complexité introduites jusque là.

Proposition 4.32. *Les classes de complexité introduites vérifient les inclusions suivantes.*

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE}.$$

Les inclusions entre ces classes et leurs classes duales sont représentées à la figure 4.11.

Preuve. Les deux inclusions NP $\subseteq$ PSPACE et NEXPTIME $\subseteq$ EXPSPACE découlent du lemme 4.2 et du théorème de Savitch.

L'inclusion PSPACE $\subseteq$ EXPTIME découle aussi du lemme 4.2. □

Le théorème de hiérarchie (corollaire 4.55 p. 233) permet de montrer que les inclusions P $\subsetneq$ EXPTIME et PSPACE $\subsetneq$ EXPSPACE sont strictes. Pour les inclusions intermédiaires, rien n'est connu même si la thèse généralement admise est qu'elles sont toutes strictes.

Théorème de Cook

Thm : SAT est NP-Complet

- **SAT est NP** : si on prend une valuation, le calcul de la valeur de vérité de la formule est polynômial

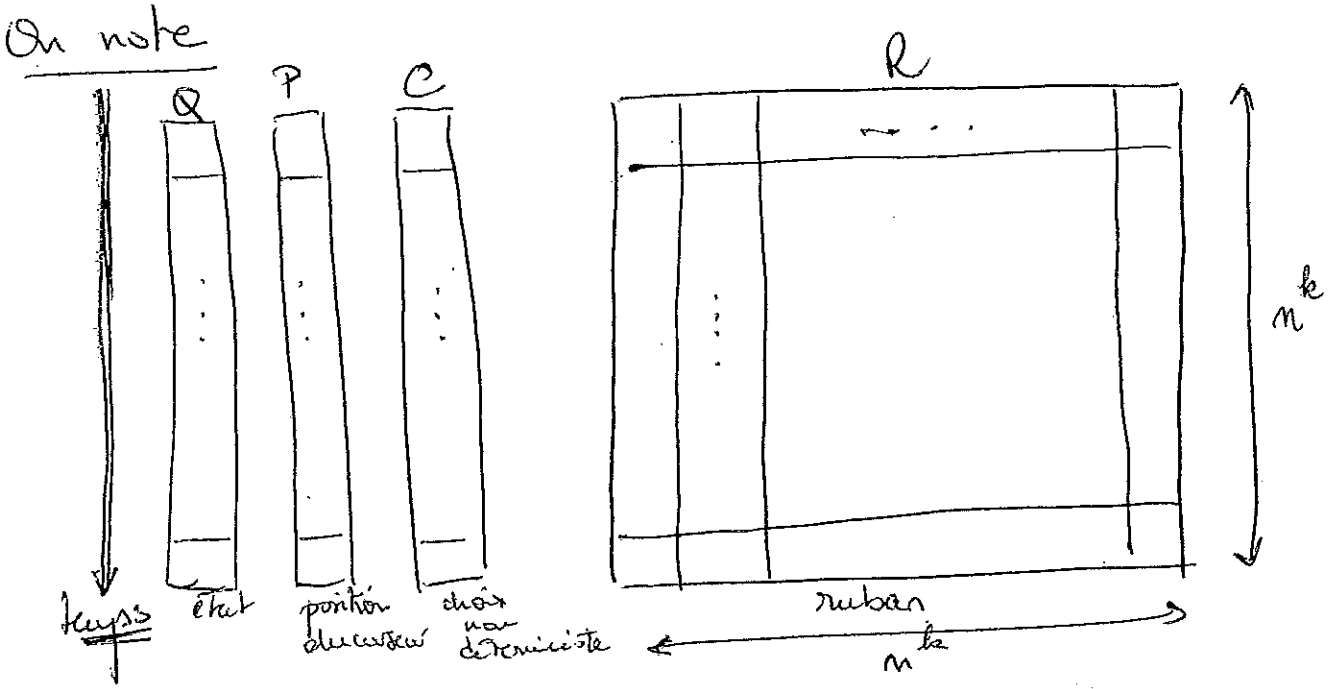
- **SAT est NP-dur** : Pour tout problème $A \in NP$, on construit une réduction de A dans SAT.

Soit $\mathcal{M}$ une MT non déterministe qui décide A en temps polynomial. Pour toute entrée w de $\mathcal{M}$, on va écrire une formule φ_w, polynomiale en $|w| = n$, telle que

φ_w satisfiable $\iff$ $\mathcal{M}$ accepte w.

OPS : • $\mathcal{M}$ décide A en temps n^k pour un entier k
• tout calcul acceptant sur w est de longueur exactement n^k.

