Modèle de Blum, Shub et Smale

Stéphane Caron

30 janvier 2009

Table des matières

Introduction

Le modèle de Blum, Shub et Smale (BSS) est un modèle de calcul algébrique qui généralise la notion de machine de Turing vue dans le cadre booléen en autorisant la manipulation directe de nombres réels.

L'intérêt de cette extension du modèle standard réside notamment dans le fait qu'elle permet de porter le problème P = NP sur $\mathbb{R}$, structure plus riche sur laquelle on dispose de nombreux outils. Nous verrons de quelle manière ce « transfert » s'effectue en définissant des classes de complexité généralisées et en établissant des liens entre ces nouvelles classes et les classes du modèle booléen.

1 Machines BSS et circuits algébriques

Nous allons dans un premier temps généraliser les notions du cadre booléen en choisissant pour alphabet non plus un ensemble fini Σ mais le corps des réels. Les mots de longueur n sont alors les éléments de $\mathbb{R}^n$ et un langage sur $\mathbb{R}$ est un sous-ensemble de $\mathbb{R}_\infty := \cup_{n\in\mathbb{N}}\mathbb{R}^n$. On appellera également *problème* un langage sur $\mathbb{R}$.

1.1 Machines BSS

Définition 1. Une *machine BSS* est une machine de Turing dont l'alphabet de bande est $\mathbb{R}$. Elle manipule les réels comme des entités atomiques (c.-à-d. sans s'intéresser à leur structure interne) et les opérations et tests qu'elle peut réaliser en temps unitaire correspondent respectivement aux fonctions de $\mathbb{R}$ et aux relations dont on dispose sur $\mathbb{R}$.

On ne munit pas en pratique les machines BSS de toutes les opérations possibles sur les réels. Au contraire, on s'intéresse généralement à des structures comme $(\mathbb{R}, +, -, =)$ ou $(\mathbb{R}, +, -, \times, <)$ où les relations et opérations disponibles sont élémentaires et peu nombreuses.

Par conséquent, on prendra garde au fait que l'égalité ne fait plus nécessairement partie de la structure sur laquelle on travaille. Le cas échéant, on remplacera donc le formalisme habituel « $q, a \to q', b, \triangleleft|\triangleright$ » par « $q, r(\cdot, a) \to q', b, \triangleleft|\triangleright$ », où la condition « la case courante est égale à a » devient simplement « les m cases à partir de la case courante vérifient la relation partielle $r(\cdot, a)$ d'arité m ».

On s'intéressera par ailleurs aux machines BSS qui s'arrêtent toujours, ce qui nous permet de définir le temps d'exécution $t(n)$, de même que dans le cadre standard, comme le nombre d'instructions exécutées par la machine sur une entrée de taille n.[1] Quand il n'y aura pas d'ambiguïté sur la structure dont on munit $\mathbb{R}$, on notera $\mathrm{P}_{\mathrm{BSS}}$ la classe des problèmes résolubles en temps polynomial par machine BSS.

1.2 Circuits algébriques

Nous aurons besoin d'un nouveau modèle au cours de notre voyage au pays des classes de complexité associées au modèle BSS : les *circuits algébriques*. On verra

1. On suppose donc en particulier qu'il ne dépend que de la taille de l'entrée.

plus loin que ces circuits sont en vérité équivalents aux machines BSS, mais ils nous permettront de définir proprement certaines classes dites « non-uniformes ».

Définition 2. Un *circuit* est un graphe orienté acyclique dont les sommets sont appelés « portes ». Une seule porte, appelée « sortie », est de degré sortant nul. Les portes de degré entrant nul sont appelées « entrées ».

Les entrées sont étiquetées par les identifiants des variables qu'elles représentent, tandis que les autres portes sont étiquetées par les opérations qu'elles effectuent (les tests peuvent être considérés comme des opérations dont le résultat est un 0 ou 1). Le degré entrant d'une porte doit donc correspondre à l'arité de la fonction ou relation qu'elle représente. On notera $C(x)$ un circuit C dont les variables d'entrée sont les éléments du n-uplet $x = (x_1, ..., x_n)$.

