Problème de séparabilité par un automate

Julie Parreaux

2018-2019

Référence du développement : Le langage des machines [1, p.555]

Leçons où on présente le développement : 909 (Automates finis) ; 915 (Classes complexité) ; 928 (NP-complétude).

1 Introduction

Le problème de séparabilité par un automate intervient en théorie de l'apprentissage par exemple. Ce problème consiste à deviner un langage à partir d'informations partielles. On va alors chercher à construire un automate qui respecte les informations que nous avons. Lorsqu'on suppose que le langage que nous devons deviner est rationnel, nous pouvons travailler avec des automates finis. Le problème devient alors un problème de construction d'automate qui accepte une liste finie de mots et en rejette une autre. L'existence d'un tel automate connaissant l'ensemble des mots acceptant et rejetant en NP-complet. Nous allons prouver sa NP-dureté en le réduisant à SAT (qui est bien NP-complet par le théorème de Cook).

Outre la théorie de l'apprentissage, ce problème permet de classifier deux langages rationnel à l'aide d'un automate fini déterministe.

Remarques sur le développement

Le développement présente une preuve du NP-dureté. Il est donc important de bien écrire la réduction et de dire qu'elle est polynomiale.

— Problème NP (attention au codage des entiers).

— Problème NP-dur.

1. Présentation des problèmes et de la réduction (faire le dessin et l'expliquer).
2. Donner la traduction (en donner l'intuition sans la montrer).
3. Preuve de la correction de la traduction
   - $\Leftarrow$ Construction de la valuation en fonction de l'automate.
   - $\Rightarrow$ Construction de l'automate à l'inverse.

2 Séparabilité par un automate

Contexte du problème (théorie de l'apprentissage) : Un individu A essaye de deviner un langage L. Un individu B possède un ensemble fini de mots positifs S (qui appartiennent à L) et un ensemble fini de mots négatifs T (qui n'appartiennent pas à L). A doit trouver la règle la plus simple possible afin d'être compatible avec les exemples de B et en espérant qu'elle soit persistante.

On se place dans ce contexte et on suppose que L est rationnel. L'individu A doit alors trouver un automate déterministe fini qui accepte tous les mots de S et rejette ceux de T (pour les autres, on ne précise pas le comportement de l'automate).

Définition. On définit le problème SÉPARATION PAR UN AUTOMATE sur les langages rationnels.

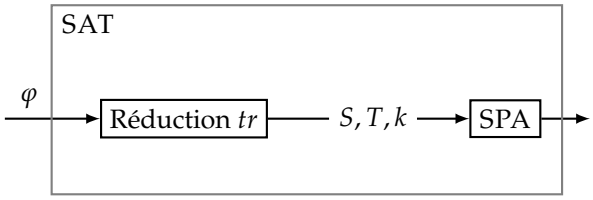

FIGURE 1 – Schéma du principe de réduction du problème SPA au problème SAT.

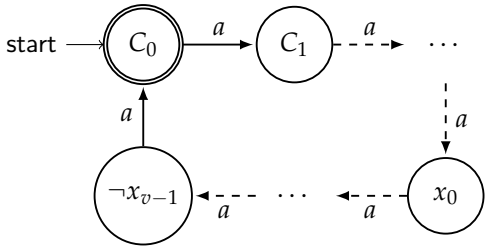

FIGURE 2 – Un automate $\mathcal{A}$ acceptant les mots de S_1 et rejetant ceux de T_1.

Problème SÉPARATION PAR UN AUTOMATE (SPA)
entrée : S et T deux ensembles finis de mots et $k \in \mathbb{N}$ (k est représenté en unaire)
sortie : Oui s'il existe $\mathcal{A}$déterministe à k états tel que $S \subseteq \mathcal{L}(\mathcal{A})$ et $T \cap \mathcal{L}(\mathcal{A}) = \emptyset$; non sinon

Théorème. *Le problème de* SÉPARATION PAR UN AUTOMATE *est NP-complet.*

Démonstration. Montrons que le problème de SÉPARATION PAR UN AUTOMATE (SPA) est NP-complet.

