Classes de complexité randomisées

Jean Goubault-Larrecq

4 décembre 2019

Résumé

Ceci est la version 30 du poly du cours de complexité II, datant du 04 décembre 2019, corrigeant un défaut de la preuve du théorème 3.14. La version 29 datait du 21 novembre 2019. La version 28 datait du 13 novembre 2019, et corrigeait un problme de dpendances entre propositions. La version 27 datait du 21 février 2019, et corrigeait quelques fautes de frappe, trouvées par Paul-Nicolas Madelaine. La version 26 datait du 17 janvier 2018, et corrigeait elle aussi quelques fautes de frappe. La version 25 datait du 08 janvier 2018. La version 24 datait du 13 décembre 2017, et simplifiait un peu la preuve du théorème de Boppana-Håstad-Zachos. La version 23 datait du 29 novembre 2017, et réorganisait la preuve du théorème de Babai $\mathbf{MA} \subseteq \mathbf{AM}$. La version 22 datait du 24 novembre 2017, et donnait principalement des références explicites aux articles fondateurs. La version 21 du datait du 15 novembre 2017. La version 20 du 14 janvier 2016 corrigeait quelques typos trouvées par Anthony Lick. La version 19, datant du 16 décembre 2015, corrigeant des problèmes sérieux dans la démonstration de l'effondrement de la hiérarchie des jeux Arthur-Merlin, ainsi que dans celle du théorème de Shamir. La version 18 datait du 25 novembre 2014. La version 17 datait du 20 novembre 2013, la version 16 datait du 15 novembre 2013, la version 15 datait du 31 janvier 2012, la version 14 datait du 25 novembre 2010, la version 23 datait du 10 novembre 2010, la version 12 datait du 11 juillet 2009, la version 11 du 16 décembre 2008, la 10 du 21 janvier 2008, la 9 du 16 janvier 2008, la 8 datait du 21 novembre 2007, la 7 datait du 10 janvier 2007, la 6 datait du 21 décembre 2006, la version 5 datait du 14 décembre 2006, la version 4 datait du 9 novembre 2006, la version 3 du 19 mai 2006, la version 2 du 11 mai 2006. Merci à Sergiu Bursuc, Fabrice Chevalier pour sa relecture attentive, et aux élèves pour leurs conseils et remarques judicieuses (Cyril Cohen, François Bobot, Mathieu Sassolas, Florent Martin, Olivier Roussel, Florent Pompigne, Guillaume Batog, puis Michaël Monerau et Marc Bagnol en 2009, Adrien Husson en 2013, Aziz Fouché Asnoun et Garance Gourdel en 2018).

Table des matières

Ce document est librement inspiré, d'une part, des notes sur un cours de Sanjeev Arora [3], et d'autre part du livre de Christos Papadimitriou [15]. La section 3.4 s'inspire d'un certain nombre de sources additionnelles, dont [10]. Le livre de Richard Lassaigne et Michel de Rougemont [13] est une lecture particulièrement recommandée qui couvre la plupart des aspects de ces notes.

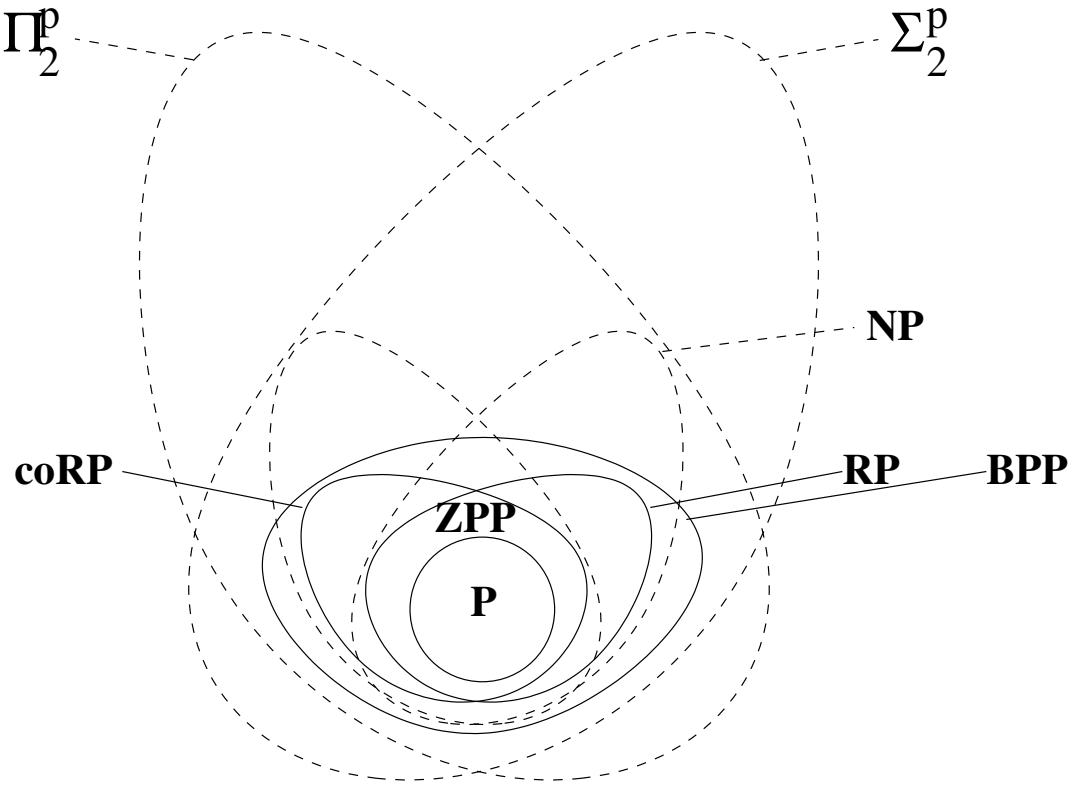

FIGURE 1 – Quelques classes randomisées, et leurs relations avec d'autres classes plus connues

1 TM randomisées

Une façon classique de définir les classes de complexité randomisées est de se fonder sur la description de "machines de Turing randomisées", qui sont des machines de Turing ayant accès à un oracle qui tire des bits aléatoires. Appelons cette "définition" la version 0 des machines de Turing randomisées.

Il est beaucoup plus simple d'utiliser des machines de Turing plus classiques et de modifier leurs conditions d'acceptation.

Définition 1.1 (TM randomisée, version 1) *Une machine de Turing randomisée est une machine de Turing* déterministe *ayant, en plus de la bande d'entrée, de la bande de sortie éventuellement, et d'un nombre fini de bandes de travail, une bande distinguée dite* bande d'aléa*, qui est en lecture seule. La tête sur cette bande est initialement à son extrémité gauche, et ne peut se déplacer que d'une case à la fois et toujours vers la droite.*

L'idée est que, plutôt que de tirer un bit aléatoire, ou plus généralement un caractère aléatoire, on lit le caractère suivant sur la bande d'aléa. Nous considérerons en général que la bande d'aléa a une certaine longueur $\ell(n)$, où n est la longueur de l'entrée, et la condition d'acceptation sera une propriété statistique sur tous les calculs de la machine, lorsque le contenu de la bande d'aléa est tiré uniformément parmi $\Sigma^{\ell(n)}$.

Pour $\mathcal{M}$ une TM randomisée qui termine, on notera $\mathcal{M}(x, r)$ le résultat du calcul de $\mathcal{M}$ sur l'entrée x, lorsque le contenu de la bande d'aléa est la chaîne r. On considérera presque exclusivement des machines résolvant des problèmes de décision : $\mathcal{M}(x, r)$ sera alors un booléen, soit $\top$ (vrai), soit $\bot$ (faux).

1.1 La classe RP

Par exemple, la classe **RP** est la classe des langages L tels qu'il existe une TM randomisée $\mathcal{M}$ travaillant en temps $p(n)$, où p est un polynôme, et telle que :

- si $x \in L$, alors $Pr_r[\mathcal{M}(x, r) = \top] \geq 1/2$, où la probabilité est prise sur tous les contenus r de longueur $p(n)$ de la bande d'aléa ;
- si $x \notin L$, alors $\mathcal{M}(x, r) = \bot$ pour tout r de longueur $p(n)$.

On dit familièrement que $\mathcal{M}$ ne se trompe jamais lorsqu'elle affirme que $x \in L$ (si $\mathcal{M}(x, r) = \top$ alors $x \in L$), et ne se trompe qu'au plus une fois sur deux lorsqu'elle affirme que $x \notin L$.

Il y a de nombreuses variantes. Notons :

Définition 1.2 *La classe* $\mathbf{RTIME}(f(n), \ell(n), accerr(n), rejerr(n))$ *est la classe des langages L tels qu'il existe une TM randomisée $\mathcal{M}$ travaillant en temps $f(n)$ avec une bande d'aléa de taille $\ell(n)$ telle que :*

- *si $x \in L$, alors $Pr_r[\mathcal{M}(x, r) = \bot] \leq rejerr(n)$;*
- *si $x \notin L$, alors $Pr_r[\mathcal{M}(x, r) = \top] \leq accerr(n)$.*

Dans les deux cas, la probabilité est prise sur tous les contenus r de longueur $\ell(n)$ de la bande d'aléa.

En particulier, **RP** est juste $\bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 0, 1/2)$.

Prenons une TM randomisée $\mathcal{M}$ décidant L en temps n^k. Soit $p(n) \geq 1$ un polynôme en la taille n de l'entrée. La machine qui calcule la disjonction des $\mathcal{M}(x, r_i)$, $1 \leq i \leq p(n)$, où la bande d'aléa r est découpée en $r_1 r_2 \dots r_{p(n)}$, termine en temps $p(n)n^k$ (plus une constante). Si $x \in L$, alors la probabilité que cette machine retourne $\perp$ vaut au plus $(1/2)^{p(n)}$. Autrement dit :

Lemme 1.3 *Dans la définition de* **RP**, *l'erreur au rejet rejerr*(n) *peut être rendue exponentiellement basse : pour tout polynôme* $p(n) \geq 1$, $\mathbf{RP} = \bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 0, 1/2^{p(n)})$.

A l'opposé, supposons que nous ayons une machine $\mathcal{M}$ ayant une probabilité d'erreur au rejet très élevée, disons $1 - \epsilon$. En répétant de nouveau l'exécution de $\mathcal{M}$, disons K fois, on peut abaisser cette probabilité d'erreur à $(1 - \epsilon)^K$. Lorsque $K = \lceil -1/\log_2(1 - \epsilon) \rceil$, $\log_2((1 - \epsilon)^K) = K \log_2(1 - \epsilon) \leq -1$, donc $(1 - \epsilon)^K \leq 1/2$. En clair, la constante $1/2$ dans la définition de **RP** n'a aucune importance, on peut la remplacer par n'importe quelle constante entre 0 et 1.

Lemme 1.4 *Pour tout* $\epsilon > 0$, $\epsilon < 1$, $\mathbf{RP} = \bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 0, \epsilon)$.

Ceci est un cas particulier du lemma 1.3 si $\epsilon \leq 1/2$. L'intérêt du lemme 1.4 est de montrer qu'on aurait pu définir **RP** avec une erreur au rejet majorée par, disons, 0.99.

Toute machine déterministe (non randomisée) $\mathcal{M}$ définit un langage : le langage des mots acceptés par $\mathcal{M}$. On remarquera que ce n'est plus le cas pour les machines randomisées. Par exemple, seules certaines machines $\mathcal{M}$ peuvent prétendre à accepter un langage au sens de **RP**, puisqu'il faut que $\mathcal{M}(x, r)$ retourne $\perp$ quel que soit r (si $x \notin L$) ou que $\mathcal{M}(x, r)$ retourne $\top$ pour au moins la moitié des valeurs de r possibles (si $x \in L$). Une machine telle que, pour un certain x, $\mathcal{M}(x, r)$ retournerait $\top$ pour, disons, une proportion $1/2^n$ des valeurs de r possibles, serait disqualifiée.

RP ressemble un peu à **NP**, et en particulier il ne semble pas qu'elle soit close par complémentaire. On peut donc définir une classe **coRP** des langages décidables par une TM randomisée qui ne se trompe jamais lorsqu'elle affirme $x \notin L$, mais ne se trompe qu'au plus la moitié du temps lorsqu'elle affirme $x \in L$:

Définition 1.5 $\mathbf{coRP} = \bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 1/2, 0)$.

Lemme 1.6 $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$.

Démonstration. $\mathbf{P} \subseteq \mathbf{RP}$ est évident : toute machine déterministe qui ne consomme pas d'aléa est une TM trivialement randomisée. $\mathbf{RP} \subseteq \mathbf{NP}$: soit $L \in \mathbf{RP}$, et $\mathcal{M}$ une TM randomisée décidant L. On construit une machine non déterministe qui se contente de deviner une bande d'aléa r telle que $\mathcal{M}(x, r) = \top$. S'il y en a une, comme $\mathcal{M}$ ne se trompe jamais lorsqu'elle accepte, c'est que $x \in L$. Réciproquement, si $x \in L$, par définition $Pr_r[\mathcal{M}(x, r) = \top] \geq 1/2$, c'est-à-dire qu'au moins la moitié de toutes les bandes d'aléa r possibles mènent à $\mathcal{M}(x, r) = \top$: il en existe donc au moins une, dès que la longueur des r envisagés est d'au moins 1 bit (donc qu'il y a au moins deux valeurs de r possibles). □

La construction du lemme précédent mène à une autre définition des machines de Turing randomisées, cette fois-ci comme des machines *non déterministes* particulières. C'est la définition de Papadimitriou [15]. Les deux définitions sont équivalentes, au sens où elles définissent les mêmes classes de langages.

Définition 1.7 (TM randomisée, version 2) *Une machine de Turing randomisée est une machine de Turing* non déterministe $\mathcal{M}$*, qui est de plus* standardisée*, au sens où :*

- $\mathcal{M}$ *est* précise *: partant de l'entrée x, toutes ses branches de calcul terminent en exactement le même nombre d'étapes $f(n)$, où n est la taille de x ;*
- *à chaque étape de calcul, $\mathcal{M}$ fait un choix non déterministe entre exactement deux transitions possibles (éventuellement menant aux mêmes configurations).*

$\mathbf{RTIME}(f(n), f(n), accerr(n), rejerr(n))$ *est alors la classe des langages L tels qu'il existe une machine $\mathcal{M}$ comme ci-dessus telle que, si $x \in L$ alors au plus $rejerr(n).2^{f(n)}$ des branches de calcul rejettent, et si $x \notin L$ alors au plus $accerr(n).2^{f(n)}$ des branches de calcul acceptent.*

1.2 La classe ZPP

La classe des langages **RP**, et parfois aussi abusivement **coRP**, est aussi appelée la classe *Monte Carlo*. La classe suivante est celles des langages *Las Vegas* :

Définition 1.8 $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.

ZPP est la classe des langages qui ont *deux* algorithmes Monte Carlo : un, $\mathcal{M}_1$, qui ne se trompe jamais lorsqu'il affirme que $x \in L$, l'autre, $\mathcal{M}_2$, qui ne se trompe jamais lorsqu'il affirme que $x \notin L$. Avant que l'on ne découvre que la primalité des entiers était dans **P**, on savait que ce problème était dans **ZPP**. Les algorithmes les plus efficaces de test de primalité, d'ailleurs, sont toujours aujourd'hui des algorithmes probabilistes.

En utilisant le lemme 1.6, on a facilement :

Lemme 1.9 $\mathbf{P} \subseteq \mathbf{ZPP}$.

De façon surprenante, **ZPP** a une caractérisation élégante :

Lemme 1.10 ZPP *est exactement la classe des langages décidables en temps* moyen *polynomial, à l'aide d'une TM randomisée qui ne fait aucune erreur.*

Ce lemme demande à opérer une légère adaptation de la définition d'une TM randomisée, qui doit maintenant disposer d'une bande d'aléa potentiellement infinie. La seule difficulté est de définir une distribution "uniforme" sur les chaînes infinies de bits aléatoires. C'est techniquement difficile, et l'on doit faire appel à un peu de théorie de la mesure. Nous resterons donc ici à un niveau intuitif : une chaîne infinie de bits est tirée uniformément si et seulement chacun de ses bits est tirée uniformément et indépendamment.

Démonstration. Si $L \in \mathbf{ZPP}$, soient $\mathcal{M}_1$ et $\mathcal{M}_2$ leurs machines Monte Carlo associées. On définit une machine $(\mathcal{M}_1.\mathcal{M}_2)^*$ comme suit. Sur l'entrée x, $(\mathcal{M}_1.\mathcal{M}_2)^*$ tire r_1 au hasard :

si $\mathcal{M}_1(x, r_1) = \top$, alors $x \in L$, stop ; sinon, tirer r_2 au hasard, si $\mathcal{M}_2(x, r_2) = \bot$, alors $x \notin L$, stop ; sinon recommencer au début (en tirant de nouveaux aléas). Lorsque la machine $(\mathcal{M}_1.\mathcal{M}_2)^*$ s'arrête (si elle s'arrête), elle donne la bonne réponse. De plus, la probabilité pour qu'elle ne s'arrête pas au cours du premier tour de boucle est la probabilité qu'au moins une des deux machines se soit trompée, qui vaut au plus $1/2$ que x soit ou non dans L. En général, la probabilité qu'elle ait effectué k tours de boucle et ne s'arrête pas est d'au plus $1/2^k$. Donc la probabilité qu'elle ne s'arrête pas vaut $\lim_{k\to+\infty} 1/2^k = 0$. (Ceci ne veut pas dire qu'elle s'arrête toujours ! Juste que les cas où elle ne s'arrête pas sont extrêmement rares.) Notons $p(n)$ le temps d'exécution maximal d'un tour de boucle (un polynôme en la taille n de x, par hypothèse). La probabilité que la machine $(\mathcal{M}_1.\mathcal{M}_2)^*$ termine au bout de k tours exactement ($k \geq 1$), donc en temps $kp(n)$, est $1/2^k$. Son temps moyen d'arrêt vaut donc au plus :

$$\sum_{k=1}^{+\infty} \frac{kp(n)}{2^k} \quad = \quad 2p(n)$$

On voit au passage que $(\mathcal{M}_1.\mathcal{M}_2)^*$ termine en deux tours de boucle en moyenne. Clairement, cette machine termine en temps moyen polynomial et ne fait jamais d'erreur.

Réciproquement, soit L un langage décidé par une TM randomisée $\mathcal{M}$ (avec bande d'aléa infinie) terminant en temps moyen n^k, et ne faisant jamais d'erreur. On définit une TM randomisée (ordinaire) $\mathcal{M}_1$ comme suit. Sur l'entrée x et l'aléa r (de longueur finie), $\mathcal{M}_1$ simule $\mathcal{M}$ tant qu'il reste des bits à lire dans r. À chaque pas de calcul, $\mathcal{M}_1$ avance la tête de la bande d'aléa, que $\mathcal{M}$ demande ou non un bit aléatoire. Lorsque la bande d'aléa est épuisée, et si $\mathcal{M}_1$ ne s'est pas arrêtée avant, $\mathcal{M}_1$ rejette. Faisons tourner $\mathcal{M}_1$ sur des bandes d'aléa de taille $n^{k'}$, pour un entier k' que nous déterminerons plus loin : $\mathcal{M}_1$ s'arrête alors en temps $n^{k'}$. Notons que $\mathcal{M}_1$ ne peut éventuellement se tromper que lorsqu'elle rejette. Notons $P(k')$ la probabilité qu'elle se trompe : c'est au plus la probabilité que $\mathcal{M}$ ne s'arrête pas en temps $n^{k'}$. On utilise l'*inégalité de Markov* : pour toute variable aléatoire X à valeurs réelles positives, de moyenne finie $E(X)$, pour tout $a > 0$, la probabilité que $X \geq a$ est inférieure ou égale à $E(X)/a$. (Démonstration rapide : $E(X) = \int_0^{+\infty} X dP(X) \geq \int_a^{+\infty} X dP(X) \geq a \int_a^{+\infty} dP(X) = aP(X \geq a)$.) La variable aléatoire est ici le temps d'arrêt de $\mathcal{M}$, et $a = n^{k'}$. On a donc $P(k') \leq n^k/n^{k'}$. Pour k' suffisamment grand, disons $k' = k + 1$, on voit que $L \in \mathbf{RTIME}(n^k, n^{k+1}, 0, 1/n) \subseteq \mathbf{RP}$. La machine $\mathcal{M}_2$ construite comme $\mathcal{M}_1$, sauf qu'à épuisement de la bande d'aléa, $\mathcal{M}_2$ accepte, montre, elle, que $L \in \mathbf{coRP}$. □

Ceci justifie le nom de la classe **ZPP** : "**Z**ero **P**robability of error **P**olynomial time".

1.3 La classe BPP

À l'opposé, on peut définir des classes randomisées avec erreurs des deux côtés : lors de l'acceptation, et lors du rejet. Ceci mène à la définition de la classe **BPP**, aussi parfois appelée *Atlantic City* :

Définition 1.11 $\mathbf{BPP} = \bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 1/3, 1/3)$.

C'est donc la classe des langages décidables par des TM randomisées qui ne se trompent qu'au plus une fois sur trois, qu'elles acceptent ou qu'elles rejettent. Remarquons tout de suite le résultat évident :

Lemme 1.12 $\mathbf{RP} \subseteq \mathbf{BPP}$. **BPP** *est close par complémentaire.* $\mathbf{coRP} \subseteq \mathbf{BPP}$.

On aurait pu penser placer des bornes d'erreur à $1/2$, comme dans la définition de **RP**. Mais l'on souhaite que les machines **BPP** décident par une *majorité claire* : que la réponse soit $\top$ ou $\bot$, au moins $2/3$ des calculs doivent être d'accord sur la bonne réponse.

On peut encore diminuer exponentiellement l'erreur, c'est-à-dire passer de $1/3$ à l'inverse d'une exponentielle en n. Ceci se fait à l'aide de la *borne de Chernoff* :

Proposition 1.13 (Chernoff) *Soient X_1, ..., X_N N variables aléatoires indépendantes, chacune prenant la valeur 1 avec probabilité p, et la valeur 0 avec probabilité $1-p$. Alors pour tout $\theta \in \mathbb{R}^+$,*

$$Pr[X_1 + \ldots + X_N \geq (1+\theta)pN] \quad \leq \quad e^{-c(\theta)pN} \tag{1}$$

où $c(\theta)$ est une fonction croissante de θ telle que, d'autre part, $c(\theta)/(1+\theta)$ est aussi croissante, et $c(\theta) \geq \frac{\theta^2}{3}$ pour tout θ, $0 \leq \theta < 1$.

Démonstration. En voici une démonstration. Par l'inégalité de Markov, pour tous $t, k > 0$,

$$Pr[e^{t(X_1+\ldots+X_N)} \geq kE(e^{t(X_1+\ldots+X_N)})] \quad \leq \quad \frac{1}{k}$$

Posons $k = e^{t(1+\theta)pN}/E(e^{t(X_1+\ldots+X_N)})$, on obtient :

$$Pr[e^{t(X_1+\ldots+X_N)} \geq e^{t(1+\theta)pN}] \quad \leq \quad E(e^{t(X_1+\ldots+X_N)})e^{-t(1+\theta)pN}$$

Le côté gauche vaut clairement $Pr[X_1 + \ldots + X_N \geq (1+\theta)pN]$, et l'espérance du côté droit vaut :

$$\begin{aligned} E(e^{t(X_1+\ldots+X_N)}) &= \prod_{i=1}^{N} E(e^{tX_i}) \qquad \text{puisque les variables sont indépendantes} \\ &= (pe^t + (1-p))^N = (1+p(e^t-1))^N \leq e^{(e^t-1)pN} \end{aligned}$$

Posons maintenant $t = \ln(1+\theta)$, on obtient alors :

$$Pr[X_1 + \ldots + X_N \geq (1+\theta)pN] \quad \leq \quad e^{(e^t-1)pN}e^{-t(1+\theta)pN} = e^{\theta pN - t(1+\theta)pN} = e^{pN(\theta-(1+\theta)\ln(1+\theta))}$$

Posons $c(\theta) = -(\theta - (1+\theta)\ln(1+\theta))$. La dérivée de c est $c'(\theta) = \ln(1+\theta) \geq 0$, donc c' est croissante. La fonction $c(\theta)/(1+\theta)$ est $-\theta/(1+\theta) + \ln(1+\theta)$, dont la dérivée est $-1/(1+\theta)^2 + 1/(1+\theta) = \theta/(1+\theta^2) \geq 0$, et est donc croissante elle aussi. Il ne reste plus qu'à montrer que $\theta - (1+\theta)\ln(1+\theta) \leq -\theta^2/3$ pour $0 \leq \theta < 1$. Voir la figure 2 pour une illustration. Or, lorsque $0 \leq \theta < 1$, le développement en série de $\theta-(1+\theta)\ln(1+\theta)$ converge,

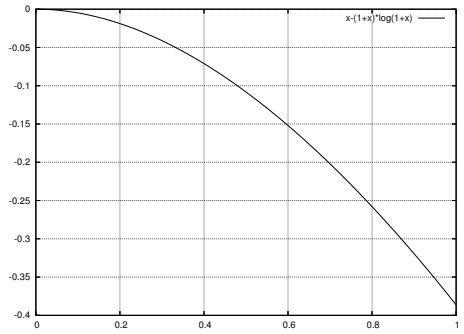

FIGURE 2 – Borne de Chernoff

et vaut $\theta - (1+\theta)(\theta - \theta^2/2 + \theta^3/3 - \ldots) = -\theta^2/2 + \theta^3/6 - \ldots$. Comme il s'agit d'une série alternante, elle est majorée par $-\theta^2/2 + \theta^3/6 \leq -\theta^2/2 + \theta^2/6 = -\theta^2/3$ (puisque $\theta \leq 1$). □

Supposons que L soit un langage dans **BPP**. On a donc une TM randomisée $\mathcal{M}$ qui décide L en ne se trompant qu'au plus une fois sur trois. Faisons tourner $\mathcal{M}$ k fois de suite sur N bandes d'aléa indépendantes et uniformément distribuées, partant de l'entrée x. Nous obtenons N résultats d'expériences $X_1, \ldots, X_N$ obéissant aux hypothèses de la borne de Chernoff, où X_i vaut 1 si l'expérience numéro i a vu $\mathcal{M}$ terminer sur $\top$ (acceptation). Décidons d'accepter si la majorité des expériences sont positives : $X_1 + \ldots + X_N \geq N/2$; et de rejeter sinon.

Supposons par exemple $x \notin L$. Alors par définition de **BPP**, la probabilité p que X_i vaille 1 est d'au plus 1/3, donc $Pr[X_1 + \ldots + X_N \geq (1+\theta)pN] \leq e^{-\frac{\theta^2}{3}pN}$. Pour $\theta = 1/2p - 1 = (1-2p)/2p$, $Pr[X_1 + \ldots + X_N \geq N/2] \leq e^{-\frac{(1-2p)^2}{12p}N} \leq e^{-N/36}$. (Plus lentement : cette probabilité est majorée par $e^{-c(\theta)pN} = e^{-c(\theta)/(1+\theta)\times N/2}$, or $p \leq 1/3$ donc $\theta \geq 1/2$, donc $c(\theta)/(1+\theta) \geq c(1/2)/(3/2) \geq 1/18$ puisque $c(\theta)/(1+\theta)$ est croissante ; donc $e^{-c(\theta)pN} \leq e^{-N/36}$.) En d'autres termes, le vote par majorité ne se trompe qu'avec probabilité au plus $e^{-N/36}$. On raisonne de même si $x \in L$. Pour $N = 36q(n)\ln 2$, on obtient donc :

Lemme 1.14 *Pour tout polynôme q,* $\mathbf{BPP} = \bigcup_{k\in\mathbb{N}} \mathbf{RTIME}(n^k, n^k, 1/2^{q(n)}, 1/2^{q(n)})$.

Contrairement à **RP**, on ne sait pas si **BPP** est inclus dans **NP**. Dans la section suivante, nous démontrons quelques résultats qui justifient qu'il est improbable que **NP** soit inclus dans **BPP**.

1.4 Circuits, classe P/poly

Une façon d'attaquer la relation entre **BPP** et **NP** est de passer par (encore) une autre classe, encore plus étrange que les précédentes... mais au moins ne sera-t-elle pas randomisée.

La classe **P/poly** est intuitivement définie comme suit. Pour décider de l'appartenance de x au langage L, on va concevoir un jeu à deux joueurs, qui ressemble un peu à un oral avec préparation. Plutôt que d'écrire x sur la bande d'entrée, et de demander à la machine de Turing $\mathcal{M}$ si x est ou non dans L, nous allons procéder en deux temps.

D'abord, je vais décrire à $\mathcal{M}$ la longueur n de x en unaire. (Supposons que x soit écrit sur l'alphabet $\{0,1\}$: n est alors le nombre de bits de x.) Autrement dit, je vais aligner n cases sur l'entrée de $\mathcal{M}$, mais le contenu des cases est *masqué*.