Définition 3. Un *circuit algébrique* est un circuit dont les portes sont soit des variables, soit des opérations réelles (p.ex. $+$, $=$, $\leqslant$, $\times$, etc. selon la structure sur laquelle on travaille) et dont la porte de sortie est une porte de test.

Dans le cas où la structure de $\mathbb{R}$ sur laquelle on travaille ne permet pas de l'exprimer, on introduit une fonction supplémentaire S à trois arguments appelée *sélecteur* et définie par $S(0, y, z) = y$ et $S(1, y, z) = z$. S est *a priori* indéfinie si son premier argument n'est pas un booléen ; toutefois, dans des structures comme $(\mathbb{R}, +, -, \times, =)$ on peut définir S « nativement » par $S(x, y, z) = x.y + (1 - x).z$.

Définition 4. Un problème X est reconnu par une famille $(C_n(x, y))_{n\in\mathbb{N}}$ de circuits algébriques s'il existe un uplet a de paramètres tel que $\forall n \in \mathbb{N},\ C_n(x, a)$ résout X pour les entrées de taille $|x| = n$, c.-à-d. que $\forall x \in \mathbb{R}^n,\ x \in X \Leftrightarrow C_n(x, a) = 1$.

Définition 5. Une famille de circuits (C_n) est dite *uniforme* s'il existe une machine de Turing *booléenne* qui, étant donné un entier n, construit en *temps polynomial* le circuit C_n (i.e. en fournit une description sur sa bande de sortie).

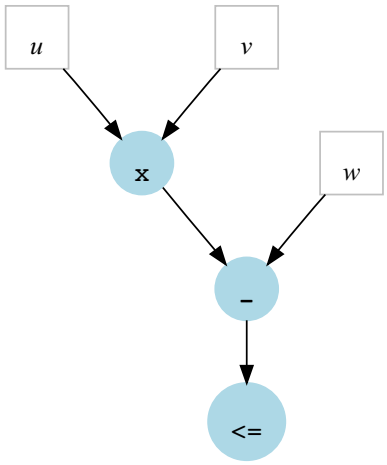

Figure 1 – Circuit algébrique décidant si $u.v \leqslant w$.

Attention toutefois : on dira d'une famille de circuits (C_n) qu'elle est *non-uniforme* dans le cas général où l'on n'impose pas de contrainte sur sa construction. En particulier, une famille uniforme de circuits est donc non-uniforme.

1.3 Équivalence entre les deux modèles

Voyons un théorème essentiel qui va nous permettre de manipuler les notions de machine BSS et de circuit algébrique que nous venons de définir.

Théorème 1. *Un problème X est dans la classe $\mathrm{P}_{\mathrm{BSS}}$ si et seulement s'il est reconnu par une famille uniforme de circuits.*

Démonstration. Il est facile de montrer qu'un problème reconnu par une famille uniforme (C_n) de circuits est dans $\mathrm{P}_{\mathrm{BSS}}$: il suffit de considérer la machine qui, pour une entrée x de taille n, construit le circuit C_n puis l'exécute sur x, ce qui se fait en temps polynomial car C_n est de taille polynomiale.

Montrons la réciproque. On considère une machine BSS qui s'exécute en temps polynomial $t(n)$ pour une entrée de taille n. Notre objectif est de construire des « blocs » [2] B_n simulant une étape de calcul de la machine pour une bande de travail de largeur au plus n. En effet, une fois qu'on disposera de tels objets, il nous suffira d'en empiler $t(n)$ pour simuler dans son intégralité le calcul effectué par la machine.

La structure d'un bloc B_n est la suivante :

- l'entrée est constituée de l'état courant q de la machine, de la position p de son pointeur d'exécution ainsi que des valeurs des n cases non-vides de la bande de travail ;
- la sortie est constituée de l'état q' résultant, de la position p' du pointeur après exécution et des $n+1$ valeurs des cases non-vides de la bande de travail (la largeur de la bande augmente d'au plus 1 au cours d'une étape).