Le problème de SÉPARATION PAR UN AUTOMATE est dans NP On devine un automate déterministe à k états et on test l'appartenance de S à son langage et la non appartenance de T. Cette construction et ses tests se fond en temps polynomiale (chaque état à un nombre borné de transition et le temps d'exécution des mots de S et de T sont de leur taille, k est représenté en unaire sinon on a un facteur exponentielle).

Le problème de SÉPARATION PAR UN AUTOMATE est dans NP-dur On effectue une réduction au problème SAT (Figure 9 (à faire et à commenter)). (Ce problème est bien NP-complet par le théorème de Cook). Soit φ une formule booléenne à m clauses $C_0, \dots, C_{m-1}$ et v variables $x_0, \dots, x_{v-1}$. On note $x_0, \dots, x_{v-1}, \neg x_0, \dots, \neg x_{v-1}$ les variables propositionnelles de la formule φ et leur négation. On se base sur l'alphabet $\Sigma = \{a, b\}$.
Réduction : construire k, S et T de tel sorte que $\mathcal{A}$ soit contraint afin qu'il décrive une valuation pour φ.

— $k = m + 2v$ (le nombre de clauses et de termes possibles)
— $S = S_1 \cup S_5$
  - $S_1 = \{\epsilon, a^k\}$ (donne un unique état initial et un unique final dans un état à k états)
  - $S_5 = \{a^i bb \mid 0 \leq i < m\}$ (pour toutes les clauses, il existe un littéral lié via b à C_0)
— $T = T_1 \cup T_2 \cup T_3 \cup T_4$
  - $T_1 = \{a^i, 0 < i < k\}$ (donne un unique état initial et un unique final dans un état à k états)
  - $T_2 = \{a^i b a^j \mid 0 \leq i < m; 0 \leq j < k\} \setminus \{a^i b a^{2v-j} \mid l_j \text{ littéral de } C_i\}$ (les clauses sont liés à un de ces littéral via b)
  - $T_3 = \{a^{m+j} b a^h \mid 0 \leq j < 2v; 0 < h < k; h \neq v\}$ (les littéraux sont liés à C_0 ou à $\neg x_0$ via b)
  - $T_4 = \{a^{m+j} b a^{m+j+v} b \mid 0 \leq j < v\}$ (les variables et leur négation ne sont pas liés en même temps à C_0 via b)

Cette réduction se calcul en temps polynomial puisque k (évident : addition), S (on construit un ensemble contenant $n + 2$ mots donc linéaire en la taille de la formule) et T (on construit un ensemble contenant $(n + 2v)k$ mots donc polynomial en la taille de la formule) peuvent être construit en temps polynomial.

Proposition. *φ est satisfiable si et seulement s'il existe $\mathcal{A}$ déterministe à k états tel que $\mathcal{A}$ accepte S et rejette T.*

Démonstration. $\Leftarrow$ Supposons qu'il existe $\mathcal{A}$ déterministe à k états tel que $\mathcal{A}$ accepte S et rejette T.
$\rightarrow$ Comme $\mathcal{A}$ accepte S_1 et rejette T_1, il possède donc k états avec un unique état initial qui est aussi l'unique état final (il possède bien k états car tout états q_i, q_j sont distingués par a^{j-i}, pour tout $1 \leq i < j \leq k$). Pour simplifier la notation des états, on les étiquette comme suit : $C_0, \dots, C_{m-1}, x_0, \dots, x_{v-1}, \neg x_0, \dots, \neg x_{v-1}$ (Figure 2).