C'est au tour de $\mathcal{M}$ de jouer : $\mathcal{M}$ ne peut pas encore décider si $x \in L$, mais il peut *compiler* un programme décidant, en temps polynomial en n, si $y \in L$ pour toutes les entrées possibles y de longueur n. Mieux : en se fondant sur le codage de Cook, $\mathcal{M}$ peut dérouler tout le calcul d'une machine $\mathcal{M}'$ décidant $x \in L$ sous forme d'une grosse formule propositionnelle, dont les entrées sont les bits (encore inconnus) de x. (Si l'on sait que $\mathcal{M}'$ termine en temps $p(n)$, ce déroulement termine lui-même en ne fabriquant qu'au plus $p(n)^3$ clauses. Mais attention : $\mathcal{M}$ aura tout le temps qu'elle veut.) On peut en réalité compiler le calcul de $\mathcal{M}'$ sous forme d'une liste de clauses de Horn de la forme $A_{ij} = b \Leftarrow \mathcal{B}$, où $A_{ij} = b$ est une variable propositionnelle dénotant si le $i^{\text{ième}}$ bit de la $j^{\text{ième}}$ configuration de $\mathcal{M}'$ est censé être vrai (lorsque b est 1) ou faux (où b est 0), et où le corps $\mathcal{B}$ ne dépend que des $A_{i'(j-1)} = b'$ ($j \geq 1$). On obtient ainsi une conjonction S_n de 3-clauses, et deux variables propositionnelles (typiquement $A_{0p(n)} = 1$ et $A_{0p(n)} = 0$) dénotant l'accessibilité de l'état acceptant, resp. rejetant de $\mathcal{M}'$.

Une autre façon de voir S_n est de le considérer comme un circuit formé de portes et, ou, non, que l'on cherche à évaluer sur des données d'entrée A_{i0} (codant x) encore inconnues.

Dans un deuxième temps, je révèle le contenu des cases codant x. Ceci revient à fabriquer, pour chaque position i d'un bit dans x, une clause $A_{i0} = b_i$, où b_i est le $i^{\text{ième}}$ bit de x (ignorant le codage de l'état de contrôle de $\mathcal{M}'$, pour plus de simplicité). Soit I_x l'ensemble de ces nouvelles clauses. Il ne reste plus qu'à évaluer le circuit, c'est-à-dire à tester si $A_{0p(n)} = 1$ ou $A_{0p(n)} = 0$ est vraie dans le plus petit modèle de $S_n \cup I_x$. Si c'est $A_{0p(n)} = 1$ qui est vraie, alors $x \in L$, si c'est $A_{0p(n)} = 0$, alors $x \notin L$.

Appelons ce genre de machine $\mathcal{M}$ une *machine à deux temps*.

Allons plus loin. On peut même imaginer que l'on fabrique S_n à partir de la longueur n par un procédé même non calculable. La définition n'est pas équivalente, mais a le mérite d'une plus grande simplicité :

Définition 1.15 *Une* famille de circuits $\mathcal{C}$ *est une suite infinie* $\mathcal{C}_0$, $\mathcal{C}_1$, ..., $\mathcal{C}_n$, ..., *de circuits booléens, où* $\mathcal{C}_n$ *a* n *bits d'entrée. Lorsque* x *est une chaîne de bits de longueur* n, *on notera* $\mathcal{C}_n[x]$ *le résultat de l'évaluation du circuit* $\mathcal{C}_n$ *lorsque le* $i^{\text{ième}}$ *bit d'entrée de* $\mathcal{C}_n$ *est affecté à la valeur du* $i^{\text{ième}}$ *bit de* x.

Un langage $\mathcal{L} \subseteq \{0,1\}^*$ a des circuits polynomiaux *si et seulement s'il existe une famille de circuits* $\mathcal{C}$ *telle que :*

— *la taille de* $\mathcal{C}_n$ *est au plus* $p(n)$, *où* $p(n)$ *est un polynôme en* n ;

— *pour tout* $x \in \{0,1\}^*$, $x \in L$ *si et seulement si* $\mathcal{C}_n[x]$ *est vrai, où* n *est la taille de* x.

Lemme 1.16 *La classe* **P/poly** *est celle des langages qui ont des circuits polynomiaux.*

Principalement à cause du fait qu'on ne demande plus que les circuits $\mathcal{C}_n$ soient calculables, la classe **P/poly** est très étrange :

Lemme 1.17 *Il existe des langages indécidables qui ont des circuits polynomiaux.*

Démonstration. Soit L un langage indécidable sur l'alphabet $\{0,1\}$. Soit U le langage des mots 1^n (la lettre 1 répétée n fois, autrement dit n écrit en unaire) tels que n, écrit en binaire, est dans L. Il est clair que U est indécidable. Pourtant U a des circuits polynomiaux : si n est dans L, alors $\mathcal{C}_n$ est la conjonction des n bits d'entrée ; sinon, $\mathcal{C}_n$ est juste le booléen faux. □

Malgré cela, il ne semble pas, et de loin, que **P/poly** capture tous les langages indécidables, ni même les langages récursivement énumérables.

À l'opposé :

Définition 1.18 *Un langage L* a des circuits uniformément polynomiaux *si et seulement si $\mathcal{C}_n$ est calculable en espace $O(\log n)$ à partir de l'entrée 1^n.*

L'entrée 1^n est juste une façon de coder les n cases d'entrée, masquées. Il se trouve que ceci ne définit rien d'autre que la classe **P** :

Lemme 1.19 *Un langage L a des circuits uniformément polynomiaux si et seulement si $L \in \mathbf{P}$.*

Démonstration. On montre d'abord que si $L \in \mathbf{P}$, disons L est décidé par une machine $\mathcal{M}$ en temps polynomial $p(n)$, alors L a des circuits uniformément polynomiaux. L'argument est celui donné au début de la section 1.4 : on peut construire un circuit qui déroule les $p(n)$ étapes de calcul de $\mathcal{M}$ sous forme de $p(n)^3$ clauses, et ceci est faisable en espace logarithmique.

Réciproquement, si L a des circuits uniformément polynomiaux, on décide si $x \in L$ en construisant $\mathcal{C}_n$, où n est la longueur de x, en espace $\log n$ (donc en temps polynomial). De façon équivalente, en produisant les clauses de S_n. On ajoute les clauses codant l'entrée x, et on résout le problème HORNSAT permettant de conclure si la valeur de $\mathcal{C}_n$ est vraie ou non. □

On peut également démontrer que l'on obtient encore la classe **P** si l'on restreint les circuits $\mathcal{C}_n$ à être construits en temps polynomial, plutôt qu'en espace logarithmique.

On notera au passage que la conjecture $\mathbf{P} \neq \mathbf{NP}$ est équivalente au fait que les problèmes **NP**-complets n'ont pas de circuits uniformémement polynomiaux. On conjecture en fait qu'ils n'ont pas de circuits polynomiaux du tout.

Ce qui va nous intéresser est le résultat suivant dû à Len Adleman :

Proposition 1.20 (Adleman [1]) BPP $\subseteq$ P/poly.

Démonstration. C'est plus compliqué. Soit $L \in \mathbf{BPP}$. Soit $\mathcal{M}$ une TM randomisée qui décide L par une majorité écrasante (au plus $1/2^{q(n)}$ de probabilité d'erreur) en temps $p(n)$ avec $p(n)$ bits sur la bande d'aléa ; p et q sont des polynômes, et nous fixerons q plus tard. On va construire $\mathcal{C}_n$, mais pas a priori en temps polynomial en n, car sinon la remarque ci-dessus montrerait $\mathbf{P} = \mathbf{BPP}$, ce qui semble improbable.

Le principe de fonctionnement de $\mathcal{C}_n$ est le suivant. Pour toute bande d'aléa r de longueur $p(n)$, la machine qui prend une entrée x de taille n et calcule $\mathcal{M}(x,r)$ est une machine de Turing déterministe. On peut donc, comme au lemme 1.19, convertir cette machine en un

circuit. Ce sera $\mathcal{C}_n$, à condition de bien choisir r, de sorte que le résultat de $\mathcal{M}(x,r)$ soit juste pour tout x de longueur n.

Un point important est que nous allons montrer que r existe, mais nous ne saurons pas le calculer ! D'où l'intérêt de ne pas demander que les circuits soient calculables dans la définition de **P/poly**. Formellement, on calcule la probabilité sur r que $\mathcal{M}(x,r)$ se trompe pour au moins un x de longueur n. C'est au plus la somme sur tous les x de longueur n de la probabilité sur r que $\mathcal{M}(x,r)$ se trompe pour cet x là. Cette dernière probabilité vaut au plus $1/2^{q(n)}$, et il y a 2^n valeurs possibles de x. (Je suppose ici l'alphabet binaire.) La probabilité sur r que $\mathcal{M}(x,r)$ se trompe pour au moins un x de longueur n est donc inférieure ou égale à $1/2^{q(n)-n}$. Si l'on pose par exemple $q(n) = n+1$, c'est inférieur ou égal à $1/2$, donc différent de 1. Il existence en conséquence au moins une valeur de r (en fait presque la moitié des valeurs de r conviennent) telle que $\mathcal{M}(x,r)$ ne se trompe pour *aucun* x. □

Rappelons que nous voulons justifier que **NP** n'est probablement pas inclus dans **BPP**. On montre d'abord quelques conséquences de l'hypothèse **NP** $\subseteq$ **P/poly**. Rappelons que **PH** est la hiérarchie polynomiale.

Proposition 1.21 (Karp et Lipton [12]) *Si* **NP** $\subseteq$ **P/poly**, *alors* **PH** *s'effondre au niveau* 2 *:* **PH** $= \Pi_2^p = \Sigma_2^p$.

Démonstration. Il suffit de montrer que $\Pi_2^p \subseteq \Sigma_2^p$. Par hypothèse, SAT est dans **P/poly**. On montre d'abord que : $(*)$ il existe une famille w_n de mots binaires de longueur n (pour tout $n \in \mathbb{N}$) et une fonction h en temps polynomial telles que pour tout ensemble de clauses S satisfiable de taille n, $h(S, w_n)$ calcule une affectation qui satisfait toutes les clauses de S.

En effet, puisque SAT est dans **P/poly**, SAT a des circuits polynomiaux $\mathcal{C}_k$ pour chaque taille k. Connaissant ces circuits, on peut calculer une affectation ρ qui satisfait toutes les clauses de S comme suit. Notons $x_1, \ldots, x_m$ ($m \leq n$) les variables propositionnelles de S.

1. Initialement, ρ_0 est l'affectation vide $\{\}$, et $S_0 = S$. Si S_0 n'est pas satisfiable, alors retourner ρ_0. Sinon, pour i variant de 1 à m faire :
2. (* Invariant : ρ_{i-1} est une affectation des variables $x_1, \ldots, x_{i-1}$; S_{i-1} est l'ensemble de clauses de S plus, pour chaque j, $1 \leq j < i$, la clause x_j si $\rho(x_j)$ est vrai, la clause $\neg x_j$ si $\rho(x_j)$ est faux ; et S_{i-1} est satisfiable. *)
   Si S_{i-1} plus la clause x_i est satisfiable, alors poser $\rho_i = \rho_{i-1}[x_i := \top]$ et S_i égal à S_{i-1} plus la clause x_i ; sinon $\rho_i = \rho_{i-1}[x_i := \bot]$ et S_i égal à S_{i-1} plus la clause $\neg x_i$.

Notons que l'on peut décider si S_{i-1} plus la clause x_i est satisfiable en évaluant en temps polynomial le circuit $\mathcal{C}_{|S_{i-1},x_i|}$, où $|S_{i-1}, x_i|$ est la taille de S_{i-1} plus la clause x_i. Supposons pour simplifier que la taille de S plus i clauses de la forme x_j ou $\neg x_j$ soit $|S| + i(K + \log n)$, où l'on utilise K bits pour coder le signe devant la variable x_j et le séparateur d'avec les clauses précédentes, et $\log n$ bits pour coder x_j. (On notera que l'on code le signe devant x_j sur un nombre fixe de bits, donc par exemple pas par "¬" si x_j est niée et par le mot vide sinon. Ceci n'est pas très important ici, mais simplifie l'argument. Ce sera important plus bas, dans notre discussion de la taille de $f(x,y)$.) On a besoin pour exécuter l'algorithme ci-dessus de $m \leq n$ circuits qui sont $\mathcal{C}_{|S|}$, $\mathcal{C}_{|S|+K+\log n}$, $\ldots$, $\mathcal{C}_{|S|+(m-1)(K+\log n)}$: on définit w_n

comme la concaténation des (codages de) ces circuits. L'algorithme ci-dessus consulte le $i^{\text{ième}}$ circuit au tour de boucle numéro i, et est clairement un algorithme en temps polynomial en S et w_n. De plus, w_n est clairement de taille polynomiale en n.

Sachant $(*)$, montrons la proposition. Soit $L \in \Pi_2^p$. On peut donc écrire $L = \{x \mid \forall y \cdot (x,y) \in L'\}$, où y est de taille $p(n)$ en fonction de la taille n de x, et $L' \in \mathbf{NP}$. Comme SAT est **NP**-complet, il existe une réduction f de L' vers SAT.

La construction de Cook d'une instance SAT à partir du langage L' montre que l'ensemble de clauses $f(z)$, pour toute instance z de L', a une taille qui est *indépendante* de z, et ne dépend *que* de la taille $|z|$ de z. (Ici il est important que pour chaque variable booléenne z_i représentant un bit de l'entrée z, $\neg z_i$ et z_i soient codés par des mots de la même taille : c'est le seul facteur possible de variation de la taille de $f(z)$.)

Lorsque la taille de x est n et celle de y est de taille $p(n)$, la taille de $f(x,y)$ est donc égale à une quantité $q(n)$ qui ne dépend que de n, pas des lettres de x ni de y. De plus, $q(n)$ est majoré par un polynôme en n. Notons $q'(n)$ la taille de $w_{q(n)}$ — c'est aussi une quantité polynomiale. On a :

$$\begin{aligned}
L &= \{x \text{ de taille } n \mid \forall y \text{ de taille } p(n) \cdot f(x,y) \in \text{SAT}\} \\
&= \{x \text{ de taille } n \mid \forall y \text{ de taille } p(n) \cdot \\
&\qquad\qquad h(f(x,y), w_{q(n)}) \text{ est une affectation satisfaisant } f(x,y)\} \quad \text{par } (*) \\
&= \{x \text{ de taille } n \mid \exists w \text{ de taille } q'(n) \cdot \forall y \text{ de taille } p(n) \cdot \\
&\qquad\qquad h(f(x,y), w) \text{ est une affectation satisfaisant } f(x,y)\}
\end{aligned}$$

Seule la dernière égalité mérite une justification. Dans un sens, si $x \in L$, alors pour tout y de taille $p(n)$, $h(f(x,y), w_{q(n)})$ est une affectation satisfaisant $f(x,y)$ (deuxième ligne) ; en posant $w = w_{q(n)}$, qui est indépendant de y et de taille $q'(n)$, on a bien que pour tout y de taille $p(n)$, $h(f(x,y), w)$ est une affectation satisfaisant $f(x,y)$. Dans l'autre sens, si $h(f(x,y), w)$ est une affectation satisfaisant $f(x,y)$, alors $f(x,y)$ est satisfiable, donc x est dans L (première ligne).

Donc $L \in \Sigma_2^p$, et l'on conclut. □

La technique ci-dessus consistant à calculer une affectation satisfaisant S lorsqu'on a juste une machine qui décide de la satisfiabilité est un classique. Le problème SAT est dit *auto-réductible* ("self-reducible" en anglais) : en utilisant un oracle de satisfiabilité, on peut réduire le problème de l'affectation à celui de trouver une affectation plus courte sur un autre problème SAT.

Proposition 1.22 (Karp et Lipton [12]) *Si* $\mathbf{NP} \subseteq \mathbf{P/poly}$ *alors* $\mathbf{PH} \subseteq \mathbf{P/poly}$.

Démonstration. Par la proposition 1.21, il suffit de montrer que $\Pi_2^p \subseteq \mathbf{P/poly}$. Soit $L \in \Pi_2^p$. Par définition, L s'écrit sous la forme $\{x \mid \exists y \cdot (x,y) \in L'\}$, où $L' \in \mathbf{coNP}$ et y est de taille $p(n)$ (un polynôme) lorsque x est de taille n. On peut de plus supposer que L' est un langage composé uniquement de tels couples (x,y). Comme $\mathbf{NP} \subseteq \mathbf{P/poly}$ et que $\mathbf{P/poly}$ est close par complémentaires, L' est dans $\mathbf{P/poly}$. Soit $\mathcal{C}_k$ le circuit traitant

des entrées de taille k pour L'. Quitte à représenter $\mathcal{C}_k$ sous forme d'une chaîne de bits w_k, $L' = \{(x,y) \mid x \text{ de taille } n, y \text{ de taille } p(n), Eval((x,y), w_{n+p(n)}) = \top\}$, où $Eval$ est la machine de Turing qui évalue le circuit en deuxième argument sur l'entrée (x,y). Donc $L = \{x \text{ de taille } n \mid (x, w_{n+p(n)}) \in L''\}$, où $L'' = \{(x,w) \mid x \text{ de taille } n, \exists y \text{ de taille } p(n), Eval((x,y),w) = \top\}$. Clairement, L'' est dans **NP**, donc dans **P/poly** par hypothèse. Soit $\mathcal{C}''_k$ le circuit pour les entrées de taille k pour L''. On fabrique un circuit $\mathcal{C}'_n$ pour L en connectant les entrées des bits $n+1$ à $2n+p(n)$ du circuit $\mathcal{C}''_{2n+p(n)}$ aux bits 1 à $n+p(n)$ du mot $w_{n+p(n)}$ décrivant $\mathcal{C}_{n+p(n)}$. □

On pense généralement que la hiérarchie polynomiale ne s'éffondre pas. Comme **BPP** $\subseteq$ **P/poly** par la proposition 1.20, il est donc improbable que **NP** soit inclus dans **BPP** :

Corollaire 1.23 *Si* **NP** $\subseteq$ **BPP** *alors* **PH** *s'effondre au niveau* 2.

Cependant, **BPP** est une classe très basse dans la hiérarchie polynomiale, par un résultat de Sipser [20] et Gács, dont nous donnons une preuve due à Lautemann [14].

Proposition 1.24 (Sipser-Gács-Lautemann) **BPP** $\subseteq \Sigma_2^p \cap \Pi_2^p$.

Démonstration. Comme **BPP** est close par complémentaire, il suffit de montrer **BPP** $\subseteq \Sigma_2^p$. Soit L un langage de **BPP**, décidé par une TM randomisée $\mathcal{M}$ en temps $p(n)$ utilisant $q(n)$ bits aléatoires, avec probabilité d'erreur au plus $1/2^n$. (Ce qui est possible par le lemme 1.14.) Soit x une entrée de taille n, et soit $m = q(n)$ le nombre de bits aléatoires requis. Soit R l'ensemble des bandes d'aléa r de longueur m telles que $\mathcal{M}(x,r) = \top$. Par hypothèse, si $x \in L$, R contient au moins $1 - 1/2^n$ de toutes les bandes de longueur m ("R est immense"), et sinon, R contient au plus $1/2^n$ de ces bandes ("R est ridicule").

Pour toute chaîne de bits t de longueur m, soit $t \oplus r$ le ou exclusif bit à bit de t et de r. Soit $t \oplus R = \{t \oplus r \mid r \in R\}$. L'opération $r \mapsto t \oplus r$ est une bijection (même une involution), donc le cardinal de R et celui de $t \oplus R$ coïncident pour tout t. De plus, si r est fixé et t est tiré au hasard uniformément, alors $t \oplus r$ est aussi soumis à une distribution uniforme.

Si R est immense, c'est-à-dire est de cardinal au moins $2^m(1 - 1/2^n)$, montrons qu'il existe des mots $t_0, t_1, \ldots, t_{\lceil m/n \rceil}$, chacun de longueur m, tels que les $t_i \oplus R$ recouvrent tous les mots de longueur m, autrement dit $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R) = \{0,1\}^m$. On raisonne par une méthode probabiliste similaire à celle utilisée à la proposition 1.20. Tirons la famille $t_0, t_1, \ldots, t_{\lceil m/n \rceil}$ au hasard uniformément. Soit r fixé. La probabilité que r ne soit pas dans $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R)$ est au plus $\prod_{i=0}^{\lceil m/n \rceil} 1/2^n \leq 1/2^{m+1}$. La probabilité qu'il existe un $r \in \{0,1\}^m$ qui ne soit pas dans $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R)$ est donc d'au plus $\sum_{r \in \{0,1\}^m} 1/2^{m+1} \leq 1/2$. En particulier, la probabilité que les $t_i \oplus R$ recouvrent tous les mots de longueur m est au moins $1/2$, et est donc non nulle. On conclut.

Si R est ridicule, c'est-à-dire est de cardinal au plus 2^{m-n}, alors le cardinal de $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R)$ vaut au plus $1 + \lceil m/n \rceil$ fois le cardinal de R. La proportion des éléments de $\{0,1\}^m$ qui sont dans $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R)$ est donc d'au plus $(1 + \lceil m/n \rceil)/2^n = O(q(n)/2^n)$. Pour n suffisamment grand, il existe donc une chaîne r de longueur m qui ne sera pas dans $\bigcup_{i=0}^{\lceil m/n \rceil}(t_i \oplus R)$.

Dans les deux cas, on conclut que pour tout x (de longueur n suffisamment grande), $x \in L$ si et seulement s'il existe des chaînes $t_0, t_1, \ldots, t_{\lceil q(n)/n \rceil}$ de longueur $q(n)$ telles que pour toute chaîne r de longueur $q(n)$, $\mathcal{M}(x, t_0 \oplus r) = \top$ ou ... ou $\mathcal{M}(x, t_{\lceil q(n)/n \rceil} \oplus r) = \top$. Ceci montre que L est dans Σ_2^p. □

La technique utilisée dans la proposition ci-dessus est souvent appelée "dérandomisation" : au lieu de tirer r au hasard, on énumère ou on devine un nombre faible (polynomial) de valeurs de r pour aboutir au même résultat.

2 Approximabilité

Résoudre exactement un problème de décision n'est pas forcément toujours ce que l'on cherche. Par exemple, plutôt que de résoudre SAT, et de trouver une affectation de valeurs de vérité qui satisfasse *toutes* les clauses de l'ensemble de clauses S en entrée, on peut se satisfaire d'un algorithme qui essaie d'en satisfaire le plus possible tout en restant en temps polynomial.

On aimerait tout de même avoir une garantie que le résultat obtenu ne soit pas trop loin de l'optimum. On définit par exemple MAXSAT_ϵ pour tout $\epsilon \geq 0$ comme suit :
ENTRÉE : un ensemble de clauses propositionnelles S ;
SORTIE : une affectation ρ qui satisfait au moins $(1-\epsilon)opt(S)$ clauses de S, où $opt(S)$ est le nombre maximal de clauses de S que l'on peut satisfaire en même temps.

Pour $\epsilon = 0$, ceci n'est autre que le problème MAXSAT, dont on ne connaît aucun algorithme polynomial. En fait, s'il en existait un, on pourrait décider de SAT en temps polynomial en appliquant l'algorithme et en vérifiant que le ρ retourné satisfait toutes les clauses de S.

La question est alors de se demander pour quelles valeurs de $\epsilon > 0$ il existe un algorithme polynomial pour MAXSAT_ϵ. Il en existe un trivial pour $\epsilon = 1$. De plus s'il en existe un pour ϵ, alors aussi pour tout $\epsilon' > \epsilon$. Il est donc sensé d'évaluer la borne inférieure de tous les ϵ pour lesquels MAXSAT_ϵ est polynomial. Ceci s'appelle le *seuil d'approximation* du problème MAXSAT.

En général, soit A un problème d'optimisation. Ceci signifie que pour chaque entrée x, il existe un ensemble $F(x)$ de solutions *faisables* (les affectations de valeurs de vérité aux variables présentes dans x pour MAXSAT), et que chaque $s \in F(x)$ a une *valeur* $c(s) \in \mathbb{N}^*$ (le nombre de clauses satisfaites pour MAXSAT). Soit $opt(x) = \max_{s \in F(x)} c(s)$ la valeur optimale.

Définition 2.1 *Soit $\mathcal{M}$ une machine de Turing telle que $\mathcal{M}(x) \in F(x)$ pour tout x. On dit que $\mathcal{M}$ est un* algorithme d'approximation à ϵ près *si et seulement si*

$$\frac{|c(\mathcal{M}(x)) - opt(x)|}{\max(opt(x), c(\mathcal{M}(x)))} \quad \leq \quad \epsilon \tag{2}$$

On définit aussi la même notion avec $opt(x) = \min_{s \in F(x)} c(s)$ pour les problèmes de minimisation, et $c(s)$ est alors appelé le *coût* de s.

L'inégalité (2) s'adapte tant aux problèmes de minimisation qu'aux problèmes de maximisation. Pour un problème de maximisation comme MAXSAT, (2) est équivalente à :

$$opt(x) - c(\mathcal{M}(x)) \quad \leq \epsilon.opt(x)$$

ou encore

$$c(\mathcal{M}(x)) \quad \geq \quad (1-\epsilon).opt(x)$$

Pour un problème de minimisation, (2) est équivalente à :

$$c(\mathcal{M}(x)) - opt(x) \quad \leq \quad \epsilon.c(\mathcal{M}(x))$$

c'est-à-dire à :

$$c(\mathcal{M}(x)) \quad \leq \quad \frac{1}{1-\epsilon} opt(x)$$

Définition 2.2 *Le* seuil d'approximation *de A est la borne inférieure des $\epsilon > 0$ tels qu'il existe un algorithme d'approximation* en temps polynomial *à ϵ près pour A.*

Le seuil d'approximation est toujours compris entre 0 et 1. Les exemples qui suivent montrent que ce seuil peut valoir essentiellement n'importe quel réel entre 0 et 1 — du moment que $\mathbf{P} \neq \mathbf{NP}$, sinon bien sûr il vaut 0 pour tous les problèmes de **NP**.

2.1 NODE COVER

Le problème NODE COVER est le problème de minimisation suivant. (On considérera toujours qu'un graphe non orienté $G = (V, E)$ a la propriété que toute arête $\{u, v\} \in E$ est de cardinal 2, c'est-à-dire que $u \neq v$: il n'y a pas d'auto-boucles.)
ENTRÉE : un graphe non orienté $G = (V, E)$.
SOLUTION FAISABLE : un ensemble de sommets $C \subseteq V$ tel que toute arête de E est incidente à C (c'est-à-dire est telle qu'une de ses extrémités est dans C).
COÛT : le cardinal $|C|$ de C.

On appelle *recouvrement* de G toute solution faisable de ce problème.

Le problème de décision associé est : étant donné G et un budget $k \in \mathbb{N}$, existe-t-il un recouvrement C de G de cardinal au plus k ? On le note traditionnellement encore NODE COVER. On a alors :

Lemme 2.3 *NODE COVER est* **NP***-complet.*

Démonstration. C est un recouvrement de G si et seulement si $V \setminus C$ est un *ensemble indépendant*, c'est-à-dire un ensemble de sommets dont aucun n'est relié à aucun autre par une arête de G. NODE COVER est donc équivalent au problème INDEPENDENT SET :
ENTRÉE : un graphe non orienté $G = (V, E)$, un entier $k \in \mathbb{N}$.
QUESTION : existe-t-il un ensemble indépendant de G de cardinal au moins k ?

On réduit 3SAT à INDEPENDENT SET comme suit. On peut supposer sans restreindre la généralité du problème que l'on part d'un ensemble de 3-clauses S qui ne contient aucune tautologie (une clause contenant à la fois x et $\neg x$) ni aucune clause unitaire (restreinte à un littéral, x ou $\neg x$). Pour chaque clause $L_1 \vee L_2 \vee L_3$, on fabrique trois sommets frais formant une clique (un triangle). De même pour les clauses de taille 2. Pour toute clause C contenant un littéral x et toute clause C' contenant le littéral opposé $\neg x$, on relie les sommets correspondants. Aucun ensemble indépendant ne peut contenir plus d'un sommet par clique, donc, s'il y a m clauses dans S, aucun ensemble indépendant n'est de cardinal strictement supérieur à m. S'il existe un ensemble indépendant I de cardinal m, il contient exactement un sommet de chaque clique. Si un sommet étiqueté x est dans I, aucun sommet de I ne peut être étiqueté $\neg x$, et réciproquement : ceci fournit directement une affectation satisfaisant S. Réciproquement, si ρ est une affectation satisfaisant S, on forme un ensemble indépendant en sélectionnant un littéral vrai dans chaque clause. □

Ce lemme montre au moins que, si $\mathbf{P} \neq \mathbf{NP}$, alors NODE COVER n'est pas dans $\mathbf{P}$. Chercher un algorithme d'approximation a donc un sens.

Il se trouve que NODE COVER est facile à approximer, jusqu'à un certain point au moins :

Proposition 2.4 *Le seuil d'approximation de NODE COVER est d'au plus* 1/2.