On va construire dans un premier temps un circuit $B_n(q,p)$ qui, pour q et p fixés, calcule les sorties attendues. Ses entrées sont les n cases non-vides $r_1 \dots r_n$ de la bande de travail. On considère les transitions $q, r(\cdot,a) \to q, b', \triangleleft|\triangleright$: pour chacune, on va appliquer la relation r aux cases $r_p \dots r_{p+k}$ (où la relation partielle $r(\cdot,a)$ est d'arité $k+1$) et injecter le résultat du test en entrée de sélecteurs comme représenté sur la figure 2. Pour ce faire, on a besoin (a) de sélecteurs sur des entrées-sorties de taille n, qu'on construit à partir du sélecteur S élémentaire (on obtient $O(n)$ portes) et (b) d'un incrémenteur (cas $\triangleright$) ou d'un décrémenteur (cas $\triangleleft$), qu'on peut également construire avec des sélecteurs (on obtient $O(\log n)$ portes). Après avoir mis en série les sous-blocs correspondant à chaque transition, et sans oublier d'ajouter une sortie supplémentaire pour la case $n+1$, on obtient le circuit $B_n(q,p)$.

On construit ensuite B_n à partir des sous-blocs $B_n(q,p)$. Pour ce faire, on commence par câbler tous les sous-blocs aux entrées $r_1 \dots r_n$, puis on définit la sortie de B_n par des sélecteurs (gigantesques) dont les entrées de contrôle sont les représentations binaires de q et de p, et dont les entrées à sélectionner sont les sorties respectives de chacun des $B_n(q,p)$

2. Qu'on peut voir comme des circuits sans contrainte sur les sorties.

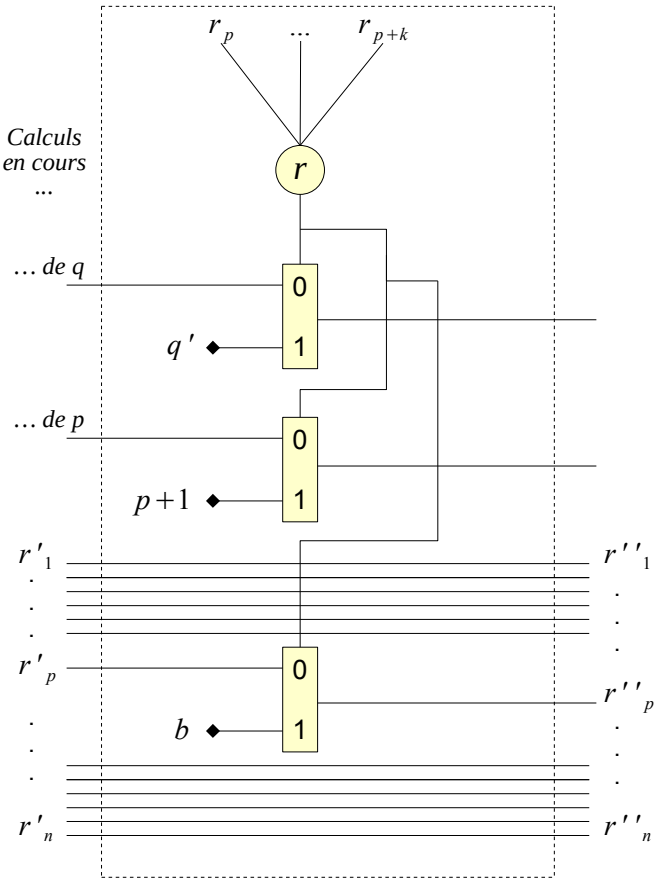

FIGURE 2 – Construction d'un bloc $B_n(q, p)$.