On note



Pour décrire le remplissage de ces tableaux, on va utiliser les variables propositionnelles suivantes :

→ $r_{ij\alpha}$, $(i,j) \in [\![0, n^k]\!]^2$, $\alpha \in \Gamma$
qui est vraie quand la case (i,j) de R contient α (ruban)

→ q_{ik} , $i \in [\![0, n^k]\!]$, $k \in Q$
qui est vraie quand la case i de Q contient k
ie quand on est à l'état k au temps i

→ p_{ij} , $(i,j) \in [\![0, n^k]\!]$ qui indique la position j du curseur au temps i

→ c_{ik} , $i \in [\![0, n^k]\!]$, $k \in [\![1, Max]\!]$ où Max est le nb maximal de choix pour entre deux transitions
qui indique le choix de transition fait par l'exécution

Rq : on a un nombre polynômial de variables propositionnelles

---

il y a une valeur dans chaque case (i,j) de R

$$\varphi(\exists! R) = \bigvee_{a \in \Gamma} r_{ij\alpha}$$

il y a au ⊕ une valeur dans la case (i,j) de R

$$\varphi(\exists! R) = \bigwedge_{\substack{a \neq a' \\ \in \Gamma}} \overline{r_{ij\alpha}} \vee \overline{r_{ij\alpha'}}$$

de même pour chaque case de Q : $\varphi(\exists! Q) = \bigwedge_{i \in [\![0, n^k]\!]} \left(\bigvee_{k \in Q} q_{ik} \wedge \bigwedge_{\substack{k \neq k' \\ k,k' \in Q}} \overline{q_{ik}} \vee \overline{q_{ik'}} \right)$

[Toute valuation vérifiant $\varphi(\exists! R) \wedge \varphi(\exists! Q) \wedge \varphi(\exists! \tau) \wedge \varphi(\exists! c) = \varphi(\exists!)$ est un remplissage de nos tableaux

pour s'assurer que le remplissage décrit bien une exécution de $\mathcal{M}$ sur w.

- la première configuration est initiale.

$$\varphi(\text{ini}) = \underbrace{\left(\bigwedge_{0 \le j \le |w|-1} r_{0,j,w_{j+1}} \wedge \bigwedge_{n \le j \le n^k} r_{0,j,\text{Bloc}} \right)}_{\text{où on a écrit } w \text{ sur le ruban puis des blancs}} \wedge \underbrace{q_{0,s}}_{\text{état initial}} \wedge \underbrace{P_{0,0}}_{\text{position}}$$

- une configuration finale apparaît avant la n^{k+1}e étape

$$\varphi(\text{fin}) = \bigvee_{\substack{i \in [\![0, n^k]\!] \\ \kappa \in Q_F}} q_{i,\kappa}$$

- on écrit bien les transitions de $\mathcal{M}$.

→ toutes les positions hors tête de lecture sont conservées

$$\varphi(\text{conserve}^k) = \bigwedge_{(i,j) \in [\![0, n^k]\!]^2} \left[\left(r_{i,j,\alpha} \wedge \overline{P_{i,j}} \right) \Rightarrow r_{i+1,j,\alpha} \right]$$

$A \Rightarrow B$ ssi $\overline{A} \vee B$

$$= \bigwedge_{\substack{i,j,\alpha \\ \alpha \in \Gamma}} \overline{r_{i,j,\alpha}} \vee P_{i,j} \vee r_{i+1,j,\alpha}$$

→ les cases de R sous tête de lecture, l'état, la position de la tête de lecture sont modifiées

si [configuration i, état κ, tête de lecture j, symbole lu α, choix de transition t] $\xrightarrow{\text{détermine de façon unique}}$ Δ la transition effectuée, donc: [l'état suivant, κ', la lettre écrite α', le déplacement de la tête d (+1 ou -1)]

Code 3

ce qui donne

$$\varphi(\text{transition}) = \bigwedge_{\substack{(i,j)\in[\![0,n^k]\!]^2 \\ K\in Q \\ \alpha\in\Gamma \\ k\in[\![1,n_{ax}]\!]}} \left(\begin{array}{ll} (q_{iK} \wedge p_{ij} \wedge r_{ij\alpha} \wedge c_{ik}) \Longrightarrow q_{(i+1)K'} & \text{] nouvel état} \\ \wedge\, (\_\_\_\_\_\_) \Longrightarrow r_{(i+1)j\alpha'} & \text{] nouvelle lettre} \\ \wedge\, (\quad) \Longrightarrow p_{(i+1)(j+d)} & \text{] nouvelle position} \end{array} \right)$$

$$\overset{\text{CNF}}{\leadsto} \bigwedge_{(i,j,K,\alpha,k)} \left[(\overline{q_{iK}} \vee \overline{p_{ij}} \vee \overline{r_{ij\alpha}} \vee \overline{c_{ik}} \vee q_{(i+1)K'}) \wedge \dots \right]$$

---

chaque $\varphi(\,)$ est $\begin{cases} \text{en taille } O((n^k)^2) \rightarrow \text{polynomial} \\ \text{en CNF} \end{cases}$

$$\varphi = \left[\varphi(\exists!) \wedge \varphi(\text{ini}) \wedge \varphi(\text{fin}) \wedge \varphi(\text{conservation}) \wedge \varphi(\text{transition})\right]$$

on a bien $\mathcal{A}$ accepte w ssi φ satisfiable

et φ polynomiale en $|w|$.