Démonstration. On considère l'algorithme (stupide) suivant. Initialement, $C = \emptyset$. Tant qu'il reste des arêtes dans G, en prendre une quelconque $\{u, v\}$, et rajouter u et v à C. Puis effacer u, v, et toutes les arêtes incidentes à u ou v de G, et recommencer.

En pratique, plutôt que d'effacer des sommets et des arêtes, il suffit de les marquer comme inutilisables par la suite, par exemple dans une table stockée sur une bande annexe.

Cet algorithme est alors clairement en temps polynomial, et il est facile de voir qu'il fournit un recouvrement de G. C est construit de telle sorte qu'il est l'union de paires $\{u, v\}$ qui sont des arêtes de G ne partageant aucun sommet. Un tel ensemble de paires est appelé un *couplage* de G. Ce couplage est de cardinal $|C|/2$, et tout recouvrement de G doit contenir au moins une extrémité de chacune des arêtes du couplage, donc être aussi de cardinal au moins $|C|/2$: l'algorithme est approximant à 1/2 près. □

Bizarrement, l'algorithme glouton, apparemment plus intelligent, qui consiste à choisir le sommet de plus grand degré de G, à l'ajouter à C, puis à l'effacer de G et à recommencer, ne fournit pas d'algorithme d'approximation à ϵ-près pour aucun $\epsilon < 1$. En fait, l'algorithme stupide de la démonstration de la proposition 2.4 est celui qui fournit le meilleur taux d'approximation connu pour ce problème.

2.2 TSP

Le problème du voyageur de commerce (TSP) est lui extrêmement difficile à approximer :
ENTRÉE : une matrice $D = (d_{ij})_{1 \leq i,j \leq n}$ de distances $d_{ij} = d_{ji} \in \mathbb{N}$, avec $d_{ii} = 0$ pour tout i, entre les villes $1, \ldots, n$.
SOLUTION FAISABLE : une permutation π de $\{1, \ldots, n\}$.

COÛT : la distance parcourue sur le chemin $\pi(1), \pi(2), \ldots, \pi(n), \pi(1)$, c'est-à-dire $c(\pi) = \sum_{i=1}^{n} d_{\pi(i),\pi(i \bmod n+1)}$.

Il s'agit d'un problème de minimisation.

Lemme 2.5 *TSP est* **NP***-complet.*

Démonstration. Par réduction à partir de HAMILTONIAN CYCLE, le problème de l'existence d'un circuit hamiltonien dans un graphe non orienté. Étant donné un graphe non orienté $G = (V, E)$, où $V = \{1, \ldots, n\}$, on pose $d_{ij} = d_{ji} = 1$ si $\{i, j\} \in E$, $d_{ij} = d_{ji} = 2$ sinon et si $i \neq j$, $d_{ii} = 0$ finalement, et l'on alloue un budget de n. □

Et il est difficile à approximer :

Proposition 2.6 *Le problème de décision TSP$_\epsilon$ ($0 \leq \epsilon < 1$), où l'on demande s'il existe une permutation π de coût $c(\pi) \leq \frac{1}{1-\epsilon} opt(D)$, est* **NP***-difficile. En particulier, si* $\mathbf{P} \neq \mathbf{NP}$*, alors le seuil d'approximation de TSP est* 1*.*

Démonstration. C'est fondamentalement la même construction que celle du lemme 2.5, sauf que $d_{ij} = d_{ji}$ vaut $opt(D)/(1-\epsilon)$ si $\{i, j\} \notin E$, plutôt que 2. Le budget est de $opt(D)/(1-\epsilon)$. Si G a un circuit hamiltonien π, alors $c(\pi) = n \leq opt(D)/(1-\epsilon)$ (noter que $opt(D)$ est nécessairement supérieur ou égal à n). Sinon, le circuit passe par au moins une paire $\{i, j\}$ à distance $opt(D)/(1-\epsilon)$, donc $c(\pi) > opt(D)/(1-\epsilon)$.

Supposons maintenant que TSP ait un algorithme d'approximation à ϵ près, $\epsilon < 1$. On peut alors, par définition, décider TSP$_\epsilon$ en temps polynomial. Donc $\mathbf{P} = \mathbf{NP}$. □

2.3 KNAPSACK

À l'opposé, KNAPSACK est un problème très facile à approximer. Il s'agit du problème de maximisation suivant.
ENTRÉE : une collection de prix $v_i \in \mathbb{N}$ et de poids $w_i \in \mathbb{N}$ d'objets $i = 1, \ldots, n$, un poids limite $W \in \mathbb{N}$.
SOLUTION FAISABLE : un sous-ensemble S de $\{1, \ldots, n\}$ de poids $\sum_{i \in S} w_i$ inférieur ou égal à W.
VALEUR : le prix total de S, $\sum_{i \in S} v_i$.

KNAPSACK est un problème **NP**-complet, voir Papadimitriou [15, théorème 9.10].

Proposition 2.7 *Le seuil d'approximation de KNAPSACK est* 0*.*

Démonstration. On commence par écarter un cas trivial : si tous les poids w_i sont strictement supérieurs à W, alors l'instance donnée de KNAPSACK est facile à résoudre, vu que l'unique solution faisable est $S = \emptyset$. On supposera donc dans la suite qu'il existe un poids $w_i \leq W$, ce qui entraînera qu'il existe toujours une solution optimale S de valeur non nulle.

Soit $V = \max_{\substack{1 \leq i \leq n \\ w_i \leq W}} v_i$ le prix maximum des objets pertinents de $\{1, \ldots, n\}$, et définissons $W(i, v)$ le plus petit poids d'un ensemble $S \subseteq \{1, \ldots, i\}$ dont la valeur est exactement v, pour tout v, $0 \leq v \leq nV$. Lorsqu'aucun S n'a le poids v, on pose $W(i, v) = +\infty$.

Lorsque $i = 0$, on a $W(0,0) = 0$, $W(0,v) = +\infty$ pour tout $v \neq 0$. Calculons ensuite par récurrence sur i, $1 \leq i \leq n$: $W(i,v) = \min(W(i-1,v), w_i + W(i-1,v-v_i))$ si $v \geq v_i$, sinon $W(i,v) = W(i-1,v)$. Ceci mène à un algorithme calculant $W(i,v)$ pour tous i, v, $0 \leq i \leq n$, $0 \leq v \leq nV$, par programmation dynamique. Cet algorithme maintient et remplit une table d'entiers de taille $O(n^2V)$, et tourne en temps $O(n^2V)$. Ceci permet de résoudre KNAPSACK : il suffit ensuite de calculer le plus grand v, $0 \leq v \leq nV$, tel que $W(n,v) \leq W$. La solution S optimale s'en déduit comme suit : pour tout i allant de n à 1, si $W(i-1,v) = W(i,v)$, alors i n'est pas dans S ; sinon i est dans S, poser $v := v - v_i$, et recommencer.

Mais ceci n'est pas un algorithme en temps polynomial, puisque V est une exponentielle de la taille de l'entrée en général.

Modifions donc l'algorithme ci-dessus en représentant les valeurs possibles v en ne gardant que leurs k bits de poids fort. Supposons pour ceci que nV s'écrive sur k_0 bits (autrement dit $k_0 = O(\log_2(nV))$), et $k \leq k_0$. Pour tout v, $0 \leq v \leq nV$, notons $\widetilde{v} = \lfloor v/2^{k_0-k} \rfloor$, le nombre v dont on n'a gardé que les k premiers bits. Remplaçons dans le calcul de W les prix v_i par les prix approchés $\widetilde{v}_i$: $W'(0,0) = 0$, $W'(0,\widetilde{v}) = +\infty$ pour tout $\widetilde{v} \neq 0$, $0 \leq \widetilde{v} \leq 2^k$; et pour tout i, $1 \leq i \leq n$, $W'(i,\widetilde{v}) = \min(W'(i-1,\widetilde{v}), w_i + W'(i-1,\widetilde{v}-\widetilde{v}_i))$, sinon $W'(i,\widetilde{v}) = W'(i-1,\widetilde{v})$.

Soit S' la solution optimale trouvée par l'algorithme décrit plus haut, mais travaillant avec W' plutôt que W. La valeur de S, $\sum_{i \in S} v_i$, est nécessairement supérieure ou égale à celle de S', $\sum_{i \in S'} v_i$, puisque S est optimal. Par construction, $\sum_{i \in S'} v_i \geq 2^{k_0-k} \sum_{i \in S'} \widetilde{v}_i$. Mais $\sum_{i \in S'} \widetilde{v}_i \geq \sum_{i \in S} \widetilde{v}_i$, puisque cette fois-ci c'est S' qui est optimal sur le problème tronqué aux k bits de poids fort. Donc $\sum_{i \in S} v_i \geq 2^{k_0-k} \sum_{i \in S} \widetilde{v}_i$. Puisque $\widetilde{v} \geq v/2^{k_0-k} - 1$, on en déduit $\sum_{i \in S} v_i \geq 2^{k_0-k} \sum_{i \in S'} \widetilde{v}_i \geq \sum_{i \in S} v_i - n2^{k_0-k}$.

Le rapport z de la valeur obtenue $2^{k_0-k} \sum_{i \in S'} \widetilde{v}_i$ sur l'optimum $\sum_{i \in S} v_i$ est donc d'au moins $1 - n2^{k_0-k} / \sum_{i \in S} v_i$. (Ceci a un sens par notre remarque liminaire sur les poids w_i, qui entraîne que toute solution optimale S a une valeur non nulle, et est non vide.) Une solution faisable, en général non optimale, consiste en le singleton $\{i\}$, où i est un objet de valeur maximale parmi ceux de poids au plus W. La valeur de cette solution est V, par définition de V. Donc $\sum_{i \in S} v_i \geq V$. On en déduit que $z \geq 1 - n2^{k_0-k}/V$.

Pour tout ϵ, $0 < \epsilon < 1$, on peut rendre le rapport z plus grand que $1-\epsilon$: il suffit de prendre $k = \lceil k_0 - \log_2(\epsilon V/n) \rceil$. L'algorithme modifié tourne alors en temps $O(n^2 2^k) = O(n^3/\epsilon)$. □

2.4 MAXSAT

On peut caractériser MAXSAT formellement comme un problème de maximisation :
ENTRÉE : un ensemble de clauses propositionnelles S, donc aucune n'est une tautologie.
SOLUTION FAISABLE : une affectation ρ de valeurs de vérité aux variables de S.
VALEUR : le nombre de clauses de S satisfaites par ρ.

Le problème de décision associé est ici encore **NP**-complet. (Ce n'est pas SAT.)

Celui-ci peut être approché de très près. Ce qui est intéressant ici, c'est que l'algorithme d'approximation va faire intervenir un argument probabiliste, quand bien même il est lui-même parfaitement déterministe. Il est important que S ne contienne pas de tautologie.

Lemme 2.8 *Le seuil d'approximation de MAXSAT est au plus* $1/2$*. Le seuil d'approximation du problème k-MAXSAT, où l'entrée S est restreinte à ne consister qu'en des clauses non tautologiques contenant au moins k littéraux distincts, est au plus* $1/2^k$*.*

Démonstration. Pour toute clause C (ayant au moins k littéraux distincts), soit $p(C)$ la probabilité $Pr_\rho[\rho \not\models C]$, lorsque ρ est tirée au hasard. On peut calculer $p(C)$ exactement : comme C n'est pas une tautologie par hypothèse, $p(C) = 1/2^{k'}$, lorsque C contient $k' \geq k$ littéraux. Ceci est parce que la seule façon de rendre une clause $+A_1 \vee \ldots \vee +A_m \vee -A_{m+1} \vee \ldots \vee -A_{k'}$ fausse est de prendre A_1 faux, ..., A_m faux, A_{m+1} vrai, ..., et $A_{k'}$ vrai.

En particulier, $p(C) \leq 1/2^k$.

On peut donc aussi calculer l'espérance $E(S)$ du nombre de clauses C non satisfaites dans S par $E(S) = \sum_{C \in S} E(\{C\}) = \sum_{C \in S} p(C)$.

Pour tout ensemble de clauses S, posons $S[x := \top]$ l'ensemble obtenu en effaçant les clauses de S contenant le littéral x, en gardant les autres, et en effaçant de chacune de ces dernières le littéral $\neg x$ s'il est présent. Symétriquement, $S[x := \bot]$ est obtenu en effaçant les clauses contenant $\neg x$, et en effaçant x des clauses restantes. Ceci revient à remplacer x par vrai ($\top$), resp. faux ($\bot$), puis à simplifier.

L'algorithme est le suivant. Soient x_1, ..., x_n les variables de S. Poser $\rho_0 = \emptyset$, $S_0 = S$. Pour i variant de 1 à n, comparer $E(S_{i-1}[x_i := \top])$ et $E(S_{i-1}[x_i := \bot])$; si la première quantité est la plus petite, poser $\rho_i = \rho_{i-1}[x_i \mapsto \top]$ et $S_i = S_{i-1}[x_i := \top]$, sinon $\rho_i = \rho_{i-1}[x_i \mapsto \bot]$ et $S_i = S_{i-1}[x_i := \bot]$. Finalement, $\rho = \rho_n$.

On prétend que pour tous S et x, $E(S) = \frac{1}{2}(E(S[x := \top]) + E(S[x := \bot]))$. En effet, chaque clause de S peut contenir x ou pas, et contenir $\neg x$ ou pas, mais pas contenir les deux. Si C ne contient ni x ni $\neg x$, $p(C)$ intervient dans la somme $E(S)$ avec coefficient 1, ainsi que dans les sommes $E(S[x := \top]$ et $E(S[x := \bot])$. Si C contient x mais pas $\neg x$, C a disparu de $S[x := \top]$, et apparaît sous la forme d'une clause C' égale à C privée de x dans $S[x := \bot]$: cette dernière contribue pour $1/2^{k'-1}$ dans la somme $E(S[x := \bot])$, et C contribue pour $1/2^{k'}$ dans la somme $E(S)$. De même si C contient $\neg x$ mais pas x.

Au vu de l'algorithme, S_i est tel que $E(S_i)$ est la plus petite des deux quantités $E(S_{i-1}[x := \top])$ et $E(S_{i-1}[x := \bot])$, donc $E(S_i) \leq E(S_{i-1})$. En particulier, $E(S_n) \leq E(S)$. Or $E(S) = \sum_{C \in S} p(C) \leq |S|/2^k$. Il y a donc à terminaison de l'algorithme au plus $|S|/2^k$ clauses non satisfaites. □

Bien que MAXSAT s'approxime bien, son seuil d'approximation ne peut valoir 0 que si $\mathbf{P} = \mathbf{NP}$. C'est un résultat compliqué (le théorème d'Arora-Safra), et qui fait appel à une nouvelle caractérisation de **NP** via une classe randomisée, qui ressemble à une restriction de la classe **RP**, avec du non-déterminisme en plus. Cette classe s'appelle **PCP**, et sera définie formellement plus loin.

Avant de définir **PCP**, réexaminons la classe **NP**. Un langage $L \in \mathbf{NP}$ est un ensemble de chaînes x tel que $x \in L$ si et seulement s'il existe une chaîne y, de taille $p(n)$, où $p(n)$ est un polynôme en la taille n de x, telle que (x, y) soit dans un certain langage L' en temps polynomial. On peut voir y comme une *preuve* que $x \in L$, et une machine décidant l'appartenance à L' est un *vérificateur* du fait que y soit bien une preuve de l'assertion

$x \in L$. Par exemple, une affectation de valeurs de vérité satisfaisant un ensemble de clauses propositionnelles S est une preuve de la satisfiabilité de S.

Un langage de **NP** est donc un langage L ayant des preuves y de taille polynomiale, et vérifiables par un algorithme déterministe en temps polynomial.

Que se passerait-il si nous autorisions le vérificateur à utiliser un peu d'aléa ? On peut espérer que ceci réduise les ressources nécessaires : des preuves plus courtes, ou bien un temps de vérification plus court. Le théorème d'Arora-Safra cherche à réduire le temps de vérification. Définissons une classe **PCP** comme celle des langages L que l'on peut décider par des machines non déterministes randomisées $\mathcal{M}$ comme suit. On définit une machine non déterministe randomisée comme une machine déterministe avec deux bandes de travail distinguées : une bande d'aléa r, et une bande de preuve y. (Ici, la machine va d'abord tirer r au hasard, puis deviner y. On peut se demander ce qui se passerait si on devinait y d'abord, puis que l'on tirait r au hasard, ou bien si l'on intercalait les deux... ceci sera le sujet de la section 3.)

- Sur l'entrée x de taille n, $\mathcal{M}$ tire $O(\log n)$ bits aléatoires r, puis calcule un nombre constant k d'entiers $p_1, \ldots, p_k$ entre 1 et $p(n)$, en temps polynomial en n. (Ces entiers p_i sont des positions sur la bande de preuve y, mais doivent être calculés de façon *non adaptative*, c'est-à-dire sans pouvoir consulter y au fur et à mesure : on n'a pas encore deviné y à ce moment.)
- $\mathcal{M}$ devine ensuite une bande y de taille $p(n)$.
- Finalement, $\mathcal{M}$ lit les k bits de y aux positions $p_1, \ldots, p_k$, et décide si y est une preuve de $x \in L$ en ne regardant que ces bits de y, et ce en temps *constant*. Si c'est le cas, $\mathcal{M}$ doit accepter. Sinon, $\mathcal{M}$ doit rejeter avec probabilité au moins $1/2$, les probabilités étant comme d'habitude prises sur les r de taille $q(n)$ possibles.

Ce mode de calcul baroque peut sembler être trop restreint pour ne permettre aucun calcul digne d'intérêt. Le théorème d'Arora-Safra énonce pourtant que $\mathbf{PCP} = \mathbf{NP}$.

Si nous devions être un peu plus formels dans la définition de **PCP**, nous devrions préciser ce que signifie lire les k bits aux positions $p_1, \ldots, p_k$ de y, et seulement eux. Ceci n'a aucun sens sur une machine de Turing usuelle, puisque la machine va de toute façon passer au-dessus tous les bits intermédiaires de y. Une façon de se débarrasser du problème est de supposer que $\mathcal{M}$ est à *lecture directe* sur la bande de preuve y. Ceci signifie que $\mathcal{M}$ dispose d'une autre bande spéciale, dite d'*adresses*, et d'un oracle $\mathcal{R}$ de lecture. $\mathcal{M}$ ne peut pas lire directement sa bande de preuve : sa fonction de transition est indépendante du contenu de la bande de preuve. En revanche, lorsque $\mathcal{M}$ consulte l'oracle $\mathcal{R}$, le contenu de la bande d'adresses est lu sous forme d'un entier a écrit en binaire sur $\{0, 1\}$, et le caractère qui est en position a sur la bande de preuve (si présent, sinon, un caractère quelconque) est stocké dans l'état de contrôle de $\mathcal{M}$. D'autres difficultés formelles sont liées au fait d'exprimer que $\mathcal{M}$ ne consulte pas du tout y dans la première phase de calcul. Ceci est laissé en exercice.

On peut maintenant définir le modèle de calcul PCP ("probabilistic checking of proofs") :

Définition 2.9 *Soient $R(n)$, $Q(n)$, $T(n)$ trois fonctions de n. Un vérificateur (R, Q, T)-*restreint *est une TM randomisée V, à lecture directe sur une bande de preuve additionnelle, qui sur l'entrée x de taille n, une bande d'aléa r de taille $R(n)$, et la bande de preuve y de*

taille polynomiale $p(n)$,

- *calcule $k = Q(n)$ positions $p_1, \ldots, p_k$ (en binaire) à l'intérieur de y en temps polynomial, sans lire y (autrement dit, sans aucun appel à l'oracle de lecture directe sur la bande de preuve) ;*
- *lit les bits y_{p_i}, $1 \leq i \leq k$, aux positions $p_1, \ldots, p_k$ de y, et calcule un booléen $f(y_{p_1}, \ldots, y_{p_k})$ en temps $T(n)$, soit $\top$ (accepte) soit $\bot$ (rejette). (Noter que ce booléen ne dépend plus ni de x ni de r. Noter aussi que le temps de lecture est inclus dans le temps de calcul, mais pas le temps de calcul des adresses.)*

On définit au total $V(x, y, r) = f(y_{p_1}, \ldots, y_{p_k})$.

La classe $\mathbf{PCP}(R(n), Q(n), T(n))$ *est la classe des langages L tels qu'il existe un vérificateur V (R, Q, T)-restreint tel que :*

- *si $x \in L$, alors il existe y de taille $p(n)$ tel que $Pr_r[V(x, y, r) = \bot] = 0$;*
- *si $x \notin L$, alors pour tout y de taille $p(n)$, $Pr_r[V(x, y, r) = \top] \leq 1/2$.*

Dans les deux cas, la probabilité est prise sur tous les contenus r de longueur $\ell(n)$ de la bande d'aléa.

On notera que la condition d'acceptation est celle de **coRP**, à part pour les quantifications sur les preuves y. En clair, si $x \in L$, alors il existe une preuve y de x qui sera acceptée à coup sûr par V. Si $x \notin L$, alors quelle que soit la "preuve" y qu'un escroc essaierait d'utiliser pour justifier $x \in L$, le vérificateur V détectera la supercherie au moins une fois sur deux.

En ce qui concerne le temps de calcul, on notera d'abord que nous avons borné séparément les temps de calcul de la première phase (calcul des positions, en temps polynomial) et de la deuxième phase (calcul de la fonction booléenne f, en temps $T(n)$). Dans la suite, on s'intéressera essentiellement au cas où $Q(n)$ et $T(n)$ sont des fonctions constantes : f sera alors une simple fonction booléenne d'un nombre fixe, constant d'entrées.

Lemme 2.10 *Pour tous polynômes $Q(n)$, $T(n)$,* $\mathbf{PCP}(k_1 . \log_2 n, Q(n), T(n)) \subseteq \mathbf{NP}$.

Démonstration. Soit L un langage de $\mathbf{PCP}(k_1 . \log_2 n, Q(n), T(n))$. Par définition $x \in L$ si et seulement s'il existe une preuve y de taille $p(n)$ tel que $Pr[f(\vec{y}) = \bot] = 0$, où $\vec{y} = (y_{p_1}, \cdots, y_{p_{Q(n)}})$ est un vecteur de bits pris au hasard dans la bande y. Mais on peut calculer $Pr[f(\vec{y}) = \bot]$ en temps polynomial en énumérant tous les r possibles (il y en a au plus $2^{k_1 . \log_2 n} = n^{k_1}$), et en calculant pour chacun les positions $Q(n)$ valeurs de positions p_1, …, $p_{Q(n)}$ en temps polynomial, puis en vérifiant que $f(y_{p_1}, \ldots, y_{p_k}) = \top$ en temps $T(n)$. □

Nous allons démontrer :

Proposition 2.11 *Pour tout ϵ, $0 \leq \epsilon < 1$, soit MAX3SAT(ϵ) le langage suivant.*
ENTRÉE : un ensemble de 3-clauses propositionnelles S.
tel que soit S est satisfiable, soit une proportion d'au plus $1 - \epsilon$ des clauses de S peut être satisfaite en même temps.
SORTIE : S est-il satisfiable ?

Les propositions suivantes sont équivalentes :

1. *Il existe ϵ pour lequel MAX3SAT (ϵ) est* **NP***-difficile.*

2. $\mathbf{NP} = \bigcup_{k_1,k_2,k_3 \in \mathbb{N}} \mathbf{PCP}(k_1.\log n, k_2, k_3)$.
3. $\mathbf{NP} \subseteq \bigcup_{k_1,k_2,k_3 \in \mathbb{N}} \mathbf{PCP}(k_1.\log n, k_2, k_3 T(n))$ *pour une certaine fonction* T.

Notons que MAX3SAT(ϵ) n'est pas MAX3SAT$_\epsilon$, qui serait la version 3SAT du problème MAXSAT$_\epsilon$ mentionné antérieurement. Ici, on sait déjà que S est soit satisfiable, soit loin d'être satisfiable.

Démonstration. $1 \Rightarrow 2$. Par le lemme 2.10, $\mathbf{PCP}(k_1.\log_2 n, k_2, k_3) \subseteq \mathbf{NP}$. Réciproquement, $\bigcup_{k_1,k_2,k_3 \in \mathbb{N}} \mathbf{PCP}(k_1.\log_2 n, k_2, k_3)$ est clairement stable par réductions en temps polynomial, donc il suffit de montrer qu'un certain problème **NP**-difficile est dans $\mathbf{PCP}(k_1.\log n, k_2, k_3)$ pour certains entiers k_1, k_2, k_3. Par hypothèse, MAX3SAT(ϵ) est **NP**-difficile. Soit S une instance de $L =$ MAX3SAT(ϵ). Décrivons un vérificateur PCP décidant si S est satisfiable. Le vérificateur devine une preuve y de la satisfiabilité de S sous forme d'une affectation de valeurs de vérité aux variables de S : une liste de booléens, indexée par les numéros de variables. Le vérificateur opère ensuite en les deux phases précisées par la définition 2.9 :

— (Phase avant lecture de y.) Le vérificateur V tire au hasard l'une des clauses C qui s'y trouve, par exemple $+x_3 \vee -x_5 \vee -x_{22}$. Il stocke les signes (ici, $+$, $-$, et $-$) dans son état interne. Les positions p_1, p_2 et p_3 à lire sont celles données par les numéros des variables de C (ici, les positions 3, 5, et 22).

— (Phase de vérification.) V consulte les 3 bits de la preuve y correspondant aux valeurs de vérité devinées des variables présentes dans C (les bits numéros 3, 5, et 22 dans notre exemple), en temps constant. En utilisant les signes stockés dans son état interne, V décide si C est vraie ou non dans l'affectation y, encore en temps constant. (Dans notre exemple, V décide si le bit numéro 3 est vrai, ou bien si le bit 5 est faux, ou bien si le bit 22 est faux.)

Si $S \in L$, alors on peut prendre pour y une affectation satisfaisant S, et V accepte alors toujours. Si $S \notin L$, par définition de MAX3SAT (ϵ), pour tout y il y a une proportion d'au moins ϵ des clauses de S dont tous les littéraux seront faux dans l'affectation y. Autrement dit, la probabilité que V rejette est d'au moins ϵ. La probabilité que V accepte est donc d'au plus $1 - \epsilon$. En répétant l'expérience au moins $-\log 2/\log(1-\epsilon)$ fois, on peut abaisser cette probabilité à $1/2$. Notons que le nombre de fois où l'expérience doit être répétée est une constante, donc le temps total de vérification est toujours constant. Pour que la première phase soit toujours non adaptative, nous devons paralléliser les expériences : tirer au hasard non pas une clause mais $-\log 2/\log(1-\epsilon)$ clauses en phase un, puis consulter les $-\log 2/\log(1-\epsilon) \times 3$ bits concernés en phase 2.

$2 \Rightarrow 3$: trivial.

$3 \Rightarrow 1$. Supposons $\mathbf{NP} \subseteq \bigcup_{k_1,k_2,k_3 \in \mathbb{N}} \mathbf{PCP}(k_1.\log_2 n, k_2, k_3.T(n))$. Donc SAT est dans la classe $\mathbf{PCP}(k_1.\log_2 n, k_2, T(n))$ pour certains entiers k_1, k_2 et une certaine fonction $T(n)$. Soit V un vérificateur de SAT correspondant. Montrons alors que MAX3SAT (ϵ) est **NP**-difficile pour un certain $\epsilon > 0$. Soit S une instance de SAT, de taille n. Pour chaque valeur r de la bande d'aléa (de taille $k_1.\log_2 n$), le résultat $V(S, y, r)$ est une fonction $f(y_{p_1}, \ldots, y_{p_{k_2}})$ des k_2 bits lus sur la bande y aux positions $p_1, \ldots, p_{k_2}$ déterminées en fonction de S et de r. Dans la suite, nous noterons $p_i(r)$ la position p_i, pour mettre en évidence sa dépendance à r (mais elles dépendent aussi de x.)