(un sélecteur pour chaque arête de sortie). On définit de tels sélecteurs comme suit par récurrence à partir de S :

$$S_{n+1}(x_1, \ldots, x_{n+1}, y_1, \ldots, y_{2^{n+1}}) = S_n(x_2, \ldots, x_{n+1}, \ldots, S(x_1, y_{2i}, y_{2i+1}), \ldots).$$

Enfin, on construit notre circuit C_n total pour toutes les entrées de taille n en empilant consécutivement des circuits B_0, B_1, ..., $B_{t(n)-1}$ et en câblant leurs entrées et sorties respectives. Comme on peut vérifier sans difficulté que la taille des circuits B_n est polynomiale, on en conclut que l'algorithme que nous venons de décrire construit en temps polynomial les circuits de la famille (C_n). □

2 Classes de complexité généralisées

La généralisation aux machines BSS s'accompagne d'une nouvelle hiérarchie de classes de complexité. Nous allons voir comment étendre la classe NP connue avant de définir de nouvelles classes dites « non-uniformes ».

2.1 Classe existentielle NP

Dans le cadre standard, on a montré que NP pouvait être définie de manière existentielle comme la classe des problèmes dont les instances admettent un certificat polynomial, soit : $X \in \mathrm{NP}$ si et seulement s'il existe $Y \in \mathrm{P}$ tel que $x \in X \Leftrightarrow \exists y \in \Sigma^*$ de taille polynomiale en $|x|$ tel que $(x, y) \in Y$. Ce point de vue existentiel nous permet de généraliser la classe NP au modèle BSS. [3]

Définition 6. Un problème X est dans la classe $\mathrm{NP}_{\mathrm{BSS}}$ si et seulement s'il existe un problème $Y \in \mathrm{P}_{\mathrm{BSS}}$ tel que

$$x \in X \iff \exists y \in \mathbb{R}_\infty \text{ de taille polynomiale en } |x| \text{ t.q. } (x, y) \in Y.$$

Nous verrons en 3.2 que cette approche de généralisation est pertinente dans la mesure où elle permet de lier les questions « $\mathrm{P}_{\mathrm{BSS}} = \mathrm{NP}_{\mathrm{BSS}}$? » et « $\mathrm{P} = \mathrm{NP}$? ».

2.2 Classes polynomiales non-uniformes

Définition 7. On note $\mathbb{P}$ la classe des problèmes $X \subset \Sigma^*$ reconnus par une famille non-uniforme de circuits *booléens* de taille polynomiale.

Définition 8. On note $\mathbb{P}_{\mathrm{BSS}}$ la classe des problèmes $X \subset \mathbb{R}_\infty$ reconnus par une famille non-uniforme de circuits *algébriques* de taille polynomiale.

Du point de vue des machines BSS, la classe $\mathbb{P}$ correspond aux problèmes que l'on peut résoudre en temps polynomial lorsqu'on dispose en sus d'un « conseil » pour chaque taille d'entrée, c'est à dire que, pour une entrée x de taille n, la machine peut également consulter au cours du calcul un uplet c_n. On notera toutefois que ce conseil est le même pour toutes les entrées de même taille. [4]

3. Et même à n'importe quelle structure sur laquelle on est capable de définir la classe P.

4. Au contraire des vérificateurs polynomiaux associés aux problèmes NP, pour lesquels le « certificat » correspondant à une entrée dépend de l'entrée elle-même et pas seulement de sa taille.

Les classes non-uniformes nous fournissent un modèle de calcul *a priori* plus puissant que les classes uniformes, mais également plus distant des considérations pratiques de calculabilité. L'un des intérêts de ces classes réside dans le fait que cette augmentation de la puissance des outils est accompagnée de la proposition suivante.

Proposition 2. *Si* NP *n'est pas un sous-ensemble de* $\mathbb{P}$*, alors* $\mathrm{P} \neq \mathrm{NP}$*. De même, si* $\mathrm{NP}_{\mathrm{BSS}}$ *n'est pas un sous-ensemble de* $\mathbb{P}_{\mathrm{BSS}}$*, alors* $\mathrm{P}_{\mathrm{BSS}} \neq \mathrm{NP}_{\mathrm{BSS}}$*.*

(La démonstration de ce résultat est facile et ne posera pas de problème au lecteur qui nous a suivis jusqu'ici.) Toutefois, l'apport de la non-uniformité dépend de la structure sur laquelle on travaille : le théorème suivant nous montre que, sur $(\mathbb{R}, +, -, <)$, cet apport est même inexistant car on peut alors profiter des paramètres autorisés dans $\mathrm{P}_{\mathrm{BSS}}$ pour coder la famille de circuits.