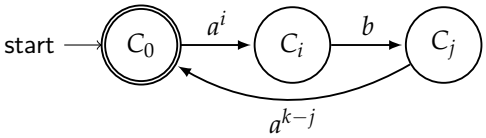

FIGURE 3 – Un morceau de l'automate $\mathcal{A}$ sous l'hypothèse de l'absurde pour T_2.

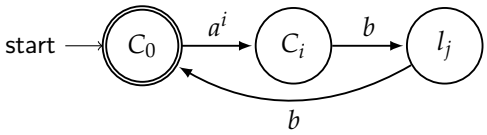

FIGURE 4 – Un morceau de l'automate $\mathcal{A}$ acceptant S_5.

→ Comme $\mathcal{A}$ rejette T_2, les clauses sont liés à un littéral. Raisonnons par l'absurde et supposons qu'il existe une transition issue d'un état C_i vers un état C_j (Figure 3). Soit $u = a^i b a^{k-j}$. Alors $u \in T_2 \subseteq T$ est accepté par $\mathcal{A}$, contradiction. Le mot $a^i b a^{2v-j}$ est accepté par $\mathcal{A}$ si et seulement si l_j est un littéral de C_i.

→ Comme $\mathcal{A}$ rejette T_3, les littéraux sont liés à C_0 ou à $\neg x_0$. Raisonnons par l'absurde et supposons qu'il existe une transition issue d'un état l_j vers un état q autre que C_0 ou $\neq x_0$ (Figure 5). On distingue alors deux cas selon la nature de l'état q.

— Si $q = C_i$ pour un certain $i \neq 0$, alors la lecture du mot $u = a^{m+j}b$ atteint l'état C_i car la lecture du mot a^{m+j}, par la nature cyclique de $\mathcal{A}$ et les m clauses, nous emmène en l_j. Donc le mot $a^{m+j}ba^{m+2v-i} \in T_3 \subseteq T$ est accepté par $\mathcal{A}$(car il y a $m + 2v = k$ état et C_i est à la position i). Absurde.

— Si $q = l_i$ pour un certain $i \neq v$, alors le mot $a^{m+j}ba^{2v-i} \in T_3 \subseteq T$ est accepté par $\mathcal{A}$ (car il y a $2v$ littéraux et l_i est à la position i). Absurde

On obtient une contradiction dans les deux cas, d'où le résultat.

→ Comme $\mathcal{A}$ rejette T_4, les variables et leurs négations ne sont pas liés à C_0 en même temps. Raisonnons par l'absurde et supposons qu'il existe une variable x_j qui soit liée à C_0 en même temps que sa négation (Figure 6). Supposons qu'il existe des transitions étiquetées entre l_j et C_0 et entre l_{j+v} et C_0. La lecture des mots $a^{m+j}b$ et $a^{m+v+j}b$ arrivent en C_0. La concaténation des deux mots $a^{m+j}ba^{m+v+j}b \in T_4 \subseteq T$ est alors acceptée. On obtient une contradiction.

→ Comme $\mathcal{A}$ accepte S_5 et rejette T, pour toute clause, il existe un littéral lié à C_0. Soit $u = a^i bb$ avec $0 \leq i < m$. Comme $uS_5 \subseteq S$, il est accepté par $\mathcal{A}$, la lecture du mot $a^i bb$ depuis C_0 nous emmène en C_0. Comme $i < m$, il existe un état C_i tel que la lecture du mot a^i depuis C_0 nous emmène en C_i, puis que la lecture du mot bb de C_i nous emmène en C_0. De l'état C_i, on va dans un état l_j qui est soit véridique soit $\neg x_0$. Comme après la lecture du dernier b on est en C_0 cet état l_j est véridique.

On construit une valuation ν. On pose

$$\nu(x_i) = \begin{cases} \text{vrai} & \text{si la transition } \delta(x_i, b) = C_0 \text{ est dans } \mathcal{A} \\ \text{faux} & \text{sinon} \end{cases}$$

On affecte la valeur vrai aux littéraux liés à C_0. Par la structure de $\mathcal{A}$, les variables et leur négation ne sont pas liés à C_0 en même temps donc leurs valuation ν est bien définie. De plus, dans $\mathcal{A}$ chacune des clauses contient un littéral lié à C_0. Toutes les clauses sont évaluées à vrai par ν. Donc il existe une valuation pour φ.