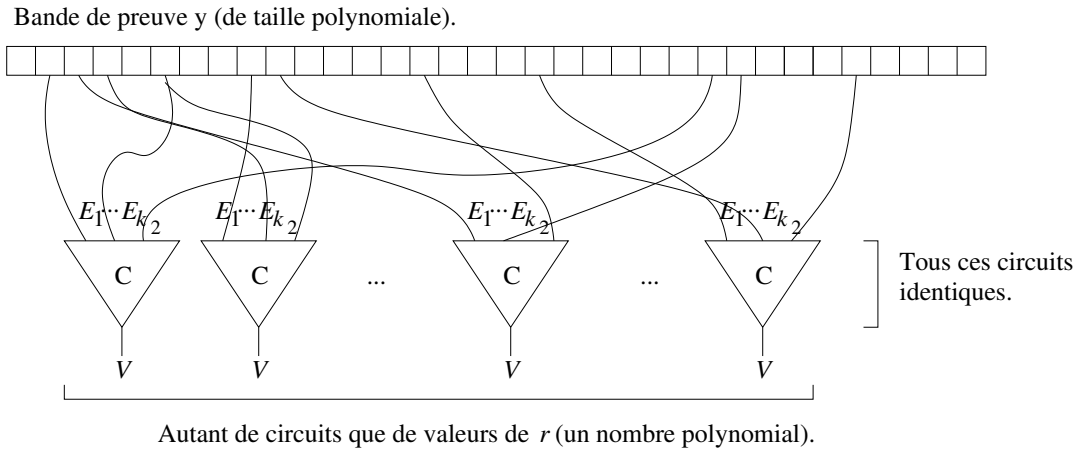

FIGURE 3 – La construction du $3 \Rightarrow 1$ de la proposition 2.11

Ce qui est important, c'est que f est une fonction booléenne de k_2 bits, et est donc représentable par un circuit $\mathcal{C}$ de taille *constante*. (Peu importe le temps $T(n)$ de calcul de la machine qui réalise f : il existe toujours un circuit de taille constante calculant la même fonction.)

Pour chacun des n^{k_1} bits possibles de y, on crée une variable fraîche Y_j, $1 \leq j \leq n^{k_1}$. (Ceci représente le bit inconnu à la position j de y.) Pour chaque r, on construit un ensemble de clauses S'_r comme suit. Pour chaque fil A de $\mathcal{C}$, on crée une variable propositionnelle A_r fraîche (en particulier $A_r \neq A_{r'}$ dès que $r \neq r'$). Pour chaque porte de $\mathcal{C}$, disons formée à partir d'un opérateur binaire $\oplus$, $A = B \oplus C$, S'_r contient la forme clausale de la formule $A \Leftrightarrow B \oplus C$. S'_r contient de plus, pour chaque fil d'entrée E_i du circuit $\mathcal{C}$, $1 \leq i \leq k_2$, la forme clausale de : $Y_{p_i(r)} \Leftrightarrow E_{i\,r}$; ceci exprime que l'entrée du circuit est le bit adéquat de la bande de preuve y. Finalement, S'_r contient une clause V_r, où V est le fil de sortie du circuit $\mathcal{C}$. Ceci exprime que S'_r est insatisfiable si et seulement si $V(S, y, r) = \bot$ pour tout y, autrement dit S'_r est satisfiable si et seulement s'il existe y tel que $V(S, y, r) = \top$.

S'_r est encore de taille constante, disons $a.2^{k_2}$ pour un certain $a \in \mathbb{N}$. De plus, on peut le calculer en temps polynomial.

Soit S' l'union des S'_r, lorsque r parcourt les n^{k_1} valeurs possibles. Noter qu'on n'écrit aucune clause pour les valeurs Y_j des $O(n^{k_1})$ bits utilisés de y, qui sont les seules indéterminées réelles de S'. Clairement, S' est un ensemble de 3-clauses de taille polynomiale en n. Si S est satisfiable, il existe y tel que $V(S, y, r) = \top$ pour tout r, donc S' est satisfiable. Sinon, pour chaque preuve y, comme $Pr[V(S, y, r) = \top] \leq 1/2$, au moins la moitié des S'_r est insatisfiable. Si S'_r est insatisfiable, au moins une clause de S'_r sur $a.2^{k_2}$ est fausse dans n'importe quelle affectation. Il y a donc une proportion au moins $1/(2a.2^{k_2})$ de clauses de S' qui sont fausses dans n'importe quelle affectation. S' est donc une instance de taille polynomiale de MAX3SAT (ϵ), dont la satisfiabilité est équivalente à celle de S. □

Le miracle est le théorème suivant :

Théorème 2.12 (Arora et Safra [4]) *Soit* **PCP** *la classe* $\bigcup_{k_1,k_2,k_3 \in \mathbb{N}} \mathbf{PCP}(k_1.\log n, k_2, k_3)$.

Alors $\mathbf{NP} = \mathbf{PCP}$.

Démonstration. La démonstration est longue et complexe. Les curieux liront [3, leçons 3 et 4]. Présentons simplement les idées maîtresses. Vu le lemme 2.10, et l'équivalence $2 \Leftrightarrow 3$ de la proposition 2.11, il suffit de démontrer que SAT est dans $\mathbf{PCP}(k_1.\log n, k_2, T(n))$ pour un certain polynôme $T(n)$.

L'idée fondamentale est de deviner une affectation y satisfaisant S, mais décrite sous forme d'un code correcteur d'erreurs qui corrige beaucoup d'erreurs. Soit $\mathbb{F}$ un corps fini, typiquement $\mathbb{Z}/p\mathbb{Z}$, avec p un nombre premier. Soit $h < p$, m un autre entier, choisis de telle sorte que $N = (h+1)^m$ soit supérieur ou égal au nombre de variables présentes dans S. On notera par exemple $x_1, \ldots, x_N$ ces variables. Plongeons $\{0, \ldots, h\}$ dans $\mathbb{F}$ (l'image de i, $0 \leq i \leq h$, étant typiquement i lui-même lorsque $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$), et fixons une bijection entre $\{1, \ldots, N\}$ et $\{0, \ldots, h\}^m$. Une affectation ρ de $x_1, \ldots, x_N$ vers $\{0, 1\}$ peut alors être vue comme une fonction ρ de $\{0, \ldots, h\}^m$ vers $\{0, 1\} \subseteq \mathbb{F}$. Par le théorème d'interpolation de Lagrange (étendu au cas à plusieurs variables : faire une récurrence sur m), on peut prolonger ρ en un polynôme $\hat{\rho}$ à m variables sur $\mathbb{F}$, de degré total au plus mh.

La preuve y que nous allons deviner est censée être le polynôme $\hat{\rho}$, décrit sous forme d'une table de toutes ses valeurs sur $\mathbb{F}^m$. Notons que cette représentation est très redondante, mais de taille au plus $p^m \log p$. Etant donnée une telle table y (une fonction $f : \mathbb{F}^m \to \mathbb{F}$), nous devrons ensuite vérifier que f est proche d'un polynôme $\hat{\rho}$ de degré faible (au plus mh ; ceci détectera les erreurs dans le codage de y). Etant donné $z \in \mathbb{F}^m$, nous aurons aussi besoin de calculer la valeur corrigée $\hat{\rho}(z)$. Les deux calculs devront examiner un nombre extrêmement réduit d'entrées de y.

Le test de faible degré. Formellement, on dira que $f : \mathbb{F}^m \to \mathbb{F}$ est proche (à η près) d'un polynôme P si et seulement si $Pr_z[f(z) \neq P(z)] \leq \eta$, autrement dit si et seulement si le nombre de points $z \in \mathbb{F}^m$ tels que $f(z) \neq P(z)$ est d'au plus ηp^m. Supposons que f soit proche de P à η près, et que P est de degré mh. Tirons deux points z_1 et z_2 de $\mathbb{F}^m$ au hasard, et regardons la droite passant par z_1 et z_2 dans l'espace vectoriel $\mathbb{F}^m$: c'est l'ensemble des points de la forme $z_1 + t(z_2 - z_1)$, $t \in \mathbb{F}$. La restriction de P à cette droite est un polynôme de degré mh au plus en t. Comme f est proche de P à η près, la probabilité qu'un point z de la droite soit tel que $f(z) = P(z)$ vaut au moins $1 - \eta$, donc la probabilité que f et P coïncident sur toute la droite est d'au moins $(1-\eta)^m$. Avec probabilité au moins $(1-\eta)^m$, la restriction de f à la droite sera un polynôme de degré mh au plus en t. Il se trouve que la réciproque est vraie : si l'on peut trouver des polynômes $P_{z_1,z_2}(t)$ de degré au plus mh tels que $f(z_1 + t(z_2 - z_1)) = P_{z_1,z_2}(t)$ avec probabilité au moins $1 - \eta$, la probabilité étant prise sur $z_1, z_2 \in \mathbb{F}^m$ et $t \in \mathbb{F}$, alors il existe un réel $\delta > 0$, $\delta < 1$, et un unique polynôme P à m variables tel que $f(z) = P(z)$ avec probabilité au moins $1 - \delta$, lorsque z est tiré au hasard. De plus, δ ne dépend que de ϵ et pas de m, h, ou p.

On peut donc tester si f est proche d'un polynôme P de degré mh en tirant des droites $z_1 - z_2$ au hasard, en tabulant complètement la fonction $t \mapsto f(z_1 + t(z_2 - z_1))$ et en calculant le degré du polynôme résultant en t, par interpolation de Lagrange. Ceci ne prend qu'un temps polynomial en m, qui est, à h fixé, un *logarithme* de N.

Par une extension de ces techniques, on peut de plus calculer les valeurs du polynôme P en k points de $\mathbb{F}^k$, où k est polynomial en $\log N$, en temps de nouveau polynomial en $\log N$, sur une TM randomisée qui se trompe au plus avec probabilité $2\sqrt{\eta} + mhk/p$. Ceci est la partie *correction d'erreur* : P est la forme corrigée de f.

Le vérificateur de nullité. La construction du vérificateur V aura aussi besoin de tester si, étant donné un polynôme P en m variables $Z_1, \ldots, Z_m$, de degré total au plus d, on a : $(*)$ $P(Z_1, \ldots, Z_m) = 0$ pour tous $0 \leq Z_1, \ldots, Z_m \leq h$. Comme d'habitude, on a le droit d'utiliser des tirages au hasard, mais on doit arriver à la conclusion rapidement.

Or l'on peut montrer (le théorème est dû à Babai, Fortnow, Levin et Szegedy) qu'il existe une famille de $p^{O(m)}$ polynômes $R_1, \ldots, R_{p^{O(m)}}$ à m variables sur $\mathbb{F}$, de faible degré, et calculables en temps $q^{O(m)}$, tels que si $(*)$ est faux, alors la probabilité que : $(\dagger)$ $\sum_{0 \leq v_1, \ldots, v_m \leq h} R_i(v_1, \ldots, v_m) P(v_1, \ldots, v_m) = 0$ est au plus $1/100$, lorsque R_i est tiré au hasard parmi la famille.

Comme on a pu le deviner, le polynôme P qui nous intéressera sera celui qui est la forme corrigée de la fonction f dont il était question plus haut. Or on ne saura évaluer P que sur un nombre k polynomial en $\log N$ de points distincts : bien moins que les $(h+1)^m = N$ points nécessaires à évaluer la somme $(\dagger)$.

Posons Q le polynôme $R_i P$, et soit d' son degré total. La condition $(\dagger)$ s'écrit $\sum_{0 \leq v_1, \ldots, v_m \leq h} Q(v_1, \ldots, v_m) = 0$. De façon générale, on va fournir un algorithme non déterministe décidant de $\sum_{0 \leq v_1, \ldots, v_m \leq h} Q(v_1, \ldots, v_m) = v$, pour chaque $v \in \mathbb{F}$. Pour ceci, on va deviner les tables des polynômes $Q_{i;v_1,\ldots,v_{i-1}}(Z) = \sum_{0 \leq v_{i+1}, \ldots, v_m \leq h} Q(v_1, \ldots, v_{i-1}, Z, v_{i+1}, \ldots, v_m)$. Notons que ceci est de taille au plus polynomiale en h^m. Plus précisément, on va deviner, pour chaque i, $1 \leq i \leq m$, et pour chaque i-uplet $v_1, \ldots, v_{i-1}$ de $\{0, \ldots, h\}^{i-1}$, un polynôme $Q'_{i;v_1,\ldots,v_{i-1}}$ en une variable Z, et l'on va vérifier avec un calcul randomisé que $Q'_{i;v_1,\ldots,v_{i-1}} = Q_{i;v_1,\ldots,v_{i-1}}$ et que $Q(v_1, \ldots, v_m) = v$. Ceci se fait comme suit :

— Initialement, $v := 0$;
— Tirer au hasard des valeurs $v_1, \ldots, v_m$ dans $\mathbb{F}$;
— Pour i variant de 1 à m : si $v \neq \sum_{0 \leq Z \leq h} Q'_{i;v_1,\ldots,v_{i-1}}$ alors rejeter, sinon $v := Q'_{i;v_1,\ldots,v_{i-1}}(v_i)$.
— Si $Q'_{m;v_1,\ldots,v_{m-1}}(v_m) \neq Q(v_1, \ldots, v_m)$ alors rejeter, sinon accepter.

On reverra un algorithme similaire à la section 3. Si les polynômes $Q'_{i;v_1,\ldots,v_{i-1}}$ ont été correctement devinés et sont donc égaux à $Q_{i;v_1,\ldots,v_{i-1}}$, et si $v = \sum_{0 \leq v_1 \leq h} Q_{1;}(v_1)$, alors l'algorithme ci-dessus accepte nécessairement. Supposons réciproquement que $v \neq \sum_{0 \leq v_1 \leq h} Q_{1;}(v_1)$, et montrons que l'algorithme ci-dessus rejette avec forte probabilité, quelles que soient les tables $Q'_{i;v_1,\ldots,v_{i-1}}$. S'il accepte, c'est que $v = \sum_{0 \leq Z \leq h} Q'_{1;}(Z)$, $Q'_{1;}(v_1) = \sum_{0 \leq Z \leq h} Q'_{2;v_1}(Z)$, $Q'_{2;v_1}(v_2) = \sum_{0 \leq Z \leq h} Q'_{3;v_1,v_2}(Z)$, $\ldots$, $Q'_{m-1;v_1,\ldots,v_{m-2}}(v_{m-1}) = \sum_{0 \leq Z \leq h} Q'_{m;v_1,\ldots,v_{m-1}}(Z)$, et $Q'_{m;v_1,\ldots,v_{m-1}}(v_m) = Q(v_1, \ldots, v_m)$. Soit i le premier indice tel que $Q'_{i;v_1,\ldots,v_{i-1}}$ n'ait pas été correctement deviné. Comme les polynômes utilisés sont de degré d', $Q'_{i;v_1,\ldots,v_{i-1}} - Q_{i;v_1,\ldots,v_{i-1}}$ est un polynôme non nul, qui n'a qu'au plus d' racines. La probabilité que l'on trouve quand même $Q'_{i;v_1,\ldots,v_{i-1}}(v_i) = \sum_{0 \leq Z \leq h} Q'_{i+1;v_1,\ldots,v_i}(Z) = Q_{i;v_1,\ldots,v_{i-1}}(v_i)$ est donc d'au plus d'/p. La probabilité d'échapper à la détection par le vérificateur en $m+1$ itérations est donc d'au plus $(m+1)d'/p$. Si p est suffisamment grand, l'algorithme ne se trompe qu'avec probabilité

au plus 1/2.

Au total. Pour résoudre SAT, étant donné S, on devine une fonction $f : \mathbb{F}^m \to \mathbb{F}$. En utilisant le test de faible degré, le vérificateur vérifie que f est proche d'un polynôme de degré au plus mh, donc du codage $\hat{\rho}$ d'une affectation ρ. On devine aussi les tables nécessaires au test de nullité, puis l'on vérifie à l'aide de ce test que l'interprétation donnée par ρ de S est vraie. L'idée est que l'on peut coder les opérations logiques sous forme d'opérations polymiales, une fois les booléens codés dans le corps $\mathbb{F}$, de sorte qu'une formule F à N variables soit rendue fausse si et seulement si le polynôme associé P vaut 0 lorsque toutes ses variables prennent des valeurs dans $\{0, 1\}$. On code x par le polynôme $1 - x$, la négation $\neg x$ par x, la disjonction $L_1 \vee L_2 \vee L_3$ par le produit des codages des L_i. En introduisant une variable x'_C par clause C, on code ensuite S sous forme de la somme des x'_C multipliés par le codage de C. (Par exemple.) □

Il est à noter qu'il existe une autre preuve de ce théorème, due à Dinur [9], qui montre directement que MAX3SAT (ϵ) est **NP**-difficile, en transformant des instances S de 3SAT petit à petit de sorte que si S est satisfiable alors sa transformée aussi, et sinon alors sa transformée a un rapport de nombre de clauses satisfaites sur nombre de clauses total strictement plus faible.

3 Preuves interactives

Le modèle de calcul PCP fait intervenir à la fois du non-déterminisme (existentiel) et du calcul randomisé : on devine une preuve y, puis on vérifie en utilisant des tirages aléatoires r. On peut voir le calcul randomisé comme un quantificateur, notons-le R_{p_+,p_-}, tel que $\mathsf{R}_{p_+,p_-} r \cdot P(x, r)$ est vrai si et seulement si $P(x, r)$ est vrai avec probabilité supérieure ou égale à p_+, et $\mathsf{R}_{p_+,p_-} r \cdot P(x, r)$ est faux si et seulement si $P(x, r)$ est faux avec probabilité supérieure ou égale à p_-.

En utilisant ce nouveau quantificateur, un problème de **RP** est donc un prédicat de l'entrée x qui s'exprime sous la forme $\mathsf{R}_{1/2,1} r \cdot P(x, r)$, où r est de taille polynomiale et P se calcule en temps polynomial. Un problème de **coRP** est lui de la forme $\mathsf{R}_{1,1/2} r \cdot P(x, r)$, un problème de **BPP** est de la forme $\mathsf{R}_{2/3,2/3} r \cdot P(x, r)$.

Le quantificateur R_{p_+,p_-} n'est pas très facile à manipuler (on en verra un meilleur à la section 3.1). Mais, de même que l'alternance entre quantificateurs $\exists$ et $\forall$ pouvait s'interpréter comme un jeu à deux joueurs, les alternances entre $\exists$ et R peuvent s'interpréter comme un jeu à deux joueurs ayant des rôles très dissymétriques :

- Le joueur R, aussi appelé Arthur, peut jouer en tirant une bande r au hasard, en temps polynomial.
- Le joueur $\exists$, aussi appelé Merlin (car ayant, en un sens, des pouvoirs surnaturels), joue en trouvant une preuve y, en un temps quelconque. La seule limitation est que Merlin doive fournir une preuve (supposée) y de taille polynomiale.

Babai [5] et Moran [6] ont introduit les jeux de Arthur contre Merlin, et de Merlin contre

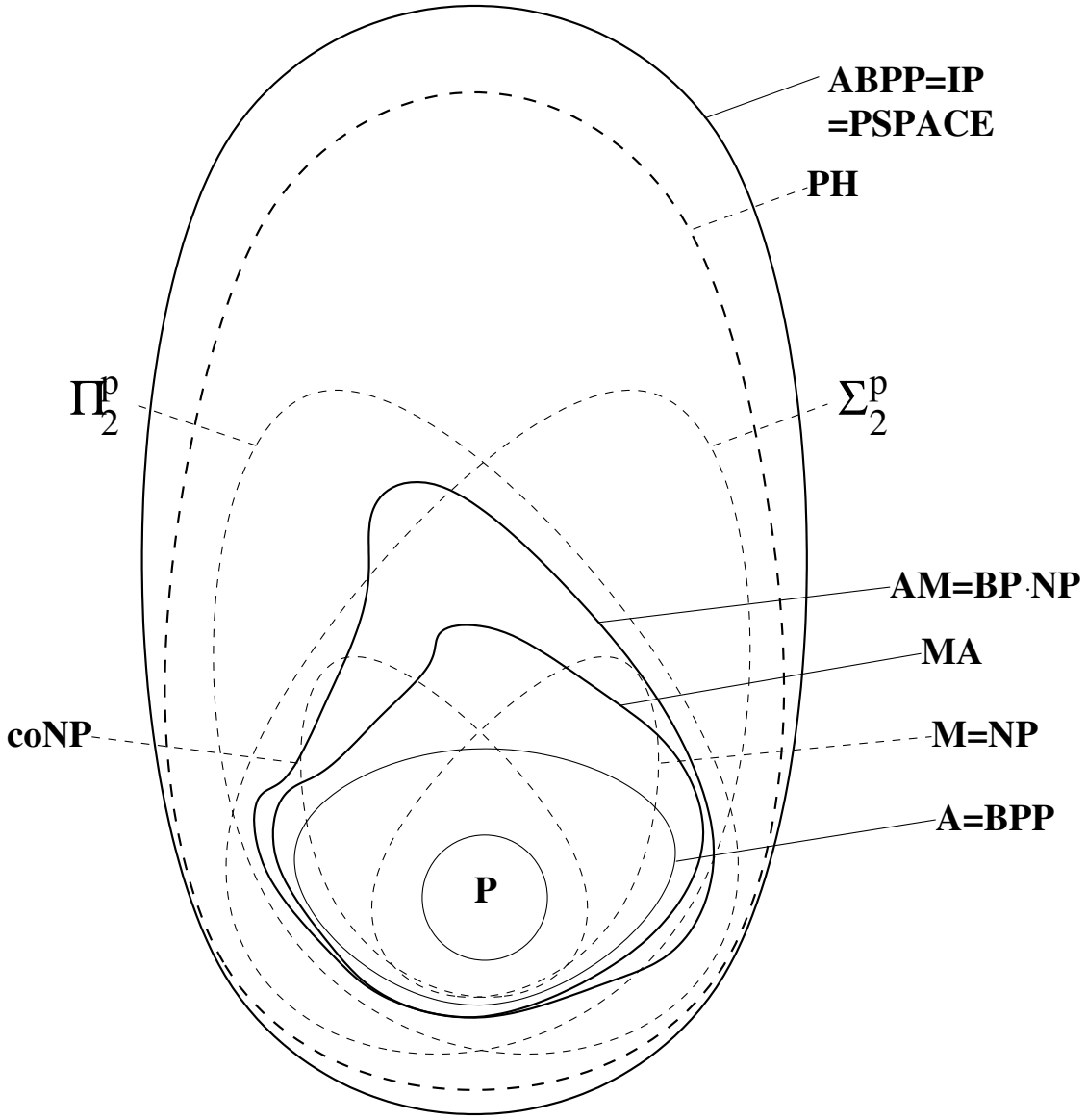

FIGURE 4 – Quelques nouvelles classes randomisées : preuves interactives

Arthur, comme formalisme permettant de créer des classes de complexité intéressantes. C'est ce formalisme qui a mené bien plus tard au théorème d'Arora.

Dans le formalisme des jeux Arthur-Merlin, Arthur et Merlin engagent une discussion pour savoir si l'entrée x est dans le langage L. Arthur commence, et calcule en temps polynomial randomisé une question $q_1 = \mathcal{A}_1(x, r_1)$ à poser à Merlin, où r_1 est l'aléa utilisé par Arthur. Merlin prétend que $x \in L$, même si ce n'est pas vrai : si x est vraiment dans M, Merlin devra pouvoir convaincre Arthur que $x \in L$, avec forte probabilité ; si x n'est pas dans L, on veut que quels que soient les efforts déployés par Merlin pour convaincre Arthur, Arthur finisse par se convaincre que Merlin est de mauvaise foi avec forte probabilité. Donc Arthur pose une question q_1 à Merlin, et selon sa réponse, va essayer de déterminer si $x \in L$. Merlin, ayant des pouvoirs surnaturels, calcule donc une réponse $y_1 = M_1(x, r_1, q_1)$, de façon déterministe mais en un temps arbitraire ; y_1 est de taille polynomiale en x. Arthur peut maintenant décider de conclure sur la base de la réponse y_1, ... ou bien poser d'autres questions. Supposons que la discussion continue : Arthur pose une question $q_2 = \mathcal{A}_2(x, r_1, q_1, y_1, r_2)$, où r_2 est un nouvel aléa de taille polynomiale en x, et Merlin trouve une réponse $y_2 = M_2(x, r_1, q_1, y_1, r_2, q_2)$. La discussion dure un certain nombre de tours, disons $k \geq 1$. Finalement, Arthur décide en fonction de x, r_1, q_1, y_1, ..., r_k, q_k, y_k, si x est dans L ou non, à l'aide d'un algorithme déterministe $\mathcal{D}$; il ne doit se tromper que rarement.

Définition 3.1 (Arthur et Merlin) *Un* jeu entre Arthur et Merlin*, est la donnée* $(M, \mathcal{A}, D)$ *d'une fonction dite* de Merlin M *(possiblement non calculable) qui à toute chaîne req associe une chaîne de taille polynomiale en celle de req ; d'une TM randomisée* $\mathcal{A}$ *en temps polynomial ; et d'un langage* $D \in \mathbf{P}$.

Pour tout mot prot $\in \{\mathtt{A}, \mathtt{M}\}^*$*, notons k le nombre d'occurrences de la lettre* `A` *dans prot. Le* protocole $prot[\mathcal{A}, M]_D$ *induit par ce jeu est le programme suivant. Pour toute entrée x de taille n, pour tout k-uplet* r_1, ..., r_k *de bandes d'aléa de tailles polynomiales en n :*

inp $:= x$ *; // contiendra l'entrée fournie à chaque joueur.*
$j := 0$ *; // nombre de fois où Arthur a déjà joué.*
for $i = 1$ *to* $|prot|$ *do*
  if $prot_i = \mathtt{A}$
    then $j := j + 1$ *;* $q := \mathcal{A}(inp, r_j)$ *;* $inp := inp\#r_j\#q$ *;*
  else /$prot_i = \mathtt{M}$ */* $y := M(inp)$ *;* $inp := inp\#y$ *;*
accepter si et seulement si $inp \in D$.

On notera $prot[\mathcal{A}, M]_D(x; r_1, \ldots, r_k) = \top$ *si et seulement si le programme ci-dessus accepte, sinon* $prot[\mathcal{A}, M]_D(x; r_1, \ldots, r_k) = \bot$.

Pour toute fonction propre f, on note $\mathbf{AM}[f]$ *la classe des langages L tels que, pour tout* $\ell \geq 0$*, il existe un jeu* $(M, \mathcal{A}, D)$ *entre Arthur et Merlin tel que pour tout x de taille n, en posant* $prot = \mathtt{AM}^k$, $k = f(n)$,

1. *si* $x \in L$*, alors* $prot[\mathcal{A}, M]_D(x; r_1, \ldots, r_k) = \top$ *avec probabilité au moins* $1 - 1/2^{n^\ell}$ *;*
2. *si* $x \notin L$*, alors* $prot[\mathcal{A}, M']_D(x; r_1, \ldots, r_k) = \bot$ *avec probabilité au moins* $1 - 1/2^{n^\ell}$*,* pour toute fonction M' de Merlin *;*

les probabilités étant prises sur toutes les suites $r_1, \ldots, r_k$.

Pour tout mot prot $\in \{\mathtt{A}, \mathtt{M}\}^*$*, contenant* k *occurrences de la lettre* **A***, on note* **prot** *la classe des langages* L *tels que, pour tout* x *de taille* n*, les conditions 1 et 2 ci-dessus sont vérifiées. Ceci définit donc en particulier les classes* **A***,* **M***,* **AM***,* **MA***, etc.*

On note encore $\mathbf{AM}[k] = \mathbf{AM}^k$ *la classe* $\mathbf{AM}[f]$ *où* f *est la fonction constante égale à* $k \geq 1$*. On note* **ABPP** *la classe* $\mathbf{AM}[poly] = \bigcup_{k\in\mathbb{N}} \mathbf{AM}[n^k]$.

On notera que Merlin et Arthur jouent avec la même machine, M ou $\mathcal{A}$, à chaque tour. Leur comportement peut cependant changer d'un tour à l'autre : en comptant le nombre de dièses dans leurs entrées respectives, ils peuvent en déduire la valeur du tour courant et agir en conséquence.

La condition d'acceptation est étrange : elle dit que si x est dans L, alors Merlin peut jouer selon la machine, dite *honnête*, M ; mais si x n'est pas dans L, alors Merlin doit jouer malhonnêtement pour espérer convaincre Arthur, et l'on demande que quelle que soit la fonction M' utilisée par Merlin pour produire les réponses y, Arthur détectera qu'il est malhonnête avec forte probabilité.

La classe **ABPP** est analogue à la classe **AP** du temps polynomial alternant, sauf qu'au lieu d'alterner quantificateurs $\exists$ et $\forall$, on alterne choix de Merlin et tirage probabiliste d'Arthur. C'est donc un analogue alternant de la classe **BPP**.

La définition 3.1 est ambiguë, au sens où nous n'avons pas précisé les polynômes majorant la taille des réponses de Merlin sur l'entrée *inp*, et majorant le temps d'exécution de $\mathcal{A}$ sur l'entrée *inp* et l'aléa r_j. Ce n'est pas un problème dans le cas de classes à nombres bornés de tours, mais c'en est un par exemple pour **ABPP**. En effet, supposons que Merlin et Arthur soient autorisés à fournir des réponses de taille quadratique en la taille de l'entrée *inp*. Sur x de taille n, le premier joueur va fournir une question ou une réponse de taille n^2, et *inp* sera de taille $O(n^2)$. Le second joueur va alors fournir une question ou une réponse de taille $O(n^2)^2 = O(n^4)$, et ainsi de suite : au $i^{\text{ième}}$ coup, l'entrée *inp* sera de taille $O(n^{2^i})$, ce qui est exponentiel en le nombre de tours. On doit donc en réalité supposer que les fonctions de Merlin $\mathcal{M}$ doivent retourner des réponses de tailles bornées par un polynôme fixé en la taille de x, qui est le préfixe de l'argument *inp* le plus long qui ne contienne pas le symbole #. On doit aussi demander que $\mathcal{A}$ s'exécute en un temps majoré par un polynôme fixé en la taille de x de nouveau, et non de toute l'entrée *inp*.