Théorème 3. *Sur* $(\mathbb{R}, +, -, <)$ *on a* $\mathrm{P}_{\mathrm{BSS}} = \mathbb{P}_{\mathrm{BSS}}$*.*

Démonstration. On a par définition $\mathrm{P}_{\mathrm{BSS}} \subset \mathbb{P}_{\mathrm{BSS}}$ quelle que soit la structure sur laquelle on travaille. Montrons l'inclusion réciproque. Soit (C_n) une suite non-uniforme de circuits de taille polynomiale : comme annoncé, on va exploiter les opérations dont on dispose sur $\mathbb{R}$ pour coder (C_n) dans une constante α qu'on saura décoder en temps polynomial.

Intéressons-nous tout d'abord au codage d'une suite arbitraire de 0 et de 1 dans une constante $\alpha \in [0, 1[$. On considère la représentation dyadique *propre* $\alpha = 0, \epsilon_1\epsilon_2 \dots \epsilon_n \dots$ où $\epsilon_i \in \{0, 1\}$ et où la suite (ϵ_i) ne stationne pas à 1. On définit les deux opérations suivantes :

$$t(x) = \begin{cases} 0 & \text{si } x + x < 1 \\ 1 & \text{sinon} \end{cases} \quad ; \quad q(x) = x + x - t(x).$$

En d'autres termes, t compare $2x$ à 1 pour récupérer le chiffre de tête ϵ_1 de la représentation tandis que q retranche son premier chiffre à $2x$ pour obtenir la queue $q(x) = 0, \epsilon_2\epsilon_3 \dots$

Pour coder dans un réel une suite quelconque (w_n) de mots sur $\{0, 1\}$, on dédouble les bits pour introduire un symbole de séparation. Formellement, on introduit un séparateur \$ et un morphisme $\mu : 0 \mapsto 00, 1 \mapsto 10, \$ \mapsto 01$. On code alors (w_n) par le réel α dont la représentation dyadique propre est $0, \mu(w_0\$w_1\$ \dots)$.

Si la croissance de la suite (w_n) est majorée par un polynôme $p(n)$, on peut définir un algorithme qui récupère le mot w_n en temps polynomial : il suffit d'appliquer successivement t et q à α jusqu'à avoir lu les n premiers séparateurs, puis de lire w_n entre le n^{e} et le $(n+1)^{\mathrm{e}}$ séparateur. Ceci se fait en un temps $2(p(0) + 1 + p(1) + 1 + \dots + 1 + p(n)) = O(n.p(n))$.

Nous avons désormais les outils nécessaires pour revenir à notre problème $X \in \mathbb{P}_{\mathrm{BSS}}$: soit $(C_n(x, y))$ une famille non-uniforme de circuits de taille polynomiale qui résout X pour un paramètre a. En codant (C_n) dans un paramètre supplémentaire α, on a un algorithme polynomial qui résout X : pour une entrée x de taille n, récupérer le circuit C_n codé dans α, puis exécuter $C_n(x, a)$. Ces opérations sont en temps polynomial, donc $X \in \mathrm{P}_{\mathrm{BSS}}$. □

Remarque. Dans le cas booléen, on a $\mathrm{P} \neq \mathbb{P}$. En effet, P est dénombrable (on peut dénombrer les machines de TURING) tandis que $\mathbb{P}$ ne l'est pas.

3 Liens avec les classes standard

On connaît désormais les extensions des classes standard au modèle BSS, mais il nous reste à établir des liens entre ces nouvelles classes et nos classes initiales pour comprendre l'intérêt de la généralisation qui nous occupe.

3.1 Problèmes booléens

Définition 9. Un *problème booléen* sur une structure quelconque de $\mathbb{R}$ est un sous-ensemble de $\{0,1\}^*$.

L'idée derrière l'étude des problèmes booléens sur une structure plus générale que $\{0,1\}$ est de voir si nos machines BSS, « enrichies » par leur capacité à calculer sur $\mathbb{R}$, sont capables de résoudre plus de problèmes que les machines booléennes classiques lorsqu'elles travaillent sur des entrées booléennes. Le théorème suivant montre qu'il n'en est rien sur $(\mathbb{R}, +, -, =)$.