⇒ Supposons que φ est satisfiable par une valuation ν. On note $t(C_i)$ le premier littéral tel que $\nu(t(C_i)) = \text{vrai}$.

On construit $\mathcal{A}$.

— $Q = \{C_0, \ldots, C_{m-1}, x_0, \ldots, x_{v-1}, \neg x_0, \ldots, \neg x_{v-1}\}$

— $i = C_0$ et $F = \{C_0\}$

— $\delta : \begin{cases} (q_i, a, q_{(i+1) \mod k}) & \forall 0 \leq i < k \\ (C_i, b, t(C_i)) & \forall 0 \leq i < m \\ (l_i, b, C_0) & \forall 0 \leq j < 2v \text{ si } \nu(l_j) = \text{vrai} \\ (l_i, b, \neg x_0) & \forall 0 \leq j < 2v \text{ si } \nu(l_j) = \text{faux} \end{cases}$

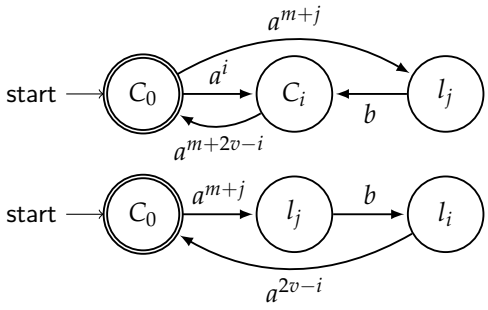

FIGURE 5 – Deux morceaux de l'automate $\mathcal{A}$ sous l'hypothèse de l'absurde pour T_3 (selon la distinction de cas).

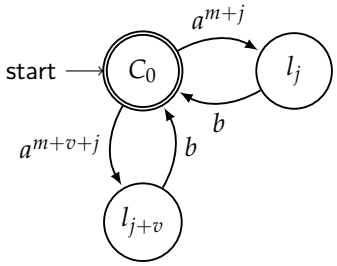

FIGURE 6 – Un morceau de l'automate $\mathcal{A}$ sous l'hypothèse de l'absurde pour T_4.

Montrons que $\mathcal{A}$ accepte S.

- → $\mathcal{A}$ accepte S_1 : structure de l'automate.
- → $\mathcal{A}$ accepte S_5 : soit $u = a^i bb$ pour un certain i. Après la lecture de a^i, on est dans la clause C_i. Puis par lecture de b, on atteint l'état l_j un littéral de valuation vrai de C_i. Donc par le lecture du second b, on atteint C_0 ($\nu(l_j) = \text{vrai}$).

Montrons que $\mathcal{A}$ rejette T.

- → $\mathcal{A}$ rejette T_1 : soit $u = a^i$ pour un certain $0 < i < k$. La lecture d'un tel mot nous amène dans l'état $q_i \neq C_0$ (car pour atteindre C_0, i doit être égal à 0 ou à k).
- → $\mathcal{A}$ rejette T_2 : soit $u = a^i b a^j$ pour i et j tels que l_j ne soit pas une clause de C_i. La lecture de a^i nous amène à l'état C_i. Puis la lecture de b nous donne l'état $l_h \neq l_j$ (l_j n'est pas un littéral de C_i) pour un certain littéral de la clause C_i. La lecture de a^j ne nous amène pas en C_0 car on n'est pas dans l'état l_j à j états de C_0.
- → $\mathcal{A}$ rejette T_3 : soit $u = a^{m+j} b a^h$ pour un certain j et un certain h. La lecture de a^{m+j} nous amène à l'état l_j. Si $\nu(l_j) = \text{vrai}$, la lecture de b donne l'état C_0. Comme $0 < h < 2v$, la lecture de a^h ne permet pas de revenir en C_0. Si $\nu(l_j) = \text{faux}$, la lecture de b donne l'état $\neg x_0$. Comme $h \neq v$, la lecture de a^h ne permet pas d'atteindre C_0.
- → $\mathcal{A}$ rejette T_4 : soit $u = a^{m+j} b a^{m+j+v} b$ pour un certain j. Supposons, sans perte de généralité (l'autre cas se traite de la même façon), que $\nu(x_j) = \text{vrai}$. La lecture de $a^{m+j}b$ nous amène en C_0. La lecture de a^{m+j+v} nous permet d'atteindre $\neg x_j$. Donc la lecture du b nous amène et $\neg x_0 \neq C_0$.