Finalement, notons que la borne d'erreur est directement définie comme étant au plus $1/2^{n^\ell}$, plutôt que $1/3$ par exemple. On pourrait en fait demander à remplacer la borne $1 - 1/2^{n^\ell}$ dans la définition par $2/3$, et ceci ne changerait pas la définition des classes $\mathbf{AM}[f]$ et **ABPP**. L'idée est que si un protocole donne la bonne réponse avec probabilité d'erreur au plus $1/3$, on peut répéter le protocole pour diminuer exponentiellement la probabilité d'erreur, comme dans le cas de **BPP** (voir le lemme 1.14). Mais ceci introduit plusieurs difficultés. D'abord, répéter le protocole change en général le nombre de tours. On doit donc effectuer une *répétition en parallèle* du protocole : par exemple, pour **AMAM**, Arthur doit tirer N aléas, poser N questions, puis Merlin doit donner les N réponses aux N premières questions ; Arthur repose N nouvelles questions, et Merlin doit donner les N réponses à ces dernières. (Techniquement, la $i^{\text{ième}}$ question et la $i$ème réponse à chaque tour forment un

tour de la $i^{\text{ième}}$ *session* du protocole parallèle.) Cependant, quand $x \notin L$, un Merlin parallèle malhonnête peut baser sa réponse dans la session i non seulement sur les questions et les réponses préalablement posées dans la session i, mais aussi dans toutes les autres sessions, ce qui rend la démonstration délicate. Dans tous les cas, la borne de Chernoff ne s'applique pas directement. Ce point est ignoré dans la plupart des publications sur le sujet. Plutôt que de l'ignorer et de commettre ainsi une erreur, j'ai préféré donner directement une définition avec borne d'erreur exponentiellement basse.

Lemme 3.2 $\mathbf{A} = \mathbf{BPP}$. $\mathbf{M} = \mathbf{NP}$. *Pour tous mots w, w' sur $\{\mathtt{A}, \mathtt{M}\}^*$,* $\mathbf{wAAw'} = \mathbf{wAw'}$ *et* $\mathbf{wMMw'} = \mathbf{wMw'}$.

C'est un exercice facile. La deuxième partie du lemme montre qu'on peut se restreindre aux jeux entre Arthur et Merlin où chacun joue en alternance. Pour les jeux à nombre fini de tours k, fixé à l'avance, on a donc a priori les classes $\mathbf{M} = \mathbf{NP}$, $\mathbf{A} = \mathbf{BPP}$, $\mathbf{MA}$, $\mathbf{AM}$, $\mathbf{MAM}$, $\mathbf{AMA}$, $\mathbf{MAMA} = \mathbf{MA}[2]$, $\mathbf{AMAM} = \mathbf{AM}[2]$, etc.

3.1 La classe $\mathbf{BP} \cdot \mathbf{NP}$, et autres présentations de AM

Considérons un protocole **AM** pour un langage L donné. Arthur tire d'abord au hasard r_1, calcule une question $q = \mathcal{A}(x, r_1)$, puis Merlin doit trouver un y, possiblement en fonction de x, r_1 et q, tel que $x\#r_1\#q\#y$ soit dans un langage en temps polynomial D. Le langage $\{z \mid \exists y \cdot z\#y \in D\}$ est alors dans **NP**. Il est donc tentant de décider L en utilisant une TM randomisée $\mathcal{A}'$ (retournant $x\#r_1\#\mathcal{A}(x, r_1)$ ici) et un langage D' dans **NP** (ici, $\{z \mid \exists y \cdot z\#y \in D\}$) tels que, si $x \in L$ alors $\mathcal{A}'(x, r_1)$ sera dans D' avec forte probabilité, et si $x \notin L$, alors $\mathcal{A}'(x, r_1)$ sera hors de D' avec forte probabilité. (Les probabilités sont calculées lorsque r_1 est tirée au hasard uniformément.)

Si $x \in L$, il existe une fonction de Merlin M telle que $\mathcal{A}'(x, r_1)\#y$ soit dans D avec probabilité au moins $2/3$, où $y = M(\mathcal{A}'(x, r_1))$. Donc $Pr_{r_1}[\mathcal{A}'(x, r_1) \in D'] = Pr_{r_1}[\exists y \cdot \mathcal{A}'(x, r_1)\#y \in D] \geq Pr_{r_1}[\mathcal{A}'(x, r_1)\#M(\mathcal{A}'(x, r_1)) \in D] \geq 2/3 : \mathcal{A}'(x, r_1)$ est bien dans D' avec forte probabilité. Réciproquement, si $x \notin L$, alors pour toute fonction de Merlin M', $Pr_{r_1}[\mathcal{A}'(x, r_1)\#M'(\mathcal{A}'(x, r_1)) \in D] \leq 1/3$. Or $Pr_{r_1}[\mathcal{A}'(x, r_1) \in D'] = Pr_{r_1}[\exists y \cdot \mathcal{A}'(x, r_1)\#y \in D] \leq Pr[\mathcal{A}'(x, r_1)\#M'(\mathcal{A}'(x, r_1)) \in D]$, où M' est n'importe quelle fonction qui à z associe un y tel que $z\#y \in D$ s'il existe, sinon retourne un y arbitraire. (L'existence d'une telle fonction est donnée par l'axiome du choix, si l'on ne veut pas se fatiguer. Mais l'on peut même voir que M' est calculable par une fonction qui énumère, en temps exponentiel, tous les y possibles.) On en déduit que $Pr_{r_1}[\mathcal{A}'(x, r_1) \in D'] \leq 1/3$. L est donc dans la classe $\mathbf{BP} \cdot \mathbf{NP}$, définie comme suit.

Définition 3.3 ($\mathbf{BP} \cdot \mathcal{C}$) *Pour toute classe de complexité $\mathcal{C}$, on note $\mathbf{BP} \cdot \mathcal{C}$ la classe des langages L tels qu'il existe une TM randomisée $\mathcal{A}'$ en temps polynomial et un langage $D' \in \mathcal{C}$ tels que, pour toute entrée x de taille n,*

— *si $x \in L$, alors $Pr_r[\mathcal{A}'(x, r) \in D'] \geq 2/3$;*
— *si $x \notin L$, alors $Pr_r[\mathcal{A}'(x, r) \notin D'] \geq 2/3$;*

la probabilité étant prise sur les aléas r de taille $q(n)$, où $q(n)$ est un polynôme en n.

La démonstration du lemme de réduction d'erreur 1.14 s'applique directement au cas étendu des classes $\mathbf{BP} \cdot \mathcal{C}$, en utilisant directement la borne de Chernoff :

Lemme 3.4 *Disons qu'une classe de complexité $\mathcal{C}$ est* démocratique *si et seulement si, pour tout $L \in \mathcal{C}$, le langage des mots $w_1 \# w_2 \# \ldots \# w_k$ ($k \geq 0$) tels que le nombre d'indices i tels que $w_i \in L$ est supérieur ou égal à $k/2$ est encore dans $\mathcal{C}$. (Ici, # est un symbole frais.)*

Si $\mathcal{C}$ est démocratique, on peut remplacer la borne $2/3$ dans la définition 3.3 par $1 - e^{-q(n)}$ pour n'importe quel polynôme $q(n)$.

L'aspect démocratique est nécessaire pour assurer que l'on peut répéter l'expérience. Il est facile de voir que **P** et **NP** sont démocratiques. Pour **NP**, il suffit de deviner un ensemble d'au moins $k/2$ indices et de vérifier que pour chacun de ces indices i, w_i est dans L.

Proposition 3.5 $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$.

Démonstration. Par l'argument précédant la définition 3.3, $\mathbf{AM} \subseteq \mathbf{BP} \cdot \mathbf{NP}$. Réciproquement, soit L un langage dans $\mathbf{BP} \cdot \mathbf{NP}$. Reprenons les notations de la définition 3.3, notamment $\mathcal{A}'$ et le langage $D' \in \mathbf{NP}$. Supposons cependant, grâce au lemme 3.4, que les deux probabilités de la condition d'acceptation soient supérieures ou égales non pas à $2/3$ mais à $1 - 1/2^{n^\ell}$, où $\ell \geq 0$ est un entier quelconque. Puisque $D' \in \mathbf{NP}$, on peut écrire $D' = \{z \mid \exists y \cdot (z, y) \in D\}$ pour un certain langage $D \in \mathbf{P}$, où y est contraint à être de taille polynomiale.

Définissons un protocole **AM** comme suit. Arthur tire r au hasard, calcule $q = \mathcal{A}'(x, r)$. Merlin répond en calculant $y = M(x \# r \# q)$, où M est une fonction qui à $x \# r \# q$ associe un y quelconque tel que $(x \# r \# q, y)$ soit dans D s'il existe, sinon retourne une chaîne quelconque de la bonne longueur.

Si $x \in L$, alors $Pr_r[\mathcal{A}'(x, r) \in D'] \geq 1 - 1/2^{n^\ell}$. Par construction, si $\mathcal{A}'(x, r) \in D'$, c'est-à-dire si $\exists y \cdot \mathcal{A}'(x, r) \# y \in D$, alors $\mathcal{A}'(x, r) \# M(x \# r \# \mathcal{A}'(x, r)) \in D$. Donc $Pr[\mathcal{A}'(x, r) \# M(x \# r \# \mathcal{A}'(x, r)) \in D] \geq 1 - 1/2^{n^\ell}$.

Si $x \notin L$, alors $Pr_r[\mathcal{A}'(x, r) \in D'] \leq 1/2^{n^\ell}$. Comme pour tout y fixé, $Pr_r[\mathcal{A}'(x, r) \# y \in D] \leq Pr_r[\exists y \cdot \mathcal{A}'(x, r) \# y \in D]$, on en déduit pour $y = M(x \# r \# \mathcal{A}'(x, r))$ que $Pr_r[\mathcal{A}'(x, r) \# M(x \# r \# \mathcal{A}'(x, r)) \in D] \leq 1/2^{n^\ell}$. □

Il est d'autre part clair que $\mathbf{BPP} = \mathbf{BP} \cdot \mathbf{P}$. La classe $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$ est donc une forme de généralisation de **BPP**, où le langage D' est dans **NP**.

On remarque aussi que, de même que **BPP** est stable par complémentaire :

Lemme 3.6 *Pour toute classe $\mathcal{C}$, notons $\mathbf{co}\mathcal{C}$ la classe des complémentaires de langages dans $\mathcal{C}$. Alors $\mathbf{co}(\mathbf{BP} \cdot \mathcal{C}) = \mathbf{BP} \cdot (\mathbf{co}\mathcal{C})$.*

La démonstration est facile, de même que celle du lemme suivant.

Lemme 3.7 *L'opérateur $\mathbf{BP} \cdot$ est monotone : si $\mathcal{C} \subseteq \mathcal{C}'$ alors $\mathbf{BP} \cdot \mathcal{C} \subseteq \mathbf{BP} \cdot \mathcal{C}'$.*

Il serait naturel de penser que chaque **A** peut être retranscrit en un opérateur **BP**, et chaque **M** en un opérateur **N** (où $\mathbf{N} \cdot \mathbf{P} = \mathbf{NP}$, définition laissée au lecteur). Donc que par exemple $\mathbf{MAM} = \mathbf{N} \cdot (\mathbf{BP} \cdot \mathbf{NP})$, ou $\mathbf{AMA} = \mathbf{BP} \cdot (\mathbf{N} \cdot \mathbf{BPP})$. C'est vrai, mais c'est loin d'être une évidence au point ou nous en sommes. (Voir le théorème 3.14 plus loin.)

Notons aussi que la diminution d'erreur pour la classe $\mathbf{BP} \cdot \mathbf{NP}$ donc aussi pour **AM**, se passe sans aucun problème, contrairement aux difficultés que nous avions annoncées plus haut pour les jeux entre Arthur et Merlin. Le lecteur sceptique est invité à montrer que les jeux **MA** avec probabilité d'erreur bornée par 1/3 peuvent être transformés en jeux **MA** avec probabilité d'erreur définie comme à la définition 3.1 par répétition parallèle ; s'il y arrive, à passer ensuite aux jeux **AMA**, pour apprécier la difficulté.

On peut caractériser les classes de jeux d'Arthur et Merlin d'une autre façon, peut-être plus simple. Comme Arthur envoie son aléa r_j à Merlin, on peut demander que ce soit Merlin qui calcule lui-même la question posée $q = \mathcal{A}(inp, r_j)$ à la ligne 5 du protocole $prot[\mathcal{A}, M]_D$ (définition 3.1). En clair, on ne restreint pas la généralité en supposant qu'Arthur ne fait *aucun* calcul (à part vérifier que $inp \in D$ à la fin du protocole), et se contente de tirer au hasard des bandes $r_1, \ldots, r_k$. Formellement :

Lemme 3.8 *On dit qu'Arthur est* paresseux *dans un jeu $(M, \mathcal{A}, D)$ si et seulement si $\mathcal{A}$ est la fonction constante qui retourne toujours la chaîne vide ϵ.*

Les classes $\mathbf{prot}_{lazy}$, $\mathbf{AM}[f]_{lazy}$, $\mathbf{ABPP}_{lazy}$ *sont définies comme les classes* **prot**, $\mathbf{AM}[f]$, **ABPP**, *à ceci près qu'Arthur y est contraint à être paresseux.*

Alors, pour tout prot $\in \{\mathtt{A}, \mathtt{M}\}^*$, $\mathbf{prot}_{lazy} = \mathbf{prot}$. $\mathbf{AM}[f]_{lazy} = \mathbf{AM}[f]$. $\mathbf{ABPP}_{lazy} = \mathbf{ABPP}$.

Démonstration. Les inclusions dans le sens $\subseteq$ sont évidentes. Montrons $\mathbf{AM}[f] \subseteq \mathbf{AM}[f]_{lazy}$. Supposons donc que L est décidé par un jeu $(M, \mathcal{A}, D)$. Définissons une fonction d'enrichissement $enr_{\mathcal{A}}$, qui à toute bande inp' d'un jeu où Arthur est paresseux ajoute les questions que poserait Arthur dans ce dernier jeu :

- Si inp' est de la forme $x\#r_1\#\#y_1\#r_2\#\#y_2\#\ldots\#r_{i-1}\#\#y_{i-1}\#r_i\#$ ("Arthur vient de jouer"), alors $enr_{\mathcal{A}}(inp') = inp\#r_i\#\mathcal{A}(inp, r_i)$, où $inp = enr_{\mathcal{A}}(x\#r_1\#\#y_1\#r_2\#\#y_2\#\ldots\#r_{i-1}\#\#y_{i-1})$.
- Si inp' est de la forme $x\#r_1\#\#y_1\#r_2\#\#y_2\#\ldots\#r_{i-1}\#\#y_{i-1}$ ("c'est à Arthur de jouer"), alors $enr_{\mathcal{A}}(inp') = enr_{\mathcal{A}}(x\#r_1\#\#y_1\#r_2\#\#y_2\#\ldots\#r_{i-1}\#)\#y_{i-1}$.
- Sinon, $enr_{\mathcal{A}}(inp')$ est arbitraire.

Ceci étant fait, considérons le jeu $(M', \mathcal{A}_0, D')$, où $\mathcal{A}_0(inp) = \epsilon$ pour tout inp (Arthur est paresseux), $M'(inp) = M(enr_{\mathcal{A}}(inp))$ (Merlin se pose maintenant toutes les questions q_j tout seul comme un grand), et où $D' = \{inp' \mid enr_{\mathcal{A}}(inp') \in D\}$. Il est clair que les deux jeux se simulent l'un l'autre, au sens où l'on peut reconstruire les bandes inp du premier jeu à partir des bandes inp' du second par $inp = enr_{\mathcal{A}}(inp')$, et reconstruire inp' à partir de inp en effaçant les questions d'Arthur dans l'autre direction. □

Sachant que l'on peut se restreindre aux Arthur qui sont paresseux, une autre caractérisation des classes de jeux entre Arthur et Merlin est donnée par une logique contenant deux quantificateurs :

Définition 3.9 (Quantificateurs E, $\exists$) *Pour toute fonction $f : X \to \mathbb{R}$ sur un ensemble fini non vide X, notons :*

— $E(f) = \mathsf{E}x \cdot f(x)$ *l'espérance de f,* $1/|X| \sum_{x \in X} f(x)$ *;*

— $\max(f) = \exists x \cdot f(x)$ *le maximum de f,* $\max_{x \in X} f(x)$*.*

Pour tout langage $D \in \mathbf{P}$, notons $D(x)$ la quantité valant 1 si $x \in D$, 0 sinon. Pour tout mot $prot \in \{\mathtt{A}, \mathtt{M}\}^*$, disons de longueur m, on définit la fonction $prot_D$ qui à toute entrée x de taille n associe

$$Q_1 x_1 \cdot Q_2 x_2 \cdot \ldots \cdot Q_m x_m \cdot D(x, x_1, x_2, \ldots, x_m) \tag{3}$$

où les variables x_1, x_2, …, x_m varient parmi les chaînes de tailles bornées par un polynôme donné en n, et $Q_i = \exists$ si la $i^{ième}$ lettre de $prot$ est $\mathtt{M}$, $Q_i = E$ si c'est $\mathtt{A}$.

La fonction $prot_D$ évalue la probabilité maximale qu'a Merlin de convaincre Arthur d'accepter. Par exemple, si $prot = \mathtt{AM}$, la formule (3) devient $\mathsf{E}x_1 \cdot \exists x_2 \cdot D(x, x_1, x_2)$. Pour chaque aléa x_1 tiré par Arthur, pour chaque réponse x_2 fournie par Merlin, $D(x, x_1, x_2)$ vaut 1 si Arthur accepte. À x_1 fixé, $\exists x_2 \cdot D(x, x_1, x_2)$ vaut donc 1 s'il existe un moyen pour Merlin de faire accepter Arthur. Finalement, $\mathsf{E}x_1 \cdot \exists x_2 \cdot D(x, x_1, x_2)$ vaut l'espérance de cette dernière quantité, c'est-à-dire exactement la probabilité maximale qu'Arthur finisse par accepter.

Proposition 3.10 (Caractérisation logique) *Pour tout langage L, L est décidé par un jeu entre Arthur et Merlin $(M, \mathcal{A}_0, D)$ à probabilité d'erreur bornée par $1/2^{n^\ell}$, où Arthur est paresseux, en suivant le protocole $prot \in \{\mathtt{A}, \mathtt{M}\}^*$, si et seulement si, pour tout x,*

— *Si $x \in L$ alors $prot_D(x) \geq 1 - 1/2^{n^\ell}$;*

— *Si $x \notin L$, alors $prot_D(x) \leq 1/2^{n^\ell}$.*

Démonstration. On observe d'abord que l'on peut permuter les quantificateurs E et $\exists$, par une forme de skolémisation :

$$\mathsf{E}x \in X \cdot \exists y \in Y \cdot F(x, y) \quad = \quad \exists f \in X \to Y \cdot \mathsf{E}x \in X \cdot F(x, f(x))$$

En effet, le membre gauche vaut $1/|X| . \sum_{x \in X} \max_{y \in Y} F(x, y)$ et le membre droit vaut $\max_{f \in X \to Y} 1/|X| . \sum_{x \in X} F(x, f(x))$. D'un côté, pour tout $x \in X$, $\max_{y \in Y} F(x, y) \geq F(x, f(x))$ pour toute fonction $f \in X \to Y$, donc le membre gauche est supérieur ou égal au membre droit. D'un autre côté, si l'on définit une fonction f qui à chaque $x \in X$ associe un y rendant $F(x, y)$ maximal, on voit que $\max_{y \in Y} F(x, y) = F(x, f(x))$ pour tout $x \in X$, donc que le membre gauche égale le membre droit.

Traitons du cas où $prot = \mathtt{AM}^k$, dans le but de simplifier les notations. En échangeant répétitivement les quantificateurs E et $\exists$, on peut ramener la formule $prot_D(x)$ à une formule de la forme :

$$\exists f_1, f_2, \ldots, f_k \cdot \mathsf{E}r_1, r_2, \ldots, r_k \cdot D(x, r_1, f_1(r_1), r_2, f_2(r_1, r_2), \ldots, r_k, f_k(r_1, \ldots, r_k)) \tag{4}$$

Dans le cas $x \in L$, dire que la formule (4) ci-dessus est supérieure ou égale à $1 - 1/2^{n^\ell}$ est équivalent à l'existence de fonctions $f_{1x}, f_{2x}, \ldots, f_{kx}$ (nous rendons explicite la dépendance

de ces fonctions en x) telles que la moyenne sur tous les aléas $r_1, r_2, \ldots, r_k$ de $D(x, r_1, f_{1x}(r_1), r_2, f_{2x}(r_1, r_2), \ldots, r_k, f_{kx}(r_1, \ldots, r_k))$ soit au moins $1 - 1/2^{n^\ell}$, c'est-à-dire telles que la probabilité qu'Arthur accepte lorsque Merlin joue $f_{jx}(r_1, \ldots, r_j)$ au tour j est au moins $1 - 1/2^{n^\ell}$. Ceci est équivalent à l'existence d'une fonction de Merlin M forçant Arthur à accepter avec probabilité au moins $1 - 1/2^{n^\ell}$: si l'on connaît un tel k-uplet $f_{1x}, f_{2x}, \ldots, f_{kx}$ pour chaque $x \in L$, il suffit de poser $M(x\#r_1\#\#y_1\#r_2\#\#\ldots\#y_{j-1}\#r_j\#) = f_{jx}(r_1, \ldots, r_j)$ lorsque $x \in L$, M retournant n'importe quoi sur les arguments qui ne sont pas de cette forme, ou avec $x \notin L$; réciproquement, si l'on connaît une fonction de Merlin M forçant Arthur à accepter avec probabilité au moins $1 - 1/2^{n^\ell}$, on définit f_{jx} par récurrence sur j par $f_{jx}(r_1, \ldots, r_j) = M(x\#r_1\#\#f_{1x}(r_1)\#r_2\#\#\ldots\#f_{j-1\,x}(r_1, \ldots, r_{j-1})\#r_j\#)$.

Lorsque $x \notin L$, la formule (4) exprime exactement le maximum, sur tous les k-uplets $f_{1x}, f_{2x}, \ldots, f_{kx}$ ou de façon équivalente sur toutes les fonctions de Merlin, de la probabilité qu'Arthur accepte. Dire que ce maximum vaut au plus $1/2^{n^\ell}$ revient donc exactement à demander que, quoi que Merlin joue, la probabilité qu'Arthur accepte est d'au plus $1/2^{n^\ell}$. □

3.2 La hiérarchie Arthur-Merlin s'effondre

De façon surprenante, la hiérarchie de Arthur et Merlin s'effondre : les seules classes **prot**, $prot \in \{\mathtt{A}, \mathtt{M}\}^*$, sont $\mathbf{M} = \mathbf{NP}$, $\mathbf{A} = \mathbf{BPP}$, **MA** et **AM**. De plus $\mathbf{M}, \mathbf{A} \subseteq \mathbf{MA} \subseteq \mathbf{AM}$, donc **AM** est la plus grande classe de la hiérarchie de Arthur et Merlin à nombre constant de tours. (Attention, ceci n'implique pas que $\mathbf{AM} = \mathbf{ABPP}$, puisque **ABPP** autorise à utiliser un nombre de tours non constant en la taille de l'entrée.)

La clé est la proposition suivante, qui permet d'échanger deux quantificateurs E et $\exists$, mais dans l'autre sens que celui permis par la skolémisation de la section précédente.

Lemme 3.11 *Soit $F(x, y, r)$ un prédicat dépendant d'un mot x de taille n, et de deux mots y et r de tailles polynomiales respectivement $p(n)$ et $q(n)$ en n. Supposons que pour tout x, $\exists y \cdot \mathsf{E}r \cdot F(x, y, r)$ est soit très grand ($\geq 1 - 1/2^n$) soit très petit ($\leq 1/2^n$), et appelons L l'ensemble des x pour lequel cette valeur est très grande.*

Soit $\ell \geq 1$. Pour $k = \lceil (p(n) + q(n) + n^\ell)/n \rceil$, et pour n suffisamment grand (précisément, dès que $p(n) + q(n) + n^\ell < n2^{n-1}$), la valeur de :

$$\mathsf{E}r_1, \ldots, r_k \cdot \exists y, r' \cdot \bigwedge_{i=1}^{k} F(x, y, r' \oplus r_i)$$

où r_1, …, r_k, r' sont de même taille que r, est très grande (vaut en fait 1) si $x \in L$ et très petite ($\leq 1/2^{n^\ell}$) si $x \notin L$.

Démonstration. C'est une variante de l'argument de Lautemann déjà utilisé pour démontrer la proposition 1.24. Fixons x de taille n, et pour tout y, posons R_y l'ensemble des r tels que $F(x, y, r)$ soit vrai.

Si $x \in L$, on montre que pour tous r_1, …, r_k, il existe y et r' tels que $F(x, y, r' \oplus r_i)$ soit vrai pour tout i, $1 \leq i \leq k$. Comme $(\exists y \cdot \mathsf{E}r \cdot F(x, y, r)) \geq 1 - 1/2^n$, il existe un y tel que

$\mathsf{E}r \cdot F(x,y,r)$, et c'est cet y que l'on choisit. Comme $\mathsf{E}r \cdot F(x,y,r)$, $Pr_r[F(x,y,r)] \geq 1-1/2^n$, donc R_y est immense : $Pr_r[r \in R_y] \geq 1 - 1/2^n$. Il ne reste qu'à montrer qu'il existe un r' tel que $F(x,y,r' \oplus r_i)$ soit vrai pour tout i entre 1 et k. Pour ceci, on calcule la probabilité sur r' que $F(x,y,r' \oplus r_i)$ soit faux pour au moins un i, et de montrer qu'elle est strictement inférieure à 1. Ceci vaut :

$$\begin{aligned} Pr_{r'}[\bigvee_{i=1}^{k} r' \oplus r_i \notin R_y] &\leq \sum_{i=1}^{k} Pr_{r'}[r' \oplus r_i \notin R_y] \\ &= \sum_{i=1}^{k} Pr_r[r \notin R_y] \leq k/2^n. \end{aligned}$$

Comme nous avons supposé $q(n) < n2^{n-1}$, $k/2^n = \lceil (p(n)+q(n)+n^\ell)/n \rceil / 2^n$ est inférieur ou égal à $1/2$, donc strictement à 1.

Si $x \notin L$, on va majorer $\mathsf{E}r_1, \ldots, r_k \cdot \exists y, r' \cdot \bigwedge_{i=1}^{k} F(x,y,r' \oplus r_i)$ directement. Ceci vaut :

$$\begin{aligned} Pr_{r_1,\cdots,r_k}[\exists y, r' \cdot \bigwedge_{i=1}^{k} F(x,y,r' \oplus r_i)] &\leq \sum_{y,r'} Pr_{r_1,\cdots,r_k}[\bigwedge_{i=1}^{k} F(x,y,r' \oplus r_i)] \\ &= \sum_{y,r'} \prod_{i=1}^{k} Pr_{r_i}[F(x,y,r' \oplus r_i)] \qquad \text{par indépendance} \\ &= \sum_{y,r'} \prod_{i=1}^{k} Pr_{r_i}[r' \oplus r_i \in R_y] \\ &= \sum_{y,r'} \prod_{i=1}^{k} Pr_r[r \in R_y] \\ &\leq 2^{p(n)} 2^{q(n)} / 2^{kn} \end{aligned}$$

Or $k = \lceil (p(n)+q(n)+n^\ell)/n \rceil$, donc $2^{kn} \geq p(n)+q(n)+n^\ell$, donc cette probabilité est inférieure ou égale à $1/2^{n^\ell}$. □

Théorème 3.12 (Babai [5]) MA ⊆ AM.

Démonstration. Soit $L \in \mathbf{MA}$. Par définition (avec $\ell = 1$), il existe un jeu entre Arthur et Merlin $(M, \mathcal{A}, D)$ tel que, pour tout x de longueur n :

— si $x \in L$, il existe un mot y tel que $\mathcal{A}(x\#y, r)$ accepte avec probabilité supérieure ou égale à $1 - 1/2^n$;

— si $x \notin L$, pour tout mot y, $\mathcal{A}(x\#y, r)$ accepte avec probabilité au plus $1/2^n$;

les mots y considérés étant d'une taille bornée par un polynôme $p(n)$, et les probabilités étant prises sur la bande d'aléa r, elle-même de taille polynomiale $q(n)$ en n. (On a ici fusionné la deuxième étape de calcul par Arthur et la décision d'être ou non dans D en un seul algorithme $\mathcal{A}$.)