Théorème 4. *Dans la structure* $(\mathbb{R}, +, -, =)$ *la classe* $\mathrm{P}_{\mathrm{BSS}} \cap \{0,1\}^*$ *des problèmes booléens de* $\mathrm{P}_{\mathrm{BSS}}$ *est la classe* P *standard, et de même la classe* $\mathbb{P}_{\mathrm{BSS}} \cap \{0,1\}^*$ *des problèmes booléens de* $\mathbb{P}_{\mathrm{BSS}}$ *est la classe* $\mathbb{P}$ *standard.*

Démonstration. Considérons un algorithme polynomial $\mathcal{A}$ résolvant un problème $X \in \mathrm{P}_{\mathrm{BSS}}$, et soient $(a_1, \ldots, a_m)$ ses paramètres. On munit $\mathbb{R}$ d'une structure de $\mathbb{Q}$-espace vectoriel. À renumérotation des indices près, on choisit alors $p < m$ tel que $(a_1, \ldots, a_p)$ soit une base de $\mathrm{Vect}(a_1, \ldots, a_m)$. On va construire un algorithme standard polynomial qui résout X en écrivant les réels manipulés comme des combinaisons linéaires à coefficients entiers des $a_1, \ldots, a_p$, qu'on pourra représenter booléennement par des uplets d'entiers.

À partir de l'écriture $a_i = \sum_{j=1}^{m} \alpha_j a_j = \sum_{j=1}^{p} \beta_j a_j$, on pose N le PPCM des dénominateurs des rationnels α_j et β_j, et on définit les réels $b_i = N a_i$. On effectue une première transformation de $\mathcal{A}$ pour manipuler les b_i à la place des a_i (ceci nous permettra d'obtenir des combinaisons linéaires à coefficients entiers). Pour ce faire, on va :

- réécrire les booléens $\{0,1\}$ (paramètres et sorties des tests) par $0 \mapsto 0$ et $1 \mapsto N$;
- remplacer les a_i d'entrée par les b_i en les multipliant par l'entier N ;
- remplacer les b_i par des combinaisons linéaires à coefficients entiers des $(a_1, \ldots, a_p)$.

L'algorithme $\mathcal{A}'$ obtenu reconnaît bien le même problème étant donné que les tests $a = b$ et $N.a = N.b$ sont équivalents. De plus, l'ajout des multiplications par N ne fait que multiplier le temps d'exécution par une constante, donc $\mathcal{A}'$ est en temps polynomial.

Enfin, on obtient un algorithme booléen $\mathcal{A}_b$ équivalent en remplaçant toutes les combinaisons linéaires $\alpha_0 a_0 + \cdots + \alpha_p a_p$ par des uplets $(\alpha_0, \ldots, \alpha_p)$. En effet, la famille étant libre, l'addition de deux nombres revient à additionner leurs coordonnées deux à deux, l'opposition revient à opposer les coordonnées et la nullité équivaut à la nullité de toutes les coordonnées. On obtient ainsi un algorithme booléen qui résout X.

Reste à vérifier qu'il est polynomial. Les opérations $+$, $-$ et $=$ s'exécutent en temps unitaire pour $\mathcal{A}'$ mais pas pour $\mathcal{A}_b$; toutefois, au cours de l'exécution, les entiers α_i sont majorés par $2^{t(n)}$ car, en une étape de calcul, $\mathcal{A}$ peut au plus doubler la taille du plus grand des α_i. Les opérations $+$, $-$ et $=$ sur les représentations binaires des entiers s'exécutent alors en temps $O(t(n))$, ce qui nous garantit que $\mathcal{A}_b$ est en $O(t(n)^2)$.