□

Donc, le problème SPA est NP-dur, ce qui implique sa NP-complétude. □

Remarque : On ne garantie pas que toutes les variables ou leur négation soient véridique si φ est satisfiable. Ce n'est pas génant pour montrer la NP-complétude car pour montrer que le formule est satisfiable un seul littéral par clause de valuation vrai suffit.

3 Théorème de Cook

Le théorème de Cook est un théorème historique car c'est le premier résultat de NP-complétude. Ce résultat n'applique donc pas une réduction polynomial à partir d'un problème NP-dur mais il explique quel est la réduction polynomiale à partir d'un problème NP quelconque.

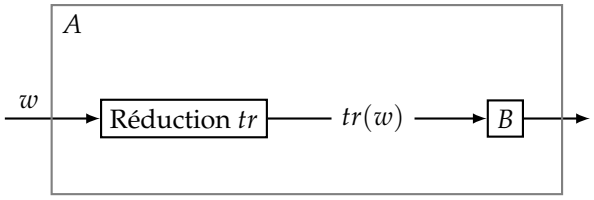

FIGURE 7 – Schéma du principe de réduction polynomiale du problème A au problème B.

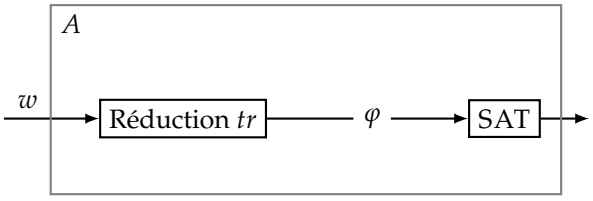

FIGURE 8 – Schéma du principe de réduction polynomiale dans le théorème de Cook d'un problème A dans NP au problème SAT.

Quelques rappels sur la NP-complétude [Papadimitriou ?]

Définition. La classe P est l'ensemble des problèmes de décision (vu comme un langage) décidé par une machine de Turing (ou un algorithme) déterministe en temps polynomial en la taille de l'entrée.

Remarque. Une définition alternative existe : elle fait apparaître la robustesse de cette classe. En effet, P est la réunion (sur $\mathbb{N}$) de tous les problèmes (langages) qui sont décidés par une machine de Turing déterministe en temps $O(n^k)$.

Définition. La classe NP est l'ensemble des problèmes de décision (vu comme un langage) décidé par une machine de Turing (ou un algorithme) non-déterministe en temps polynomial en la taille de l'entrée.

Définition. Un vérifieur pour un problème A est une machine de Turing $\mathcal{M}$ déterministe telle que w est une instance positive de A si et seulement s'il existe c tel que $\mathcal{M}$ accepte (w, c). Un tel c est appelé certificat.

Proposition (Définition alternative de NP [2, p.294])**.** *$A \in NP$ si et seulement si il existe un vérifieur en temps polynomial de A.*

Idée de la démonstration. L'idée est de montrer comment convertir un vérifieur en une machine de Turing non-déterministe et réciproquement.