Autrement dit, $x \in L$ si et seulement si $\exists y \cdot \mathsf{E}r \cdot F(x,y,r)$ est très grand, o $F(x,y,r)$ est vraie si et seulement si $\mathcal{A}(x\#y,r)$ accepte, et la formule $\exists y \cdot \mathsf{E}r \cdot F(x,y,r)$ étant très grande ou très petite au sens du lemme 3.11.

Ce lemme nous indique que, pour n assez grand, la valeur de $\mathsf{E}r_1,\ldots,r_k \cdot \exists y, r' \cdot \bigwedge_{i=1}^{k} F(x,y,r' \oplus r_i)$ vaudra 1 si $x \in L$, et sera inférieure ou égale à $1/2^{n^\ell}$ sinon. Ceci revient à dire que le protocole **AM** suivant décide L : Arthur tire au hasard non pas une bande d'aléa r, mais k bandes d'aléa $r_1, \ldots, r_k$, et les envoie directement à Merlin, sans faire d'autre calcul ; Merlin devine ensuite un couple (y, r'). Le langage D final est celui des $x\#r_1 \ldots r_k\#y\#r'$ tels que $\mathcal{A}(x\#y, r' \oplus r_i)$ accepte pour tout i, $1 \leq i \leq k$.

Une note finale. La justification du protocole dans le cas $x \in L$ nécessite que n soit suffisamment grand, disons $n \geq N$. Pour $n < N$, on peut directement tabuler toutes les entrées x de tailles inférieures à N telles que $x \in L$ et utiliser cette table dans la décision finale, ignorant l'interaction entre Arthur et Merlin. □

Lemme 3.13 $\mathbf{MAM} \subseteq \mathbf{AMM} = \mathbf{AM}$

Démonstration. Un langage $L \in \mathbf{MAM}$ est caractérisé par une formule $\exists y \cdot \mathsf{E}r \cdot F(x,y,r)$, où cette fois-ci $F(x,y,r)$ n'est plus décidable en temps polynomial mais est de la forme $\exists z \cdot G(x,y,r,z)$, avec $G(x,y,r,z)$ décidable en temps polynomial. Comme plus haut, on peut utiliser le lemme 3.11 pour remplacer $\exists y \cdot \mathsf{E}r \cdot F(x,y,r)$ par (en posant $k = \lceil m/n \rceil$, et en notant $\wedge$ la fonction minimum) :

$$\begin{aligned} & \mathsf{E}r_1,\ldots,r_k \cdot \exists y, r' \cdot \bigwedge_{i=1}^{k} F(x,y,r' \oplus r_i) \\ = \; & \mathsf{E}r_1,\ldots,r_k \cdot \exists y, r' \cdot \bigwedge_{i=1}^{k} \exists z \cdot G(x,y,r' \oplus r_i, z) \\ = \; & \mathsf{E}r_1,\ldots,r_k \cdot \exists y, r', z_1,\ldots,z_k \cdot \bigwedge_{i=1}^{k} G(x,y,r' \oplus r_i, z_i). \end{aligned}$$

□

Théorème 3.14 (Effondrement de hiérarchie [5, 6]) *Pour tout prot* $\in \{\mathtt{A},\mathtt{M}\}^*$, **prot** $\subseteq$ **AM**.

Démonstration. Au lieu de langages L, considérons plus généralement des problèmes à promesses : deux langages L^+ et L^- disjoints ; pour chaque x, dont on nous promet qu'il est dans L^+ ou dans L^-, on cherche à décider si $x \in L^+$. La généralité vient du fait qu'on ne demande pas que $L^+ \cup L^- = \Sigma^*$.

Étendons la définition des conditions d'acceptance pour les langages L aux langages à promesses. Au lieu de demander "si $x \in L$ alors (A), et si $x \notin L$ alors (B)", demandons "si $x \in L^+$ alors (A), et si $x \in L^-$ alors (B)". Ceci mène à des extensions naturelles des classes de la hiérarchie de Arthur contre Merlin, que nous noterons avec un symbole prime. Par

exemple, $\mathbf{AM}'$ est la classe des problèmes à promesses (L^+, L^-) tels que, pour tout $\ell \geq 1$, il existe $D \in \mathbf{P}$, tel que si $x \in L^+$ alors $Er \cdot \exists y \cdot (x, r, y) \in D \geq 1 - 1/2^{n^\ell}$, et si $x \in L^-$ alors $Er \cdot \exists y \cdot (x, r, y) \in D \leq 1/2^{n^\ell}$.

La construction du théorème 3.12 montre que $\mathbf{MA}' \subseteq \mathbf{AM}'$.

Nous montrons que $\mathbf{prot}' \subseteq \mathbf{AM}'$ pour tout $prot \in \{\mathtt{A}, \mathtt{M}\}^*$, par récurrence sur la longueur de $prot$. C'est clair si cette longueur est au plus 2.

Si $prot$ est de la forme $\mathtt{A}prot_2$ pour un certain mot $prot_2$, fixons un problème à promesses (L^+, L^-) dans $\mathbf{prot}'$. Nous souhaitons montrer que pour tout polynôme $q(n)$, on peut décider (L^+, L^-) dans $\mathbf{AM}'$, avec erreur au plus $1/2^{q(n)}$. Comme (L^+, L^-) est dans $\mathbf{prot}'$, et que l'on peut supposer que ceci se fait avec une erreur d'au plus $1/2^{2q(n)+2}$, on a, explicitement : si $x \in L^+$, alors $Er \cdot prot_{2\,D}(x, r) \geq 1 - 1/2^{2q(n)+2}$, et si $x \in L^-$, alors $Er \cdot prot_{2\,D}(x, r) \leq 1/2^{2q(n)+2}$. La fonction $prot_{2\,D}$ est définie en définition 3.9. Par l'inégalité de Markov appliquée à la fonction $f \colon r \mapsto 1 - prot_{2\,D}(x, r)$, si $x \in L^+$ alors la probabilité que $f(r) \geq 1/2^{q(n)+1}$ est inférieure ou égale à $Er \cdot f(r)/(1/2^{q(n)+1}) \leq 2^{q(n)+1-2q(n)-2} = 1/2^{q(n)+1}$. Donc, si $x \in L^+$, alors $prot_{2\,D}(x, r) \geq 1 - 1/2^{q(n)+1}$ avec probabilité (sur r) au moins $1 - 1/2^{q(n)+1}$. Soit L_2^+ le langage des couples (x, r) tels que $x \in L^+$ et $prot_{2\,D}(x, r) \geq 1 - 1/2^{q(n)+1}$. De façon similaire, par l'inégalité de Markov appliquée à la fonction $r \mapsto prot_{2\,D}(x, r)$, si $x \in L^-$ alors $prot_{2\,D}(x, r) \leq 1/2^{q(n)+1}$ avec probabilité (sur r) au moins $1 - 1/2^{q(n)+1}$. Soit L_2^- le langage des couples (x, r) tels que $x \in L^-$ et $prot_{2\,D(x,r)} \leq 1/2^{q(n)+1}$. Le couple (L_2^+, L_2^-) est encore un problème à promesses, et on peut lui appliquer l'hypothèse de récurrence : on peut trouver $D' \in \mathbf{P}$ tel que, si $(x, r) \in L_2^+$ alors $Er' \cdot \exists y' \cdot (x, r, r', y') \in D' \geq 1 - 1/2^{q(n)+1}$, et si $(x, r) \in L_2^-$, alors $Er' \cdot \exists y' \cdot (x, r, r', y') \in D' \leq 1/2^{q(n)+1}$. Donc, si $x \in L^+$, $Er, r' \cdot \exists y' \cdot (x, r, r', y') \in D' \geq (1 - 1/2^{q(n)+1})^2 \geq 1 - 1/2^{q(n)}$, et si $x \in L^-$, $Er, r' \cdot \exists y' \cdot (x, r, r', y') \in D' \leq 1/2^{q(n)+1} + 1/2^{q(n)+1} = 1/2^{q(n)}$.

Si $prot$ est de la forme $\mathtt{M}prot_2$, fixons un problème à promesses (L^+, L^-) dans $\mathbf{prot}'$: si $x \in L^+$, alors $\exists y \cdot prot_{2\,D}(x, y) \geq 1 - 1/2^{q(n)}$, et si $x \in L^-$, alors $\exists y \cdot prot_{2\,D}(x, y) \leq 1/2^{q(n)}$. Soit L_2^+ le langage des couples (x, y) tels que $prot_{2\,D}(x, y) \geq 1 - 1/2^{q(n)}$, L_2^- celui des couples (x, y) tels que $prot_{2\,D}(x, y) \leq 1/2^{q(n)}$. Par hypothèse de récurrence, on peut trouver $D' \in \mathbf{P}$ tel que si $(x, y) \in L_2^+$ alors $Er' \cdot \exists y' \cdot (x, r', y, y') \in D' \geq 1 - 1/2^{q(n)}$, et si $(x, y) \in L_2^-$ alors $Er' \cdot \exists y' \cdot (x, r', y, y') \in D' \leq 1/2^{q(n)}$. On en déduit que, si $x \in L^+$, alors $\exists y \cdot Er' \cdot \exists y' \cdot (x, r', y, y') \in D' \geq 1 - 1/2^{q(n)}$, et si $x \in L^-$, alors $\exists y \cdot Er' \cdot \exists y' \cdot (x, r', y, y') \in D' \leq 1/2^{q(n)}$. Donc (L^+, L^-) est dans $\mathbf{MAM}'$. Par une adaptation immédiate du théorème 3.13 aux problèmes à promesses, (L^+, L^-) est donc dans $\mathbf{AM}'$. □

À nombre *constant* de tours, le théorème 3.14 montre qu'il n'y a au plus que quatre classes de jeux entre Arthur et Merlin : $\mathbf{A} = \mathbf{BPP}, \mathbf{M} = \mathbf{NP} \subseteq \mathbf{MA} \subseteq \mathbf{AM}$. Attention, ceci ne permet pas de conclure quoi que ce soit sur les protocoles à nombre variable de tours, sauf un théorème de speedup : pour toute constante $k \in \mathbb{N}$, et toute fonction polynomiale $f \geq 1$, $\mathbf{AM}[f + k] = \mathbf{AM}[f]$. En particulier, nous ne pouvons rien en conclure de particulier sur $\mathbf{ABPP} = \mathbf{AM}[poly]$.

On notera aussi que la démonstration du théorème 3.12 établit que L est dans $\mathbf{AM}$, mais même que L est décidable par un jeu de $\mathbf{AM}$ dans lequel, si $x \in L$ alors le protocole doit accepter *toujours*. Autrement dit, le protocole ne se trompe jamais lorsqu'il rejette,

c'est-à-dire lorsqu'il affirme $x \notin L$. C'est un analogue de la classe **coRP**.

Ceci n'est pas une particularité de **MA** : on peut en général démontrer que la condition 1 dans la définition de la classe **AM** peut être remplacée par une acceptation inconditionnelle (avec probabilité 1). Ceci sera une conséquence des techniques de la section 3.3.

3.3 Les techniques de hachage universel

Les techniques de hachage universel sont dues à Carter et à Wegman [8], et ont trouvé une application étonnante en théorie de la complexité grâce à Goldwasser et Sipser [11].

Soit $\Sigma = \mathbb{Z}/2\mathbb{Z}$ l'alphabet des booléens. Il est remarquable que $\Sigma = \mathbb{Z}/2\mathbb{Z}$, muni du "ou" exclusif comme addition et du "et" logique comme multiplication, est un corps. Σ^m sera alors en particulier un espace vectoriel sur $\mathbb{Z}/2\mathbb{Z}$.

Une *fonction de hachage linéaire* h de Σ^m vers $\Sigma^{m'}$ est par définition une matrice $m' \times m$ de booléens, vus comme des éléments de $\mathbb{Z}/2\mathbb{Z}$, c'est-à-dire d'une application linéaire de $\mathbb{Z}/2\mathbb{Z}^m$ vers $\mathbb{Z}/2\mathbb{Z}^{m'}$. Appliquée à une chaîne (un vecteur) x de Σ^m, elle fournit une chaîne de $\Sigma^{m'}$. Comme la multiplication dans $\mathbb{Z}/2\mathbb{Z}$ est le et logique, et l'addition est le ou exclusif, h opère donc des calculs de parité sur les bits de x. C'est le cadre classique des codes correcteurs (linéaires).

Soit X une partie de Σ^m. Une *collision* dans X pour une fonction de hachage h est la donnée d'une entrée $x \in X$ telle qu'il existe $x' \in X$, $x' \neq x$ avec $h(x) = h(x')$. Le nom vient du problème suivant : on veut fabriquer une table qui à tout $x \in X$ associe un objet $y = f(x)$. Pour ceci, on se donne une fonction $h : \Sigma^m \to \Sigma^{m'}$ comme ci-dessus, et on alloue un tableau de $2^{m'}$ entrées ; si h n'a pas de collision dans X, pour décrire f il suffit de stocker le couple $f(x)$ à l'entrée $h(x)$ du tableau, pour chaque $x \in X$.

On peut généraliser à plusieurs fonctions de hachage. Soit $H = \{h_1, \ldots, h_\ell\}$ une famille finie de fonctions de hachage, $\ell \geq 1$. Une *collision* dans X pour H est la donnée d'une entrée $x \in X$ telle que, pour tout j, $1 \leq j \leq \ell$, il existe une autre entrée $x_j \in X$, $x_j \neq x$ telle que $h_j(x_j) = h_j(x)$.

Pour toute partie X de Σ^m, on dit que X *a une collision* pour H si et seulement s'il existe un x dans X qui est une collision pour H dans X.

Les deux lemmes suivants forment collectivement le *lemme du codage* de Sipser.

Lemme 3.15 (Sipser I) *Soit X une partie de Σ^m de cardinal au plus $2^{m'-1}$. Lorsque H est tirée au hasard uniformément parmi les familles de $\ell \geq m'$ fonctions de hachage linéaires de Σ^m vers $\Sigma^{m'}$, la probabilité que X ait une collision pour H est d'au plus $1/2^{\ell - m' + 1}$.*

Démonstration. Soit $y \in \Sigma^m$ un vecteur non nul fixé. La probabilité $P(y.z = 0)$ sur z que le produit scalaire $y.z$ soit nul, vaut $1/2$. En effet, puisque $y \neq 0$, il existe une de ses coordonnées y_i qui est non nulle, et la bijection qui à z associe $z + (0 \ldots 010 \ldots 0)$ (avec un seul 1 en position i) met clairement en correspondance les z tels que $y.z = 0$ et ceux tels que $y.z = 1$.

Lorsque x et x' sont fixés dans X, $x \neq x'$, notons $y = x - x'$. Soit h une fonction linéaire de Σ^m dans $\Sigma^{m'}$, tirée au hasard. Comme h est linéaire, $P(h \mid h(x') = h(x))$ est

la probabilité que $h(y)$ soit le vecteur nul de $\Sigma^{m'}$. En notant z_j le $j^{\text{ième}}$ vecteur ligne de h, $h(y) = 0$ si et seulement $y.z_j = 0$ pour tout j, $1 \leq j \leq m'$. Par la remarque ci-dessus, $y.z_j = 0$ avec probabilité $1/2$. Comme les vecteurs lignes de h sont tirés indépendamment, $P(h \mid h(x') = h(x)) = 1/2^{m'}$.

À x fixé, la probabilité $P(h \mid x$ collision pour h dans $X)$ est maintenant inférieure ou égale à la somme sur tous les $x' \in X$, $x' \neq x$, de $P(h \mid h(x') = h(x))$, donc à $(2^{m'-1}-1)/2^{m'} \leq 1/2$.

La probabilité $P(H \mid x$ collision pour H dans $X)$ que x soit une collision pour H dans X est le produit, pour i de 1 à ℓ, de la probabilité $P(h_i \mid x$ collision pour h_i dans $X)$ que x soit une collision pour h_i dans X, h_i étant prise au hasard. C'est donc au plus $1/2^{\ell}$.

La probabilité que X ait une collision pour H est donc au plus $1/2^{\ell}$ multiplié par le nombre d'éléments de X, soit au plus $1/2^{\ell-m'+1}$. □

Donc, si X est petit, il est facile de trouver des familles de fonctions de hachage linéaires sans collision dans X : au moins la moitié des familles convient. Si X est gros, en revanche, il n'y en a pas :

Lemme 3.16 (Sipser II) *Soit X une partie de Σ^m de cardinal strictement plus grand que $\ell.2^{m'}$, $\ell \geq m'$. Pour toute famille H de ℓ fonctions de hachage linéaires de Σ^m vers $\Sigma^{m'}$, X a une collision pour H.*

Démonstration. Soient $h_1, \ldots, h_\ell$ les éléments de H. On montre que si X n'a aucune collision pour H, alors $|X| \leq \ell.2^{m'}$. Posons, pour tout $x \in X$, $\kappa(x)$ le plus petit indice i, $1 \leq i \leq \ell$, tel que $h_i(y) \neq h_i(x)$ pour tout $y \in X$, $y \neq x$. Comme X n'a aucune collision pour H, ceci est bien défini. Posons f la fonction qui à tout $x \in X$ associe le couple $(i, h_i(x))$ avec $i = \kappa(x)$. Clairement, f est injective. Donc X a au plus autant d'éléments que l'espace d'arrivée, de cardinal $\ell.2^{m'}$. □

On peut donc en particulier tester le cardinal de X et chercher s'il existe des familles de fonctions de hachage linéaires sans collision pour X. Ce qui est intéressant, c'est que l'on pourra estimer si la taille de X est d'environ $2^{m'}$, en ne testant des familles que de m' fonctions de hachage.

Ceci s'applique immédiatement pour démontrer que le problème $\overline{\text{ISO}}$ du *non-isomorphisme de graphes* est dans **AM**. Définissons-le.

Considérons le problème suivant, appelé GRAPH-ISOMORPHISM, ou ISO en abrégé :
ENTRÉE : deux graphes orientés $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$.
QUESTION : G_1 et G_2 sont-ils isomorphes ?

Il est entendu qu'un isomorphisme de graphes est une bijection π entre V_1 et V_2 telle que $(v_1, v_1') \in E$ si et seulement si $(\pi(v_1), \pi(v_1')) \in E_2$. ISO demande donc s'il existe une telle bijection π, qui est clairement de taille polynomiale, et vérifier que π est une bijection se fait en temps polynomial. Donc ISO est dans **NP**.

On ne connaît aucun algorithme en temps polynomial pour ISO. En fait, on ne sait même pas si ISO est dans **BPP**. D'un autre côté, on ne sait pas non plus si ISO est **NP**-complet.

Considérons la négation de ce problème, appelé GRAPH-NON-ISOMORPHISM, ou en bref $\overline{\text{ISO}}$. Sans perte de généralité, on va supposer que G_1 et G_2 ont le même nombre de som-

mets et le même nombre d'arêtes. On peut même supposer que $V_1 = V_2 = \{0, 1, \ldots, N-1\}$:
ENTRÉE : deux graphes orientés $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E')$, avec $V_1 = V_2 = \{0, 1, \ldots, N-1\}$.
QUESTION : G_1 et G_2 sont-ils non isomorphes ?

On ne sait donc pas si $\overline{\text{ISO}}$ est dans **BPP**, ou s'il est **coNP**-complet. (Il est clairement dans **coNP**.) Mais :

Proposition 3.17 $\overline{\text{ISO}}$ *est dans* **AM**.

Démonstration. Soient G_1, G_2 deux graphes ayant comme sommets 0, 1, ..., $N-1$. Considérons l'ensemble X_0 des graphes G sur $\{0, 1, \ldots, N-1\}$ isomorphes à G_1 ou à G_2. Si G_1 et G_2 sont isomorphes, le cardinal $|X_0|$ de X_0 vaut $N!$, sinon il vaut $2N!$... enfin, pas tout à fait. Il se peut que G_1 par exemple ait des automorphismes non triviaux, c'est-à-dire des isomorphismes π de G_1 sur G_1 lui-même, et non triviaux, c'est-à-dire différents de l'identité. Par exemple, un graphe complet G_1 sur N sommets n'a qu'un seul graphe sur ces N sommets qui lui soit isomorphe, mais a d'un autre côté $N!$ automorphismes.

On va donc considérer l'ensemble X_0 des couples (G, φ) de graphes G qui sont isomorphes à G_1 ou à G_2, et d'automorphismes φ de G. Disons qu'un tel couple est un *graphe décoré*. Il y a exactement $N!$ graphes décorés (G, φ) tels que G soit isomorphe à un graphe donné G_1. L'argument court est le suivant : considérons l'action du groupe symétrique sur les graphes ; l'ensemble des graphes isomorphes à G_1 est l'orbite de G_1, et l'ensemble des automorphismes de G_1 est le stabilisateur de G_1, on conclut alors car on sait que l'orbite est isomorphe au quotient du groupe (ici, le groupe symétrique) par le stabilisateur. L'argument long, mais plus terre à terre, est le suivant. Pour toute permutation π de $\{0, 1, \ldots, N-1\}$, notons $\pi \cdot G$ le graphe dont les arêtes sont les $(\pi(v), \pi(w))$, lorsque (v, w) parcourt les arêtes de G. (C'est l'action du groupe symétrique mentionnée plus haut.) Un automorphisme de G est une permutation φ telle que $\varphi \cdot G = G$. Un graphe G est isomorphe à G_1 si et seulement s'il existe une permutation π telle que $G = \pi \cdot G_1$. Pour chaque graphe G isomorphe au graphe fixé G_1, fixons une permutation π_G telle que $G = \pi_G \cdot G_1$. Soit g la fonction qui à un graphe décoré (G, φ) associe $\varphi^{-1} \circ \pi_G$, et f la fonction qui à une permutation π associe $(G, \pi_G \circ \pi^{-1})$, où $G = \pi \cdot G_1$. On vérifie d'abord que $f(\pi)$ est un graphe décoré : les arêtes de $G = \pi \cdot G_1$ sont exactement les $(\pi(v), \pi(w))$, où (v, w) est une arête de G_1 ; or $(\pi(v), \pi(w))$ est une arête de G si et seulement si $(\pi_G(v), \pi_G(w))$ est une arête de $(\pi_G \circ \pi^{-1}) \cdot G = \pi_G(G_1) = G$, par définition de l'action de $\pi_G \circ \pi^{-1}$ sur G ; donc $\pi_G \circ \pi^{-1}$ est bien un automorphisme de G. Ensuite, f et g sont mutuellement inverses. D'abord, $f(g(G, \varphi)) = f(\varphi^{-1} \circ \pi_G)$ est le graphe décoré $(G', \pi_{G'} \circ (\varphi^{-1} \circ \pi_G)^{-1})$, où $G' = (\varphi^{-1} \circ \pi_G) \cdot G_1$; or $G' = \varphi^{-1} \cdot \pi_G \cdot G_1 = \varphi^{-1} \cdot G = G$, car φ donc aussi φ^{-1} est un automorphisme de G ; et comme $G' = G$, $\pi_{G'} \circ (\varphi^{-1} \circ \pi_G)^{-1} = \pi_G \circ (\varphi^{-1} \circ \pi_G)^{-1} = \varphi$. Ensuite, $g(f(\pi)) = g(G, \pi_G \circ \pi^{-1})$ (où $G = \pi \cdot G_1$) $= (\pi_G \circ \pi^{-1})^{-1} \circ \pi_G = \pi$. Le couple f, g forme donc une bijection entre l'ensemble des graphes décorés et l'ensemble des permutations de $\{0, 1, \ldots, N-1\}$. Il y a donc bien $N!$ graphes décorés.

Nous définissons donc l'ensemble X_0 des graphes décorés (G, φ) tels que G est isomorphe à G_1 ou à G_2. Cette fois-ci, X_0 est bien soit "petit" (de cardinal $N!$) soit "grand" (de cardinal $2N!$). On va utiliser le lemme de codage de Sipser pour détecter laquelle des deux

affirmations est vraie. Cependant, l'écart entre $N!$ et $2N!$ est trop faible pour qu'on puisse y arriver directement.

Utilisons l'idée de répéter des tests aléatoires en parallèle. L'ensemble $X = X_0^k$ des k-uplets de graphes décorés isomorphes à G_1 ou à G_2 est de cardinal $(N!)^k$ si G_1 et G_2 sont isomorphes, $2^k.(N!)^k$ sinon. On peut coder chaque graphe décoré par une matrice booléenne de N^2 bits (pour le graphe) plus $N(1+\log_2 N)$ bits (pour l'automorphisme, en listant ses N valeurs $\varphi(0), \ldots, \varphi(N-1)$ de taille $1+\log_2 N$). On peut donc coder les k-uplets de graphes décorés sur $k(N^2+N+N\log_2 N)$ bits. Posons donc $m = k(N^2+N+N\log_2 N)$.

Fixons un polynôme $\delta(n) \geq 0$ en n (dont nous verrons la valeur plus loin), et montrons qu'il existe deux entiers k et m' tels que $(N!)^k \leq 2^{m'-1}$ et $2^k(N!)^k > (m'+\delta(n)).2^{m'}$. Posons $k = N$, et $m' = \lceil 1+k\log_2 N!\rceil$, ce qui assurera la première inégalité. La deuxième inégalité est équivalente à $2^N(N!)^N > \lceil 1+N\log_2 N! + \delta(n)\rceil 2^{\lceil 1+N\log_2 N!\rceil}$. Or $2^{\lceil 1+N\log_2 N!\rceil} \leq 2^{2+N\log_2 N!} = 4(N!)^N$, donc cette deuxième inégalité est vraie dès que $2^{N-2} \geq 1+N\log_2 N! + \delta(n)$, ce qui est vrai dès que N est assez grand, disons $N \geq N_0$. Notons que N_0 dépend uniquement de δ.

Fixons un entier $\ell \geq 0$. Nous cherchons un algorithme avec probabilité d'erreur au plus $1/2^{n^\ell}$. Posons alors $\delta(n) = n^\ell - 1$, où $n = O(N^2)$ est la taille de l'entrée. Le N_0 ci-dessus dépend de δ, donc de ℓ, mais à chaque ℓ fixé, N_0 est une constante. Si $N \leq N_0$, Arthur peut décider directement si G_1 et G_2 sont non isomorphes en temps constant, en tabulant toutes les réponses. Si $N > N_0$, Arthur tire au hasard une famille H de $m'+\delta(n) = m'-1+n^\ell$ fonctions de hachage linéaires, et Merlin doit y trouver une collision. Noter qu'on peut vérifier que c'est effectivement une collision en temps polynomial. Noter en particulier que la taille de H est polynomiale, parce que m' est borné par un polynôme en la taille $n = O(N^2)$ de l'entrée : $m' = O(N\log N!) = O(N^2\log N)$, par la formule de Stirling.

Si G_1 et G_2 sont non isomorphes, $|X|$ est de cardinal $2^k.(N!)^k > (m'+\delta(n)).2^{m'}$, donc par le lemme 3.16, H a une collision, et Arthur doit accepter. Sinon, $|X|$ est de cardinal $(N!)^k \leq 2^{m'-1}$, donc par le lemme 3.15, H n'a aucune collision avec probabilité au moins $1/2^{n^\ell}$, donc Arthur doit refuser avec probabilité au moins $1/2^{n^\ell}$. □

Proposition 3.18 *Dans la définition de* **AM** *(définition 3.1), on peut remplacer le cas $x \in L$ de la condition d'acceptation par :*

1. *si $x \in L$, alors $prot[\mathcal{A}, M]_D(x; r_1, \ldots, r_k) = \top$,* toujours *(c'est-à-dire, il existe y de taille polynomiale tel que $\mathcal{A}(x, r_1)\#y \in D$).*

Ceci reste valable même lorsque l'on suppose Arthur paresseux.

Démonstration. D'abord, on rappelle que $\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP}$ (proposition 3.5). Si $L \in \mathbf{AM}$, il existe donc un langage $L' \in \mathbf{NP}$ tel que, sur l'entrée x de taille n :

— si $x \in L$ alors $x\#r \in L'$ avec probabilité au moins $1-1/2^n$;

— si $x \notin L$ alors $x\#r \in L'$ avec probabilité au plus $1/2^n$.

Les probabilités sont dans les deux cas prises sur tous les $r \in \Sigma^m$, où m est un polynôme $q(n)$ en n.

Décider si $x \in L$ revient donc à décider si $R_x = \{r \mid x\#r \in L'\}$ est immense (premier cas) ou ridicule (deuxième cas). Pour ceci, Arthur tire une famille H de ℓ fonctions de hachage linéaires de Σ^m vers $\Sigma^{m'}$, où $m' = m-n+1$ et $\ell = m+f(n)$ pour un certain polynôme f,

$f(n) \geq 1$. (On supposera que $q(n) \geq n$ pour tout n, de sorte que $m' \geq 1$. Sinon, on peut forcer $q(n) \geq n$ en tirant plus d'aléa que nécessaire.)

Si $x \in L$, le cardinal de R_x est d'au moins $2^m(1 - 1/2^n)$, ce qui est supérieur strictement à $\ell.2^{m'}$ dès que $2^m(1 - 1/2^n) > (m + f(n)).2^{m-n+1}$, c'est-à-dire dès que $2^{n-1}(1 - 1/2^n) > q(n) + f(n)$, ce qui arrive dès que n est assez grand. Merlin peut donc trouver une collision pour H dans R_x par le lemme 3.16.