In fine, sur $(\mathbb{R}, +, -, =)$ on a $\mathrm{P}_{\mathrm{BSS}} \cap \{0,1\}^* \subset \mathrm{P}$ et, l'inclusion réciproque étant triviale, on a bien l'égalité. En termes de circuits, les opérations réalisées sur les algorithmes reviennent à transformer un circuit algébrique sur $(\mathbb{R}, +, -, =)$ de taille t en un circuit booléen [5] de taille t^2 ayant le même comportement sur les entrées booléennes et on peut donc étendre directement le résultat aux classes non-uniformes : $\mathbb{P}_{\mathrm{BSS}} \cap \{0,1\}^* = \mathbb{P}$. □

Pour illustrer le propos de cette section, on citera également les deux théorèmes suivants qui sont dus à Koiran. Leurs démonstrations respectives sont trop longues pour qu'on se permette de les donner ici, mais le lecteur insatiable pourra les trouver en [3].

Théorème 5 (premier théorème de Koiran)**.** *Sur* $(\mathbb{R}, +, -, =, <)$*, les problèmes booléens de* $\mathrm{P}_{\mathrm{BSS}}$ *sont les problèmes* $\mathbb{P}$ *au sens standard, soit* $\mathrm{P}_{\mathrm{BSS}} \cap \{0,1\}^* = \mathbb{P}$*.*

Théorème 6 (deuxième théorème de Koiran)**.** *Sur* $(\mathbb{R}, +, -, \times, =)$*, les problèmes booléens de* $\mathbb{P}_{\mathrm{BSS}}$ *sont les problèmes* $\mathbb{P}$ *au sens standard, soit* $\mathbb{P}_{\mathrm{BSS}} \cap \{0,1\}^* = \mathbb{P}$*.*

Ces comparaisons avec les classes standard nous indiquent que la puissance des machines BSS dépend essentiellement des opérations dont on dispose. Notamment, les théorèmes 3 et 5 montrent que la présence de l'ordre $<$ permet de calculer bien plus de choses en temps polynomial.

3.2 Liens entre P et NP

Nous nous sommes principalement intéressés aux classes non-uniformes et à la comparaison entre les classes polynomiales standard et généralisées. Toutefois, le modèle

5. Dont les portes sont des entrées ou des opérations booléennes ($\wedge$, $\vee$, $\neg$).

BSS nous permet également d'aborder la question « P = NP ? » sur de nouvelles structures. Le théorème suivant nous montre que l'on sait parfois même l'y résoudre.

Théorème 7. *Dans la structure* $(\mathbb{R}, +, -, =)$*, on a* $\mathrm{P}_{\mathrm{BSS}} \neq \mathrm{NP}_{\mathrm{BSS}}$.

Toutefois, on ne sait pas pour l'instant relier la question standard « P = NP ? » à son analogue sur $(\mathbb{R}, +, -, =)$ et l'intérêt de ce résultat est donc relatif. Par ailleurs, on ne donnera pas la démonstration de ce théorème ici faute de temps et d'espace : le lecteur pourra trouver la preuve en [2].

Fort heureusement, on sait parfois établir un lien direct entre les classes généralisées et leurs analogues standard, comme l'indique le théorème suivant.

Théorème 8 (Fournier, Koiran, 1999)**.** *Dans la structure* $(\mathbb{R}, +, -, <)$*, on a :*

$$\mathrm{P} = \mathrm{NP} \iff \mathrm{P}_{\mathrm{BSS}} = \mathrm{NP}_{\mathrm{BSS}}$$

On se contentera malheureusement de mentionner ce résultat car sa preuve est trop longue et difficile pour être donnée ici. Le lecteur curieux pourra se reporter à [1] pour une démonstration complète.

Remerciements

Je tiens à remercier chaleureusement Sylvain Périfel pour son aide précieuse à la rédaction de cet article.

Références

[1] Hervé Fournier and Pascal Koiran, *Lower bounds are not easier over the reals : Inside ph*, Proceedings of the 27th International Colloquium on Automata, Languages and Programming, vol. 1853, Springer-Verlag, March 2000, pp. 832–843.

[2] Pascal Koiran, *Computing over the reals with addition and order*, Theoretical Computer Science (1994), 35–47.

[3] Bruno Poizat, *Les petits cailloux*, Aléas, 1995.