Algorithm 1 Algorithme convertissant un vérifieur en une machine de Turing non-déterministe.

```
1: procedure M(w)
2:     Choisir c de longueur f(|w|)
3:     if V(w, c) accepte then
4:         Accepter
5:     else
6:         Rejeter
7:     end if
8: end procedure
```

Algorithm 2 Algorithme convertissant un vérifieur en une machine de Turing non-déterministe.

```
1: procedure V(w, c)
2:     Simuler l'exécution M(w) en prenant
       les choix conseillés par c.
3:     if l'exécution accepte then
4:         Accepter
5:     else
6:         Rejeter
7:     end if
8: end procedure
```

□

Remarque. On a $P \subseteq NP$ et si $NP - C \cap P \neq \emptyset$ alors $P = NP$ où $NP - C$ est l'ensemble des problèmes NP-complets.

Définition. Une réduction polynomiale (Figure 7) de A vers B est une fonction tr telle que

— tr est calculable en temps polynomial ;

— w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B.

On note alors $A \preccurlyeq B$.

Théorème. *Si le problème A se réduit au problème B, alors :*

— *si $B \in P$ alors $A \in P$;*

— *si $A \in NP$ alors $B \in NP$.*

Définition. Un problème A est dit NP-dur si pour tout problème $B \in NP$, il existe une réduction polynomiale de B à A. Si, de plus, $A \in NP$, alors A est dit NP-complet

Le théorème en lui-même Nous allons maintenant énoncé et donner l'idée de la preuve du théorème de Cook (qui fut le premier à exhiber un problème NP-complet).

Définition. On définit le problème SAT pour les formules de la logique propositionnelle.

Problème : SAT
entrée une formule ϕ de la logique propositionnelle
sortie Oui si ϕ est satisfiable ; non sinon

Théorème. *Le problème* SAT *est NP-complet.*

Idée de la démonstration. Montrons que e problème SAT est NP-complet.

SAT $\in$ NP Choisir une valuation pour chacune des variables de la formule (choix non-déterministe) et vérifier si c'est une valuation. On accepte lorsqu'on a trouvé une valuation et sinon on rejette.

SAT est NP-dur Comme c'est la première démonstration qu'un problème est NP-complet, il nous faut utiliser la définition. On prend donc un problème NP B et on va le réduire au problème SAT (Figure 8). Par définition, il existe une machine de Turing non-déterministe $\mathcal{M}$ qui décide B en temps polynomial. On va alors coder l'exécution de cette machine dans des formules de la logique propositionnelle. Nous énonçons les propriétés que nous souhaitons coder sans en donner la traduction en formule de la logique propositionnelle.

1. La machine de Turing $\mathcal{M}$ et dans un état à tout instant t.
2. La machine de Turing $\mathcal{M}$ n'est jamais dans deux états à la fois.
3. Le curseur est positionné quelque part à tout instant t.
4. Le curseur n'est jamais à deux positions différentes.
5. À tout instant, toute case du ruban contient une lettre.
6. Une case du ruban ne contient pas plus d'une lettre.
7. À tout instant, on tire une transition pour aller vers l'instant $t + 1$.
8. On ne tire jamais plus d'une transition.
9. À l'instant 0, le ruban contient l'instance donnée à la machine.
10. À l'instant 0, la machine est dans l'état initial q_0 et son curseur est à la position 1.
11. La machine atteint son état d'acceptation.
12. On ne change pas le contenu du ruban, si le curseur n'y ait pas.
13. On ne tire qu'une transition de son état courant lisant la lettre sur le ruban.
14. On change s'état lorsqu'on applique une transition (le bon état).
15. On écrit la nouvelle valeur du ruban sur la case i.
16. Le curseur change de position lors d'une transition.