Si $x \notin L$, le cardinal de R_x est d'au plus 2^{m-n}. Par le lemme 3.15, la probabilité que R_x ait une collision pour H est d'au plus $1/2^{\ell-m+n} = 1/2^{n+f(n)}$.

Comme L' est dans **NP**, on peut écrire $L' = \{z \mid \exists y \cdot z\#y \in D\}$, où D est dans **P**. Merlin doit deviner une collision pour H dans R_x. Ceci signifie deviner un $z \in R_x$ et ℓ éléments $z_j \in R_x$, $1 \leq j \leq \ell$, tels que $h_j(z_j) = h_j(z)$ pour tout j. Pour ceci, Merlin devine z et une (prétendue) preuve y de $z\#y \in D$, chacun des z_j et pour chacun une (prétendue) preuve y_j de $z_j\#y_j \in D$. Tout ceci est de taille polynomiale, $\ell = q(n) + f(n)$ étant polynomial en n. Finalement, Arthur vérifie que ceci est bien une collision pour H dans R_x, en vérifiant que $h_j(z_j) = h_j(z)$ pour tout j d'une part, et d'autre part que $z\#y$ et tous les $z_j\#y_j$ sont dans D. □

On en déduit immédiatement :

Théorème 3.19 $\mathbf{AM} \subseteq \Pi_2^p$.

Démonstration. Soit $L \in \mathbf{AM}$, décidé par un jeu **AM** où, si $x \in L$, alors le protocole accepte toujours. Arthur tire r au hasard, calcule $q = \mathcal{A}(x, r)$, Merlin devine $y = M(x\#r\#q)$ de sorte que, si $x \in L$, alors $x\#r\#q\#y \in D$, sinon $x\#r\#q\#y \notin D$ avec probabilité au moins 2/3. Considérons le langage $L' = \{x \mid \forall r \cdot \exists y \cdot x\#r\#\mathcal{A}(x, r)\#y \in D\}$, un langage Π_2^p. (Les quantificateurs portent sur des objets de longueurs polynomiales.) Si $x \in L$, par construction $x \in L'$. Si $x \notin L$, alors au moins 2/3 des valeurs de r possibles sont telles qu'il n'existe aucun y tel que $x\#r\#\mathcal{A}(x, r)\#y \in D$, donc $x \notin L'$. Puisque $L = L'$, on conclut. □

Donc, non seulement **BPP**, mais aussi **MA** et **AM** sont inclus dans Π_2^p.

La proposition 3.17 entraîne qu'il est improbable qu'ISO soit **NP**-complet. En fait, si c'était le cas, alors la hiérarchie polynomiale s'effondrerait au niveau 2.

Théorème 3.20 (Boppana, Håstad, et Zachos [7]) *Si* $\mathbf{coNP} \subseteq \mathbf{AM}$*, alors la hiérarchie polynomiale s'effondre au niveau* 2*, sur* **AM** *:* $\mathbf{PH} = \mathbf{AM} = \Pi_2^p = \Sigma_2^p$.

Démonstration. Supposons $\mathbf{coNP} \subseteq \mathbf{AM}$. Montrons que $\Sigma_2^p \subseteq \mathbf{AM}$. En effet, tout langage L de Σ_2^p est de la forme $\{x \mid \exists y \cdot (x, y) \in L'\}$, avec $L' \in \mathbf{coNP} \subseteq \mathbf{AM}$. On peut décider $x \in L$ en laissant Merlin deviner y, puis en s'engageant dans un protocole **AM** : si $x \in L$, alors Merlin peut trouver un tel y, sinon il se fera repérer comme malhonnête avec forte probabilité. Donc L est dans $\mathbf{MAM} = \mathbf{AM}$.

Plus précisément, en utilisant la définition 3.9, il existe une formule $Er \cdot \exists y' \cdot (x, y, r, y') \in D$, où $D \in \mathbf{P}$, caractérisant L', au sens où si $(x, y) \in L'$ alors cette formule a une valeur très proche de 1 pour au moins un y, et sinon elle a une valeur très proche de 0 pour tout y. Alors L est caractérisé par la formule $\exists y \cdot Er \cdot \exists y' \cdot (x, y, r, y') \in D$ (le $\exists y$ est un max), au

sens où la valeur de cette formule est très proche de 1 si $x \in L$, et très proche de 0 sinon. Donc $L \in \mathbf{MAM} = \mathbf{AM}$.

En utilisant le théorème 3.19, $\Sigma_2^p \subseteq \mathbf{AM} \subseteq \Pi_2^p$. De $\Sigma_2^p \subseteq \Pi_2^p$ on déduit $\Pi_2^p \subseteq \Sigma_2^p$: tout langage L de Π_2^p est le complémentaire d'un langage $\overline{L}$ de Σ_2^p, et comme $\overline{L} \in \Sigma_2^p \subseteq \Pi_2^p$, L est dans Σ_2^p. Donc $\Pi_2^p = \mathbf{AM} = \Sigma_2^p$. Le fait que ces classes soit aussi égales à **PH** s'en déduit facilement, en montrant que $\Sigma_k^p \subseteq \Sigma_2^p$ par récurrence sur $k \geq 2$. □

Corollaire 3.21 *Si ISO est* **NP***-complet alors la hiérarchie polynomiale s'effondre au niveau* 2*.*

Démonstration. Si ISO était **NP**-complet, alors $\overline{\text{ISO}}$ serait **coNP**-complet. Comme $\overline{\text{ISO}}$ est dans **AM**, et **AM** est stable par réductions polynomiales, on aurait **coNP** $\subseteq$ **AM**. □

3.4 Preuves interactives, la classe IP

Dans le cadre des jeux entre Arthur et Merlin, Arthur peut être paresseux, et ne faire aucun calcul. En un sens, c'est dommage : Arthur n'a aucun rôle. Pour qu'Arthur ait un rôle décisif, il faudrait qu'Arthur ne divulgue pas ses bits aléatoires r_j à Merlin, et les garde secrets. On aboutit alors à une variante légère des jeux entre Arthur et Merlin, dite des *preuves interactives* (sous-entendu, que $x \in L$).

Définition 3.22 (IP) *Pour tout mot prot* $\in \{\mathtt{A}, \mathtt{M}\}^*$*, notons k le nombre d'occurrences de la lettre* A *dans prot. Le* protocole $prot[\mathcal{A}, M]_D^{secret}$ *induit par ce jeu est le programme suivant. Pour toute entrée x de taille n, pour tout k-uplet* $r_1, \ldots, r_k$ *de bandes d'aléa de tailles polynomiales en n :*

public $:= x$ *; // contiendra l'entrée fournie à Merlin, sans les bits secrets.*
secret $:= \epsilon$ *; // contiendra les bits secrets d'Arthur.*
$j := 0$ *; // nombre de fois où Arthur a déjà joué.*
for $i = 1$ *to* $|prot|$ *do*
  if $prot_i = \mathtt{A}$
    then $j := j + 1$ *;* $q := \mathcal{A}(public, secret, r_j)$ *;* $public := public\#q$ *;* $secret := secret\#r_j$ *;*
      // note : r_j *n'est* pas *ajouté à public.*
  *else /*** $prot_i = \mathtt{M}$ ***/* $y := M(public)$ *;* $public := public\#y$ *;*
accepter si et seulement si $(public, secret) \in D$*.*

On notera $prot[\mathcal{A}, M]_D^{secret}(x; r_1, \ldots, r_k) = \top$ *si et seulement si le programme ci-dessus accepte, sinon* $prot[\mathcal{A}, M]_D^{secret}(x; r_1, \ldots, r_k) = \bot$*.*

Pour toute fonction propre f, on note $\mathbf{IP}[f]$ *la classe des langages L tels que pour tout* $\ell \geq 0$*, il existe un jeu* $(M, \mathcal{A}, D)$ *entre Arthur et Merlin tel que pour tout x de taille n, en posant* $prot = \mathtt{AM}^k$*,* $k = f(n)$*,*

1. *si* $x \in L$*, alors* $prot[\mathcal{A}, M]_D^{secret}(x; r_1, \ldots, r_k) = \top$ *avec probabilité au moins* $1 - 1/2^{n^\ell}$ *;*
2. *si* $x \notin L$*, alors* $prot[\mathcal{A}, M']_D^{secret}(x; r_1, \ldots, r_k) = \bot$ *avec probabilité au moins* $1 - 1/2^{n^\ell}$*,* pour toute fonction M' de Merlin *;*

les probabilités étant prises sur toutes les suites $r_1, \ldots, r_k$.

On note $\mathbf{IP}[k]$ *la classe* $\mathbf{IP}[f]$ *où* f *est la fonction constante égale à* $k \geq 1$. *On note* $\mathbf{IP}$ *la classe* $\mathbf{IP}[poly] = \bigcup_{k\in\mathbb{N}} \mathbf{IP}[n^k]$.

Notons au passage une légère incohérence dans les notations : alors que $\mathbf{AM}$ dénotait $\mathbf{AM}[1]$, et que $\mathbf{AM}[poly]$ était noté $\mathbf{ABPP}$, ici $\mathbf{IP}$ dénote $\mathbf{IP}[poly]$, et $\mathbf{IP}[1]$ n'a aucune autre notation.

La tradition veut que, dans le cadre de $\mathbf{IP}$, on appelle Merlin le *prouveur* et Arthur le *vérificateur*. Merlin fournit en effet des preuves y, qu'Arthur cherche à vérifier, avec des moyens calculatoires limités. La dénomination vient du monde de la cryptographie, et en particulier des protocoles cryptographiques.

Donnons un exemple canonique de ce que l'on peut faire dans $\mathbf{IP}[k]$, et que l'on ne sait pas faire en temps polynomial, ni même dans $\mathbf{BPP}$. C'est notre ami le problème du non-isomorphisme de graphes :

Proposition 3.23 $\overline{\text{ISO}}$ *est dans* $\mathbf{IP}[1]$.

Démonstration. Considérons le protocole suivant. Arthur tire un entier $i \in \{1, 2\}$ au hasard (c'est-à-dire juste un bit), et une permutation π aléatoire de $\{0, 1, \ldots, N-1\}$. C'est son secret. Il fabrique le graphe permuté $\pi(G_i)$, c'est-à-dire le graphe dont l'ensemble des sommets est encore $\{0, 1, \ldots, N-1\}$, mais dont les arcs sont les $(\pi(v_i), \pi(v_i'))$, lorsque (v_i, v_i') parcourt l'ensemble des arcs E_i. Il envoie à Merlin le graphe $\pi(G_i)$ (c'est la partie publique), et Merlin doit deviner à quel graphe, G_1 ou G_2, $\pi(G_i)$ est isomorphe. Merlin renvoie donc un entier dans $\{1, 2\}$. On accepte si et seulement si cet entier égale i.

Si G_1 et G_2 ne sont pas isomorphes, $\pi(G_i)$ est isomorphe à G_i mais pas à l'autre graphe. Merlin, en donnant la bonne réponse, force l'acceptation du protocole. Si G_1 et G_2 sont isomorphes, Merlin a exactement une chance sur deux de trouver la bonne valeur de i.

La probabilité d'erreur dans le cas où l'entrée n'est pas dans $\overline{\text{ISO}}$ est bornée par $1/2$, pas encore par $1/2^{n^\ell}$. On ne peut pas appliquer de théorème de réduction d'erreur pour les problèmes de $\mathbf{IP}[k]$ en général, mais on peut le faire dans ce cas particulier. On corrige le protocole ci-dessus comme suit : Arthur tire maintenant N entiers $i_1, \ldots, i_N \in \{1, 2\}$ au hasard, N permutations alératoires π_1, ..., π_N, et fournit $\pi_1(G_{i_1})$, ..., $\pi_N(G_{i_N})$ à Merlin. Merlin retourne ensuite une liste d'entiers $j_1, \ldots, j_N$. Arthur accepte enfin si et seulement si $i_1 = j_1$ et ... et $i_N = j_N$. Lorsque G_1 et G_2 sont isomorphes, Arthur accepte de façon erronée avec probabilité au plus $1/2^N$. On conclut en prenant $N \geq n^\ell$. □

Cette construction a été trouvée avant celle de la proposition 3.17, et est essentiellement inutile : nous allons démontrer tout de suite que $\mathbf{AM} \subseteq \mathbf{IP}[1]$, la proposition 3.17 est donc plus forte. Mais l'algorithme de la proposition 3.23 reste intéressant : il est plus simple, et de plus il est "zero-knowledge", c'est-à-dire que si G_1 et G_2 sont en fait isomorphes, Arthur en sera convaincu, sans avoir la moindre idée de l'isomorphisme en question. (Mais ceci commence à nous entraîner un peu trop loin.)

A priori, le fait que Merlin n'ait plus accès aux bits aléatoires d'Arthur (à moins qu'Arthur ne décide de les divulguer, bien entendu) devrait faire de $\mathbf{IP}[f]$ une classe plus grande que $\mathbf{AM}[f]$, pour toute fonction f :

Lemme 3.24 *Pour toute fonction propre f,* $\mathbf{AM}[f] \subseteq \mathbf{IP}[f]$. $\mathbf{ABPP} \subseteq \mathbf{IP}$.

Démonstration. Tout protocole entre Arthur et Merlin en $f(n)$ tours, où Arthur joue en posant des questions $q = \mathcal{A}(inp, r_j)$, peut être simulé par un protocole **IP** où Arthur pose la question $\mathcal{A}(public, secret, r_j)\#r_j$, forçant *public* dans le protocole **IP** à être égal à *inp* dans le protocole **AM**. □

Ce qui est beaucoup moins évident, c'est que les protocoles **IP** à nombre constant de tours définissent exactement les mêmes langages que les protocoles entre Arthur et Merlin à nombre constant de tours. En clair, contrairement à l'intuition, garder des secrets n'offre essentiellement aucun avantage à Merlin — du moment que le nombre de tours est constant. (On verra plus loin que ceci n'offre pas non plus d'avantage à Arthur si l'on a un nombre polynomial de tours.)

En fait, on peut déduire que $\overline{\text{ISO}}$ est dans **AM** en montrant le théorème plus général ci-dessous. Il est dû à Goldwasser et Sipser. Sa démonstration est relativement technique, et repose aussi sur le lemme du codage de Sipser. Sachant que $\overline{\text{ISO}}$ est dans $\mathbf{IP}[1]$, on en déduit que $\overline{\text{ISO}}$ est dans $\mathbf{AM}[3]$. Or $\mathbf{AM}[3] = \mathbf{AM}$ par le théorème 3.14.

Théorème 3.25 (Goldwasser et Sipser [11]) *Pour tout $k \geq 1$,* $\mathbf{IP}[k] \subseteq \mathbf{AM}[k+1]$.

Démonstration. La démonstration est complexe, et nous nous contenterons de montrer que $\mathbf{IP}[1] \subseteq \mathbf{AM}[2]$, qui contient l'essentiel de l'argument.

Soit donc $L \in \mathbf{IP}[1]$. Sur l'entrée x de taille n, Arthur tire un aléa r, de taille $p(n)$, calcule une question $q = \mathcal{A}(x, \epsilon, r)$ de taille $q(n)$. Merlin, en utilisant la fonction de Merlin M, calcule la réponse $y = M(x\#q)$. Arthur décide enfin si $(x\#q\#y, x\#r) \in D$. Ce doit être le cas si $x \in L$ avec probabilité au moins $1 - 1/2^{n^\ell}$, et ce doit être vrai si $x \notin L$ avec probabilité au plus $1/2^{n^\ell}$.

Pour simplifier, on va supposer que Merlin joue avec une fonction de Merlin unique, que $x \in L$ ou $x \notin L$, mais qui maximise la probabilité qu'Arthur accepte.

On va essayer de compter le nombre d'aléas r qui font accepter Arthur à la fin du protocole. Ce nombre est soit énorme soit ridicule, et l'on pourra le décider en utilisant le lemme du codage de Sipser comme plus haut.

La difficulté, c'est qu'il est possible que deux aléas distincts r et r' puissent donner lieu à la même question q d'Arthur. Or, puisque Merlin ne connaît que la question et pas r ni r', il doit alors fabriquer la même réponse y.

Définissons donc une relation d'équivalence $\sim$ entre aléas par : $r \sim r'$ si et seulement si ils donnent lieu à la même question, autrement dit $\mathcal{A}(x, \epsilon, r) = \mathcal{A}(x, \epsilon, r')$. Soit Q l'ensemble des questions q qu'Arthur peut poser, c'est-à-dire celles telles qu'il existe r avec $q = \mathcal{A}(x, \epsilon, r)$. Chaque question $q \in Q$ détermine une classe d'équivalence $R_q = \{r \mid \mathcal{A}(x, \epsilon, r) = q\}$ pour $\sim$. Définissons le *poids* $w(q)$ comme étant le nombre d'éléments $r \in R_q$ tels qu'Arthur accepte, c'est-à-dire tels que $(x\#q\#y, x\#r) \in D$, où $y = M(x\#q)$.

Le poids $w(q)$ de la question $q \in Q$ peut varier a priori de 1 à $2^{p(n)}$. On peut classer les questions selon qu'elles ont un poids élevé ou faible. On va montrer que, si $x \in L$, il y a nécessairement beaucoup de questions q de forts poids. (C'est l'essentiel de l'argument.) Par

hypothèse, le nombre N d'aléas r tels qu'Arthur accepte à la fin vaut au moins $2^{p(n)}(1 - 1/2^{n^\ell})$. D'autre part, par définition, $N = \sum_{q \in Q} w(q)$. Séparons la somme en rangeant les questions q par grandes catégories de tailles de $w(q)$. Disons que q est dans la *catégorie numéro* j si et seulement si $2^j \leq w(q) < 2^{j+1}$. Notons Q_j les questions de Q de catégorie j. Alors $N = \sum_{j=0}^{p(n)} \sum_{q \in Q_j} w(q)$. Comme $N \geq 2^{p(n)}(1 - 1/2^{n^\ell})$, l'un des termes $\sum_{q \in Q_j} w(q)$ est supérieur ou égal à $2^{p(n)}/(p(n)+1)(1 - 1/2^{n^\ell})$. Mais lorsque $q \in Q_j$, $w(q)$ vaut au plus 2^{j+1}, donc la somme $\sum_{q \in Q_j} w(q)$ contient au moins $2^{p(n)}/(p(n)+1)(1 - 1/2^{n^\ell})/2^{j+1}$ termes. D'autre part, $w(q)$ vaut au moins 2^j. Il y a donc au moins $2^{p(n)}/(p(n)+1)(1 - 1/2^{n^\ell})/2^{j+1}$ questions q de poids au moins 2^j, pour un certain j, dès que $x \in L$. (Et réciproquement, s'il y a au moins $2^{p(n)}/(p(n)+1)(1 - 1/2^{n^\ell})/2^{j+1}$ questions q de poids au moins 2^j, alors $N \geq 2^{p(n)}/(2(p(n)+1))(1 - 1/2^{n^\ell})$, ce qui implique $x \in L$, pour n assez grand : si, en effet, x n'était pas dans L, c'est que $2^{p(n)}/(2(p(n)+1))(1 - 1/2^{n^\ell}) \leq N \leq 1/2^{n^\ell}$, ce qui est asymptotiquement faux.)

À l'opposé, si $x \notin L$, alors toutes les questions sont de poids au plus $2^{p(n)}/2^{n^\ell}$, ce qui est bien plus petit que $2^{p(n)}/(2(p(n)+1))(1 - 1/2^{n^\ell})/2^{j+1}$ pour tout $j \leq p(n)$, dès que n est suffisamment grand.

On ne connaît pas j, et l'on pourrait demander à Merlin de le deviner. Comme il n'y a pas beaucoup de valeurs de j possibles (au plus $p(n)+1$), on va lancer $p(n)+1$ instances en parallèle du protocole à définir ci-dessous, une pour chaque valeur de j. Chaque instance décidera s'il y a au moins $2^{p(n)}/(2(p(n)+1))(1 - 1/2^{n^\ell})/2^{j+1}$ questions q de poids au moins 2^j. Finalement, Arthur acceptera si l'une des instances accepte.

Supposons donc j fixé, $0 \leq j \leq p(n)$.

Au premier tour, Arthur et Merlin s'engagent dans un protocole fondé sur le lemme de codage de Sipser pour décider avec forte probabilité si le nombre de questions q de poids entre 2^j et 2^{j+1} est bien supérieur ou égal à $2^{p(n)}/(2(p(n)+1))(1 - 1/2^{n^\ell})/2^{j+1}$. L'espace considéré est Σ^m, avec $m = q(n)$ la taille de q, et le sous-ensemble X est celui des q tels que $w(q) \geq 2^j$. Pour ceci, Arthur tire au hasard ℓ' fonctions de hachage, $\ell' \geq p(n) - j$, et Merlin y cherche des collisions dans X, disons q, q_1, …, $q_{\ell'}$. Nous demandons aussi que Merlin fournisse ses réponses $y = M(x\#q)$, $y_1 = M(x\#q_1)$, …, $y_{\ell'} = M(x\#q_{\ell'})$ lors de ce tour. (Noter qu'Arthur ne calcule pas du tout d'aléa r tel que $q = \mathcal{A}(x, \epsilon, r)$ par exemple : sinon il serait obligé de le rendre public !)

Il reste à vérifier que ceci définit effectivement une collision dans X. Par souci d'uniformité de notation, notons $q_0 = q$ et $y_0 = y$. Que ceci soit une collision est facile à vérifier, mais nous devons encore vérifier que $q_0, q_1, \ldots, q_{\ell'}$ sont dans X, autrement dit que $w(q_k) \geq 2^j$ pour tout k. Ceci se fait lors du second tour. Vérifier que $w(q_k) \geq 2^j$, avec $y_k = M(x\#q_k)$ fourni au tour précédent, c'est vérifier que le nombre d'aléas r tels que $\mathcal{A}(x, \epsilon, r) = q_k$ et $(x\#q_k\#y_k, x\#r) \in D$ est d'au moins 2^j. Pour ceci, Arthur tire $\ell'+1$ familles de $\ell'' \geq j$ fonctions de hachage (une famille pour chaque q_k, y_k), et Merlin doit deviner une collision dans chaque famille. Cette fois-ci, X est l'ensemble des r tels que $\mathcal{A}(x, \epsilon, r) = q_k$ et $(x\#q_k\#y_k, x\#r) \in D$, et Arthur peut facilement vérifier que $r \in X$.

Ce protocole en deux tours a la propriété que si $x \in L$, alors Arthur doit accepter avec probabilité forte. (Nous laissons le soin au lecteur de régler les détails, typiquement d'ajuster

ℓ' et ℓ'' de sorte à obtenir une probabilité de rejet faible.)

Si en revanche $x \notin L$, alors quel que soit j, toutes les questions sont de poids bien plus petit que $2^{p(n)}/(2(p(n)+1))(1-1/2^{n^\ell})/2^{j+1}$, donc Arthur doit échouer avec probabilité forte. Il est ici important que toutes les questions soient de poids non seulement inférieur, mais bien plus petit que $2^{p(n)}/(2(p(n)+1))(1-1/2^{n^\ell})/2^{j+1}$: c'est ce qui garantira que la construction de Sipser s'applique.

Les détails de calculs explicites de bornes sont laissées en exercice. Nous avons dit plus haut que le protocole ci-dessus serait exécuté $p(n)+1$ fois en parallèle, avec une instance par valeur de j, $0 \leq j \leq p(n)$. Ceci revient à ce qu'Arthur devine non pas une famille de ℓ' fonctions de hachage au premier tour, mais $p(n)+1$ familles, et que Merlin devine une collision dans chaque ; au deuxième tour, Arthur doit tirer au hasard non plus $\ell'+1$ familles de ℓ'' fonctions de hachage, mais $(p(n)+1)(\ell'+1)$. □

3.5 ABPP, IP, et PSPACE

Lorsque le nombre de tours n'est plus borné par une constante, a priori tout change. On a d'abord :

Proposition 3.26 **ABPP** $\subseteq$ **IP** $\subseteq$ **PSPACE**.

Démonstration. On peut écrire un simulateur de n'importe quel protocole entre Arthur et Merlin, ou **IP**, en espace polynomial, qui au lieu de tirer des bandes d'aléa r_j itère sur toutes les bandes possibles. Le simulateur termine en comptant le nombre d'instances du protocole qui accepte. Comme **ABPP** $\subseteq$ **IP** (lemme 3.24), il suffit de simuler un protocole **IP**, tel que défini à la définition 3.22. Cependant, il est plus facile de démontrer d'abord **ABPP** $\subseteq$ **PSPACE**, puis de modifier notre argument pour établir que **IP** $\subseteq$ **PSPACE**.

Supposons donc que L soit décidé par un protocole **ABPP** à $f(n)$ tours, en utilisant des bandes d'aléa de $q(n)$ bits, et où Merlin produit des réponses y de taille $p(n)$, où $f(n)$, $q(n)$ et $p(n)$ sont des polynômes en n. On écrit le programme non-déterministe suivant, paraphrasant la définition 3.22, et écrit en style fonctionnel et récursif, avec une variable globale *prot* (en lecture seule) et une variable globale *cnt*, qui compte le nombre de configurations finales acceptantes :

```
let rec simul(inp, j, i) =
        if i = |prot| // fin du protocole, comptons :
          then if inp ∈ D
                  then cnt := cnt + 1 ;
               else ;
        else if prot_i = A
               then for r ∈ {0,1}^{q(n)} do simul(inp#r#A(inp, r),
                                                  j + 1, i + 1) ;
        else { guess y ∈ {0,1}^{p(n)} ; simul(inp#y, j, i + 1) ; }
```

On calcule ensuite $cnt := 0$; $simul(x, 0, 1)$; et on accepte si $cnt > \frac{1}{2}2^{f(n)q(n)}$. Tous les compteurs sont de taille polynomiale, au plus $f(n)q(n)$ pour *cnt*. La récurrence s'arrête à

profondeur $f(n)$, ce qui nécessite de fabriquer $f(n)$ copies des divers compteurs utilisés au sein de *simul*. En dépliant la récurrence, on obtient ainsi un algorithme en espace polynomial non déterministe. On conclut car **NPSPACE** = **PSPACE**, par le théorème de Savitch.

Pour montrer **IP** $\subseteq$ **PSPACE**, on pourrait penser, de même, paraphraser la définition 3.22, et écrire :

let rec $simul(public, secret, j, i) =$
  if $i = |prot|$ // fin du protocole, comptons :
    then if $(public, secret) \in D$
      then $cnt := cnt + 1$;
    else ;
  else if $prot_i = \mathtt{A}$
    then for $r \in \{0,1\}^{q(n)}$ do $simul(public\#\mathcal{A}(public, secret, r),$
      $secret\#r, j+1, i+1)$;
  else { guess $y \in \{0,1\}^{p(n)}$; $simul(public\#y, secret, j, i+1)$; }

Cependant, le y deviné en dernière ligne peut dépendre à la fois de *public* et de *secret*, alors qu'il ne devrait dépendre que de *public*.

À la place, on va appliquer *simul* juste sur *public*, et pas sur *secret* ; au lieu de parcourir tous les $r \in \{0,1\}^{q(n)}$, on va parcourir directement toutes les questions possibles $quest \in \{0,1\}^{p'(n)}$. Lorsque l'on termine le protocole, il n'y a plus qu'à compter le nombre d'historiques *secret* correspondant à *public* tels que $(public, secret) \in D$. Notons $q'(n)$ la taille polynomiale des questions. On écrit donc :

let rec $simul\_ip(public, j, i) =$
  if $i = |prot|$ // fin du protocole, comptons :
    then $compte(public, \epsilon, 1)$;
  else if $prot_i = \mathtt{A}$
    then for $quest \in \{0,1\}^{p'(n)}$ do $simul\_ip(public\#quest,$
      $j+1, i+1)$;
  else { guess $y \in \{0,1\}^{p(n)}$; $simul\_ip(public\#y, j, i+1)$; }

où la fonction de comptage des secrets (calculant sur *public*, de droite à gauche, ici) est définie comme suit. L'idée est que $compte(public, secret, i)$ compte combien d'historiques secrets s correspondant à *public* et ayant *secret* comme préfixe des $(i-1)$ premiers aléas, sont tels que $(public, s) \in D$. (On notera la ressemblance de cette technique avec celle du théorème 3.25.)

let rec $compte(public, secret, i) =$
  if $i = |prot|$ // fin.
    then if $(public, secret) \in D$
      then $cnt := cnt + 1$;
    else ;
  else if $prot_i = \mathtt{A}$
    then for $r \in \{0,1\}^{q(n)}$ do

if $message(public, i) = \mathcal{A}(history(public, i), r)$
then $compte(public, secret\#r, i+1)$;
else ;
else $compte(public, secret, i+1)$;

Accessoirement, les fonctions *message* et *history* extraient respectivement le $i$ème message de la liste *public*, et la liste des $i-1$ premiers messages : si *public* est de la forme $m_0\#m_1\#\ldots\#m_n$, et $0 \leq i \leq n$, $message(public, i)$ retourne m_i, et $history(public, i)$ retourne $m_0\#m_1\#\ldots\#m_{i-1}$. On laisse leur écriture en exercice.

On calcule ensuite, comme plus haut, $cnt := 0$; $simul\_ip(x, 0, 1)$; et on accepte si $cnt > \frac{1}{2}2^{f(n)q(n)}$. □

On peut aller plus loin. Le théorème 3.30 ci-dessous, montrant qu'en fait **IP** = **PSPACE**, est dû à Shamir. Comme **IP** ⊆ **PSPACE**, il suffit de montrer qu'un problème **PSPACE**-complet donné a une preuve interactive en nombre polynomial de tours. Le problème QBF s'impose. Cependant, la technique originale de Shamir [17] ne s'applique pas directement à QBF, mais à une restriction de QBF, elle aussi **PSPACE**-complète. On peut éviter cette indirection en utilisant à la place un argument dû à Shen [18].

Rappelons qu'une *formule QBF* est une formule de la forme $F = Q_1x_1 \cdot \ldots \cdot Q_mx_m \cdot G(x_1, \ldots, x_m)$, où $G(x_1, \ldots, x_m)$ est un ensemble de 3-clauses en $x_1, \ldots, x_m$, et $Q_1, \ldots, Q_m \in \{\forall, \exists\}$. Le problème QUANTIFIED-BOOLEAN-FORMULAS, ou QBF en abrégé, est le suivant :
ENTRÉE : une formule QBF F
QUESTION : cette formule est-elle valide ?

Une formule QBF est sans variable libre, elle est donc soit valide soit insatisfiable.

Les deux démonstrations, celles de Shamir et de Shen, dépendent d'un lemme qu'il est bon de connaître, le lemme de Schwartz-Zippel [16, 21], énoncé ci-dessous. Rappelons qu'un polynôme P à m variables $x_1, \ldots, x_m$ détermine une fonction polynomiale. Si $P = 0$, la fonction polynomiale associée est identiquement nulle, mais la réciproque n'est pas vraie. Par exemple, lorsque p est premier, le polynôme $x^p - x$ est un polynôme non nul de $\mathbb{Z}/p\mathbb{Z}[x]$, qui détermine une fonction identiquement nulle sur $\mathbb{Z}/p\mathbb{Z}$, par le théorème de Fermat. En revanche, si le degré de P en chaque variable x_i est strictement plus petit que p, alors P est entièrement déterminé par sa fonction polynomiale associée :

Lemme 3.27 *Soit fun la fonction qui à tout polynôme P de $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_m]$ associe sa fonction polynomiale associée. Alors fun détermine une bijection de l'espace $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_m]_{<p}$ des polynômes en $x_1, \ldots, x_m$ de degré inférieur strictement à p en chaque variable vers l'espace $(\mathbb{Z}/p\mathbb{Z})^m \to \mathbb{Z}/p\mathbb{Z}$ des fonctions de $(\mathbb{Z}/p\mathbb{Z})^m$ vers $\mathbb{Z}/p\mathbb{Z}$.*

Démonstration. Pour tout $k \in \mathbb{Z}/p\mathbb{Z}$, soit $\delta_k(x)$ le polynôme $\prod_{j \in \mathbb{Z}/p\mathbb{Z}, j \neq k}(x-j)(k-j)^{-1}$, de degré $p-1$. Pour chaque fonction f de $(\mathbb{Z}/p\mathbb{Z})^m$ vers $\mathbb{Z}/p\mathbb{Z}$, posons $P = poly(f)$ le polynôme $\sum_{k_1,\ldots,k_m \in \mathbb{Z}/p\mathbb{Z}} f(k_1, \ldots, k_m)\delta_{k_1}(x_1)\ldots\delta_{k_m}(x_m)$. P est de degré au plus $p-1$ en chaque variable, et sa fonction polynomiale est exactement f. Ceci définit une fonction *poly* de $(\mathbb{Z}/p\mathbb{Z})^m \to \mathbb{Z}/p\mathbb{Z}$ vers $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_m]_{<p}$. De plus, $fun(poly(f)) = f$ pour tout f, par

construction. En particulier, *poly* est injective. Mais $(\mathbb{Z}/p\mathbb{Z})^m \to \mathbb{Z}/p\mathbb{Z}$ et $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_m]_{<p}$ ont le même cardinal, à savoir p^{p^m}, donc *poly* est bijective. Il en est donc de même de fun. □

Un résultat proche et bien connu est le lemme d'interpolation de Lagrange. Rappelons qu'une *racine* d'un polynôme $P(x_1, \ldots, x_m)$ est un m-uplet de valeurs $v_1, \ldots, v_m$ telles que $P(v_1, \ldots, v_m) = 0$. Rappelons aussi que si p est premier, alors $\mathbb{Z}/p\mathbb{Z}$ est un corps. En particulier, l'inverse k^{-1} de tout nombre k non nul (modulo p) existe. Celui-ci peut se calculer à l'aide du théorème de Bezout : puisque k et p sont premiers entre eux, il existe deux entiers a et b tels que $ak + bp = 1$; alors a modulo p est l'inverse k^{-1} cherché.

Lemme 3.28 (Lagrange) *Soit p un entier premier et $d < p$. Soient $v_0, \ldots, v_d$ $d+1$ éléments distincts de $\mathbb{Z}/p\mathbb{Z}$, et $w_0, \ldots, w_d$ des éléments, non nécessairement distincts, de $\mathbb{Z}/p\mathbb{Z}$. Il existe un unique polynôme P dans $\mathbb{Z}/p\mathbb{Z}$ de degré au plus d tel que $P(v_i) = w_i$ pour tout i, $0 \leq i \leq d$. En particulier, si P est un polynôme de degré au plus d qui a strictement plus de d racines, alors P est le polynôme nul.*

Démonstration. Posons $L_{v_i}(x)$ le polynôme $\prod_{0 \leq k \leq d, k \neq i}(x - v_k)(v_i - v_k)^{-1}$: $L_{v_i}(v_i) = 1$ et $L_{v_i}(v_k) = 0$ pour tout $k \neq i$. De plus, $L_{v_i}(x)$ est de degré d. Le polynôme souhaité est alors $P = \sum_{i=0}^{d} w_i L_{v_i}(x)$, qui est bien de degré au plus d. Montrons que c'est le seul polynôme répondant à la question. Pour chaque famille $\vec{w}$ de valeurs $w_0, \ldots, w_d$, notons $L(\vec{w})$ ce polynôme de degré au plus d. Réciproquement, pour chaque polynôme P de degré au plus d, notons $W(P)$ la famille des valeurs $P(v_0)$, …, $P(v_d)$. Par construction, $W(L(\vec{w})) = \vec{w}$, donc L est injective. Comme il y a autant de polynômes de degré au plus d que de familles $\vec{w}$, à savoir p^{d+1}, L est bijective, donc W aussi. □

Une des conséquences de ce lemme est que, sous les hypothèses données, et si $P(X) \neq 0$, alors la probabilité que $P(v) = 0$, lorsque v est tirée aléatoirement, uniformément dans $\mathbb{Z}/p\mathbb{Z}$, est d'au plus d/p. C'est tout ce dont nous aurons besoin dans la suite, mais il est bon de savoir que ce résultat se généralise à plusieurs variables :

Lemme 3.29 (Schwartz-Zippel [16, 21]) *Soit p un nombre premier, et P un polynôme non nul de $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_m]$ ($m \geq 1$), de degré total $d < p$. La probabilité que $P(v_1, \ldots, v_m) = 0$, lorsque $v_1, \ldots, v_m$ sont tirés aléatoirement, uniformément et indépendamment dans $\mathbb{Z}/p\mathbb{Z}$, est au plus d/p.*

Démonstration. Ceci revient à démontrer que P a au plus dp^{m-1} racines. On le montre par récurrence sur m. Si $m = 1$, c'est une conséquence du lemme 3.28 d'interpolation de Lagrange : si le polynôme P avait strictement plus de d racines, il serait nul. Si $m > 1$, on peut écrire $P(x_1, \ldots, x_{m-1}, x_m)$ sous forme d'un polynôme en x_m, à coefficients dans $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_{m-1}]$. Soit d_m le degré de ce polynôme en x_m seul, et $Q_m(x_1, \ldots, x_{m-1})$ le coefficient de $x_m^{d_m}$. Notons que Q_m est de degré total au plus $d - d_m$. Si $P(v_1, \ldots, v_{m-1}, v_m) = 0$, alors soit $Q_m(v_1, \ldots, v_{m-1}) = 0$ (ce qui arrive pour au plus $(d-d_m)p^{m-2}$ valeurs du $(m-1)$-uplet $(v_1, \ldots, v_{m-1})$, par hypothèse de récurrence) soit le polynôme $P(v_1, \ldots, v_{m-1}, x)$ en x, de degré d_m, s'annule en v_m (ce qui arrive pour au plus d_m valeurs de v_m, par interpolation de

Lagrange). Le nombre de racines de P est donc d'au plus $(d-d_m)p^{m-2}.p+d_m.p^{m-1} = dp^{m-1}$. □

Il faudra savoir, ce que nous admettrons, que le calcul de la somme, de la différence, du produit modulo p, ainsi que des coefficients de Bezout de deux nombres x et y, c'est-à-dire des coefficients a et b tels que $ax+by$ soit égal au pgcd de x et y, se font en temps polynomial. Lorsque p est premier, le calcul des coefficients de Bezout permet de calculer l'inverse de x modulo p lorsque x n'est pas multiple de p (prendre $y = p$, alors a est l'inverse de x cherché). Donc toutes les opérations du corps $\mathbb{Z}/p\mathbb{Z}$ s'effectuent en temps polynomial, en la taille $\log_2 p$ de p.

Nous devrons aussi admettre le *postulat de Bertrand* : pour tout $n \geq 1$, il existe un nombre premier p entre n et $2n$. Malgré le nom, le postulat de Bertrand est un théorème. On peut de plus dire mieux : il existe en fait strictement plus de $n/(3\log(2n))$ nombres premiers p tels que $n < p \leq 2n$ pour tout $n \geq 1$.

Nous aurons finalement besoin de tester si un entier donné p est premier. L'algorithme naïf qui énumère tous les j, $2 \leq j \leq \lfloor\sqrt{p}\rceil$, et teste si j divise p, ne fonctionne *pas* en temps polynomial en la taille $\log_2 p$. Cependant, tester si p est premier peut se faire en temps polynomial [2]. N'importe quel algorithme randomisé pour tester la primalité de p suffirait aussi en pratique. Même en ignorant tout cela, nous pourrions aussi compter sur Merlin : la primalité est dans **NP**, par un ancien résultat de Pratt, et l'on peut donc demander à Merlin une preuve de taille polynomiale du fait que p est premier. On consultera par exemple le cours de Shoup sur l'algorithmique des nombres [19] pour davantage d'informations.

Théorème 3.30 (Shamir [17]) **ABPP = IP = PSPACE**.

Démonstration. On va montrer que QBF, le prototype des problèmes **PSPACE**-complets, est dans **ABPP**. Rappelons que QBF est le problème de décision suivant :
ENTRÉE : une formule de la forme $F = Q_1x_1 \cdot \ldots \cdot Q_mx_m \cdot G(x_1, \ldots, x_m)$, où $G(x_1, \ldots, x_m)$ est un ensemble de 3-clauses en x_1, …, x_m, et $Q_1, \ldots, Q_m \in \{\forall, \exists\}$.
QUESTION : cette formule est-elle valide ?

Nous donnons la variante de Shen [18] de la preuve de Shamir.

Notons que F est une formule close : il est équivalent de demander qu'elle soit valide ou bien qu'elle soit satisfiable.

Soit $\mathbb{F}$ le corps fini $\mathbb{Z}/p\mathbb{Z}$, avec p un nombre premier suffisamment grand. On va plonger l'ensemble $\{0, 1\}$ des booléens dans $\mathbb{F}$, et interpréter chaque formule QBF comme un polynôme sur $\mathbb{F}$. Ceci rappelle les techniques du théorème 2.12, et effectivement le théorème de Shamir a été l'un des travaux ayant mené plus tard au théorème d'Arora.

On peut ainsi coder le et de x et de y et par le produit xy, la négation de x par $1-x$, le ou de x et y par $x+y-xy$ — ce que nous noterons $x \vee y$ dans la suite, sans qu'on courre un risque de confusion. Lorsque x et y sont dans $\{0, 1\}$, ceci calcule le bon booléen. On peut ainsi convertir n'importe quelle formule propositionnelle (sans quantificateurs) $G(x_1, \ldots, x_m)$ en un polynôme $\hat{G}(x_1, \ldots, x_m)$ suivant ce principe. Lorsque G est une conjonction de 3-clauses, $\hat{G}$ est un polynôme de degré total au plus $3k$, où k est le nombre de clauses de G. Attention : nous ne développerons pas le polynôme sous forme de somme de produits, ce qui fabriquerait

une expression de taille exponentielle en général. Nous le laisserons sous forme symbolique, en utilisant des opérateurs $\vee$, $\wedge$, $\neg$.

Pour tout polynôme P dans $\mathbb{F}[x_1, \ldots, x_m]$, et toute variable $x \in \{x_1, \ldots, x_m\}$, on peut définir les trois autres polynômes :

$$\begin{aligned} \forall x \cdot P &= P[x := 0] \times P[x := 1] \\ \exists x \cdot P &= P[x := 0] \vee P[x := 1] \\ Rx \cdot P &= P \bmod (x^2 - x) \end{aligned}$$

où $P[x := a]$ dénote le polynôme obtenu à partir de P en remplaçant x par la valeur $a \in \mathbb{F}$, et en simplifiant. L'idée est que $\forall x \cdot P$ code la quantification universelle sur $x \in \{0, 1\}$, $\exists x \cdot P$ code la quantification existentielle sur $x \in \{0, 1\}$. Le polynôme $Rx \cdot P$ coïncide avec P lorsque x est un booléen, mais est de degré au plus 1 en x.

L'entrée QBF $F = Q_1 x_1 \cdot \ldots \cdot Q_m x_m \cdot G(x_1, \ldots, x_m)$ est alors vue comme un polynôme. Sur l'entrée QBF $F = Q_1 x_1 \cdot \ldots \cdot Q_m x_m \cdot G(x_1, \ldots, x_m)$, l'idée sera de fabriquer le polynôme

$$\begin{aligned} & Q_1 x_1 \cdot Rx_1 \cdot \\ & Q_2 x_2 \cdot Rx_1 \cdot Rx_2 \cdot \\ & \ldots \\ & Q_m x_m \cdot Rx_1 \cdot Rx_2 \cdot \ldots \cdot Rx_m \cdot \hat{G} \end{aligned} \tag{5}$$

De nouveau, on ne cherchera pas à simplifier l'expression ci-dessus, ce qui fournirait une expression trop grosse. On peut cependant borner le degré en z d'un tel polynôme P par la fonction $deg_z(P)$, calculable en temps polynomial par la récurrence :

$$\begin{aligned} deg_z(\forall x \cdot P) = deg_z(\exists x \cdot P) &= 2deg_z(P) \quad \text{si } z \neq x \\ deg_x(\forall x \cdot P) = deg_x(\exists x \cdot P) &= 0 \\ deg_z(Rx \cdot P) &= deg_z(P) \quad \text{si } z \neq x \\ deg_x(Rx \cdot P) &= \min(1, deg_x(P)) \end{aligned}$$

L'idée du protocole est de demander à Merlin de stocker tous les polynômes

$$\begin{aligned} P_{i,j}(x_1, \ldots, x_i) = \quad & Rx_{j+1} \cdot \ldots Rx_i \cdot \\ & Q_{i+1} x_{i+1} \cdot Rx_1 \cdot \ldots \cdot Rx_{i+1} \cdot \\ & \ldots \\ & Q_m x_m \cdot Rx_1 \cdot Rx_2 \cdot \ldots \cdot Rx_m \cdot \hat{G} \end{aligned}$$

pour tous $0 \leq j < i \leq m$, ainsi que le polynôme $\hat{G}$, et (5). Par souci d'uniformité, notons $P_{0,0}()$ le polynôme (5), et $P_{m+1,0}()$ le polynôme $\hat{G}$. Le fait d'enlever le premier quantificateur à P_{ij} fournit un polynôme $P_{s(i,j)}$, où le successeur $s(i, j)$ est défini par $s(0, 0) = 1, 0$ et dans les autres cas, $s(i, j) = i, j+1$ si $j+1 < i$, et par $s(i, j) = i+1, 0$ sinon. Arthur devra ensuite vérifier que l'expression $P_{0,0}()$ vaut 1, et que pour tous $j \leq i$, $P_{i,j}$ et le polynôme suivant

$P_{s(i,j)}$ vérifient certaines conditions de cohérence, que l'on testera en évaluant les polynômes sur des valeurs aléatoires (modulo p) des variables. Il est important que ces valeurs ne soient pas limitées à des valeurs booléennes : c'est ainsi qu'Arthur pourra vraiment vérifier que Merlin a les bons polynômes, et ne triche pas.

Fixons $\ell \geq 0$. Merlin commence le protocole en choisissant un nombre p de taille $\log_2(2^{n^\ell+1}dq)$, où d le degré maximal des polynômes apparaissant comme sous-expressions de $P_{00}()$; d est lui-même calculable en temps polynomial. Arthur vérifie que p est un nombre premier entre $2^{n^\ell}dq$ et $2^{n^\ell+1}dq$, en temps polynomial en n (éventuellement probabiliste). Notons que d sera toujours au plus $3k$, donc dq est borné par un polynôme en la taille de F. Ceci se fait donc bien en temps polynomial.

Initialement, le polynôme P est $P_{00}()$, donné par l'équation (5), la valeur objectif w est 1. On va aussi tirer au hasard des valeurs $v_1, \ldots, v_m$ des m variables $x_1, \ldots, x_m$ dans la suite, que l'on conservera dans des variables globales (publiques) du même nom. Initialement, aucune valeur v_i n'est définie. On itère ensuite la procédure suivante pour (i,j) valant $(1,0)$, $s(1,0) = (2,0)$, $s(2,0) = (2,1)$, $\ldots$, $s(m,m-1) = (m+1,0)$. Un invariant est qu'au tour (i,j), les valeurs définies sont exactement $v_1, \ldots, v_i$. Au tour (i,j), donc :

- Merlin fournit un polynôme $\widetilde{P}_{i,j}(x_j)$, et prétend que c'est $P_{i,j}(v_1, \ldots, v_{j-1}, x_j)$. (Comme les seuls polynômes qui passeront le test suivant sont de taille polynomiale, typiquement bornée par $O(d \log p)$, ceci est bien une réponse acceptable de Merlin.)
- Arthur vérifie que $\widetilde{P}_{i,j}(x_j)$ est de degré au plus d (en x_j, l'unique variable du polynôme).
- Si $j = 0$, Arthur vérifie que $Q_i x_i \cdot \widetilde{P}_{i,0}(x_i) = w$. Si $Q_i = \forall$, ceci revient à vérifier que $\widetilde{P}_{i,0}(0) \times \widetilde{P}_{i,0}(1) = w$, et si $Q_i = \exists$, que $\widetilde{P}_{i,0}(0) + \widetilde{P}_{i,0}(1) - \widetilde{P}_{i,0}(0) \times \widetilde{P}_{i,0}(1) = w$.
- Si $j \neq 0$, Arthur vérifie que $(Rx_j \cdot \widetilde{P}_{i,j}(x_j))(v_j) = w$. Noter que R n'est pas un vrai quantificateur : le polynôme $Rx_j \cdot \widetilde{P}_{i,j}(x_j)$ a une variable, qui est x_j, et on évalue ce polynôme pour $x_j = v_j$. C'est facile : ce polynôme est, par construction, une fonction affine, qui vaut la même chose que $\widetilde{P}_{i,j}(x_j)$ lorsque x_j vaut 0 ou 1, donc $\widetilde{P}_{i,j}(0) + (\widetilde{P}_{i,j}(1) - \widetilde{P}_{i,j}(0))x_j$. Arthur vérifie donc que $\widetilde{P}_{i,j}(0) + (\widetilde{P}_{i,j}(1) - \widetilde{P}_{i,j}(0))v_j = w$.
- Enfin, Arthur tire une valeur aléatoire v_j pour x_j, la stocke (dans la variable v_j, donc), et calcule le nouvel objectif à vérifier $w := \widetilde{P}_{i,j}(v_j)$. On passe ensuite au tour suivant.

Les équations $Q_i x_i \cdot \widetilde{P}_{i,0}(x_i) = w$ $(j = 0)$ et $(Rx_j \cdot \widetilde{P}_{i,j}(x_j))(v_j) = w$ $(j \neq 0)$ seront appelées les *conditions de cohérence*.

A la fin, il ne reste plus à Arthur qu'à vérifier que $\hat{G}(v_1, \ldots, v_m) = w$. Si oui, Arthur accepte, sinon il rejette. Arthur peut le faire en temps polynomial. Noter qu'Arthur ne pourrait pas le faire si $\hat{G}$ contenait encore des quantificateurs.

Si la formule F est vraie, alors Merlin a une stratégie gagnante : il joue le bon polynôme à chaque tour : $\widetilde{P}_{i,j} = P_{i,j}$.

Sinon, quelle est la probabilité qu'Arthur accepte, lorsque F est fausse ?

Disons que Merlin *triche* au tour (i,j) si et seulement si $\widetilde{P}_{i,j} \neq P_{i,j}$, et *joue honnêtement* sinon. Si Merlin jouait honnêtement au premier tour $(1,0)$, il donnerait un polynôme $\widetilde{P}_{1,0}(x_1) = P_{1,0}(x_1)$, donc $P_{0,0} = Q_1 x_1 \cdot P_{1,0}(x_1)$ vaut 0, qui est différent de la valeur objectif initiale, 1.

C'est impossible : Merlin doit tricher au premier tour.

Regardons le dernier tour (i,j) où Merlin triche. Si (i,j) est le tout dernier tour, (m,m), Merlin a fourni un polynôme $\widetilde{P}_{m,m}(x_m)$, différent de $P_{m,m}(x_m) = \hat{G}(v_1, v_2, \cdots, v_{m-1}, x_m)$. La probabilité qu'Arthur vérifie la condition finale $\hat{G}(v_1, \ldots, v_m) = w$ dans ce cas est d'au plus la probabilité que ces deux polynômes distincts, en une variable x_m ($\widetilde{P}_{m,m}(x_m)$ et $P_{m,m}(x_m)$) coïncident sur la valeur v_m, donc au plus d/p.

Si ce dernier tour est de la forme (i,i), Merlin a fourni un polynôme $\widetilde{P}_{i,i}(x_i)$ différent de $P_{i,i}(v_1, v_2, \cdots, v_{i-1}, x_i)$, et il joue honnêtement au tour suivant : $\widetilde{P}_{i+1,0}(x_{i+1}) = P_{i+1,0}(v_1, \cdots, v_i, x_{i+1})$, parce que (i,i) est le dernier tour où Merlin triche. Comme $P_{i,i}(v_1, v_2, \cdots, v_{i-1}, x_i) = Q_{i+1}x_{i+1} \cdot P_{i+1,0}(v_1, \cdots, v_{i-1}, x_i, x_{i+1})$, les deux polynômes $\widetilde{P}_{i,i}(x_i)$ et $Q_{i+1}x_{i+1} \cdot \widetilde{P}_{i+1,0}(x_{i+1})$ sont différents, et la probabilité qu'Arthur vérifie l'équation de cohérence correspondante est donc de nouveau au plus d/p.

Si le dernier tour (i,j) considéré est tel que $j < i$, l'argument est identique, à ceci près que le quantificateur impliqué est R.

Notons $A_{i,j}$ l'événement "l'équation de cohérence au tour (i,j) est vérifiée" pour $(i,j) \neq (m,m)$, et "la condition finale $\hat{G}(v_1, \ldots, v_m) = w$ est vérifiée" si $(i,j) = (m,m)$. Si la formule F est fausse, en résumé, et qu'Arthur accepte, alors il existe un indice (i,j) (le dernier tour où Merlin triche) tel que $A_{i,j}$ soit vrai. De plus, la probabilité que $A_{i,j}$ soit vrai est inférieure ou égale à d/p. La probabilité qu'Arthur accepte est donc majorée par la somme de ces probabilités, $(1+q)d/p$.

Or on a choisi p de sorte que $(1+q)d/p \leq 1/2^{n^\ell}$. Rappelons que si F est vraie, alors Merlin a une stratégie pour qu'Arthur accepte, avec probabilité 1.

Donc QBF est dans **ABPP**. On conclut que **PSPACE** $\subseteq$ **ABPP** parce que **ABPP** est stable par réduction polynomiales, puis en utilisant la proposition 3.26. □

Revenons un instant sur la taille de l'entier p. Dans la démonstration ci-dessus, nous devons pouvoir tester si p est premier en temps polynomial en la taille de p, parce que p est de l'ordre de $2^{n^\ell}d(1+q)$, de taille $O(n^\ell \log n)$. On pourrait aussi demander un p premier plus petit, typiquement entre $3d(1+q)$ et $6d(1+q)$ dans la démonstration ci-dessous. L'algorithme naïf fonctionne alors en temps polynomial en n. La probabilité d'erreur finale serait de $1/3$ au lieu de $1/2^{n^\ell}$, mais l'on peut diminuer la probabilité d'erreur de toute façon en répétant le protocole séquentiellement et en votant, comme pour **BPP**.

Notons que nous avons au passage montré que QBF était décidable par un protocole entre Arthur et Merlin où, si $x \in L$, alors Merlin peut s'arranger pour qu'Arthur accepte toujours. Ceci entraîne que, comme pour la classe **AM** (proposition 3.18), on peut remplacer le cas $x \in L$ dans la définition de **ABPP** et de **IP** par une condition demandant qu'Arthur accepte toujours, plutôt qu'avec probabilité supérieure ou égale à $1 - 1/2^{n^\ell}$.

Notons aussi du coup qu'il est relativement improbable que **AM** = **ABPP**, contrairement à l'intuition. On aurait pu en effet croire que ce serait le cas, vu que **AM** = **AMAM** = **AMAMAM** = ... = $\mathbf{AM}^k$ = ..., et **ABPP** = **AM**[*poly*] ; mais **AM** $\subseteq \Pi_2^p$ (théorème 3.19) et **ABPP** = **PSPACE**. **AM** = **ABPP** entraînerait donc que la hiérarchie polynomiale s'écroule au niveau 2, et en plus coïncide avec **PSPACE**.

Références

[1] Leonard M. Adleman. Two theorems on random polynomial time. In *Proc. 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS'78)*, pages 75–83, 1978.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. Manuscrit, `www.cse.iitk.ac.in/news/primality.html`, 2002.

[3] Sanjeev Arora. Around the PCP theorem. `http://www.cs.princeton.edu/~arora/pubs/barbados.ps`, 1996. McGill Workshop on Complexity Theory, Barbados.

[4] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs : A new characterization of NP. *Journal of the ACM*, 45(1) :70–122, 1998. Preliminary version published at the IEEE FOCS 1992 Conference.

[5] László Babai. Trading group theory for randomness. In *Proc. 17th Annual ACM symposium on the Theory of Computing (STOC'85)*, pages 421–429, 1985.

[6] László Babai and Shlomo Moran. Arthur-Merlin games : A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36 :254–276, 1988.

[7] R. B. Boppana, Johan Håstad, and S. Zachos. Does co-NP have short interactive proofs ? *Information Processing Letters*, 25(2) :127–132, 1987.

[8] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2), 1979.

[9] Irit Dinur. The PCP theorem by gap amplification. Technical Report TR05-046, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[10] Uriel Feige. Lecture notes on complexity. Available on the author's Web page, January 2004.

[11] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proc. 18th Annual ACM symposium on the Theory of Computing (STOC'86)*, pages 59–68, 1986.

[12] Richard M. Karp and Richard J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28 :191–209, 1982.

[13] Richard Lassaigne and Michel de Rougemont. *Logic and Complexity*. Springer-Verlag, 2004.

[14] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4) :215–217, 1983.

[15] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[16] Jack Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27 :701–717, 1980.

[17] Adi Shamir. IP=PSPACE. *Journal of the ACM*, 39(4) :869–877, 1992.

[18] Alexander Shen. IP=PSPACE : Simplified proof. *Journal of the ACM*, 39(4) :878–880, 1992.

[19] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005. Voir `http://shoup.net/ntb/`.

[20] Michael Sipser. A complexity theoretic approach to randomness. In *Proc. 15th ACM Symposium on Theory of Computing (STOC'83)*, pages 330–335, 1983.

[21] Richard Zippel. Probabilistic algorithms for sparse polynomials. *International Symposium on Symbolic and Algebraic Manipulation (EUROSAM'79)*, pages 216–226, 1979.