La formule que l'on cherche est la conjonction de toutes ses formules, ce qui prouve le théorème de Cook. □

Application La première application de ce théorème est une technique de preuve plus agréable pour montrer la NP-dureté. En effet, maintenant que nous avons un premier problème NP-complet, nous allons pourvoir réduire les autres à celui et par transitivité de la relation est réductible à, on prouvera la NP-dureté de ce problème. Depuis on en a trouvé des nombreux.

4 Technique de preuve : la réduction

Pour montrer qu'un problème apparaît dans une certaine classe de complexité (et même sa dureté) ou qu'il est indécidable, nous utilisons une technique de preuve : la réduction. Pour appliquer le principe de réduction il nous faut connaître un premier problème possédant les propriétés que l'on souhaite montrer sur le deuxième.

Nous présentons ici le principe de la réduction dans sa généralité puis nous verrons comment le spécialiser pour en faire ce que nous souhaitons.

Définition. Une réduction d'un problème A à un problème B (Figure 9) est une fonction tr calculable telle que pour tout w instance de A, w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B. On note $A \leq B$.

On dit que A se réduit à B s'il existe une réduction de A à B (intuitivement, A est plus facile que B).

Remarque. En fonction des propriétés sur la fonction tr, on obtient différentes réductions qui vont nous permettre de spécialiser la réduction au résultat que nous souhaitons montrer.

Théorème (Principe de la réduction)**.** *Si A se réduit à B, alors si P est une propriété sur B alors P est une propriété sur A (dans notre cas, P peut être l'appartenance à une classe de complexité ou être le caractère indécidable d'un problème, ...).*

Démonstration. On raisonne par l'absurde et grâce à la fonction de traduction, on obtient une contradiction. □

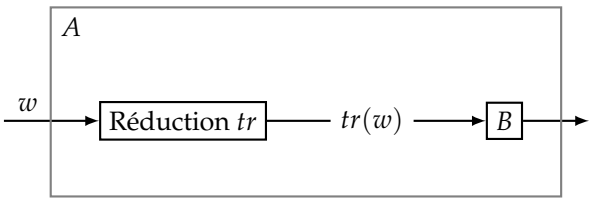

FIGURE 9 – Schéma du principe de réduction du problème A au problème B.

Nous allons donner quelques réductions de A à B et leurs propriétés. On note C une classe de complexité telle que $P \subseteq C$.

Réduction	Propriétés	Conséquences
Calculable	tr est calculable par une machine de Turing (variante de MT : équivalence)	A indécidable $\Rightarrow$ B indécidable B décidable $\Rightarrow$ A décidable
Temps polynomial	tr est calculable par une machine de Turing déterministe en temps polynomial	A est C-dur $\Rightarrow$ B est C-dur $(C \neq P)$ $B \in C \Rightarrow A \in C$
Espace logarithmique	tr est calculable par une machine de Turing déterministe en espace logarithmique (la MT a trois rubans)	A est D-dur $\Rightarrow$ B est D-dur $B \in D \Rightarrow A \in D$ $(D \neq P)$ avec $D \in \{L, NL, co-NL, P\}$

Remarque. La réduction en espace logarithmique est une réduction très spéciale car une machine qui travail en espace logarithmique a au moins deux rubans (il ne faut pas que l'entrée rentre dans la calcul). Mais pour la réduction, la sortie n'est pas non plus dans la borne de la mémoire utilisé (notons qu'elle est polynomiale (car une réduction en espace logarithmique s'exécute en temps polynomial)), il nous faut donc un troisième ruban. La machine de Turing effectuant la réduction a donc trois rubans : un ruban contenant l'entrée en lecture seule et en une seule passe ; un ruban de travail logarithmique et un ruban de sortie polynomial en écriture seule et en une seule passe.

Proposition. *Une réduction en espace logarithmique est une réduction en temps polynomial.*

Références

[1] R. Biegel R. Floyd. *Le langage des machines*. International Thomson Publishing, 1994.

[2] M. Sipser. *Introduction to the Theory of Computation*. Cengage learning, 1133187811.